Optical Micro Disaggregated Data Centers Capable of Executing Many Tasks Simultaneously

Submitted to
Graduate School of Information Science and Technology
Osaka University

December 2024

Akishige IKOMA

List of publication

Journal papers

1. Akishige Ikoma, Yuichi Ohsita, Masayuki Murata, "Resource Allocation Considering Impact of Network on Performance in a Disaggregated Data Center," *IEEE Access*, vol. 12, pp. 67600–67618, May 2024.

2. Akishige Ikoma, Yuichi Ohsita, and Masayuki Murata, "Optical network topology design to execute many tasks simultaneously in a disaggregated data center," *IEEE/OSA Journal of Optical Communications and Networking*, Vol. 16, No.7, pp. 764–780, July 2024.

Refereed Conference Papers

1. Akishige Ikoma, Yuichi Ohsita, Masayuki Murata, "Impact of remote memory and network performance on execution performance of disaggregated micro data centers," in *Proceedings of 2021 International Conference on Emerging Technologies for Communications*, December 2021.

2. Akishige Ikoma, Yuichi Ohsita, and Masayuki Murata, "Disaggregated Micro Data Center: Resource Allocation Considering Impact of Network on Performance," in *Proceedings of 2023 IEEE 20th Consumer Communications Networking Conference (CCNC)*, pp. 360–365, January 2023.

Non-Refereed Technical Papers

1. Akishige Ikoma and Yuichi Ohsita and Masayuki Murata, “Performance evaluation of remote memory utilization method in disaggregated micro data center,” *Technical Reports of IEICE (IN2020-6)*, vol. 120, no. 30, pp. 25–30, May 2020 (in Japanese).

2. Akishige Ikoma and Yuichi Ohsita and Masayuki Murata, “Resource allocation method considering future resource requests in a disaggregated micro data center,” *Technical Reports of IEICE (IN2021-36)*, vol. 121, no. 434, pp. 31–36, March 2022 (in Japanese).

3. Akishige Ikoma and Yuichi Oshita and Masayuki Murata, “Simulator of communication between resources for evaluation of network structure in a disaggregated micro data center,” *Technical Reports of IEICE (IN2023-96)*, vol. 123, no. 398, pp. 183–188, February 2024 (in Japanese).

4. Akishige Ikoma and Yuichi Ohsita and Masayuki Murata, “Dynamic resource management considering workload and network changes in a disaggregated data center,” *Technical Reports of IEICE (IN2024-33)*, vol. 124, no. 180, pp. 1–6, September 2024 (in Japanese).

Preface

In recent years, many services have been provided through cloud computing. However, cloud computing faces high communication delays. Edge computing addresses this problem. This technology deploys small-scale data centers (hereafter called micro data centers) near the users. However, micro data centers have fewer resources (e.g., central processing units (CPUs), and graphic processing units (GPUs)) than those with larger cloud data centers. To execute many service tasks, efficient resource utilization is key.

One approach to enable efficient resource usage is resource disaggregation. Resource disaggregation is the architecture constructed from resources that are connected by a network. In resource disaggregation, resources can be flexibly used by allocating only the required number of resources to each task. Therefore, we configure a micro data center applying resource disaggregation (hereafter called micro disaggregated data center (μDDC)) to execute many tasks simultaneously.

When tasks are executed in a μDDC, communication between allocated resources occurs whenever data are exchanged, which is handled by the motherboard in traditional data centers. Because the frequency of such communication is high, communication delays have a significant impact on service performance. Therefore, communication between resources must be sufficiently low latency to satisfy the performance requirements of tasks. To achieve this, an interconnect using low-latency transmission technology is required. A low-latency interconnect can be achieved by an optical interconnect that enables high-bandwidth communication at the speed of light with negligible jitter. However, if allocated resources are far from each other on the network, communication delays become larger even in a low-latency interconnect. Resources must be allocated to ensure low-latency communication that can satisfy performance requirements. Furthermore, if the resources are far

apart on network topology, it forces establishing a communication path with high latency, regardless of resource allocation method. A network topology to enable resource allocation that can satisfy performance requirements is required. In this thesis, we propose resource allocation methods and network topology in a μDDC with an optical network.

First, to execute many tasks simultaneously, we propose a resource allocation method to preserve resources required for future tasks while satisfying the performance requirements. We call this method RA-CNP. First, we model the impact of the network on the performance of tasks and verify whether the resource allocation can satisfy the performance requirements. We also define resource allocation costs for each resource based on whether it is necessary for future task execution. RA-CNP allocates resources to minimize costs while satisfying performance requirements. We demonstrate that RA-CNP can reduce task blocking to 0 even in environments where task blocking occurs in conventional methods.

Next, we propose an optical network topology that enables flexible resource allocation. First, we define a metric for evaluating an optical μDDC network called the capability of simultaneous task execution (CSTE). CSTE represents the ratio of resources that could be used as a resource communicating with other resources without violating the performance requirements in a situation where tasks up to the maximum number of executable tasks are executed. We formulate an optical topology design problem aimed at generating an optical network topology capable of maximizing the number of tasks that can be executed simultaneously based on CSTE. We demonstrate that an optimal network topology based on CSTE reduces task blockages by over 50% compared to conventional topologies.

Finally, we extend RA-CNP to estimate the suitable resources for each task. We focus on the deep learning-based tasks that can be partitioned and executed in parallel and extend the RA-CNP so that it optimizes the number of resources allocated to each task by optimizing the partitioning. We call this extended version of RA-CNP the resource aware model partitioning and allocation (RAMPA). First, we extend the model of the impact of the network on performance to consider the impact of model partitioning on performance. We also extend the resource allocation in RA-CNP to determine the combination of model partitioning and resource allocation that minimizes the resource allocation costs while satisfying the service performance requirements. We demonstrate that RAMPA can execute more tasks in any environment and improve the number of executed tasks by up to 30% compared to conventional methods.

Acknowledgments

This thesis could not have been accomplished without the assistance of the people who have supported me in many ways. I would like to take this opportunity to express my deepest appreciation to them.

First of all, I express my deepest gratitude to my supervisor, Professor Masayuki Murata of Graduate School of Information Science and Technology, Osaka University, for his valuable guidance and encouragement throughout my research career. His creative and insightful suggestions and guidance have made my research life very meaningful.

I am heartily grateful to the members of my thesis committee, Professor Takashi Watanabe, Professor Hirozumi Yamaguchi of Graduate School of Information Science and Technology, Osaka University, and Professor Hideyuki Shimonishi of D3 Center, Osaka University, for their multilateral reviews and perceptive comments.

Furthermore, I would like to express my special thanks to Associate Professor Yuichi Ohsita of D3 Center, Osaka University, for his continuous warm and enthusiastic guidance. Many of the suggestions he gave me provided new insights and perspectives that were crucial to my research.

Moreover, I am heartily grateful to Associate Professor Shin'ichi Arakawa, Associate Professor Daichi Kominami, Assistant Professor Tatsuya Otoshi, and Assistant Professor Masaaki Yamauchi of Osaka University for their appreciated comments and support. Their kind support has made my research fun and fruitful.

I am grateful to all of past and present colleagues, friends, and secretaries of the Advanced Network Architecture Research Laboratory, Graduate School of Information Science and Technology, Osaka University. Discussions with them have made my research life exciting.

I would also like to express my gratitude to my partner who always encouraged and supported me when my research was not going well.

Finally, I would like to thank my family for their warm support and many opportunities throughout my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

In recent years, numerous services have been provided through cloud computing. However, cloud computing faces several problems such as high communication delays and increased network traffic. These problems are severe for time-sensitive services and processing data close to service users is important. Edge computing is a solution to address these problems [1]. In edge computing, small-scale data centers (hereafter called micro data centers) are deployed near service users. Micro data centers provide edge service providers with computing environments that include computing, storage, networking equipment, and management systems such as power supply and cooling. Their scale ranges from installations of 1 rack to installations of container type consisting of about 10 racks [2–4]. In this thesis, we focus on a micro data center with up to 16 racks considering increasing demand for edge services in the future.

In micro data centers, multi-tenancy, where service providers share resources in a data center for service delivery, is required because it is costly to manage their own edge infrastructures [5]. To provide each service, a series of associated internal processes (hereafter called tasks) must be executed in the micro data center. Thus, multiple tasks must be executed simultaneously for multi-tenancy. However, micro data centers are small-scale, and available resources (e.g., central processing units (CPUs), graphic processing units (GPUs) and memory) are limited compared to larger cloud data centers. To execute many service tasks simultaneously in a micro data center, limited resources must be fully utilized. Unfortunately, efficient use of limited resources is impossible in today's common data center architectures (hereafter called traditional data centers). Traditional data centers consist of multiple servers. Within each server, resources are tightly connected. Due to this architecture, whether the resource is available for tasks depends on other resources in the same server. For example, we consider the case where tasks requiring 2 CPU cores and 3 GB of memory are executed in a traditional data center configured by two servers with an 8-core CPU and 8 GB memory. In this traditional data center, only 2 tasks can be executed on a server because it does not have the necessary memory to execute 3 or more tasks. Therefore, the maximum number of tasks that can be executed in this data center is 4 even though all the resources that this data center possesses are sufficient to execute more tasks. An architecture that enables efficient use of resources is required.

To address this problem, resource disaggregation has been proposed [6]. In resource disaggregation, resources are decoupled from each server and connected by a network, as shown in Fig. 1.1. Therefore, we can flexibly allocate only the resources required for each task without considering other resources and achieve higher resource utilization than traditional data centers [7, 8]. To execute many service tasks simultaneously, we configure a micro data center applying resource disaggregation (hereafter referred to as a micro disaggregated data center (μDDC)).

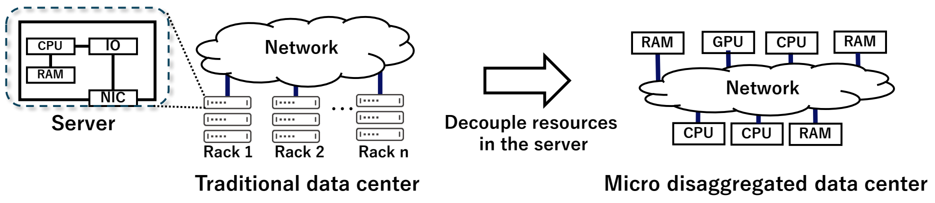


Figure 1.1: Transition from traditional data center to micro disaggregated data center.

While μDDCs have the advantage of efficient resource use, there are several challenges. Because resources are connected via a network, communication between allocated resources is required to execute tasks. Communication between allocated resources occurs whenever data are exchanged, which is handled by the motherboard in traditional architectures. Because the frequency of such communication is high, communication delays have a significant impact on service performance. Gao et al. demonstrated that communication in a few microseconds is necessary for resource disaggregation [9]. The communication delay between resources depends on the technology used for the interconnect and the location of the allocated resources in the network. Therefore, these must be considered for μDDCs to execute many tasks simultaneously.

Optical interconnects are one promising technology for achieving low latency communication [10–12]. These studies have proposed interconnects using optical circuit switches and optical packet switches and demonstrated that communication between resources in the order of nanoseconds can be achieved.

However, low latency communication between resources cannot be achieved by only optical interconnect. If allocated resources are far from each other on the network, communication delays become larger even in low-latency networks. Furthermore, if many resources are allocated for tasks and only far apart resources are available for newly requested tasks, it forces communicating with high latency. Therefore, a resource allocation method to execute many tasks while satisfying performance requirements is required. Several resource allocation methods for resource disaggregation architecture have been proposed [7, 13, 14]. These methods aimed to minimize communication delays regardless of the performance requirements of the task. However, their objectives are insufficient to execute many tasks simultaneously. For example, tasks with short time constraints are requested after resources are allocated for tasks with long time constraints. They allocate resources that can communicate in fewer hops even for tasks with long time constraints. As a result, resources for future tasks with short time constraints may be forced to communicate with larger latency. To address this problem, we focus on whether the resource allocation satisfies the performance requirements, that have been overlooked in conventional methods. If the performance requirements are satisfied, the performance degradation of the task is not a problem. By expanding the solution space through this consideration, we optimize the allocated resources to execute many tasks simultaneously. In this thesis, we allocate resources to preserve resources required by future tasks while satisfying the performance requirements.

Resource allocation depends on the network topology. If the resources are far apart on network topology, it forces establishing a communication path with high latency, regardless of resource allocation method. A network topology that enables resource allocation to execute many tasks while satisfying performance requirements is required. Several network topologies for resource disaggregation architecture have also been proposed [10, 15]. These topologies also aimed to minimize communication delays between allocated resources. However, they do not consider the number of resources that can establish a communication path that can satisfy the performance requirements. Resources that can establish a communication path with low latency may be exhausted in these topologies. To address this problem, we focus on whether resource allocation that can satisfy the performance requirements can be achieved in the network topology. By this consideration, we can optimize network topology based on objectives other than minimizing communication delays. In this thesis, we design an optical network topology with many resource pairs that can simultaneously establish communication paths that satisfy performance requirements.

1.2 Outline of thesis

Fig. 1.2 shows the overview of this thesis. To execute many tasks simultaneously, we need to allocate suitable resources to each task. That is, we need to estimate the required amount of resources and allocate physical resources to each task. In addition to the resource allocation method, the network topology is also important because resource allocation depends on the network topology. In this thesis, we construct a μDDC that can execute many tasks simultaneously by developing resource estimation and allocation methods and a suitable network topology.

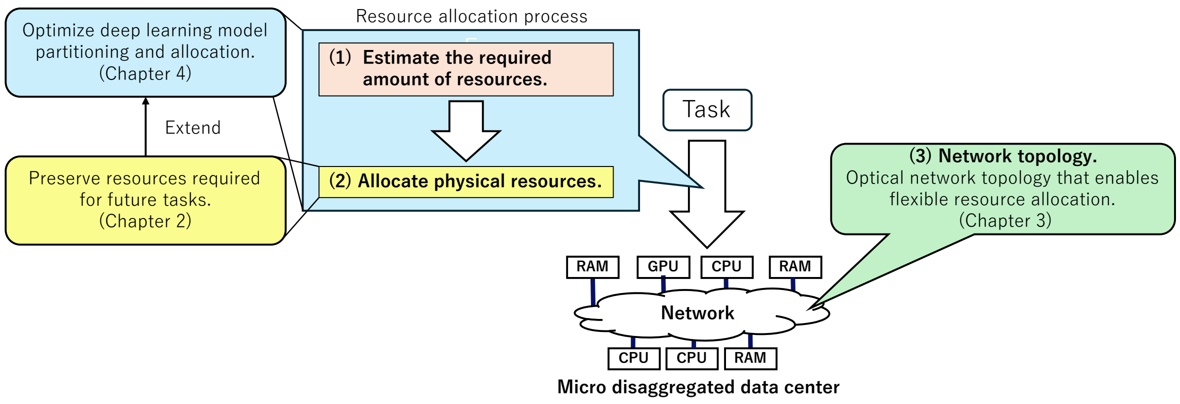

Optimize deep learning model partitioning and allocation. (Chapter 4)
Extend
Preserve resources required for future tasks. (Chapter 2)
Resource allocation process
(1) Estimate the required amount of resources.
(2) Allocate physical resources.
Task
(3) Network topology. Optical network topology that enables flexible resource allocation. (Chapter 3)
RAM
GPU
CPU
RAM
Network
CPU
CPU
RAM
Micro disaggregated data center

Figure 1.2: Overview of this thesis.

Resource Allocation Considering Impact of Network on Performance in Micro Disaggregated Data Center [16–19]

We first propose a resource allocation method for a μDDC in Chapter 2. We call this method RA-CNP. RA-CNP preserves resources required for future tasks while satisfying the performance requirements of requested tasks. First, we model the impact of the network on the performance of tasks and verify whether the resource allocation can satisfy performance requirements. Furthermore, we define resource allocation costs for each resource based on whether it is necessary for future task execution. Based on these, RA-CNP allocates resources that minimize resource allocation costs while satisfying the performance requirements. We evaluate the effectiveness of our method by simulating various μDDC networks. The results demonstrate that RA-CNP can reduce task blocking to 0 even in environments where task blocking occurs in conventional methods.

Optical Network Topology Design to Execute Many Tasks Simultaneously in Micro Disaggregated Data Center [20]

In Chapter 3, we address a suitable network structure for a μDDC. In this chapter, we propose an optical network topology that enables flexible resource allocation. First, we define a metric for evaluating an optical μDDC network called the capability of simultaneous task execution (CSTE). CSTE represents the ratio of resources that could be used as a resource communicating with other resources without violating the performance requirements in a situation where tasks up to the maximum number of executable tasks are executed. μDDC with high CSTE can have a large number of resource pairs capable of communicating satisfying task performance requirements and flexible resource allocation is possible. We formulate an optical topology design problem aimed at generating an optical network topology capable of maximizing the number of tasks that can be executed simultaneously based on CSTE. By solving this problem, we generate optical network topologies and validate their effectiveness via resource allocation simulations. We evaluate the effectiveness of the proposed network topology design by resource allocation based on RA-CNP. The results demonstrate that an optimal network topology based on CSTE reduces task blockages by over 50% compared to conventional topologies.

Resource Aware Deep Learning Model Partitioning and Allocation to Execute Many Deep Learning Tasks

Finally, we extend RA-CNP to estimate the suitable resources for each task in Chapter 4. In this chapter, we focus on the deep learning-based tasks that can be partitioned and executed in parallel and extend the RA-CNP so that it optimizes the number of resources allocated to each task by optimizing the partitioning. We call this extended version of RA-CNP the resource aware model partitioning and allocation (RAMPA). First, we extend the model of the impact of the network on performance to consider the impact of model partitioning on performance. We also extend the resource allocation in RA-CNP to determine the combination of model partitioning and resource allocation that minimizes the resource allocation costs while satisfying the service performance requirements. By these extensions, RAMPA can minimize the allocation of resources required for future tasks. We evaluate the effectiveness of RAMPA by simulating the deep learning tasks execution. The results demonstrate that RAMPA can execute more tasks in any environment and improve the number of executed tasks by up to 30% compared to conventional methods.

Chapter 2

Resource Allocation Considering Impact of Network on Performance in Micro Disaggregated Data Center

2.1 Introduction

In recent years, numerous services provided via cloud computing have emerged. However, cloud-based services are associated with problems such as latency and dense network traffic. Edge computing addresses these problems [1]. This technology deploys micro data centers near the users. Because these data centers are located near the user and can process data locally, they are effective for time-sensitive services such as automated driving and face recognition [21]. The number of edge devices is expected to increase further in the future, and a data center on the edge must execute more service tasks [21]. Nevertheless, micro data centers have fewer resources than those with larger cloud data centers. Therefore, optimal resource utilization for each task is key [22].

Flexible resource allocation via infrastructure virtualization and optimization of resource utilization have been considered potential ways to achieve this goal [23]. However, in traditional architectures where resources such as CPU and memory are aggregated on a server, per-resource flexible management is limited [24]. For example, if four tasks requiring 2 cores and 4 GB of memory are allocated to a server with a 16-core CPU and 16 GB of memory, 8 CPU cores are unavailable for other tasks owing to the absence of available memory resources. One approach to solving such inefficient resource usage is resource disaggregation [6]. Resource disaggregation refers to the use of an architecture constructed from resources such as CPUs and memory that are connected by a network. The resources in this architecture can be easily upgraded and flexibly used by allocating only the required number of resources to each task. This can be done because each resource is independent, in contrast to the case of a traditional data center where resources are aggregated into servers [9]. Owing to these advantages, resource disaggregation has been considered in various areas, such as serverless computing, big data processing, and database processing [25–27]. Therefore, we focused on a micro data center applying resource disaggregation (hereafter referred to as a micro disaggregated data center (μDDC)) and aim to configure a μDDC that can execute many service tasks simultaneously.

When a task for a service is executed in a data center applying resource disaggregation (DDC), after the required resources to execute the task are selected, the task must be executed via communication between the selected resources. Because resources are connected by a network, the task execution time increases with the duration of the communication delay between resources. In particular, communication delay between the CPU and memory has a significant effect on task execution time [9]. A DDC may not be able to provide the service in the required time because of this problem. Therefore, a DDC consisting of an optical network that has been configured with optical circuit switches and optical packet switches and that enables communication with low latency and high bandwidth has been proposed [10, 12]. In this DDC, resource disaggregation has been demonstrated to be more effective than traditional architectures in terms of resource utilization and energy consumption. Furthermore, in [28], resource disaggregation via optical interconnects was evaluated using actual equipment. Resource disaggregation is a feasible approach for improving resource utilization.

However, achieving efficient resource utilization in a DDC only by improving network performance is difficult. Communication delay between resources also depends on routing between resources [7]. The resource allocation method, which determines the CPUs and memories that execute tasks and the network links that constitute the communication path, is also important for a DDC. In particular, allocation of path has a significant impact on the performance of tasks [13]. If the path between the CPU and memory to execute a task has many hops, it takes more time for the CPU to retrieve data from memory. Furthermore, if a communication path with high traffic is allocated as a path between resources, congestion may occur. In these cases, the time required to complete a task increases, and the performance requirements of tasks may no longer be satisfied. As a result, the execution of many tasks becomes more difficult. Because this problem can occur in any network configuration, an efficient resource allocation method considering the impact of the network on the performance of tasks is required for a DDC.

However, only considering the impact of the network on the performance of tasks is not sufficient to execute many tasks. For example, we assume a case where low latency communication paths are allocated regardless of the performance requirements of tasks to minimize performance degradation. Because available network links are not infinite, available paths for tasks with strict time constraints may be exhausted even if there are available resources for task execution. If tasks with long time constraints avoided using communication path with low latency, this situation could have been prevented. Preserving the resources required by future tasks considering the impact of the network on the performance of tasks and performance requirements is important.

We emphasize that resource allocation methods for a traditional data center are insufficient for resource allocation in a DDC. In a DDC, communication delays can occur just as a computational resource, such as a CPU or GPU, reads data from memory, resulting in increased task execution time. By contrast, because resources are connected on the motherboard in traditional data centers, network communication is absent between resources involved in task execution. Because of this difference, a resource allocation method for a DDC is required. Resource allocation methods have been proposed for a DDC [7,13,14,29], but these methods do not consider the impact of the network on the performance of each task and future tasks. Instead, they allocate communication path to minimize performance degradation. Because it is important to preserve the resources required for future tasks, these methods are not sufficient to execute many tasks in a μDDC.

We propose a resource allocation method that considers the impact of the network on performance (hereafter called RA-CNP). We model the impact of the allocated resources on the time required to complete a task based on the communication delay between execution resources. In addition, when multiple candidate resources exist, we avoid allocating high-performance and low-latency communication path that may be requested in the future to execute more tasks. We define the resource allocation cost in terms of resource importance; moreover, we formulate a resource allocation problem to satisfy the performance requirements of a task and to select the candidate with the smallest cost. By not using resources with high costs, a μDDC preserves those that can be used in future task requests, thereby executing more tasks. Then, we evaluated the effectiveness of RA-CNP by comparing it with other methods in networks configured by circuit and packet switches. Moreover, to verify optimal solution to the defined resource allocation problem, we also evaluated the optimal solution for RA-CNP. Finally, we investigated whether RA-CNP can allocate resources within a practical computation time.

The main contributions of this chapter are as follows:

- We modeled the impact of network on task performance in a general form.
- We proposed a resource allocation method to execute many tasks simultaneously, RA-CNP, based on our model and the resource allocation problem.
- We demonstrated that RA-CNP can enable the completion of many tasks within their acceptable time and can allocate resources in a practical computation time.

The remainder of this chapter is organized as follows: Section 2.2 discusses related work. Section 2.3 examines the impact of the network on the performance of tasks in a μDDC. Section 2.4 provides an overview of the resource allocation method. Section 2.5 discusses the simulations used to evaluate the effectiveness of RA-CNP and the computational time. Finally, Section 2.6 concludes this chapter.

2.2 Related work

A DDC is constructed using resources such as a CPU, a GPU, and memory connected by a network. Resource disaggregation improves resource utilization and scaling flexibility [30]. After the necessary execution resources are determined, a task executed in a DDC is processed via communication between the selected resources.

A DDC architecture must consider the following aspects: (1) processing tasks in a DDC, (2) connection of resources, and (3) allocation of resources. In the remainder of this section, we discuss existing reports on DDC architectures.

2.2.1 Processing system

In a DDC, resources are distributed. A system is required to manage these resources and execute tasks.

LegoOS has been proposed as an operating system for resource disaggregation [31]. This system divides operating system functions according to each disaggregated resource and manages them in a decentralized manner. Furthermore, the operating system demonstrates compatibility with Linux and the feasibility of application deployment. This system can be used to run existing applications and feasibly implement a DDC.

2.2.2 Resource connection

In a DDC, performance degradation due to communication delays between resources is significant, and nanosecond resource communication is required [9]. Therefore, DDCs require high-bandwidth and low-latency switches to reduce performance degradation.

Optical switching has been proposed to enable high-bandwidth and low-latency communication [10–12, 32]. Zervas et al. proposed a network architecture for a DDC using optical circuit switches [10]. Owing to the configuration of the optical circuit switches, the resources could communicate at low latency. The researchers demonstrated that the blocking rate for resource requests was lower than that of traditional data centers. Yan et al. also proposed a disaggregated architecture configured by an optical circuit switch for machine learning and demonstrated that optical interconnection can improve the utilization of disaggregated resources [32]. Optical circuit switches must establish a direct connection between input and output ports for communication between resources. Because of this characteristic, the path between resources is dedicated to that resource pair. DDC configured by optical packet switches as well as optical circuit switches has been studied [11, 12]. Terzenidis et al. proposed a network for a DDC configured by optical packet switches [11]. Switching delays were reduced to nanoseconds, demonstrating the feasibility of resource disaggregation using packet switches. Guo et al. proposed a DDC architecture based on hybrid switches, including an optical circuit switch with many ports and an optical packet switch with few ports, and they achieved efficient resource utilization [12]. Because packet switches can route data to the appropriate port based on the destination address of the packets, the connection between input and output ports is not fixed. Therefore, the path between resources is not dedicated, and a network link can be used for communication between multiple resources. However, the latency between resources is greater than that in optical circuit switch networks.

Resources connected via optical networks can communicate with low latency, thereby reducing performance degradation. However, if resource allocation is inefficient, the number of tasks that can be executed is limited, even on an optical network. Fig. 2.1 shows examples of inefficient and efficient resource allocation. This example assumes that tasks with short time constraints are requested after resources are allocated for tasks with long time constraints. In the case of inefficient resource allocation, resource pairs that can communicate in fewer hops are allocated for tasks with long time constraints. Because of this, the next requested task is forced to use resource pairs that require many hops to communicate. As a result, all the requested tasks cannot be executed. In the case of efficient resource allocation, avoid allocating resource pairs that can communicate in fewer hops because tasks with long time constraints do not necessarily require minimizing performance degradation due to communication between resources. As a result, tasks with short time constraints can also be executed because resource pairs that can communicate at the lowest hop exist. Thus, when available resources are severely limited due to inefficient resource allocation, future tasks may be forced to utilize resource pairs that cannot satisfy performance requirements. We emphasize that this can occur regardless of network architecture. This is because available resources and network links are finite, regardless of network architecture. Individual execution resources can only handle a limited number of tasks. Then, available network links are not also infinite due to bandwidth constraints or dedicated network links (in the case of a network with optical circuit switches). Therefore, when many tasks are executed, available resources are reduced, and flexible selection of execution resources becomes difficult in any network. This can create a situation where inefficient allocation of a task inhibits the allocation of other tasks, as shown in Fig. 2.1. An efficient resource allocation method considering the impact of the network on performance and future tasks is required to execute many tasks simultaneously for a DDC.

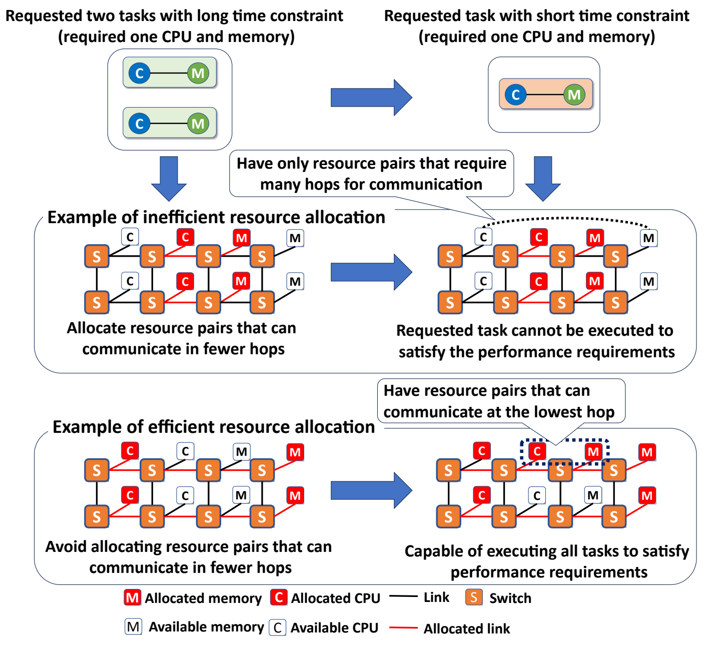

Requested two tasks with long time constraint (required one CPU and memory)
Requested task with short time constraint (required one CPU and memory)
Have only resource pairs that require many hops for communication
Example of inefficient resource allocation
Allocate resource pairs that can communicate in fewer hops
Requested task cannot be executed to satisfy the performance requirements
Have resource pairs that can communicate at the lowest hop
Example of efficient resource allocation
Avoid allocating resource pairs that can communicate in fewer hops
Capable of executing all tasks to satisfy performance requirements
Allocated memory
Allocated CPU
Link
Switch
Available memory
Available CPU
Allocated link

Figure 2.1: Example of inefficient resource allocation and efficient resource allocation.

2.2.3 Resource allocation

To execute a service task, the resources that will be used to run the task and the paths that will be used to communicate between those resources must be determined. Although resource allocation methods have been proposed for traditional data centers, they are not suitable for DDCs. Because resources are connected on the motherboard in traditional data centers, network communication is lacking between resources involved in task execution. On the other hand, in a DDC, network

Table 2.1: Objective and approach of each resource allocation method for disaggregated data center.

Author	Objective	Approach
Papaioannou et al. [13]	Minimize performance of impact of network latency in requested task	Minimize latency between resources for requested task based on bandwidth, hops, delay of link
Zervas et al. [7]	Minimize round trip latency between execution resources in requested task	Minimize latency between resources for requested task based on bandwidth and link distance
Amaral et al. [14]	Minimize execution time of requested task while avoiding performance degradation of running tasks	Minimize performance interference of running tasks and requested task considering the execution time calculated based on network load
Guo et al. [29]	Maximize requests that satisfy the failure probability requirement of allocated resources in a given set of resource allocation requests	Minimize resources used as backup in case of failure of resources considering the failure probability requirement of requests
Ikoma et al. (RA-CNP)	Minimize use of resources required for future requested tasks	Avoid allocating resources and path required for future tasks considering the impact of the network on performance based on communication delay between execution resources

communication between resources occurs when tasks are executed. Because task execution time is affected by the communication delay between resources, a resource allocation method for a DDC considering this aspect is required [14].

Several resource allocation methods have been proposed for a DDC [7, 13, 14, 29]. The characteristics of each method, in terms of objective and approach, are shown in Table 2.1. We analyze whether existing studies are sufficient to execute many tasks while satisfying performance requirements based on Table 2.1. Papaioannou et al. proposed a resource allocation to minimize the performance degradation of requested tasks by minimizing a metric based on the bandwidth and latency of paths [13]. The authors demonstrated that this method can improve resource utilization without affecting task performance in a DDC configured by an optical circuit switch and an electrical packet switch. Zervas et al. proposed a resource allocation method to minimize round-trip latency between execution resources in requested tasks by minimizing a metric based on bandwidth and link distance [7]. The authors demonstrated that this method can improve resource utilization in a DDC configured with optical circuit switches. These methods [7, 13] consider only the task requested at that time and preferentially allocate resources that can communicate with low latency regardless of the performance requirements of the task. Resources required for tasks with strict performance requirements, where low-latency communication is essential, may soon be depleted.

As a result, many tasks cannot be executed. Amaral et al. proposed a resource allocation method to minimize the execution time of requested tasks while avoiding performance degradation of running tasks [14]. This method prevents performance interference between running tasks and the requested task, considering the execution time calculated based on network load. In this method, resources are allocated to minimize the execution time of the requested task at that time. Therefore, it does not also consider future tasks. Furthermore, this method calculates execution time directly from network load by using statistical data. Therefore, this method cannot consider the impact of communication delays between resources on performance. It is not possible to optimize routing between resources while considering task performance. Guo et al. proposed a resource allocation method to maximize requests that satisfy the failure probability requirement of allocated resources in a given set of resource allocation requests [29]. This method enables higher resource utilization while guaranteeing the reliability of tasks. However, this method assumes that all resource allocation requests are given in advance. When new tasks are requested in a situation where multiple tasks are executed, there may be no available resources for those tasks. Furthermore, this method does not consider performance degradation due to communication between resources. It is difficult to execute many tasks while satisfying performance requirements.

To execute many tasks simultaneously, consideration for future tasks is essential. If resources are allocated without considering whether future tasks can be executed with the required performance, available resources to satisfy the requirements of new tasks may be exhausted soon. In reality, allocating resources to ensure the best performance of running tasks or requested tasks is not always necessary, as in the efficient allocation in Fig. 2.1. Any resources are sufficient if the performance requirements of the task are satisfied. By avoiding unnecessarily allocating resources and network links required for future tasks, more tasks can be executed. Note that we must consider whether the performance requirements of tasks are satisfied. Existing methods did not sufficiently consider it. Instead, they simply allocate resources to maximize the performance of tasks. In this chapter, we propose a resource allocation method for a μDDC to execute many tasks simultaneously while satisfying performance requirements. This method allocates resources to minimize the use of resources required for future requested tasks. Furthermore, it models the impact of the network on performance based on the communication delay between execution resources. Thus, this method can allocate resources and network links, considering both task performance and future tasks. The major difference from existing methods is the consideration of future resource allocation based on the impact of the network on performance rather than simply minimizing performance degradation to execute many tasks simultaneously.

2.3 Impact of network on performance of tasks in micro disaggregated data center

In this section, we examine the impact of the network on the performance of tasks in a μDDC. We emulate the execution of tasks by CPUs and memories connected via a network.

2.3.1 Experimental Settings

We assume a μDDC that CPU and memory are connected by a network. Each CPU has a small cache and executes tasks using the data in the cache. When the required data don't exist in the cache, the CPU obtains the data from memory. Furthermore, we assume that the μDDC uses a paging technique. That is, if required data cannot be found in the cache, the CPU obtains the page that contains the required data from the memory. We examine the impact of the communication that occurs at this time on the execution task performance.

Experimental environment

We implement an environment for emulating a μDDC. This environment contains a CPU with a cache and a memory. Although they are implemented in a single computer, we add communication delays when the CPU accesses memory, to emulate a μDDC where communication delays occur when accessing memory. This environment is constructed by creating a swap device that acts as a disaggregated memory based on the program proposed by Gao et al. [9]. This swap device makes a process wait for the communication delay and then handles the request. In this experiment, this swap device is created by using the memory installed in the computer. We allocate 200 MB of memory as the cache attached to the CPU and the rest of the memory is used as a swap device.

A computer with an Intel(R) Xeon(R) CPU E5-2687W and 64 GB DDR3 SDRAM is used for emulation and the page size is set to 4 KB.

Execution tasks

We use ResNet and Inception-v3, and execute their inference step by using TensorFlow. We execute each of these processes 10 times and measure the time from inputting one image to obtaining the output as the processing time.

Metrics

We measure the performance degradation rate as a metric for the impact of network performance. The performance degradation rate is $\frac{T^{\text{disaggregated}}-T^{\text{traditional}}}{T^{\text{traditional}}}$, where $T^{\text{disaggregated}}$ is the process time of μDDC and $T^{\text{traditional}}$ is the process time of the traditional computer. We obtain $T^{\text{disaggregated}}$ by executing the process in the experimental environment and $T^{\text{traditional}}$ by executing the process on the same computer without adding communication delay.

2.3.2 Results

We first investigate the impact of bandwidth by setting the bandwidth to 40 Gbps and 100 Gbps with the latency set to 8 μs. We next investigate the impact of the latency. We set the latency to 0.2 μs, 2 μs, and 8 μs with the bandwidth set to 100 Gbps.

Impact of bandwidth on performance

Figure 2.2 shows the impact of the network bandwidth on the performance of μDDC. This figure shows all performance degradation rates monitored in our experiment. The median values of the performance degradation rates are also shown as a bar graph.

ResNet-50 is a relatively small model and the number of page faults is small. As a result, the impact of the communication delay is small. However, the communication delay has a large impact on the other models. The median of the performance degradation rates is over 10 % for all models except ResNet-50. We can discuss the impact of the bandwidth by comparing the performance degradation rates in the cases of 40 Gbps and 100 Gbps. Fig. 2.2 shows increasing the bandwidth from 40 Gbps to 100 Gbps does not reduce the performance degradation rate. That is, the bandwidth has only a small impact on the performance of the task.

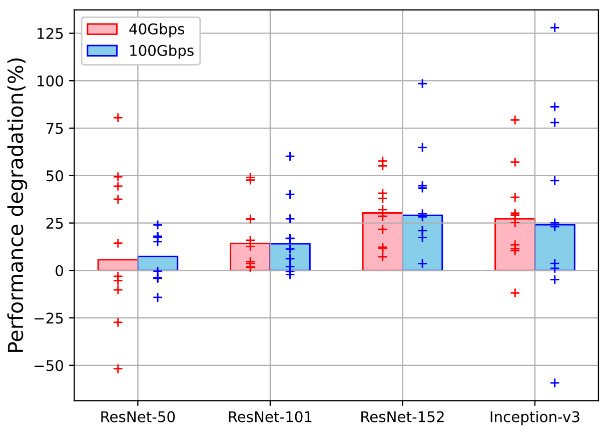


Figure 2.2: Impact of network bandwidth on performance.

Impact of latency on performance

Fig. 2.3 shows the impact of the latency on the performance of the μDDC. Like Fig. 2.2, all performance degradation rates monitored in our experiment are shown in this figure. The median values of the performance degradation rates are also shown as a bar graph.

Fig. 2.3 shows that the performance degradation rate decreases as the latency decreases. In all models, we can decrease the performance degradation rate by half by reducing the latency from 8 μs to 0.2 μs. That is, unlike the bandwidth, the latency has a large impact on the performance. Fig.2.3 also indicates that the performance degradation rate depends on the model. The performance degradation rate of a small model like ResNet-50 is small, while the performance degradation rate of a relatively large model like ResNet-152 is large.

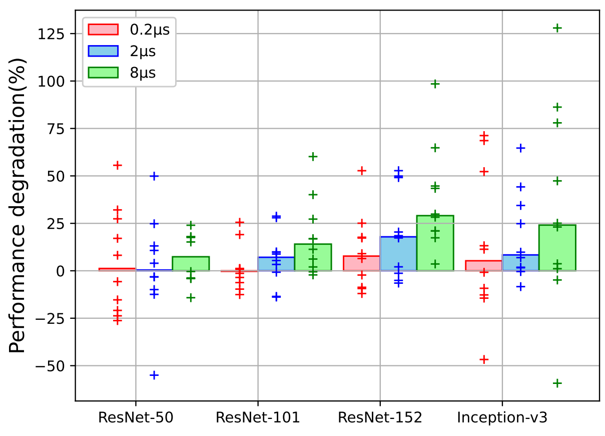


Figure 2.3: Impact of latency on performance.

2.3.3 Discussion

We showed that communication delay to obtain the data from the memory degrades task performance in a μDDC. Therefore, we verify the proportion of time to obtain the data from the memory in the total execution time of a task. Table 2.2 compares the ratio of the total time used to obtain the data from the memory. This shows that more than double the time is required to obtain the data when the latency is 8 μs, compared with the case of 0.2 μs. This comparison correlates well with the comparison of the performance degradation rates. That is, the time to obtain the required data is the cause of the performance degradation. Furthermore, our results indicate that the computational resources are also important. Even in the worst case, 3/4 of the process time is consumed by the CPU while the memory access only consumes 1/4 of the process time. The execution time of a task in a μDDC is dominated by the execution time in the computational resources and the time to obtain data from memory.

Table 2.2: Ratio of total time spent reading from memory to execution time for each model.

	0.2μs	2μs	8μs
ResNet-50	7.94%	10.15%	18.44%
ResNet-101	10.96%	14.94%	27.73%
ResNet-152	12.50%	16.16%	27.31%
Inception-v3	9.93%	12.31%	22.24%

2.4 Resource allocation considering impact of network on performance (RA-CNP)

In this section, we model the impact of the network on performance. Thereafter, we formulate the resource allocation problem and present an example of a method for addressing it.

2.4.1 Overview of micro disaggregated data center

We show the assumed μDDC in this section.

Components of micro disaggregated data center

We assume a μDDC in which the memory and computational resources (CPUs and GPUs) are connected by a network, as shown in Fig. 2.4. This μDDC includes resource pools, in which several resources of the same type are collected. Each resource pool is connected via packet or circuit switches. The components of a μDDC are as follows:

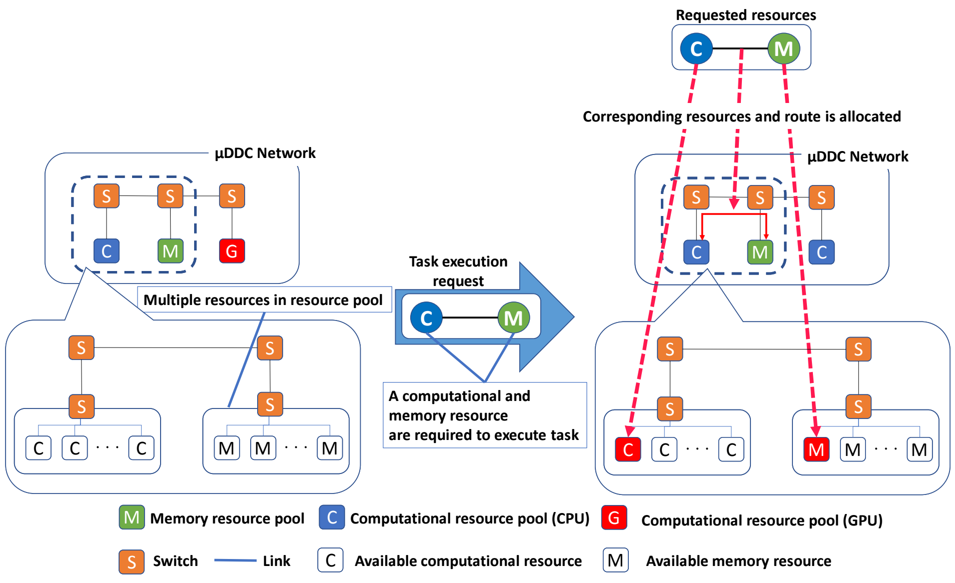

Requested resources
Corresponding resources and route is allocated
µDDC Network
µDDC Network
Multiple resources in resource pool
Task execution request
A computational and memory resource are required to execute task
Memory resource pool
Computational resource pool (CPU)
Computational resource pool (GPU)
Switch
Link
Available computational resource
Available memory resource

Figure 2.4: Overview of micro disaggregated data center.

Memory resources A memory resource is a device that stores the data required by computational resources. We divide memory into blocks and treat each block as a memory resource. Data in memory are managed via paging.

Computational resources (CPUs and GPUs) A computational resource has a small cache. When the data required to execute a process do not exist in the cache and a page fault occurs, the computational resource obtains the data from the memory resource. Data are transmitted at the granularity of a page. In a μDDC, the data read time from disaggregated memory resources is longer than the data read time from the cache. Therefore, the impact of the latter on performance is negligible. In this chapter, cache levels and the latency between a cache and computational resources are not considered. Each CPU core, or GPU, is treated as a single computational resource.

Network The network consists of resource pools and switches that connect them. Links connect pools to switches and one switch to another.

A resource pool holds multiple resources of the same type. A computational resource pool holds multiple computational resources, and a memory resource pool holds multiple memory resources.

Each resource in the pool connects to a switch via a common link between the pool and the switch.

Optical circuit switches, or optical packet switches, are used in the μDDC. In a network configured with optical circuit switches, once a path for optical signals is established to execute a task, the path is occupied by the task. Therefore, each optical circuit switch can immediately relay data to the next port without blocking, according to prior routing. In a network configured with optical packet switches, each switch has a buffer. If the next port is available, the switch immediately relays the packet to it before the entire packet is received. If the next port is busy, the switch stores the packet in its buffer to prevent communication that exceeds the bandwidth. The switch then waits for the next port to become available. We allow the construction of an aggregated virtual link from multiple links between the same nodes. Aggregating the links can reduce the delay, even if some links in the aggregated link are busy. The switch can still relay the packet without storing it in the buffer, as long as it has at least one link available.

Execute task in micro disaggregated data center

We assume that users randomly request the execution of tasks for services provided by a μDDC. When a μDDC receives a task execution request, it allocates the computational and memory resources required to execute the task from among the available resources in the resource pool shown in Fig. 2.4. Thereafter, it determines the communication path between the allocated resources. We treat this process as resource allocation. In this thesis, one resource can execute up to one task, and each resource runs in parallel. The waiting time for task execution does not occur.

Multiple types of processing with different resource requirements may be required to complete a task. In this chapter, we divide tasks into processes according to the need to use resources flexibly. A task utilizes a set of processes that are allocated by selecting the necessary resources for each process.

2.4.2 Modeling micro disaggregated data center and resource allocation request

The notations used for modeling a μDDC and resource allocation request are listed in Table 2.3.

Table 2.3: Notation of micro disaggregated data center and resource allocation request.

Symbols	Definition
	μDDC network
N^s	Set of nodes
E^s	Set of links
N^c	Set of computational resource pools
N^m	Set of memory resource pools
C_c^s	Number of available resources in the computational resource pool c
M_m^s	Number of available resources in the memory resource pool m
R^s	Number of resource pools
K_c	Performance metric (FLOPS) of resources in the computational resource pool $c \in N^c$
F_c	Clock frequency of resources in the computational resource pool $c \in N^c$
V_c	Page size of resources in the computational resource pool $c \in N^c$
$R_{i,j}$	Set of configurable paths between nodes $i, j \in N^s$
B	Bandwidth of μDDC link
T_n^N	Delay to send the entire packet in node $n \in N^s$
T_n^I	Delay to process I/O in node $n \in N^s$
T_m^R	Delay of memory processing in a memory resource $m \in M^s$
N_e^o	Number of links existing between adjacent nodes of link $e \in E^s$
T_e^P	Propagation delay of link $e \in E^s$
$\lambda_{e,n}^s$	Arrival rate of packets from node $n \in N^s$ in link $e \in E^s$
	Resource graph
N^v	Set of resource graph nodes
E^v	Set of resource graph links
C^v	Set of nodes corresponding to computational resources
M^v	Set of nodes corresponding to memory resources
	Process graph
N_t^p	Set of process graph nodes in task t
E_t^p	Set of process graph links in task t
σ_p^f	Number of page faults of a process $p \in N^p$
σ_p^n	Pages per a page fault of process $p \in N^p$
σ_p^c	Clock counts of process $p \in N^p$
λ_p^r	Arrival rate of packets from the memory of process $p \in N^p$
λ_p^w	Arrival rate of packets to the memory of process $p \in N^p$
S	Set of tasks
N_t^p	Set of processes required for task $t \in S$
T_t^a	Acceptable time of task $t \in S$
P_t	Set of paths of the process graph in task $t \in S$
c_p^v	Set of computational resources required to run the process $p \in N^p$
m_p^v	Set of memory resources to run process $p \in N^p$

Modeling micro disggregated data center network

A μDDC network is represented as a graph $G^s(N^s, E^s)$, where N^s and E^s denote sets of nodes and undirected links, respectively. Three types of nodes exist: computational resource pools, memory resource pools, and switches. N^c and N^m are the sets of computational and memory resource pools, respectively. C_n^s and M_n^s represent the number of available computational and memory resources in the computational and memory resource pools corresponding to node $n \in N^s$, respectively. For each resource in the computational resource pool $c \in N^c$, we define $K_c > 0$ as the number of floating-point operations per second, $F_c > 0$ as the clock frequency, and $V_c > 0$ as the page size. For each resource in the memory resource pool $m \in M^s$, we define $T_m^R \geq 0$ as the delay in reading data in one memory access. Then, let R^s be the number of resource pools. We also define $T_n^N \geq 0$ as the processing delay until a packet is relayed to the next port in node $n \in N^s$. If node n does not have switching capability, T_n^N is infinite. In addition, we define the I/O processing delay $T_n^I \geq 0$ that occurs during communication at each node $n \in N^s$.

For each link $e \in E^s$, we define $N_e^o > 0$ as the number of links existing between adjacent nodes, $T_e^P \geq 0$ as the propagation delay, and $\lambda_{e,n}^s \geq 0$ as the arrival rate of packets from node $n \in N^s$. We define $R_{i,j}$ as the set of configurable paths between nodes $i, j \in N^s$ on the DDC. $r \in R_{i,j}$ denotes the set of links on path r. The bandwidth of all links is $B > 0$.

Modeling resource allocation request

Resources required for a task are requested before running the task. We model a resource request using two graphs, where one indicates the relationships between the required resources (resource graph) and the other indicates the relationships between the processes required to execute the task (process graph). An example request is shown in Fig. 2.5.

A resource graph is given a graph structure $G^v(N^v, E^v)$, where N^v and E^v denote the sets of nodes and undirected links, respectively. Each node corresponds to the requested computational or memory resource. C^v and M^v denote the sets of requested computational and memory resources, respectively. Links are added between computational and memory resources that execute the same process.

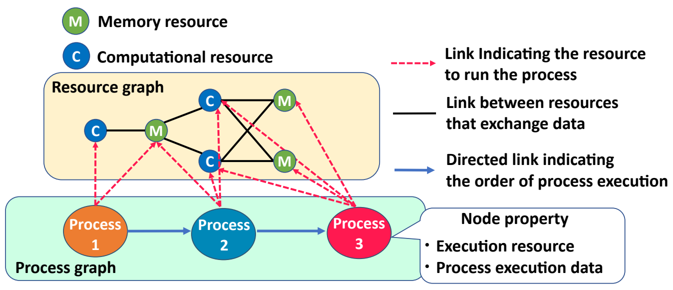


Figure 2.5: Example of resource and process graphs.

Process graphs are provided for each task. We define a set of tasks as S. For task $t \in S$, a process graph is defined as a directed graph structure $G_t^p(N_t^p, E_t^p)$, where N_t^p and E_t^p denote the sets of nodes and directed links, respectively. Each node $p \in N_t^p$ represents the process required to execute the requested task. Node $p \in N_t^p$ has a set of resource graph nodes c_p^v and m_p^v corresponding to the computational and memory resources required to run the process corresponding to node p.

For each node $p \in N_t^p$, we define the number of page faults ($\sigma_p^f \geq 0$), the number of pages transmitted per page fault ($\sigma_p^n \geq 0$), and the clock counts required to execute a process ($\sigma_p^c \geq 0$). For a process corresponding to node $p \in N^p$, $\lambda_p^r > 0$ denotes the arrival rate of packets from the memory, and $\lambda_p^w > 0$ denotes the arrival rate of packets to the memory. Arrival rates were obtained in advance by monitoring the task in the test environment. Note that if multiple resources run a process, the amount of communication and the number of clock counts for a resource pair will be reduced. However, in this chapter, we set the same value as the worst-case value. Each link $e \in E_t^p$ is a directed link that indicates the process order. Each path from the first to the final process provides the sequence of processes required to complete a particular task. We define the set of paths in the process graph for task $t \in S$ as P_t.

In addition, for each task $t \in S$, acceptable time T_t^a is defined as a performance requirement. All tasks should be completed within an acceptable time.

2.4.3 Relationship between resource allocation and task execution time

In a μDDC, resource allocation defines the performance of a task. Here, we model the relationship between resource allocation and task execution time.

Mapping resources

$\delta^N_{i,j}$ denotes the mapping between the requested resources and those in the μDDC. $\delta^N_{i,j} = 1$ when resource graph node $i \in N^v$ is mapped to the μDDC network node $j \in N^s$ and $\delta^N_{i,j} = 0$ otherwise.

Mapping network links

$\delta^E_{x,y}$ denotes the mapping between the resource graph links and paths in the μDDC. $\delta^E_{x,y} = 1$ when the resource graph links $x \in E^v$ are mapped to path $y \in R_{k,t}$ between nodes $k, t \in N^s$ in the μDDC and $\delta^E_{x,y} = 0$ otherwise.

Modeling task execution time

The execution time of task $t \in S$ is the sum of the times required to complete all processes in task t. Furthermore, as shown in Section 2.3, task execution time depends on the processing time of the computational resources and the processing time required to read the data from the memory resource. Therefore, the execution time for each process is estimated as the sum of the processing time of the computational resources and the processing time required to read the data from the memory resource. In this chapter, we compare the worst-case execution time with the acceptable time to allocate resources that satisfy the requirements of the request. The worst-case execution time T^e_t for task $t \in S$ is obtained as follows:

$$T^e_t = \max_{y \in P_t} \sum_{p \in y} \max_{c' \in c^v_p, m' \in m^v_p} \left(T^c_{c',p} + T^m_{c',m',p} \right) . \tag{2.1}$$

where $T^c_{c',p}$ denotes the processing time of process $p \in N^p_t$ in the computational resource mapped to $c' \in N^v$, and $T^m_{c',m',p}$ denotes the processing time to read the data for process p from the memory resource mapped to $m' \in N^v$ in the computational resource mapped to $c' \in N^v$.

Processing time in a computational resource The processing time $T^c_{c',p}$ for a process $p \in N^p$ in a computational resource mapped to $c' \in N^v$ is calculated by dividing the clock count σ^c_p required to complete process p by the clock frequency F_j of a resource in the computational resource pool $j \in N^c$ as follows:

$$T^c_{c',p} = \sum_{j \in N^s} \left(\delta^N_{c',j} \frac{\sigma^c_p}{F_j} \right) . \tag{2.2}$$

$T^c_{c',p}$ denotes the processing time of the computational resource mapped to $c' \in N^v$ because $\delta_{c',j} = 1$ only if c' is mapped onto $j \in N^s$.

Processing time required to read the data from the memory resource A computational resource accesses a memory resource via I/O interfaces such as PCIe and must perform address processing on the access data. Then, read processing is performed in the memory resource, and the data are transferred to the computational resource. Therefore, the processing time required to read the data from the memory resource is the sum of the I/O processing time T^I_c of the resource in computational resource pool $c \in N^c$, processing time T^R_m of the resource in memory resource pool $m \in N^m$, and communication delay $T^d_{c',m',p}$ required to transmit the data from a memory resource mapped to $m' \in N^v$ to a computational resource mapped to $c' \in N^v$ in process $p \in N^p$.

$$T^m_{c',m',p} = T^d_{c',m',p} + \sum_{j \in N^s} (\delta^N_{c',j} \cdot T^I_j + \delta^N_{m',j} \cdot T^R_j) . \tag{2.3}$$

where c^s_j and m^s_j denote the computational resource and memory resource on node $j \in N^s$, respectively. The communication delay $T^d_{c',m',p}$ is the sum of the time required to obtain the head of the page and the transmission delay. In process $p \in N^p$, the communication delay $T^d_{c',m',p}$ required to transmit the data from a memory resource mapped to $m' \in N^v$ to a computational resource mapped to $c' \in N^v$ is obtained as follows:

$$T^d_{c',m',p} = \left\{ \left(\frac{\sum_{j \in N^s} \delta^N_{c',j} \cdot V_j}{B} \right) \sigma^n_p + T^l_{e^r_{c',m'},p} \right\} \cdot \sigma^f_p$$

where $e^r_{c',m'}$ denotes the link between nodes $c', m' \in N^v$. $\sum_{j \in N^s} \delta^N_{c',j} \cdot V_j$ denotes the page size of the computational resource mapped to $c' \in N^v$ because $\delta_{c',j} = 1$ only if c' is mapped to a computational resource in node $j \in N^s$. $T^l_{e^r_{c',m'},p}$ denotes the latency in the path mapped to $e^r_{c',m'}$ in process $p \in N^p$. In a resource graph link $e' \in E^v$ and process $p \in N^p$, $T^l_{e',p}$ is obtained as follows:

$$T^l_{e',p} = \sum_{i,j\in N^s} \sum_{y\in R_{i,j}} \delta^E_{e',y} \left\{ \sum_{e\in y} \left(T^p_e + T^S_{n^{ss}_e,e,p}) \right) \right\}$$

where n^{ss}_e denotes the source node of link $e \in E^s$ when reading data from memory. T^p_e denotes the propagation delay in link e, and $T^s_{n^{ss}_e,e,p}$ denotes the switching and buffering delay in transferring the data of process p from node n^{ss}_e to link e.

The switching and buffering delays depend on the type of switch. For a packet switch, buffering is required to avoid packet collisions if the link is busy. Therefore, we model it as the sum of the switching process delay and the buffering delay. For a circuit switch, we consider only the switching delay because buffering does not occur. The switching and buffering delay $T^s_{n,e,p}$ is obtained as follows:

$$T^s_{n,e,p}=\begin{cases} T^N_n & if\ n\ is\ circuit\ switch \\ T^N_n+T^R_n(\lambda^e_{e,n,p}, N^o_e, T^N_n) & if\ n\ is\ packet\ switch \end{cases}$$

where $\lambda^e_{e,n,p}$ denotes an estimate of the packet rate to link e after resource allocation in node n. $\lambda^e_{e,n,p}$ is the sum of the current packet rate $\lambda^s_{e,n}$ on link $e \in E^s$ occurring from node $n \in N^s$ and the packet rate λ^r_p from the memory to a computational resource in process $p \in N^p$, that is, $\lambda^e_{e,n,p} = \lambda^s_{e,n}+\lambda^r_p$.

$T^R_n(\lambda^e_{e,n,p}, N^o_e, T^N_n)$ is a function that returns the buffering time in node $n \in N^s$ based on three arguments: an estimate of the packet rate $\lambda^e_{e,n,p}$ to link e at node n, the number of links forwarding packets N^o_e, and the switching delay T^N_n at the node. We estimate the buffering time by using the M/D/C queuing model. This is because the communication of each resource pair occurs randomly and the size of the packet processed by a switch does not change. We assume buffering as a situation where packets arriving according to the Poisson process are waiting to be processed until one of the C links that can process them in a fixed time D is ready to forward them. However, obtaining an accurate response time using the M/D/C queuing model is difficult. We use the approximation from [33]. $T^R_n(\lambda, J, D)$ is obtained as follows:

$$T^R_n(\lambda, J, D) = \frac{\left\{1+f^Q(\lambda,J,D)g^Q(\lambda,J,D)\right\}h^Q(\lambda,J,D)}{2} ,$$

where

$$f^Q(\lambda, J, D) = \frac{\left(1-\frac{\lambda D}{J}\right)(J-1)\left(\sqrt{4+5J}-2\right)}{16\lambda D} \ ,$$

$$g^Q(\lambda, J, D) = 1 - \exp\left\{-\frac{J-1}{(J+1)f^Q(\lambda,J,D)}\right\} \ ,$$

$$h^Q(\lambda, J, D) = \frac{D\cdot(\lambda D)^J}{J\cdot J!\left(1-\frac{\lambda D}{J}\right)^2}\left[\sum_{i=0}^{J-1}\frac{(\lambda D)^J}{i!} + \frac{(\lambda D)^J}{\left(1-\frac{\lambda D}{J}\right)J!}\right]^{-1} \ .$$

2.4.4 Resource allocation problem

At the edge, task execution requests are made continuously, and resources are allocated.

To execute many tasks in such an environment, the resources required for future task requests must remain available at the appropriate time. Therefore, we avoid allocating important resources that may be required by future requests. In this chapter, to avoid the allocation of important resources, we define the resource allocation cost based on the importance of the resources to future resource requests and minimize the costs of the allocated resources under the constraint that the performance requirements are satisfied.

In the remainder of this section, we first define the allocation costs. Thereafter, we define the resource allocation problem based on the defined costs and the execution time model of the task defined in Section 2.4.3.

Resource allocation costs

Here, we define resource allocation costs for computational resources, memory resources, and network links.

Computational resources that can execute tasks with the minimum acceptable processing times are important. In addition, computational resources in resource pools that accommodate numerous resources are important because they can execute tasks that demand substantial computational resources. Therefore, we define the cost as the product of the available computational resources and FLOPS. The allocation cost W_c^c of computational resources in the computational resource pool $c \in N^c$ is obtained as follows:

$$W_c^c = C_c^s \cdot K_c. \tag{2.4}$$

A memory unit with several available memory blocks can execute tasks that require extensive memory resources. The allocation cost W_m^m of a memory resource in the memory resource pool $m \in N^m$ is obtained as follows:

$$W_m^m = M_m^s. \tag{2.5}$$

Network links that are likely to be used as paths between important resources are important. In addition, finding the shortest path is important to satisfy performance requirements because it minimizes communication delays between resources. Therefore, we increase the cost of network links which are possibly the shortest paths between critical resource pairs.

We define the possibility to be a network link on the shortest path as the ratio of the number of shortest paths between resources to the number of shortest paths through that network link. The larger this value, the higher the probability that it is the shortest path. When a resource in computational resource $c \in N^c$ and a resource in memory resource $m \in N^m$ are paired, the possibility of being a network link on the shortest path between resources in resource pool c and m $u_{c,m}(e)$ is

$$u_{c,m}(e) = \frac{N_{c,m}^r(e)}{N_{c,m}^r}.$$

where $N_{c,m}^r$ denotes the number of shortest paths between resources in resource pools c and m, and $N_{c,m}^r(e)$ denotes the number of shortest paths between resources in resource pools c and m passing through network link e.

If the resources are close to each other and are of high cost, they are an important resource pair. Therefore, when a resource in computational resource $c \in N^c$ and a resource in memory resource $m \in N^m$ are paired, the importance of the resource pair $v_{c,m}$ is

$$v_{c,m} = \frac{W_c^c \cdot W_m^m}{H_{c,m}}.$$

where $H_{c,m}$ denotes the smallest hop count between resources in resource pool c and m.

The allocation cost W_e^e of link e is obtained as follows:

$$W_e^e = \begin{cases} \sum_{c \in N^c, m \in N^m} u_{c,m}(e) \cdot v_{c,m} & e \notin E^{alc} \\ \epsilon & e \in E^{alc} \end{cases}, \tag{2.6}$$

where E^{alc} denotes the set of network links that are already allocated. ϵ is a small cost defined for the links used by previously started tasks. By using ϵ instead of 0, we avoid allocating large paths.

Defining resource allocation problem

We define a resource allocation problem to avoid allocating resources required by tasks in the future. In this problem, the network information of the μDDC and resource allocation request is given; this outputs the mapping δ^N, δ^E between the requested resource and allocated resource defined in Sections 2.4.3 and 2.4.3.

Resource mapping constraints A request graph node is mapped as a node, and a request graph link is mapped as a path in the μDDC network as follows:

$$\forall i \in N^v, \quad \sum_{j \in N^s} \delta_{i,j}^N = 1 \ \cdot \tag{2.7}$$

$$\begin{aligned} &\forall x \in E^v, \forall k, s \in N^s, \\ &\sum_{y \in R_{k,s}} \delta_{x,y}^E = \delta_{n_x^{vs},k}^N \cdot \delta_{n_x^{vd},s}^N \end{aligned} \ \cdot \tag{2.8}$$

where n_x^{vs} and n_x^{vd} denote the source and destination nodes of link $x \in E^v$ from memory to computational resources.

Resources other than those available in the resource pool cannot be allocated, which can be represented as follows:

$$\forall c \in N^c, \quad C_c^s - \sum_{c' \in C^v} \delta_{c',c}^N \geq 0 \ \cdot \tag{2.9}$$

$$\forall m \in N^m, \quad M_m^s - \textstyle\sum_{m' \in M^v} \delta_{m',m}^N \geq 0 \ \cdot \tag{2.10}$$

Time constraints All tasks in provided services must be executed within an acceptable time; therefore,

$$\forall t \in S, \quad T_t^e \leq T_t^a \tag{2.11}$$

Objective In this method, resources are allocated to minimize the costs, that is,

$$\begin{aligned}&minimize \quad \textstyle\sum_{c \in N^c} \sum_{c' \in C^v} \delta_{c',c}^N (W_c^c) + \\ &\textstyle\sum_{m \in N^m} \sum_{m' \in M^v} \delta_{m',m}^N (W_m^m) + \\ &\textstyle\sum_{i,j \in N^s} \sum_{y \in R_{i,j}} \left[1_{\sum_{x \in E^v} \delta_{x,y}^E > 0} \left(\sum_{e \in y} W_e^e \right) \right],\end{aligned} \tag{2.12}$$

where $1_{\sum_{x \in E^v} \delta_{x,y}^E > 0}$ is 1 when $\sum_{x \in E^v} \delta_{x,y}^E > 0$ and 0 otherwise.

Solving this problem enables a μDDC to avoid allocating resources required by future tasks while satisfying the performance requirements of the tasks: this is a binary combinatorial optimization problem. A resource allocation problem based on a binary combinatorial optimization problem has been proven to be NP-hard and metaheuristic methods have been used to solve such problems [34]. We solve this problem using ant colony optimization (ACO). ACO can respond flexibly to changes in the environment [35]. ACO is suitable in DDCs where flexible resource utilization is available and a network is likely to change. However, any method that can find a solution can be used.

2.4.5 Resource allocation based on ant colony optimization

We solve the resource allocation problem defined in Section 2.4.4 based on ACO; however, any method that can find a solution can be used.

ACO is a population-based metaheuristic method in which multiple agents probabilistically search for a solution. First, pheromone values are assigned to candidate resources. The higher the pheromone value of a resource, the more likely it is to be selected by the agent. After multiple agents probabilistically search for a solution based on pheromones, the optimal solution is selected from among the searched solutions. Finally, the pheromone value of the resource in the optimal solution is increased. These processes are repeated multiple times.

A resource allocation method based on ACO (VNE-AC) has already been proposed [34]. However, VNE-AC targets traditional architectures and does not target μDDC. In traditional architectures, there is no need to consider performance degradation due to communication delays between resources, unlike μDDC. Therefore, VNE-AC does not consider the impact of the network on the performance of tasks. From this difference, we arrange and use VNE-AC in terms of network link allocation to solve our resource allocation problem. Note that VNE-AC allocates network links by solving the shortest path problem. Because the impact of the network on performance is nonlinear, it is difficult to make optimal allocations based only on the shortest path. We use ACO for network link allocation as well as to consider the impact of the network on performance. The processing steps for each agent include the (1) resource search, (2) network link search, (3) execution time calculation, and (4) pheromone update phases. These steps are repeated t^{itr} times. To reduce unnecessary searching, if the allocation cost is greater than the current best solution during the search, the resource allocation of the agent is rejected at that time. The notation used in the following equations is listed in Table 2.4.

Table 2.4: Notation of resource allocation based on ant colony optimization.

Symbols	Definition
τ_r	Pheromone of resource r
α	Pheromone weight
β	Resource allocation cost weight
ρ	Pheromone decrease rate
ϕ	Pheromone increase rate
H	Maximum number of network link search
t^{itr}	Number of search iterations
C^{cd}	Set of candidate computational resources
M^{cd}	Set of candidate memory resources
E_n^{cd}	Set of all candidate links adjacent to node n
C^b	Set of computational resources in current best solution
M^b	Set of memory resources in current best solution
E^b	Set of links in current best solution

Resource search phase

In this phase, an agent probabilistically allocates the resources corresponding to the nodes in the resource graph from the available resources. Because we aim to find a low-cost solution, we set a high allocation probability for a low-cost resource. We define the allocation probabilities p^c_c and p^m_m for computational resources in the resource pool $c \in C^s$ and memory resources in the resource pool $m \in M^s$ as follows:

$$p^c_c = \frac{(\tau_c)^\alpha \left(\frac{1}{(W^c_c)^\beta}\right)}{\sum_{x \in C^{cd}} \left[(\tau_x)^\alpha \frac{1}{(W^c_x)^\beta}\right]}, \quad p^m_m = \frac{(\tau_m)^\alpha \left(\frac{1}{(W^m_m)^\beta}\right)}{\sum_{x \in M^{cd}} \left[(\tau_x)^\alpha \frac{1}{(W^m_x)^\beta}\right]}$$

where $\alpha > 0$ and $\beta > 0$ denote the weight of the pheromone and the cost, respectively. C^{cd} and M^{cd} denote the sets of candidate computational and memory resources, respectively. τ_x, τ_c, and τ_m denote the pheromone of the resource x, c, m, respectively.

Network link search phase

In this phase, the agent searches for paths between the resources selected in the resource search phase. To search for these paths, the agent generates subagents. Each subagent probabilistically allocates paths corresponding to links in the resource graph from the links in the μDDC. The search is performed starting with the source resource. First, the link from the source resource is selected. The next link from the destination node of the first link is then selected. This process continues until a link to the destination resource is identified. At each step of this process, a link $e \in E^s$ is selected based on the probabilities $p_{e,n}$ in node $n \in N^s$. Note that if a route between resources cannot be determined in the Hth search, the search is terminated because the communication delay increases as the route length increases.

$$p_{e,n} = \frac{(\tau_e)^\alpha \frac{1}{(W^e_e)^\beta}}{\sum_{x \in E^{cd}_n} \left[(\tau_x)^\alpha \frac{1}{(W^e_x)^\beta}\right]}$$

Here, $\alpha > 0$ and $\beta > 0$ denote the relative importance of the pheromone and the cost, respectively. E^{cd}_n denotes the set of candidate links adjacent to node n, and τ_x and τ_e denote the pheromones of the links x, e, respectively.

Execution time calculation phase

After finding the resources, the agent determines the predicted execution time for tasks whose performance is affected by latency due to network link allocation. This value is derived from the equation presented in Section 2.4.3. If the predicted execution time is less than or equal to the acceptable time, this may be a solution. When one task is allocated, communication occurs between the newly allocated resources, which may increase communication delays between other resources. This calculation is performed for the requested task and all other executing tasks whose performance is affected by resource allocation because all tasks allocated to the μDDC must be able to complete processing within an acceptable time.

Pheromone update phase

After obtaining the resources, the agent updates the pheromone based on the pheromone decrease rate, ρ $(0 < \rho < 1)$. The pheromones of the resources and links of the best solution for each iteration are enhanced based on the pheromone increase rate ϕ and resource allocation cost. The pheromone-enhanced value h is obtained as follows:

$$h = \frac{\phi}{\sum_{c \in C^b} W_c^c + \sum_{m \in M^b} W_m^m + \sum_{e \in E^b} W_e^e}$$

where C^b, M^b, and E^b denote the sets of computational resources, memory resources, and links, respectively, in the resource allocation with the smallest cost.

Using these values, we update the pheromones for each resource in the computational and memory resource pools $c \in N^c$, $m \in N^m$, and each network link e as follows:

$$\tau_c = \rho\tau_c + h, \ \ \tau_m = \rho\tau_m + h, \ \ \tau_e = \rho\tau_e + h$$

2.5 Evaluation

We evaluated RA-CNP by simulating the μDDC networks. First, we discuss the effectiveness of the resource allocation problem defined in Section 2.4.4 by evaluating the optimal solution in a small network. Thereafter, we discuss the evaluation of RA-CNP in the networks of the scale that we envisioned. Finally, to demonstrate the practicality of RA-CNP, we investigate whether the resource allocation problem can be solved within a feasible computational time.

2.5.1 Environment

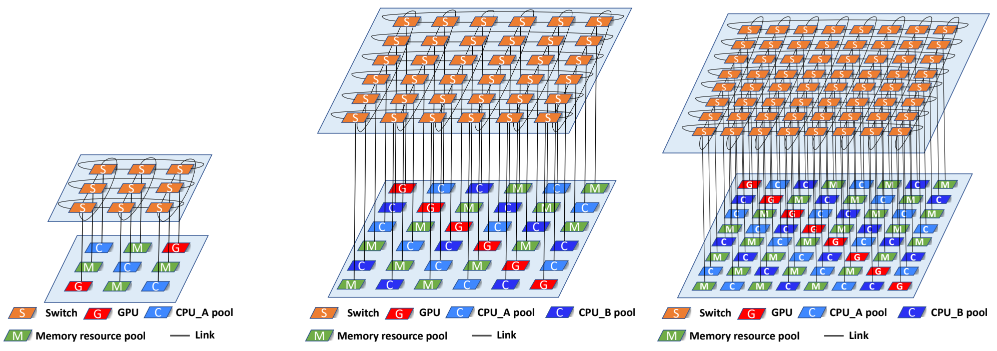
Here, we describe the evaluation networks, resource requests, and comparative methods used to evaluate RA-CNP.

Network

In this chapter, we considered μDDCs with 20–552 computational resources. This scope is similar to the number of servers in the data center we assumed.

Fig. 2.6 shows the network structures used in the performance evaluation. These networks are composed of resource pools containing multiple resources and switches. Each CPU pool had 16 computational resources, and each memory pool had 24 memory resources. We assumed a 2D torus interconnect because this topology is widely used and can be configured at various scales. We connected each resource pool to a switch. When connecting the resource pools, we avoided adjacent switches connected to the same resource type, such that many types of resources could be connected with a short path.

Optical packet switches and optical circuit switches have been proposed for resource disaggregation networks [7, 11]. In this chapter, we evaluated two cases: one configured with optical packet switches (packet switch network) and the other configured with optical circuit switches (circuit switch network).



(a) 3 × 3 2D torus network. (b) 6 × 6 2D torus network. (c) 8 × 8 2D torus network.

Figure 2.6: Networks used in evaluation.

We set the parameters of the μDDC network as listed in Table 2.5. CPU_A represents an Intel® Xeon® W-3335 processor, CPU_B represents an Intel® Xeon® Silver 4314 processor, and GPU represents an NVIDIA GeForce RTX 3090. We used these values to calculate the execution time of the task and the resource allocation cost. We referred to the I/O latency and memory latency measured in [7]. In addition, we referred to the optical circuit switch proposed in [7] and the optical packet switch proposed in [11]. The bandwidth of each link was set to 10 Gbps based on these studies. Each link length was assumed to be 5 m, and the propagation delay was set to 0.025 μs. The page size was set to a default size of 4 KB.

Table 2.5: Parameter settings for network.

Parameters	Value
CPU_A FLOPS	108.8 GFLOPS
CPU_A clock speed per core	3.4 GHz
CPU_B FLOPS	76.8 GFLOPS
CPU_B clock speed per core	2.4 GHz
GPU FLOPS	35.7 TFLOPS
GPU clock speed per core	1.7 GHz
Memory processing time	50 ns
I/O processing time	350 ns
Propagation delay	0.025 μs
Switch latency of the optical circuit switch	5 ns
Switch latency of the optical packet switch	550 ns
Page size	4 KB
The bandwidth of each link	10 Gbps

Resource allocation request

Resources are requested when a task execution request arrives at the μDDC. In this chapter, we assumed that tasks for two services are executed. One is image classification for face recognition using ResNet [36] (service 1) and the other is real-time object identification for automated driving using YOLO [37] (service 2). YOLO and ResNet are commonly used machine learning models for real-time object identification and image classification, respectively. In addition, these are typical of services running at the edge [21], making them prime candidates for being offered as services by a data center located at the edge [38].

All tasks include three processes: Process 1 selects the resource to execute the task, Process 2 loads the required data, and Process 3 executes the main process in the task. Considering the roles of the processes, we allocate the same memory resources to Processes 1 and 2 and the same computational resources to Processes 2 and 3. In addition, Processes 1 and 2 use small amounts of data and do not cause page faults. The parameters for each process, such as clock count and number of page faults, were set on the basis of values obtained by running the task using an Intel(R) Xeon(R) CPU E5-2687W. The parameters for each task are shown in Table 2.6. We generate four types of resource requests for each task, with different acceptable times and required resources.

- Request type 1: Resource request for service 1 with a long acceptable time
- Request type 2: Resource request for service 2 with a medium acceptable time
- Request type 3: Resource request for service 2 with a short acceptable time
- Request type 4: Request for service 2 that requires a GPU with a very short acceptable time

All resource request types have the same structure, shown in Fig. 2.5. We demonstrate the effectiveness of RA-CNP in various cases by changing the required resources and acceptable time. Table 2.7 shows the pattern of the number of resources required for evaluation. Acceptable time values are shown in Table 2.8.

Table 2.6: Parameter settings for task.

	Service 1	Service 2
	Process 1	
Clock count	0.035	0.035
Packet rate to memory (/ms)	0.00033	0.0020
Packet rate from memory (/ms)	0.00033	0.0020
	Process 2	
Clock count	0.054	0.054
Packet rate to memory (/ms)	0	0
Packet rate from memory (/ms)	0.00033	0.0020
	Process 3	
Clock count	2371.33	1960.36
Packet rate to memory (/ms)	1.87	1.90
Packet rate from memory (/ms)	3.71	3.43
Number of page faults	67543.25	56661.29
Number of pages per page fault	5.27	4.84

Table 2.7: Number of resources required for each request type.

	Pattern A		Pattern B		Pattern C		Pattern D	
	computational resources	memory resources	computational resources	memory resources	computational resources	memory resources	computational resources	memory resources
Total(Request type 1/2/3/4)	2/2/2/2	2/2/2/3	4/4/4/2	4/4/4/3	7/7/7/2	7/7/7/3	10/10/10/2	10/10/10/3
Process1(Request type 1/2/3/4)	1/1/1/1	1/1/1/1	1/1/1/1	1/1/1/1	1/1/1/1	1/1/1/1/1	1/1/1/1	1/1/1/1/1
Process2(Request type 1/2/3/4)	1/1/1/1(GPU)	1/1/1/1	3/3/3/1(GPU)	1/1/1/1	6/6/6/1(GPU)	1/1/1/1	9/9/9/1(GPU)	1/1/1/1
Process3(Request type 1/2/3/4)	1/1/1/1(GPU)	1/1/1/2	3/3/3/1(GPU)	3/3/3/2	6/6/6/1(GPU)	6/6/6/2	9/9/9/1(GPU)	9/9/9/2

Table 2.8: Acceptable time for each request type.

	Request type 1	Request type 2	Request type 3	Request type 4
Pattern 1	1000 ms	500 ms	250 ms	200 ms
Pattern 2	500 ms	300 ms	200 ms	180 ms
Pattern 3	500 ms	250 ms	150 ms	100 ms

Compared methods

We compared RA-CNP with two resource allocation methods. The results of these methods were obtained using ACO, the parameters of which are shown in Table 2.9. The two compared methods are described below.

Resource allocation using the shortest path (SP) This method allocates resources based on the shortest path between them. To achieve this allocation, the link cost is defined as $W_e^e = 1$.

This method is extremely simple, and we used it to evaluate whether simple routing is sufficient for DDC resource allocation.

Resource allocation by considering network performance (NP) This method allocates paths based on low traffic volumes and short path lengths between computational and memory resources. It allocates resources by focusing on performance and corresponds to the resource allocation policy proposed by Zervas et al. [7] and Amaral et al. [14]. The NP solution is obtained by identifying the solution with the minimum cost by setting the cost of link $e \in E^s$ with node $n \in N^s$ as the source as follows:

$$W^e_{e,n} = \frac{\frac{\lambda^s_{e,n}}{N^{core}_e}}{\lambda^{max}} + \frac{D_{i,j}}{D^{max}}$$

where λ^{max} denotes the maximum traffic volume, $D_{i,j}$ denotes the SP length from node $i \in N^v$ to node $j \in N^v$, and D^{max} denotes the maximum path length between any two resources in a μDDC.

Table 2.9: Parameter settings for ant colony optimization.

Parameters	Value
Number of agents	20
Number of agent generations	20
Pheromone decrease rate	0.1
Pheromone increase rate	100
Pheromone weight	2
Allocation cost weight	1
Initial pheromone value	1000

2.5.2 Optimal solution of RA-CNP

We determined the optimal solution of RA-CNP by finding the solution among all solutions (hereafter called BFS) with the solutions of other allocation methods to demonstrate the effectiveness of RA-CNP. In a μDDC, the ability to execute many tasks simultaneously using limited resources is desired. Therefore, we investigated how many resources were ultimately allocated by generating resource requests up to the limit of allocation.

In addition, we investigated whether the solution of RA-CNP could be derived using ACO. We compared the solutions obtained using ACO and BFS.

Network

We used a small μDDC network, as shown in Fig. 2.6a, because of the significant computational time required to obtain the solutions of BFS. Furthermore, if the number of resources in the resource pool is large, a significant amount of time is required to obtain the solutions of BFS. In this evaluation, we reduced the number of resources in the resource pool. Each CPU pool had six computational resources, and each memory pool had six memory resources. Note that for the packet switch network, the number of links between a given pair of nodes was set to one; for the circuit switch network, the number of links between a given pair of nodes was set to three to allocate more tasks.

Resource allocation request

In this evaluation, the acceptable time for each request corresponded to Pattern 1 in Table 2.8. The number of computational and memory resources required for each request per process was set as listed in Pattern A of Table 2.7. We generated resource requests up to the number of tasks that could be executed in the μDDC. The number of generated resource requests were the two patterns listed in Table 2.10. The order in which the requests arrived in each case was uniformly random.

Table 2.10: Number of requests generated in each pattern.

	Request type 1	Request type 2	Request type 3	Request type 4	Total
Generated pattern 1	4	2	2	2	10
Generated pattern 2	2	2	4	2	10

Metrics

We measured the worst-case resource utilization and total allocation cost.

Worst-case resource utilization We investigated whether RA-CNP could allocate resources to a limit. Therefore, we measured resource utilization after the allocation of resource requests in Table 2.10.

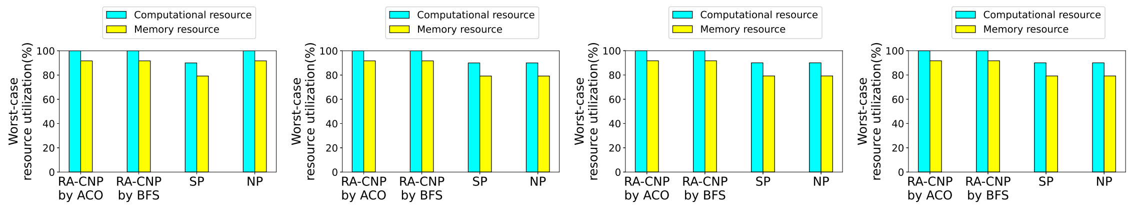
Memory resource utilization u^c and computational resource utilization u^m are defined as follows: $u^m = \frac{m^{alc}}{m^{all}}$ and $u^c = \frac{c^{alc}}{c^{all}}$. c^{all} and m^{all} denote the computational and memory resources in a μDDC, respectively, and c^{alc} and m^{alc} denote the allocated computational and memory resources, respectively. We assumed that a request is blocked if the resources required to satisfy the performance requirements cannot be allocated; that is, if some of the requests are dropped, resource utilization becomes small. In this evaluation, the number of requested computational resources is the same as the number of computational resources in the network. Therefore, if all requests are accepted, the computational resource utilization becomes 100% and no more requests can be accepted.

Total allocation cost To compare the solutions obtained using ACO and BFS, we measured the total allocation cost. If the total cost of the solution obtained using ACO was the same as that obtained using BFS, we concluded that ACO derived the optimal solution for RA-CNP.

The total allocation cost is the sum of the costs of the resources allocated to all generated requests. The total allocation cost W^{all} is defined as follows: $W^{all} = \sum_{r \in R^{req}} W_r^{alc}$. R^{req} is the set of generated requests, and W_r^{alc} is the resource allocation cost for request r.

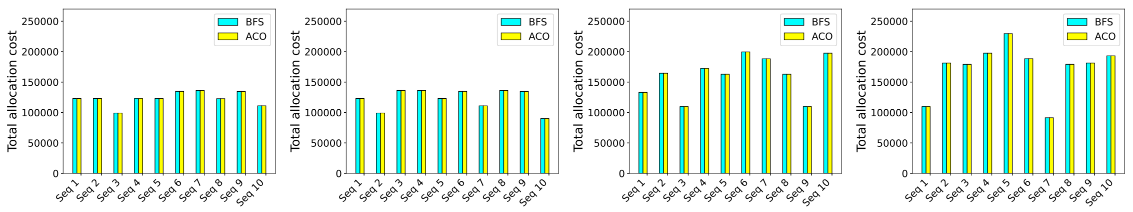
Result

Fig. 2.7 shows the worst-case resource utilization according to 10 measurements in two cases: the packet and circuit switch networks. In this evaluation, when all generated requests were allocated, the resource utilization of computational resources was 100% and it was impossible to allocate more resources. Fig. 2.8 shows a comparison between the allocation costs of the solutions obtained using ACO and BFS.



(a) Generated pattern 1 (Packet switch network). (b) Generated pattern 2 (Packet switch network). (c) Generated pattern 1 (Circuit switch network). (d) Generated pattern 2 (Circuit switch network).

Figure 2.7: Worst-case resource utilization and number of blocked requests in each pattern.



(a) Generated pattern 1 (Packet switch network). (b) Generated pattern 2 (Packet switch network). (c) Generated pattern 1 (Circuit switch network). (d) Generated pattern 2 (Circuit switch network).

Figure 2.8: Total allocated cost per request sequence in BFS and RA-CNP.

As shown in Fig. 2.7, RA-CNP had 100% worst-case computational resource utilization without blocking resource requests in the packet switch network and the circuit switch network. By contrast, SP and NP did not have 100% worst-case computational resource utilization because they caused blocking in some cases. This is because the resources and paths required to execute the task have been exhausted as a result of not considering future requests. RA-CNP derived the optimal solution in a situation in which other methods caused blocking, regardless of the switch type.

In addition, Fig. 2.8 shows that allocation costs similar to those in the solution obtained using BFS can be achieved using ACO. Hence, ACO can identify one of the best solutions to the resource allocation problem. In the following evaluation, we compared RA-CNP derived using ACO with that of other methods.

2.5.3 Effectiveness of RA-CNP

We demonstrated that RA-CNP could execute many tasks from the current network information, regardless of the switch type composing the network. Therefore, we compared RA-CNP with the two resource allocation methods in two cases: the circuit and packet switch networks.

Network

We used two μDDC networks of different scales: 6×6 and 8×8 2D torus networks, as shown in Fig. 2.6b and Fig. 2.6c, respectively. Each CPU pool had 16 computational resources, and each memory pool had 24 memory resources.

In this chapter, we used multicore optical fibers. For this evaluation, the nodes were connected via multicore optical fibers with four optical fiber cores, i.e., the number of links between a pair of nodes was four.

Resource request

We continuously generated the requests listed in Table 2.6 for 300 min. The lifetime of each task was 90 min. In this evaluation, we set the probability that request types 1, 2, 3, and 4 were generated to 0.3, 0.3, 0.3, and 0.1, respectively. We evaluated RA-CNP in four cases to demonstrate its effectiveness in various situations. Each case is shown below.

- Case 1: Neutral case for comparison.
- Case 2: Case in which many resources are required per resource request.
- Case 3: Case in which the performance requirements of requests are strict.
- Case 4: Case in which requests arrive frequently.

The combination of generated requests, required resources, and performance requirements for each case is shown in Table 2.11. Because the 8×8 2D torus network holds more resources, we allocated more resources to compare the methods.

Table 2.11: Combination of generated requests, required resources, and performance requirements for each case.

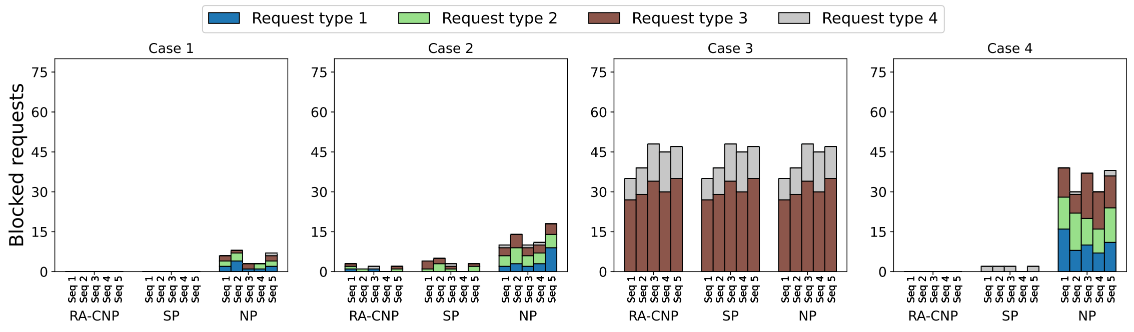
	6 × 6 2D torus network			8 × 8 2D torus network		
	Generated requests	Required resources (Table 2.7)	Performance requirements (Table 2.8)	Generated requests	Required resources (Table 2.7)	Performance requirements (Table 2.8)
Case 1	120	Pattern B	Pattern 1	150	Pattern C	Pattern 1
Case 2	120	Pattern C	Pattern 1	150	Pattern D	Pattern 1
Case 3	120	Pattern B	Pattern 3	150	Pattern C	Pattern 3
Case 4	170	Pattern B	Pattern 1	200	Pattern C	Pattern 1

Metrics

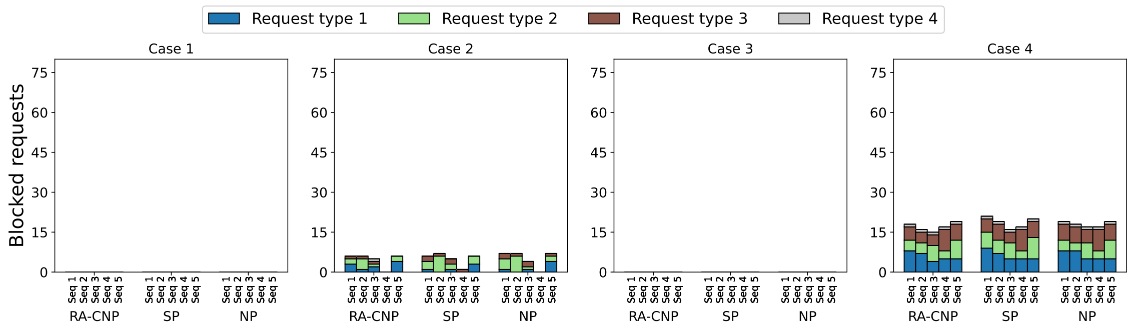
We defined blocked requests as a metric to evaluate whether RA-CNP could allocate many tasks, which refers to the number of requests that could not find resources to satisfy their performance requirements. A larger number of blocked requests implies that resource allocation is insufficient to accommodate many requests.

Result

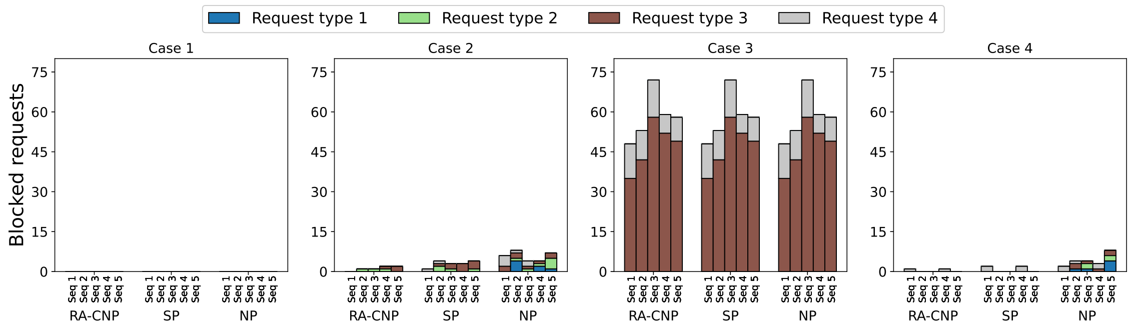
We measured the blocked requests for the five request sequences for each case in two networks: the packet switch network and the circuit switch network. Fig. 2.9 illustrates the blocked requests for each allocation method.



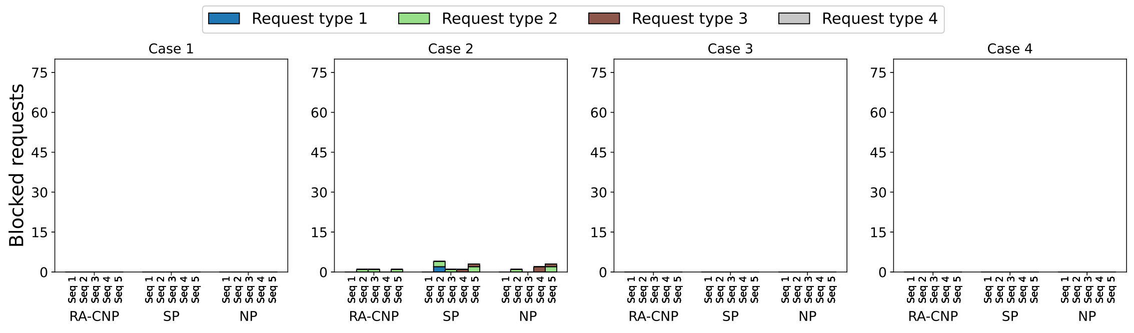
(a) Blocked requests for each case in the packet switch network (6 × 6 2D torus network).



(b) Blocked requests for each case in the circuit switch network (6 × 6 2D torus network).



(c) Blocked requests for each case in the packet switch network (8 × 8 2D torus network).



(d) Blocked requests for each case in the circuit switch network (8 × 8 2D torus network).

Figure 2.9: Number of blocked requests.

RA-CNP had fewer blocked requests than the other methods, regardless of network and case. By contrast, blocking occurred in SP and NP, even in environments where blocking did not occur in RA-CNP (case 4 of Fig. 2.9a). This difference is attributed to the availability of resources when requests require several resources. SP does not consider future requests and cannot accommodate requests with strict performance requirements. As shown in case 4 in Fig. 2.9a and Fig. 2.9c, Request type 4, which had the strictest performance requirement, was blocked. NP preferentially allocated paths between resources with low communication delays, regardless of the performance requirements of the request. Consequently, resource pairs that can satisfy performance requirements were depleted. In particular, NP caused more blocking in the packet switch network for cases 1, 2, and 4 than did the other methods. These blocks are attributed to packet switch processing delays being large and the likely depletion of resource pairs with small communication delays. RA-CNP can allocate more tasks than other methods in various environments by allocating resources in consideration of future requests. At the assumed μDDC scale, RA-CNP was effective.

A comparison of Fig. 2.9a and Fig. 2.9b in RA-CNP shows that the packet switch network was superior in cases 2 and 4 and that the circuit switch network was superior in case 3. In cases 2 and 4, because many resources were requested, many resource pairs existed for communication. Therefore, in the circuit switch network, where network links are occupied, the paths between resources are depleted. In case three, the processing delay of the packet switch was too large to satisfy the performance requirements of the tasks. By contrast, a comparison of Fig. 2.9c and Fig. 2.9d shows that the circuit switch network could allocate more tasks in all cases because more resources were held than requested. In such cases, the circuit switch network, which can reduce communication delays between resources, has more resource pairs that can satisfy the performance requirements.

2.5.4 Computational time for resource allocation by RA-CNP

For practical resource allocation, RA-CNP must allocate resources within a practical computational time at various μDDC scales. We have presented an example solution based on ACO in Section

2.4.5. We evaluated the computation time when solving using ACO. We investigated the relationship between the computation time and the factors involved in the computational complexity of each step, as shown in Section 2.4.5, as the computational time depends on computational complexity.

Computational complexity of Resource allocation based on ant colony optimization

The resource allocation process is divided into four steps. The computational complexity for each step is shown below. Note that each symbol is based on Tables 2.3 and 2.4.

Resource search phase This phase continues until resources are found for all nodes in the resource graph. For each requested resource, one agent selects resources from each resource pool. Therefore, the computational complexity per agent in this phase is the product $O(R^s(|C^v|+|M^v|))$ of the number of requested resources $|C^v| + |M^v|$ and number of resource pools R^s.

Network link search phase In this chapter, because the resource graph connected all memory and computational resources in the same process, the number of allocated paths was $O(|C^v||M^v|)$. As described in Section 2.4.5, the network link search is repeated a maximum of H times for each link in the resource graph. However, H is a constant parameter for ACO. Therefore, the computational complexity per agent in this phase is $O(|C^v||M^v|)$.

Execution time calculation phase This calculation is performed for the requested task and tasks whose performance is affected by resource allocation. Let A^{deg} be the number of tasks affected by resource allocation. The computational complexity per agent in this phase was $O(A^{deg})$.

Pheromone update phase Pheromones are updated on all resource pools and network links in the μDDC. Therefore, the computational complexity is the sum $O(R^s + |E^s|)$ of the number of resource pools R^s and network links $|E^s|$.

We summarize the computational complexity of each step in Table 2.12. This series of phases is iterated t^{itr} times. However, t^{itr} is a parameter for ACO. This value is constant and does not affect the computational time. R^s and E^s depend on the network scale, whereas C^v and M^v depend on the number of requested resources. A^{deg} increases as more tasks are allocated. We investigated whether computational time can be practical for the number of required resources, the number of accommodated requests, and the μDDC network scale.

Table 2.12: Computational complexity in each phase.

Step	computational complexity
Resource search phase	$O(R^s(\lvert C^v\rvert + \lvert M^v\rvert))$
Network link search phase	$O(\lvert C^v\rvert\lvert M^v\rvert)$
Execution time calculation phase	$O(A^{deg})$
Pheromone update phase	$O(R^s + \lvert E^s\rvert)$

Impact of the number of required resources on allocation time

Environment We used the DDC network shown in Fig. 2.6b. The number of resources for request types 1, 2, and 3 shown in Table 2.7 was changed to five patterns, as shown in Table 2.13. In each measurement, 100 requests were randomly generated within 300 min.

Table 2.13: Number of resources required in resource allocation request.

Resource request pattern	1	2	3	4	5
Required computational resources	2	4	6	8	10
Required memory resources	2	4	6	8	10

Result Fig. 2.10 shows the relationship between the required resources and allocation time in the packet switch and circuit switch networks. The 95% confidence interval is included in Fig. 2.10.

In both networks, the allocation time increased almost linearly with the product of the required number of computational and memory resources. This result matches the computational complexity of the network link search phase $O(|C^v||M^v|)$. RA-CNP required less than 10 s, even when 10 CPUs and 10 memory blocks were requested, which is considered acceptable for resource allocation before task execution.

In addition, the rate of increase in the allocation time differed between the circuit and packet switch networks. Each agent stops searching for resources if the currently selected resource allocation cost becomes greater than the current best solution. This means that if a low-cost solution can be found, the search time can be significantly reduced. In packet networks, the number of candidate network link is greater than the number of circuits because link sharing and aggregation are also possible. Consequently, the increase in computational time is high in the packet switch network.

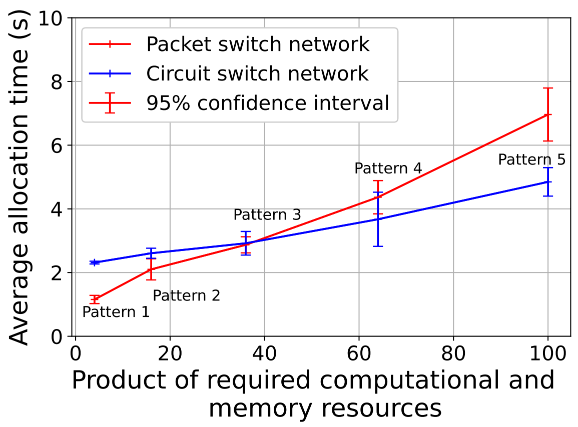


Figure 2.10: Number of required resources and allocation time.

Impact of the number of accommodated requests on allocation time

Environment We used the μDDC network, as shown in Fig. 2.6b. In this evaluation, the number of resources required for each request was set according to Pattern 2 in Table 2.7. We changed the number of generated requests to investigate the effect of the number of accommodated requests. Table 2.14 lists the number of generated requests. Resource requests were generated randomly.

Table 2.14: Number of generated resource allocation requests.

Generated request pattern	1	2	3	4	5
Generated requests	150	180	210	240	270

Result Fig. 2.11 shows the relationship between the number of accommodated requests and allocation time. The 95% confidence interval is included in Fig. 2.11.

Fig. 2.11 shows that there was no significant difference in the allocation time in each case. First, because links are not shared in the circuit switch network, they are not affected by an increase in the number of requests. In addition, the number of requests sharing the same link is limited to prevent incurring a large latency between resources in the packet switch network. The number of accommodated requests only has a limited impact on computational time.

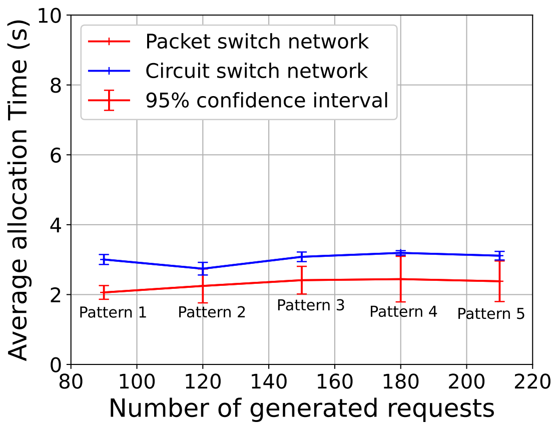


Figure 2.11: Number of resource allocation requests and allocation time.

Impact of μDDC network scale on allocation time

Environment We used 5×5, 6×6, 7×7, and 8×8 2D torus networks. The parameters for each structure are listed in Table 2.15. In this evaluation, the number of resources required for each request was set according to Pattern B in Table 2.7. In each measurement, 100 requests were generated within 300 min.

Table 2.15: Parameter settings for each network.

2D torus network	5 × 5	6×6	7×7	8×8
Switches	25	36	49	64
CPU pools	12	18	24	34
GPU pools	5	6	7	8
Memory resource pools	8	12	18	22
Links	75	108	147	192

Result Fig. 2.12 shows the impact of the scale of the μDDC network on the allocation time. The 95% confidence interval is included in Fig. 2.12.

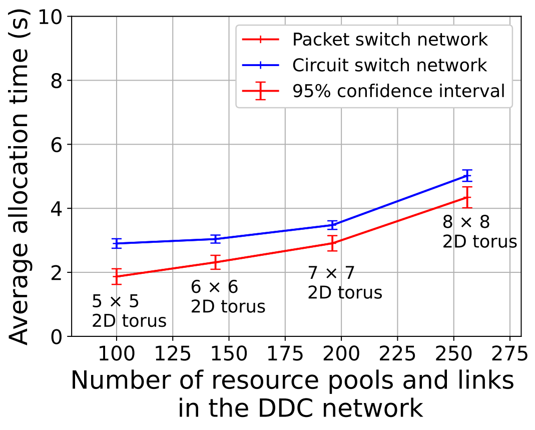


Figure 2.12: Network scale and allocation time.

Fig. 2.12 shows that the allocation time quadratically increased with the number of resource pools and links $R^s + |E^s|$. This result does not match the computational complexity of the resource search and pheromone update phases in Table 2.12. Because the scales of the 5 $\times$ 5 and 6 $\times$ 6 2D torus networks are small, the number of candidate resources is small. Therefore, RA-CNP can find the solution quickly, thereby reducing the computational time. Conversely, as the scale of the network increases, the reduction in computational time may be slight. Consequently, a large difference occurs in the computational time. However, even in the case of an 8 $\times$ 8 2D torus, the computational time is less than 6 s.

2.6 Conclusion

μDDCs improve resource utilization and scaling flexibility. However, network significantly influence task performance, and an efficient resource allocation method is required. We modeled the impact of allocated resources on task performance and defined the resource allocation cost, considering future resource requests. We then defined the resource allocation problem and resource allocation based on this model and costs RA-CNP. In RA-CNP, by avoiding unnecessary allocation of important resources, we can preserve these resources to fulfill future requests and execute more tasks simultaneously.

We conducted simulations to evaluate the effectiveness of RA-CNP. The results demonstrated that RA-CNP could allocate more tasks than other methods in various environments. This method enables the execution of many tasks in a μDDC and the evaluation of architectures. Finally, we measured the allocation time of RA-CNP and demonstrated that this method can allocate resources within a practical time.

Chapter 3

Optical Network Topology Design to Execute Many Tasks Simultaneously in Micro Disaggregated Data Center

3.1 Introduction

In recent years, cloud computing has enabled a plethora of services. Despite its benefits, cloud computing faces problems (e.g., communication delays to users and increased network traffic). Edge computing has emerged as a solution to these problems [1]. We employ micro data centers located at the network's edge to mitigate latency and traffic concerns [4, 39]. However, micro data centers often have fewer CPUs, GPUs, and memory, compared to their cloud data centers. Thus, efficient resource use is crucial for delivering many services at the edge.

Flexibility in resource management per task is important for efficiency. Conventional data center architectures aggregate multiple resources within each server. This hinders flexible resource management [24]. For example, allocating four tasks, each requiring two cores and 4 GB of memory, to a server with a 16-core CPU and 16 GB of memory renders half of the CPU cores unusable owing to memory constraints. Resource disaggregation offers a solution for such inefficient resource use [6]. In resource disaggregation, by decoupling resources (e.g., CPU and memory) into independent units connected via a network. In such micro disaggregated data centers (μDDCs), resources are no longer confined to servers, allowing tasks to use only the necessary resources. This approach improves resource utilization [8]. Therefore, we configure a μDDC that could execute many tasks simultaneously.

In μDDCs, after resources are selected to execute a task, the task is executed via communication between the execution resources. Because resources are connected by a network, the task execution time increases owing to communication delay between resources. Communication delay occurs whenever data are exchanged, which is conventionally handled by the motherboard. These types of communications frequently occur, and network latency has a significant impact on task execution time. According to measurements, resource communication delays in certain networks can increase task execution time by a factor of two or more [9]. Therefore, if the network architecture is not appropriate, resource allocation for a task is constrained due to the inability to satisfy performance requirements. We aimed to construct an optical network topology for a μDDC that prevents resource allocation from being constrained by the network even if many tasks are executed simultaneously.

Because the communication delay can impact task execution time, maintaining low-latency communications is vital to satisfy task performance requirements. Networks based on optical circuit switches (OCS) are particularly effective for achieving low latency. They establish direct end-to-end paths for optical signals. Thus allowing high-bandwidth communication at the speed of light with negligible jitter [40]. Several DDC networks using OCS technology have been proposed [7, 15, 41]. However, OCS lacks packet switching functionality. This implies that the number of resource pairs that can simultaneously establish end-to-end paths for optical signals (i.e., optical paths) is limited, as transmission paths are dedicated to these optical connections. Optical packet switches (OPS) have been proposed as a solution for DDCs offering a compromise between delay and switching speed [11, 12]. Although OPS incurs higher delay compared to OCS, it achieves faster switching than conventional electrical packet switches (EPS). Therefore, enhancing resource allocation flexibility without significantly reducing performance. Furthermore, networks that integrate both OCS and packet switches (PS), such as OPS and EPS, have been developed [42]. These hybrid networks combine the low-latency communication benefits of optical networks with the flexibility of packet switching, crucial for efficient resource management in a DDC.

The network topology plays a pivotal role in task execution within a DDC. In OCS-based networks, where optical paths monopolize transmission paths, a topology that enables communication between resources without being hindered by path dedication is required. Conversely, in networks using PS, where the traffic capacity of each PS port is a limiting factor, a topology that prevents traffic concentration in specific locations is required. To satisfy the performance requirements of tasks, each pair of communicating resources must be proximately located within the network topology, ensuring efficient task execution. An efficient network topology is crucial for managing multiple tasks effectively in a DDC.

Such a topology should ensure ample transmission paths for routing between resources, while also positioning resource pairs close enough to satisfy performance requirements. While there are existing network topology designs tailored for conventional data centers, these are insufficient for the unique demands of a DDC. In a DDC, every task requires network communication between resources, introducing additional latency and increasing network traffic compared to conventional data centers [10, 15]. Consequently, a DDC demands networks with both higher bandwidth and lower latency to overcome these challenges. Several studies have proposed optical network topologies specifically for DDCs [10, 15, 41]. In these studies, the main focus is on minimizing resource latency. Therefore, these studies often overlook the critical aspects of transmission path availability and the impact of task performance requirements on network delays.

We propose a physical optical network topology design incorporating OCS and PS to facilitate the execution of numerous tasks simultaneously in a μDDC. Our approach begins with the introduction of a metric for evaluating the μDDC network topology's capability for simultaneous task execution. We name this metric as capability of simultaneous task execution (CSTE). CSTE represents the ratio of the resources that can be used as a resource communicating with the other resources without violating the performance requirements in a situation where tasks up to the maximum number of executable tasks are executed. A high CSTE value indicates a large number of resource pairs capable of communicating satisfying task performance requirements even if many tasks are executed. Therefore, resource allocation for each task is unlikely to be constrained. That is, the simultaneous execution of many tasks is enabled by flexible resource allocation. We define an optimization problem aimed at configuring an optical network topology that maximizes CSTE.

By comparing topologies generated through this optimization with existing topologies, using our previously proposed resource allocation method RA-CNP [19], we validate the effectiveness of CSTE. Furthermore, we explore important considerations for μDDC physical topology design, emphasizing the combined use of OCS and PS. Our primary contributions are as follows:

- The introduction of CSTE, a metric to evaluate a μDDC optical network topology's ability to execute many tasks simultaneously.
- The definition of an optimization problem to design an optical network topology that maximizes CSTE.
- A demonstration of the effectiveness of optical network topology design based on CSTE through resource allocation simulations.

This chapter is structured as follows: Section 3.2 outlines related work. Section 3.3 provides an overview of CSTE and the physical topology design based on CSTE. Section 3.4 validates the physical topology design's effectiveness based on CSTE and discusses the optimal optical network topology for a μDDC. Section 3.5 discusses on the effective use of OCS and PS, the number of optical fiber cables on physical topology design and, CSTE's limitations. Finally, Section 3.6 concludes the chapter.

3.2 Related work

In a DDC, tasks are executed through communicating between allocated resources. The communication delay, which depends on the path and network between resources, must be decreased to satisfy the performance requirements of tasks. This necessitates a network design that addresses these considerations because communication between resources depends on the network.

In DDC, the impact of communication delays between resources is significant, necessitating resource communication in the order of nanoseconds [9]. Optical networks, known for their low-latency and high-bandwidth communication capabilities, have been proposed to address this challenge. Saljoghei et al. demonstrated the feasibility of nanosecond-level communication between resources in a DDC network configured with OCS [28]. Similarly, Yan et al. proposed a resource disaggregation architecture using OCS for machine learning applications, showcasing improvements in resource utilization [32]. However, networks configured solely with OCS face challenges in flexible resource allocation due to the absence of packet switching functionality and lengthy reconfiguration times, which can reach up to milliseconds [12]. Therefore, the potential of OPS for DDCs has been studied [11, 12, 43]. Terzenidis et al. proposed a high-port OPS capable of supporting up to 256 ports [11]. It achieved nanoseconds switching latency and high throughput. Guo et al. proposed a DDC architecture based on nanosecond buffer-less OPS [44], which included high-speed switching and optical flow control, achieving rapid reconfiguration [43]. In addition, Guo et al. proposed a hybrid architecture combining OCS with multiple ports and OPS with fewer ports, which led to efficient resource utilization and flexible resource reallocation [12].

Despite these advancements, current studies primarily showcase specific network architectures without conclusively determining the appropriateness of OCS or PS connections. Furthermore, there is a lack of discussion regarding the optimal placement of each resource within the network. Therefore, in networks utilizing OCS, the availability of optical fibers for establishing communication paths might become a limiting factor. Meanwhile, networks relying on PS may experience concentrated communication in certain areas, resulting in bandwidth shortages. Therefore, a comprehensive network topology design that addresses these issues is essential for DDCs.

Several physical topology designs have been proposed for DDCs aiming to optimize communication efficiency and resource allocation [10, 15, 41]. Mishra et al. proposed a network topology named MONet, which employs a parallelized spine-leaf topology [10]. In MONet, resources are aggregated into trays, with a two-tier spine-leaf architecture facilitating both inter-tray and intra-tray communications. This design ensures that any two resources can communicate within a maximum of three hops, offering low-latency communication. However, the physical topology does not specifically address the challenge of enabling simultaneous communication across multiple resources. A critical aspect considering the limitation of available transmission paths as the number of communicating resources increases. The availability of optical fiber may become a limiting factor, making it challenging to establish routes between multiple resources simultaneously for the execution of numerous tasks. Ajibola et al. presented NetCoD, a topology for DDCs that integrates both OCS and

EPS [15]. In this design, resources within a rack are disaggregated. Resources are connected via OCS for low-latency in intra-rack communication. While inter-rack communication is facilitated through a spine-leaf architecture connected to an EPS for more flexible routing. The discussion does not cover the simultaneous execution of multiple tasks. When a large number of resources are allocated, communication may need to occur between resources across racks, potentially failing to satisfy the performance requirements of tasks. Shan et al. proposed another physical topology using both OCS and EPS [41]. In this topology, multiple resources are aggregated by either OCS or EPS. These aggregated resources are interconnected via a spine-leaf architecture, where the leaf switch can be either EPS or OCS. This consolidation enables low-latency communication among the aggregated resources. Furthermore, the spine-leaf configuration ensures that any resource can communicate with all other resources in the data center within a predefined number of hops. However, these work does not address the challenge of executing multiple tasks simultaneously. The availability of resource pairs facilitating low-latency communication might become limited as more resources are allocated and additional communication routes are required.

Previous research has introduced and validated the effectiveness of network topologies using OCS and PS. However, these studies often overlooked the availability of transmission paths and resource pairs for communication, especially critical for satisfying task performance requirements during high task volumes. Consequently, a shortage of necessary transmission paths for establishing communication paths that satisfy task performance requirements may result. The resources may also be forced to communicate over longer, less efficient paths that require multiple hops. As a result, flexible resource allocation becomes more challenging, constraining the tasks that can be executed are constrained. A topology that supports flexible resource allocation without exhausting available resources and paths for task execution is required. To effectively design a DDC topology capable of handling numerous tasks, it is essential to assess the feasibility of establishing communication paths between resources that satisfy task performance requirements, even under the load of multiple ongoing tasks. We propose an optical network topology design tailored for μDDC, with a focus on this critical aspect. We base our design on a hybrid network using both OCS and PS, building on the proven effectiveness of such configurations in preceding research.

3.3 Optical network topology design for micro disaggregated data center

In this section, we propose an evaluation metric to identify a suitable optical network topology for a μDDC and a method for designing such a topology based on the evaluation metric.

3.3.1 Overview of micro disaggregated data center

The structure of the μDDC under consideration is shown in Figure 3.1. A μDDC consists of computational and memory resources interconnected by an optical network. A μDDC network employs OCS and PS. As shown in Figure 3.1, a PS aggregates multiple same-type resources, similarly to that in the architecture proposed by Zervas et al [7]. Hereafter, we refer to the set of aggregated resources as a resource board. Further, the set of resource boards connected to the same OCS is reffred to as a resource pool. Each resource board connects to OCS via PS. In this chapter, we design the μDDC physical topology design to execute many tasks simultaneously. To this end, we determine the connections between among the switches and the, resource pools, and as well as the number of optical fibers required for these connections.

The key components of a μDDC are as follows:

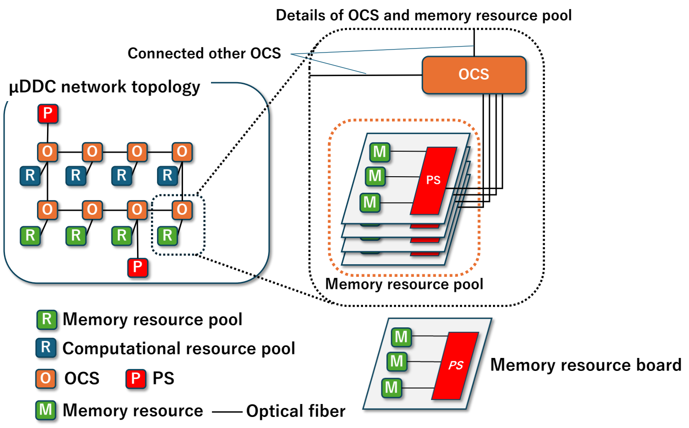


Figure 3.1: Overview of micro disaggregated data center.

Optical circuit switch (OCS) OCS: Serving as the foundational element of the μDDC network, the OCS facilitates connections to other OCS, PS, and resource boards. During task execution, the OCS establishes a direct connection between input and output ports, creating a dedicated communication path through the optical path. We assume the optical path remains active for the duration of the task.

Packet switch (PS) PS: The PS routes data to the appropriate port based on the destination address of the packets. Unlike OCS, PS ports are not pre-assigned to specific input or output lines, allowing traffic from multiple ports to be combined and sent through a single port. This flexibility enables the PS to consolidate traffic from multiple optical paths onto one path. Therefore, establishing optical paths between ports of a PS pair reduces dedicated optical fibers. However, if traffic through a PS port becomes too heavy, packet loss may occur due to congestion. DDC performance with zero packet loss is required to ensure stable operation [45]. We incorporate PS units equipped with buffers, despite the potential for increased delay. This setup aims to maintain consistent task execution within the μDDC.

Optical fiber In this chapter, all connections between switches and between resources and switches are facilitated by optical fiber. Additionally, it is permissible to connect multiple optical fibers between the same pairs of switches or between a switch and a resource, provided there are available ports. When an optical path is established, the optical fiber allocated for that path is exclusively used for communications over that path.

Resource pool A resource pool is a set of resource boards connected to the same OCS. Each resource board aggregates multiple resources of the same type by PS. The total number of resources in each resource pool can be changed by adjusting the number of resource boards connected to OCS. For resources aggregated by OCS, optical paths must be individually established for each resource's communication within the pool. However, because optical paths monopolize the ports of the OCS, a significant number of ports are required, thus limiting the number of resources that can be connected. Therefore, this chapter primarily examined the use of PS for aggregation. Resource pools are categorized based on the type of resources they aggregate: computational resource pools and memory resource pools.

Computational resource Computational resources, which include CPUs and GPUs, are designated for task execution.

They are equipped with a small cache. If the required data is not in the cache, the computational resource retrieves it from a memory resource. This chapter does not focus on cache levels or latency between the cache and computational resource due to the relatively minor impact of cache read times compared to memory resource read times.

Memory resource Memory resources are responsible for storing the data required by computational resources. It encompasses memory and storage components. For the purposes of this chapter, memory space is divided into blocks of a certain size, with each block representing a single memory resource.

Modeling of micro disaggregated data center

In this chapter, we constructed a model of a μDDC. Table 3.1 shows the notations used in this modeling.

Table 3.1: Notation of micro disaggregated data center network topology.

	Definition
N	Set of nodes
E	Set of adjacent node pairs
S^c	Set of OCS
η^c_s	Available port count in a OCS $s \in S^c$
S^p	Set of PS
η^p_s	Available port count in a PS $s \in S^p$
V	Set of PS pairs
κ_g	Threshold of traffic flows in optical path established of a PS pair $g \in V$
J_c	Set of resources corresponding to resource type c
x	Adjacency matrix representing node connection
T	Set of execution task types
R_t	Set of pairs of resource types requiring communication in task type $t \in T$
A_t	Proportion of type $t \in T$ tasks to total execution tasks
$H^o_{t,b,c}$	Maximum number of passed OCS in resource type pair $b, c \in R_t$ for task type $t \in T$
$H^p_{t,b,c}$	Maximum number of passed PS in resource type pair $b, c \in R_t$ for task type $t \in T$

The μDDC optical network topology is represented as a graph $G(N, E)$, where N denotes the set of nodes and E represents the set of pairs of adjacent nodes. The network consists of three types of nodes: OCS, PS, and resource pools. S^c and S^p denote the sets of OCS and PS, respectively. η_i^c and η_j^p represent the available port counts for OCS $i \in S^c$ and PS $j \in S^p$, respectively. J_c represents the set of resources corresponding to resource type c. Different performance resources are considered as different resource types, even if they have the same functionality (e.g., CPU and GPU are both computational resources, but are treated as different resource types). This is because task performance is affected by resource performance. Each resource is associated with a corresponding resource pool.

Physical node connections are represented by the adjacency matrix x. x indicates the physical connections and the number of optical fibers between nodes. If adjacent nodes $i, j \in E$ are connected by two optical fibers, $x_{i,j} = 2$. Conversely, if nodes $i, j \in E$ are not connected, $x_{i,j} = 0$.

V denotes the set of pairs of PSs including PSs in resource boards. Each port of a PS has capacity, and high utilization of a port causes a large queuing delay. Therefore, we keep the utilization of each port below a certain level to ensure small delay. In this thesis, we define the threshold to flow between resources instead of using the threshold to the utilization for simplicity. We denote κ_g as the threshold of traffic flows on optical paths connecting each port in a PS pair $g \in V$. If the traffic flows of resource pairs communicating on the optical path exceeds κ_g, a new optical path must be established in the PS pair.

To derive a metric shown in Section 3.3.2, we assume a situation in which tasks up to the maximum number of executable tasks are executed. In this situation, all computational resources are allocated for task execution. Execution tasks in this situation are categorized by performance requirements. We represent the set of task types as T. The proportion of type $t \in T$ tasks to total execution tasks in the assuming situation is denoted by A_t and $\sum_{t \in T} A_t$ is 1. R_t is the set of pairs of resource types requiring communication for task type t execution. In a μDDC, resources for each task are allocated so that the process can be completed within the acceptable time. Task execution time depends on the processing time within the execution resource and the communication delay to exchange data between resources. Of these, the physical topology affects only the communication delay between resources. Therefore, we consider performance requirements focusing on the communication among resources. The communication delay between resources depends on the number of optical fibers and switches passed through and the queuing delay in PS. We set the maximum numbers of passed OCS and PS in each connection between resources, considering the average queuing delay in each PS when the utilization of the port is k_g. Note that this value can be derived based on a communication delay that is small enough to satisfy the performance requirement of a task considering the processing time within the resource. In addition, we can consider traffic due to the execution of the task to set this constraint. This prevents high-traffic tasks from passing through the PS and causing overloading in the PS. $H^o_{t,b,c}$ and $H^p_{t,b,c}$ denote the maximum number of OCS and PS passed for task type $t \in T$ by the resource pairs $b, c \in R_t$.

3.3.2 Capability of simultaneous task execution (CSTE)

We aim to design a μDDC optical network topology capable of executing numerous tasks simultaneously. To design such an optical network topology, it is essential to assess the capability of simultaneous task execution. We define a metric to evaluate it.

Overview of metric

In a μDDC, network topology affects the communication performance between resources and prevents resource allocation to satisfy the performance requirements of the task. Therefore, we define a metric focusing on communication between task execution resources. To execute many tasks simultaneously, resource allocation must be less constrained by the communication between resources. We assume a situation in which many tasks are executed simultaneously to evaluate whether the topology can execute many tasks simultaneously. We evaluate the network topology in terms of whether resource allocation is constrained in such a situation.

We define the evaluation metric for a μDDC optical network topology capability of simultaneous task execution (CSTE). CSTE is the ratio of the resources that can be used as a resource communicating with each resources without violating the performance requirements. To derive CSTE, we assume a situation wherein tasks up to the maximum number of executable tasks are executed as a situation many task are executed simultaneously. The optical network topology with higher CSTE values has a large number of resource pairs capable of satisfying task performance requirements even if many tasks are executed simultaneously. Resource pairs that can satisfy task performance requirements are unlikely to be depleted, even under the load of multiple ongoing tasks. Therefore, resource allocation for each task is unlikely to be constrained. As a result, flexible resource allocation is possible, and many tasks can be executed simultaneously in the physical topology. In the topology wherein CSTE is 0, resource allocation to use all resources while satisfying the performance requirements is impossible. However, if the number of execution tasks is small, it may be possible to allocate resources to satisfy the performance requirements.

Derivation of capability of simultaneous task execution (CSTE)

We calculate the CSTE, denoted as $CSTE(x, T)$, at node connection x and for the set of execution task types T. Two resources are defined as a pair for explaining CSTE derivation: target resource and candidate resource. A pair of target and candidate resources represents resources requiring communication to execute a task. Each target resource communicates with only resources within a certain hop range (HR) to derive CSTE. If each target resource communicates solely within HR with candidate resources, all resource pairs determined based on this assumption can establish routes simultaneously.

First, for each task type $t \in T$ and pair of resource type $b, c \in R_t$ communicating to execute task type t, we determine the number of candidate resources corresponding to resource type c within hops to satisfy the maximum number of passed OCS and PS $H^o_{t,b,c}$ and $H^p_{t,b,c}$, and the maximum value of HR from each target resource corresponding to resource type b. This value is normalized by the total number of candidate resources. Subsequently, the total product of this value for each combination of target resource and candidate resource is derived. The weighted average based on the proportion of tasks of each type to total tasks yields the CSTE. Therefore, $CSTE(x, T)$ in node connection x and set of execution task types T is obtained as follows:

$$CSTE(x,T) = \sum_{t \in T} A_t \cdot \left(\prod_{b,c \in R_t} \frac{V_{b,c}(t, \theta(x,T))}{|S_c|} \right), \tag{3.1}$$

where $\theta(x, T)$ denotes the maximum HR in node connection x and set of execution task types T. $V_{b,c}(t, \theta(x, T))$ represents the minimum number of candidate resources corresponding to resource type c within hops to satisfy the maximum number of passed OCS and PS $H^o_{t,b,c}$ and $H^p_{t,b,c}$, and within $\theta(x, T)$ hops for each target resource corresponding to resource type b.

Maximum HR To determine the maximum HR, we assess the number of optical paths passing through adjacent nodes $i, j \in E$ when each target resource communicates with any candidate resource within k hops (k is a natural number). If adjacent node pairs possess a sufficient number of optical fibers to establish routes, each route between resource pairs can be established simultaneously. The maximum $\theta(x, T)$ is the highest value of k where the estimated optical path count does not exceed the number of optical fibers $x_{i,j}$ for each pair of adjacent nodes $i, j \in E$. The maximum HR $\theta(x, T)$ is obtained as follows:

$$\theta(x, T) = \max\left(k \in \mathbb{N} \mid \forall i, j \in E,\ D_{i,j}(x, k, T) \leq x_{i,j}\right), \tag{3.2}$$

where $D_{i,j}(x, k, T)$ represents the estimated number of optical paths passing through adjacent nodes $i, j \in E$ when the target resource communicates with any candidate resource within k hops in node connection x. $\mathbb{N}$ denotes the set of natural numbers.

Estimated number of optical paths that pass through adjacent nodes Optical paths are established between PS pairs when a resource pair communicates. First, we estimate the number of resource pairs passing through adjacent nodes $i, j \in E$ on the optical path for each PS pair $g \in V$ when communicating a pair of resource type $b, c \in R_t$ to execute task type $t \in T$. To estimate the number of optical paths passing through adjacent nodes i, j within the optical paths established in PS pair g, we divide the estimated number of resource pairs by the threshold κ_g in PS pair g. The sum of these values across all PS pairs yields the estimated total number of optical paths passing through the adjacent nodes. The estimation of the number of optical paths $D_{i,j}(x, k, T)$ passing through adjacent node pairs $i, j \in E$, when each target resource communicates only within k hops from the candidate resource in node connection x and set of execution task types T, is obtained as follows:

$$D_{i,j}(x,k,T) = \sum_{g \in V} \frac{\sum_{t \in T} \sum_{b,c \in R_t} Q_{i,j,g}(x,k,t,b,c)}{\kappa_g}, \tag{3.3}$$

where $Q_{i,j,g}(x,k,t,b,c)$ represents the estimated number of resource pairs corresponding to resource types $b, c \in R_t$ communicating on the optical path of PS pair $g \in V$ in adjacent nodes $i, j \in E$ when task type t is executed.

Estimated number of resource pairs communicating on the optical path of the PS pair in the adjacent node pair For task type $t \in T$ and node connection x, we calculate the estimated number of resource pairs, corresponding to resource types $b, c \in R_t$, communicating via the optical path of PS pair $g \in V$ through adjacent nodes $i, j \in E$. This estimation is the sum of probabilities that each target and candidate resource pair communicate through these adjacent nodes $i, j \in E$. The probability $Q_{i,j,g}(x,k,t,b,c)$ is determined by:

$$Q_{i,j,g}(x,k,t,b,c) = \sum_{s \in J_b} \sum_{p \in C_{s,c}(t,k)} P^c_{i,j,g}(x,k,t,s,p,c), \tag{3.4}$$

where J_b represents the set of resources for resource type b, and $C_{s,c}(t,k)$ is the set of resources for resource type c that the target resource s can communicate with within k hops. The probability $P^c_{i,j,g}(x,k,t,s,p,c)$ represents the probability that the target resource $s \in S_b$ corresponding to resource type b communicates with a candidate resource p corresponding to resource type c on the optical path established by PS pair g during the execution of task type t. It is calculated as the total product of the following probabilities: (1) $P^t(t)$: Probability that task type t is executed, which is equivalent to the proportion of type t tasks to total tasks A_t. (2) $P^s_{s,c}(t,k)$: Probability that the target resource s selects the candidate resource p as the communication partner from resources corresponding to resource type c within k hops, (3) $P^o_{x,s,p}(t,g)$: Probability that the target resource s and candidate resource p communicate via a path passing through PS pair g, (4) $P^p_{x,g}(i,j)$: Probability that an optical path established at PS pair g passes through the adjacent nodes i, j. Therefore, the probability $P^c_{i,j,g}(x,k,t,s,p,c)$ is obtained as follows:

$$P^c_{i,j,g}(x,k,t,s,p,c) = P^t(t) \cdot P^s_{s,c}(t,k) \cdot P^o_{x,s,p}(t,g) \cdot P^p_{x,g}(i,j). \tag{3.5}$$

The explanations for the individual probabilities $P^t(t), P^s_{s,c}(t,k), P^o_{x,s,p}(t,g), P^p_{x,g}(i,j)$ are explained below.

The probability $P^t(t)$, indicating the likelihood of task type t being executed, is equivalent to the proportion of type t tasks to total tasks A_t. Therefore,

$$P^t(t) = A_t. \tag{3.6}$$

Subsequently, we calculate the probability $P^s_{s,c}(t,k)$, which represents the chance that the target resource s selects the candidate resource p as its communication partner from the resources corresponding to resource type c. This probability is determined by the decision policy of the resource pair. We assume each target resource randomly selects one communication partner from the candidate resources satisfying the task's performance requirements. The resource pairs determined by this process are sufficiently close to satisfy the performance requirements. This allows for the creation of a situation where each resource pair can establish a route that satisfies the performance requirements. The probability $P^s_{s,c}(t,k)$ is the inverse of the set $C_{s,c}(t,k)$, representing resources of type c that the target resource s can communicate within k hops. Therefore,

$$P^s_{s,c}(t,k) = \frac{1}{|C_{s,c}(t,k)|}. \tag{3.7}$$

Subsequently, we derive the probability $P^o_{x,s,p}(t,g)$, which represents the chance that the target resource s and candidate resource p communicates on a path through the PS pair g. This probability depends on the routing between resources. Routes are established to minimize the number of optical fibers used. If multiple routes satisfy the policy, one is randomly selected. This routing policy helps prevent the depletion of optical fiber required for establishing optical paths. Considering this aspect is crucial execute many tasks, it is appropriate to derive CSTE. This probability represents randomly selecting one of the switch combinations passing through PS pair g as a route between resources

s, p. Therefore,

$$P^o_{x,s,p}(t, g) = \frac{w_{x,s,p}(t, g)}{w_{x,s,p}(t)}, \tag{3.8}$$

where $w_{x,s,p}(t, g)$ denotes the number of switch combinations passing through PS pair g as a route between resources s, p satisfying the maximum number of passed OCS and PS $H^o_{t,b,c}$ and $H^p_{t,b,c}$ of task t. This considers whether task t can set a route through PS. $w_{x,s,p}(t, g)$ represents the total number of such combinations.

The probability $P^p_{x,g}(i, j)$ represents the likelihood that an optical path established at PS pair g will pass through adjacent nodes i, j. This probability is determined by the optical path establishment policy. We assume the shortest path between PS is chosen. If multiple shortest paths satisfy this policy, one is randomly selected. Similar to the routing policy, this policy prevents optical fiber depletion. The probability is calculated as the likelihood of randomly selecting one of the paths passing through adjacent nodes i, j among the shortest paths of PS pair g. Therefore,

$$P^p_{x,g}(i, j) = \frac{z_{x,g}}{z_{x,g}(i, j)}, \tag{3.9}$$

where $z_{x,g}$ denotes the total number of shortest paths of PS pair g, and $z_{x,g}(i, j)$ represents the number of paths passing through adjacent nodes i, j among those.

3.3.3 Optical network topology design

Definition of optical network topology design problem

We define an optical network topology design problem to maximize CSTE and generate an optical network topology capable of executing numerous tasks simultaneously. This problem takes input parameters such as network information on the DDC topology, performance requirements, and communication resources for task execution, and outputs the adjacency matrix x representing the connections between each adjacent node. This matrix corresponds to the physical topology. Tasks are executed simultaneously by allocating resources in the physical topology.

Constraints on available ports of OCS The number of optical fibers connected to OCS $s \in S^c$ must be less than or equal to the number of available ports η_s^c on the OCS s. The number of optical fibers connected to the OCS s is $\sum_{n \in N} x_{s,n}$. Therefore,

$$\forall s \in S^c \quad \sum_{n \in N} x_{s,n} \leq \eta_s^c \tag{3.10}$$

Constraints on available ports of PS The number of optical fibers connected to PS $s \in S^p$ must be less than or equal to the number of available ports η_s^p on the PS s. The number of optical fibers connected to the PS s is $\sum_{n \in N} x_{s,n}$. Therefore,

$$\forall s \in S^p \quad \sum_{n \in N} x_{s,n} \leq \eta_s^p \tag{3.11}$$

Objective In this problem, the optical network topology is generated to maximize the CSTE, that is,

$$maximize \quad CSTE(x, T) \tag{3.12}$$

By addressing this problem, we can generate an optimal network topology for a μDDC. However, physical topology design is an NP-hard optimization problem [46]. Therefore, several metaheuristic methods have been employed to solve such a problem [47]. In this chapter, we address this problem using a genetic algorithm (GA) as an exemplar.

Optical network topology configuration by genetic algorithm

While we focus on the physical topology design problem based on the GA, it was noteworthy that it could be solved using various methods. GA is a metaheuristic method leveraging biologically inspired operators (e.g., mutation and crossover). Notably, GA facilitated rapid convergence to near-optimal solutions and adeptly solved discrete problems [46]. We explored GA-based methods for designing an optical network topology [46, 48].

In this chapter, we explored a optical network topology comprising OCS and PS. However, in a network that exclusively used an OCS, PS use was unnecessary if sufficient transmission paths existed. Therefore, we configured a topology optimizing the CSTE solely with OCS, subsequently integrating PS into this topology. All topologies at each phase were devised using GA. Should a topology attain the maximum CSTE ($CSTE = 1$) with only an OCS, we designate this OCS-exclusive configuration as the solution. Our GA methodology unfolded in four phases: (1) Generation of the initial topology phase; (2) Encoding phase; (3) Selection, crossover, and mutation phase; and (4) Evaluation phase. Following the creation of the initial topologies, we aim to maximize CSTE by cycling through steps (2) to (4) for a predetermined number of iterations. This chapter focuses on generating a topology based on the number of optical fibers between adjacent nodes. However, to streamline the procedure in Phases (1) to (3), we consider only the presence or absence of node connections during encoding. Subsequently, in Phase (4), we ascertained the number of optical fibers between the nodes. Upon achieving a topology with an optimal CSTE or concluding the iterative process, the optical network topology with the highest CSTE is the solution.

Generate initial topology configured with OCS only Any initial topology was applicable. In this chapter, we constructed the initial topology through the following steps. First, we established the connection between the resource pool and the OCS. For each resource pool, we randomly selected one OCS from those with the fewest connected resource pools and connected it to the resource pool. This procedure was applied to all resource pools. Second, we determined the connections between the OCS units. We linked each OCS in a ring topology to enable communication among all resources. The pairing of adjacent OCS units was performed randomly. The initial topology was generated using this method. It was important to note that the number of optical fibers connecting each adjacent node was not considered in this phase. These details were determined in the evaluation phase.

Generate initial topology configured with OCS and PS After generating topologies with only OCS, we proceeded to generate topologies configured with both OCS and PS. The initial topologies were based on those generated exclusively with OCS. For each PS, we randomly selected an OCS to connect to and proceeded to generate the initial topologies. Similar to the initial topology generation with only OCS, the number of optical fibers connecting each adjacent node was not set in this phase.

Encoding We uniquely encoded each topology for crossover and mutation. For every topology, we created a binary adjacency matrix to represent the connections between nodes. In this adjacency matrix, 1 represented the presence of optical fibers at that node, and 0 represented the absence. The code for each topology was formed by concatenating each row of the upper triangular portion of this adjacency matrix. For example, when, the adjacency matrix was as follows:

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

the code for this topology was 010101.

Selection In this process, we did not modify the codes themselves; rather, we selected the physical topology code to be used for the next round of mutations and crossovers. Selection was made probabilistically based on the CSTE of each topology. The selection probability of a topology was $\frac{CSTE(x^{target},T)}{\sum_{x\in X} CSTE(x,T)}$, where X represented the set of connections in the current physical topology and x^{target} was the connection in the target topology. This approach prioritized superior topologies for subsequent iterations, thereby expediting convergence.

Crossover In this phase, we randomly chose two topologies from the encoded topologies. Then, we swapped the values at any position in the codes of the two topologies. This action generated two new topologies. Table 3.2 presents an example.

Table 3.2: Example of crossover when replacing the third, fifth, and sixth values.

Number	0	1	2	3	4	5	6	7
Code 1	1	0	0	***0***	0	***1***	***0***	1
Code 2	1	0	1	***1***	0	***0***	***1***	0
generated Code 1	1	0	0	***1***	0	***0***	***1***	1
generated Code 2	1	0	1	***0***	0	***1***	***0***	0

Mutation During mutation, a topology is randomly selected from a pool of encoded topologies. The value at any position in the physical topology code was inverted; a 0 became a 1; if 1, it was set to 0. Table 3.3 presented an example of this process.

Table 3.3: Example of mutation when replacing the third, fifth, and sixth values.

Number	0	1	2	3	4	5	6	7
Code	1	0	0	***0***	0	***1***	***0***	1
generated Code	1	0	0	***1***	0	***0***	***1***	1

Evaluation In this phase, the CSTE for each topology was calculated based on the formula described in Section 3.3.2. However, immediately following a crossover or mutation, the exact number of optical fibers between nodes was unknown. The optical fiber numbers between the nodes were determined in this phase. First, we estimated the number of optical paths crossing each adjacent node pair using equation 3.3. If the node pair had more available ports than the estimated number of optical paths, the optical fibers were connected accordingly. This step was repeated until it was no longer possible to connect additional optical fibers between nodes, considering the limits of the available ports. Finally, the CSTE for each topology was derived.

3.4 Evaluation

We evaluate physical topology design for a μDDC focusing on CSTE through simulations of μDDC networks and resource allocation.

3.4.1 Environment

Network

We assumed resource disaggregation within an edge data center. For evaluation, we considered a μDDC with 16 resource pools interconnected by an optical network with OCS and OPS. This scope was similar to that of an edge data center comprising 16 racks.

We detail the resources and switches integral to the physical topology formation. Table 3.4 lists the network parameters. These values facilitated the generation of optical network topology and the computation of task execution times within the resource allocation framework.

Table 3.4: Parameter settings for micro disaggregated data center network.

Parameters	Value
Clock speed of computational resource	3.4 GHz
FLOPS of computational resource	76.8 GFLOPS
Memory processing delay	$0.125\mu s$
I/O delay	$0.101\mu s$
Page size	4 KB
Total computational resource pools	14
Total memory resource pools	2
Switching latency in EPS	1.11 μs
Ports of EPS used to connect to other switch	20
Threshold of traffic flows in each EPS port	140
Switching latency in OCS	0.03 μs
Ports of OCS used to connect to other switch	80
Threshold of traffic flows in each OPS port	140
Switching latency in OPS	0.55 μs
Ports of OPS used to connect to other switch	64
Propagation delay (Switch - Switch)	0.5 μs
Propagation delay (Resource - Switch)	0.05 μs
The bandwidth of each optical fiber	10 Gbps

Resource We treated each core of the CPU as a single computational resource. We set the performance of computational resources based on the datasheet of the Intel Xeon Silver 4314 Processor. Because this CPU has 16 cores, each CPU included 16 computational resources. We treated memory blocks of 2GB as a single memory resource. We set the performance of memory resources based on a study of the μDDC architecture [12]. Because each memory device had 256 GB of memory space, each memory device included 128 memory resources.

Resource pool A resource board comprising a resource pool was assumed to aggregate resources by EPS, given that OPS aggregation would be prohibitively expensive and impractical. For the parameter settings, we referenced the S5860-20SQ switch [49]. Using four ports for resource connections and the remaining 20 for inter-switch links. In the EPS, we limited the traffic that could be handled in each port to 70% of the capacity. On this basis, the threshold of traffic flows in each EPS port was set at 140. Within each computational resource board, three CPUs were connected per EPS. This equates to 48 computational resources connected to each computational resource board. Within each memory resource board, three memory devices were connected per EPS. This equated to 384 memory resources connected to each memory resource board. By changing the number of resource boards in a resource pool, we changed the number of resources in the resource pool. We set the number of computational resource pools to 14 and the number of memory resource pools to 2 out of 16 to minimize the difference between the number of computational and memory resources.

OCS Referencing CALIENT ’ s optical circuit switch [50], we established a switching delay of 0.03 μs. The switch features 320 ports. In this chapter, we used 80 ports to connect to other switches, and the remaining 240 ports were used to connect the resource boards.

OPS Networking between resource pools predominantly uses OPS owing to its faster switching capabilities, without presupposing the use of a large number of OPS units. OPS was proposed in [11]. In OPS, we limit the traffic that can be handled in each port to 70$\%$ of the capacity. On this basis, the threshold of traffic flows in each OPS port is set at 140. Following [50], a switching delay of 0.55 μs was set, with 64 ports available for simulation purposes.

Optical fiber In our evaluation, we assumed the interconnection of the resource pool and OCS/OPS via 10 m optical fiber cables, while switches were interconnected by 100 m optical fiber cables. These lengths represented the maximum cable distances intra- and inter-racks, respectively. Data transfer was presumed to occur at the speed of light. Therefore, resulting in a propagation delay of 0.5 μs between switches and 0.05 μs between the resource pool and switch. In addition, the bandwidth of each cable was configured to be 10 Gbps.

Execution task

Resource requests are initiated upon the arrival of a task execution request for a service being offered by the μDDC. In this chapter, we consider the provision of an image classification service using ResNet [36]. It is a common application for edge computing scenarios [21].

Each task for this service includes three processes: Process 1 involves selecting the resource for task execution, Process 2 entails loading the required data, and Process 3 consists of executing the task’s main processing. Given the distinct functions of these processes, we allocate identical memory resources to Processes 1 and 2, and the same computational resources are shared by Processes 2 and 3. In addition, Processes 1 and 2 are designed to use minimal data and are configured to avoid causing page faults. The parameters for each process, including clock count and page fault numbers, are determined based on empirical data obtained from experiments run on an Intel(R) Xeon(R) CPU E5-2687W. To evaluate the topology's adaptability in resource allocation, we introduce two scenarios characterized by varying resource demand patterns. The parameters defining the execuion service are shown in Table 3.5. These values inform the physical topology generation and derive the calculation of task execution times in simulation-based resource allocation scenarios. Task execution necessitates communication between computational and memory resources. We designate the computational resources as the target and the memory resources as the candidate. Therefore, we derive CSTE based on the availability of memory resources within a specified HR and the hop count required to satisfy the performance requirement for each computational resource.

Table 3.5: Parameter settings for task.

Parameter	Process 1	Process 2	Process 3
Clock count	0.035	0.054	2371.33
Packet rate to memory (/ms)	0.00033	0	1.87
Packet rate from memory (/ms)	0.00033	0.00033	3.71
Number of page faults	0	0	67543.25
Number of pages per page fault	0	0	5.27
Memory resources	1	4 or 6	4 or 6
Computational resources	1	1	4 or 6

Settings of resource allocation request

We simulate an environment receiving three types of resource allocation requests, each with different performance requirement thresholds. This chapter introduces two hypothetical scenarios to evaluate optical network topology performance under various conditions:

- Case1 Scenarios where requests with longer acceptable completion times occur more frequently.
- Case2 Scenarios characterized by a higher frequency of requests with shorter acceptable completion times.

In Table 3.6, the ratio of each request generated to all requests in each case is shown.

We set the acceptable time for each request type of task as shown in Table 3.6. Based on this acceptable time, we set the maximum number of passed OCS and OPS by each connection between resources by the following steps:

Phase 1: Determine the maximum number of OPS

- Step 1-1: Initialize the maximum number of OPS to the maximum number.
- Step 1-2: Estimate the process time required by the application if the communication between the resources passes the maximum number of OPS.
- Step 1-3: If the process time estimated in Step1-2 exceeds the acceptable time, reduce the number of OPS. Otherwise, end.

Phase 2: Determine the maximum number of OCS

- Step 2-1: Initialize the maximum number of OCS to 1.
- Step 2-2: Estimate the process time required by the application if the communication between the resources passes the maximum number of OCS and the maximum number of OPS defined by the Phase 1
- Step 2-3: If the process time estimated in Step2-2 is less than the acceptable time, increase the number of OCS. Otherwise, decrease the number of OCS and end.

By the above steps, we estimate the process time by using the model of a μDDC we proposed in Chapter 2, assuming that the utilization of each port of the EPS and OPS is 70 %. In this evaluation, because the number of used OPS to generate optical network topology is 1, the maximum number in Step 1-1 is 1.

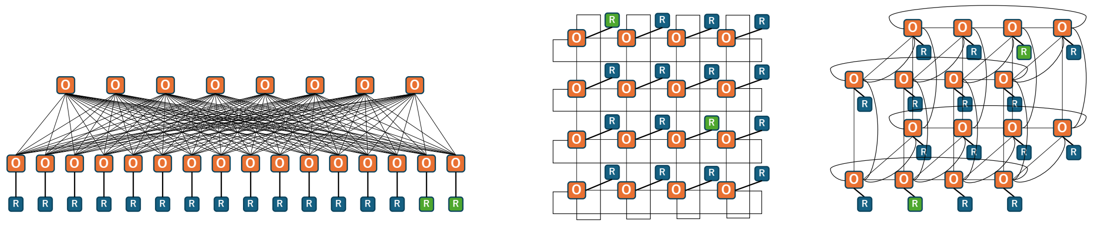
Table 3.6: Parameter settings for each request type.

	Request type 1	Request type 2	Request type 3
Acceptable time	400 ms	300 ms	200 ms
Maximum number of passed OCS	6	3	3
Maximum number of passed OPS	1	1	0
Proportion of each request generated to all requests (Case1/2)	0.6/0.2	0.2/0.2	0.2/0.6

Comparative optical network topology

To evaluate our network topology design, we conducted a comparative analysis against three established topologies alongside our generative approach. The prevalent network topology proposed for a μDDC was tree topology, as highlighted in several studies [10, 15, 41]. Therefore, we used the fat-tree architecture, a quintessential tree topology, as a benchmark. In addition, torus topology represented another conventional data center structure. Our comparison included both 2D and 3D torus topologies. It was important to note that all switches, except those within the resource pool, were configured as OCS.

Table 3.4 shows that our hypothetical μDDC comprises 16 connected resource pools. In the fat-tree configuration, each resource pool was connected to an edge switch. This results in a structure with 16 edge switches connected to a resource pool complemented by eight core switches. For the torus topology, a resource pool was connected to each OCS. Therefore, using 16 OCS, we established a 4×4 2D torus and a $4 \times 2 \times 2$ 3D torus configuration. The placement of each resource pool was optimized to maximize the CSTE value. Should it prove impossible to achieve a network topology with $CSTE > 0$, we strove to position as many computational resource pools in proximity to the memory resource pools as possible. This proximity facilitated task constraint satisfaction for each resource pair. (Figure 3.2) shows the visual representations of each optical network topology. It was noted that 10, 20, and 13 optical fibers connected each adjacent node in the fat-tree, 2D torus, and 3D torus topologies, respectively.



(a) Fat-Tree. (b) 2D Torus. (c) 3D Torus.

Figure 3.2: Comparative optical network topologies.

Settings for optical network topology generation

We evaluate generated topology in a case wherein the resource boards are connected to each OCS to use all ports of the OCS. Each OCS uses 80 ports to connect to other switches, the remaining 240 ports are used to connect the resource boards. Assuming that each resource board uses 20 ports, 12 resource boards are connected to each OCS. In this case, the number of resource pairs that can communicate simultaneously is the largest, because the number of connected resources is the maximum. This environment demonstrates that CSTE can generate an optical network topology that can satisfy task performance requirements even when the network load increases. However, we cannot generate an optical network topology whose CSTE is higher than 0 without OPS. To evaluate the generated topology without OPS, we also evaluate the generated topology in a case where 10 resource boards are connected to each OCS. The parameters in each case are shown in the table 3.7.

Table 3.7: Parameter settings for number of connected resources.

Resource boards connected to each OCS	12	10
Computational resources connected to each OCS	576	480
Memory resources connected to each OCS	4608	3840
Total computational resources	8064	6720
Total memory resources	9216	7680

Furthermore, the comparative topology shown in Section 3.4.1 exhibits a variation in the number of OCS between the Fat-Tree topology using 24 OCS and other considered topologies using 16 OCS. To compare these topologies, we use the same number of OCS to generate topology. We generate and evaluate topologies in four environments with different number of connected resources and the number of used OCS.

- Environment 1: 12 resource boards connected to each OCS and using 16 OCS
- Environment 2: 12 resource boards connected to each OCS and using 24 OCS
- Environment 3: 10 resource boards connected to each OCS and using 16 OCS
- Environment 4: 10 resource boards connected to each OCS and using 24 OCS

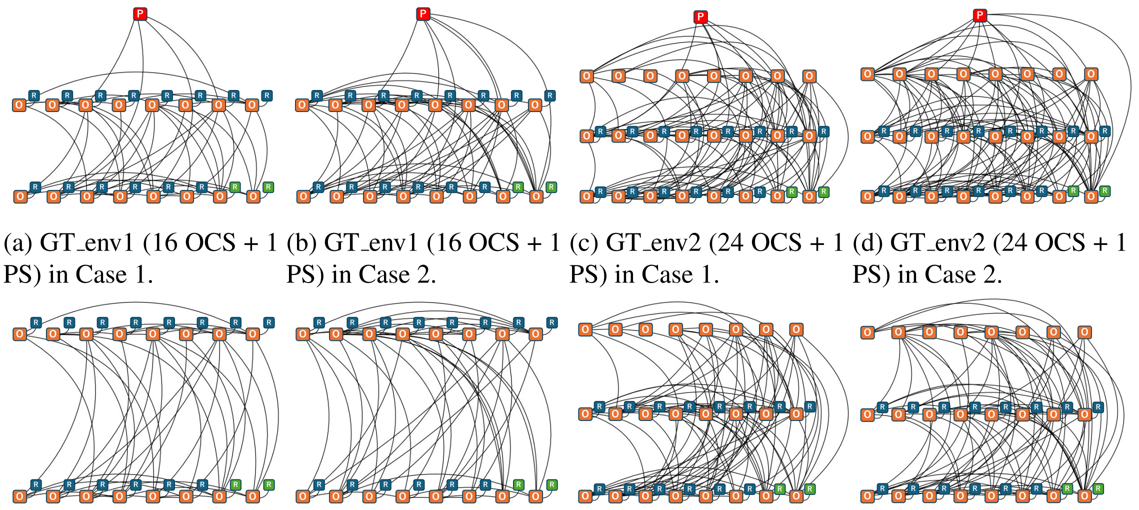
In addition, the parameters governing the GA utilized in chapter are shown in Table 3.8.

Table 3.8: Parameter settings for genetic algorithm.

Parameter	Value
Iterations to generate physical topology with only OCS	2000
Iterations to generate physical topology with OCS and OPS	2000
Topologies generated in each iteration	400
Proportions of select	0.1
Proportions of crossover	0.1
Proportions of mutation	0.8

3.4.2 Generated optical network topology

We generated the optimal network topology for each case and environment, as shown in Sections 3.4.1 and 3.4.1. These topologies are shown (Figure 3.3).


(a) GT_env1 (16 OCS + 1 PS) in Case 1. (b) GT_env1 (16 OCS + 1 PS) in Case 2. (c) GT_env2 (24 OCS + 1 PS) in Case 1. (d) GT_env2 (24 OCS + 1 PS) in Case 2.

(e) GT_env3 (16 OCS) in Case 1. (f) GT_env3 (16 OCS) in Case 2. (g) GT_env4 (24 OCS) in Case 1. (h) GT_env4 (24 OCS) in Case 2.

Figure 3.3: Generated optical network topologies.

In Environments 1 and 2, achieving an optical network topology with a CSTE of 1 using only OCS was not feasible. Therefore, we generated topologies incorporating both OCS and OPS, which achieved a CSTE of 1. Conversely, in Environments 3 and 4, the topologies we generated using only OCS achieved a CSTE of 1 across all cases. We refer to the resulting topologies for environments 1, 2, 3, and 4 as GT_env1, GT_env2, GT_env3, and GT_env4, respectively. The comparative topologies were configured exclusively with OCS. Because they were inappropriate for direct comparison, we compared our network topology against these comparative topologies by connecting one OPS to maximize the CSTE in Environments 1 and 2. In addition, we used topologies that were in the process of being generated to facilitate a comparison between GT across various CSTE values. These interim topologies are referred to as GPT_env1, GPT_env2, GPT_env3, and GPT_env4. Table 3.9 presents the CSTE values for the topologies in each environment.

Table 3.9: CSTE of each optical network topology.

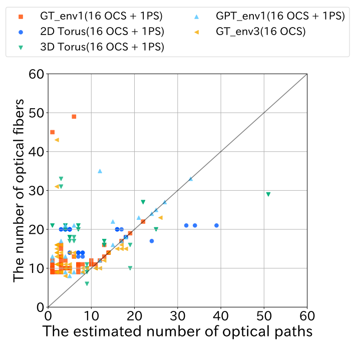
	Case 1	Case 2
2D Torus (16 OCS + 1 PS)	0.5	0.5
3D Torus (16 OCS + 1 PS)	0.5	0.5
GT_env1 (16 OCS + 1 PS)	1	1
GPT_env1 (16 OCS + 1 PS)	0.8	0.4
Fat-Tree (24 OCS + 1 PS)	1	1
GT_env2 (24 OCS + 1 PS)	1	1
GPT_env2 (24 OCS + 1 PS)	0.9	0.7
2D Torus (16 OCS)	0.9	0.7
3D Torus (16 OCS)	0	0
GT_env3 (16 OCS)	1	1
GPT_env3 (16 OCS)	0.5	0.4
Fat-Tree (24 OCS)	1	1
GT_env4 (24 OCS)	1	1
GPT_env4 (24 OCS)	0.8	0.7

3.4.3 Characteristics of optical network topology design based on CSTE

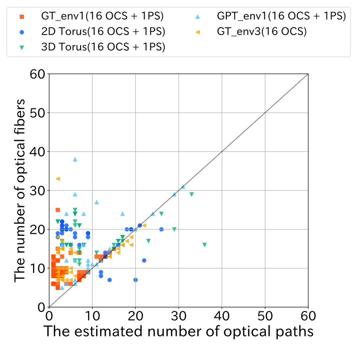
To describe the characteristics of the physical topology design predicated on CSTE, we analyzed the hop counts between computational and memory resources and the number of optical fibers connecting adjacent nodes. These factors are crucial for satisfying the performance requirements and establishing communication routes between resources, which are important for executing numerous tasks efficiently. Table 3.10 details the average and maximum hops between the computational and memory resources across each network topology. Figure 3.4 shows the correlation between the estimated number of optical paths when each computing resource communicates with any memory resource, as calculated using Equation 3.3, and the actual number of optical fibers for each adjacent node pair. Figure 3.4 shows that each point represents an adjacent node pair, with the black line indicating the equilibrium point where the number of optical fibers matches the estimated number of optical paths. If a point is located above the black line, it signifies that the adjacent node pair has the optical fibers necessary to establish the required optical paths.

Table 3.10: Average and maximum hops between all resources in each optical network topology.

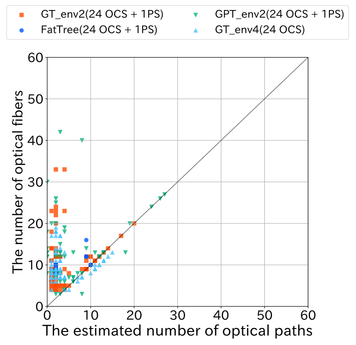
	Case 1		Case 2	
	Average	Max	Average	Max
2D Torus(16 OCS + 1PS)	4	5	4	5
3D Torus(16 OCS + 1PS)	4.07	6	4.07	6
GT_env1(16 OCS + 1PS)	3.57	4	3.50	4
GPT_env1(16 OCS + 1PS)	3.71	5	3.79	5
FatTree(24 OCS + 1PS)	4	4	4	4
GT_env2(24 OCS + 1PS)	3.68	4	3.46	4
GPT_env2(24 OCS + 1PS)	3.75	5	3.89	5
2D Torus(16 OCS)	4	5	4	5
3D Torus(16 OCS)	4.07	6	4.07	6
GT_env3(16 OCS)	3.57	4	3.64	4
GPT_env3(16 OCS)	4.07	6	3.57	5
FatTree(24 OCS)	4	4	4	4
GT_env4(24 OCS)	3.57	4	3.46	4
GPT_env4(24 OCS)	3.89	5	3.86	5



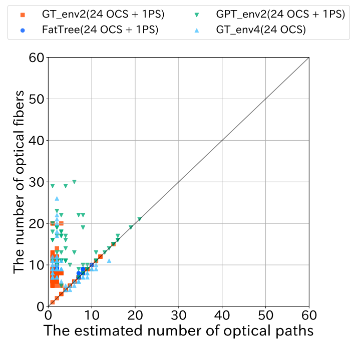
(a) Environment 1 (Case 1).



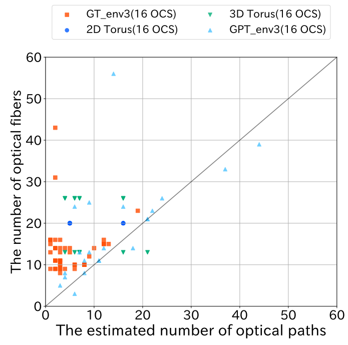
(b) Environment 1 (Case 2).



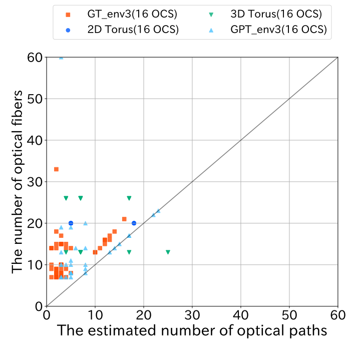
(c) Environment 2 (Case 1).



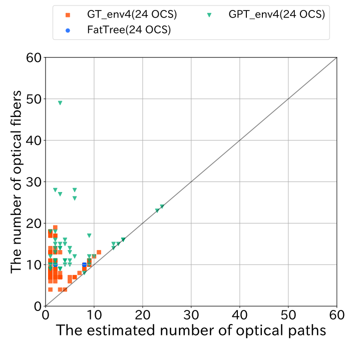
(d) Environment 2 (Case 2).



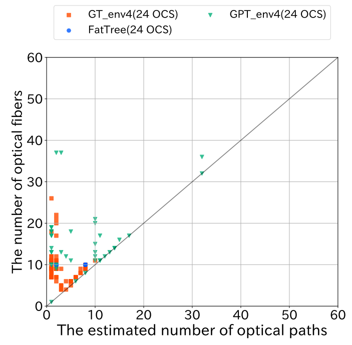
(e) Environment 3 (Case 1).



(f) Environment 3 (Case 2).



(g) Environment 4 (Case 1).



(h) Environment 4 (Case 2).

Figure 3.4: Relationship between estimated number of optical paths passed each adjacent node pair and number of optical fibers between adjacent node pair.

Topologies achieving a CSTE of 1, e.g. GT_env1, and Fat-Tree exhibit a maximum hop count of 4 (Table 3.10). This is the minimum value in the evaluated topologies. Thus, any pair of computational and memory resources can communicate within lower hops, provided that routing is not constrained by a shortage of available optical fibers. Therefore, the selection of resource pairs for task execution is not constrained by the number of hops between them. Conversely, certain topologies, e.g. GPT_env1 and GPT_env2 display small average hop counts but larger maximum hops. The CSTE of these topologies is lower than that of topologies with smaller maximum hops (Table 3.11). Thus, minimizing the maximum hop count is crucial.

Topologies with a CSTE of 1 exhibit no points below the black line (Figure 3.4). This indicates a lower likelihood of optical fiber depletion when establishing optical paths compared with other topologies. However, although there are several points significantly above the black line, the 2D and 3D torus topologies have points below it. This suggests that optical fibers may be overly abundant between nodes that communicate infrequently. This underscores the importance of allocating transmission paths judiciously, considering the communication needs between resources. Furthermore, it is shown that even GT_env1 and GT_env3 with 16 OCS can ensure an adequate number of optical fibers between each adjacent node for the establishment of optical paths, similar to topologies utilizing 24 OCS.

3.4.4 Relationship between optical network topology design based on CSTE and number of tasks that can be executed simultaneously

The effectiveness of the physical topology design based on CSTE is validated through a resource allocation simulation. In this evaluation, we generated requests up to the number of tasks that could be executed in the μDDC, assessing the simultaneous execution capability of tasks.

Resource allocation method used for evaluation

We utilized the cost-based resource allocation method RA-CNP for a μDDC in Chapter 2. This method enhances the capacity to execute multiple tasks simultaneously by prioritizing the preservation of important resources for future task requests. In RA-CNP, the resource allocation cost for each computational and memory resource, as well as for optical fibers between adjacent node pairs, is determined based on their importance to impending tasks. RA-CNP aims to allocate resources and transmission paths to minimize their total associated costs. In RA-CNP, the resource allocation cost for each computational and memory resource, as well as for optical fibers between adjacent node pairs, is determined based on their importance in impending tasks. RA-CNP aims to allocate resources and transmission paths to minimize their total associated costs. In RA-CNP, the objectives are as follows.

$$minimize \quad \textstyle\sum_{c\in C^a} W_c^c + \sum_{m\in M^a} W_m^m + \sum_{e\in E^a} W_e^e \;\; , \tag{3.13}$$

where C^a, M^a, and E^a represent the sets of allocated computational resources, memory resources, and optical fibers, respectively. The allocation costs for a computational resource c, a memory resource m, and an optical fiber e are denoted by W_c^c, W_m^m, and W_e^e, respectively. Resource allocation proceeds to satisfy the objective defined in Equation 3.13. However, RA-CNP does not inherently account for a network comprising both OCS and PS, as considered in this chapter. Therefore, we adjusted the allocation cost metrics for computational and memory resources from those defined in RA-CNP [19]. In RA-CNP [19], the allocation costs for computational resource c and memory resource m were represented as $\mathcal{W}_c^c and \mathcal{W}_m^m$, respectively. The cost $\mathcal{W}_c^c$ correlates with the computational resource's FLOPS multiplied by the number of available resources in the resource pool. Similarly, $\mathcal{W}_m^m$ is derived from the number of available resources in the memory-resource pool. The revised allocation costs W^c and W^m for the computational and memory resources c, m are obtained as follows:

$$W_c^c = \frac{L}{A} \cdot \mathcal{W}_c^c,\; W_m^m = \frac{L}{R} \cdot \mathcal{W}_m^m. \tag{3.14}$$

The costs are adjusted based on L, the total number of allowable connections between resources along optical paths from the resource pool, and A, the number of resources already allocated within that pool. Resources within a pool that have already established optical paths capable of supporting extensive communication between many resource pairs are believed to incur higher costs. This approach aims to prevent the unnecessary establishment of new optical paths.

In addition, for the cost of each optical fiber, fibers that are likely to constitute the shortest path between proximate resources with high costs are assigned higher costs. To reduce the usage of optical fibers, those already allocated as routes for other resource pairs are assigned a minimal value ϵ. The allocation cost W_e^e for an optical fiber e is obtained as follows:

$$W_e^e = \begin{cases} \sum_{c \in N^c, m \in N^m} u_{c,m}(e) \cdot v_{c,m} & e \notin E^{alc} \\ \epsilon & e \in E^{alc} \end{cases}, \tag{3.15}$$

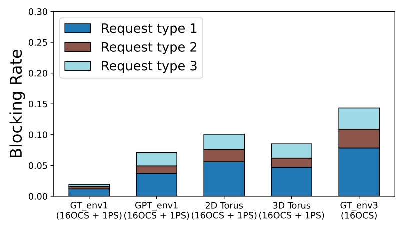
N^c and N^m are the sets of computation and memory resources, respectively. E^{alc} represents the set of transmission paths allocated as a route to any resource pair. $u_{c,m}(e)$ is the ratio of the number of shortest paths between resources c, m to the number of shortest paths through optical fiber e. $v_{c,m}$ is the product of the cost of computational and memory resources, divided by the shortest hop between resources c, m.

Metrics

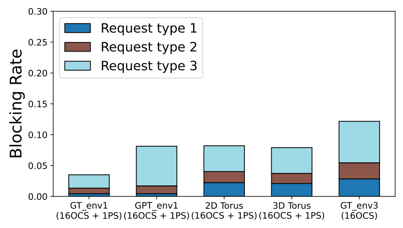
We utilized the blocking rate and computational and memory resource utilization as metrics to evaluate the topology's capability to execute numerous tasks simultaneously. In scenarios where a task cannot be completed within an acceptable time, the request is considered blocked. The execution time for tasks was calculated based on the network performance impact model defined in Chapter 2. The blocking rate represents the ratio of blocked requests to the total number of generated requests. A higher blocking rate indicates a reduced capacity for simultaneous task execution. Computational and memory resource utilization measures the proportion of the total allocated resources to the overall available resources following the resource allocation simulation. A higher utilization signifies that more resource pairs are simultaneously engaged in communication. Resource utilization was derived for the memory and computational resources.

Relationship between optical network topology design based on CSTE and blocking rate

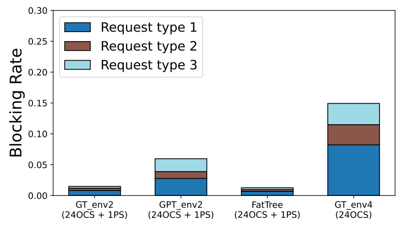
Figure 3.5 shows the blocking rate for each type of execution request in the resource allocation simulation. In addition, for Environments 1 and 2, we include the results for GT_env3 and GT_env4 to facilitate comparison with the OCS-only case.



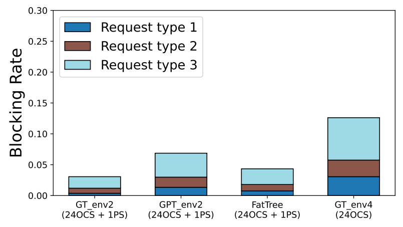
(a) Environment 1 (Case 1).



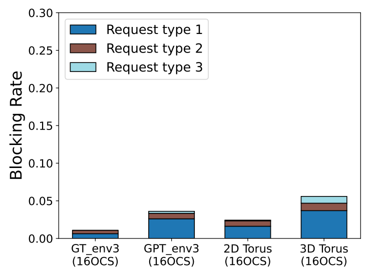
(b) Environment 1 (Case 2).



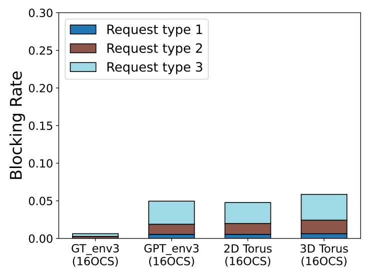
(c) Environment 2 (Case 1).



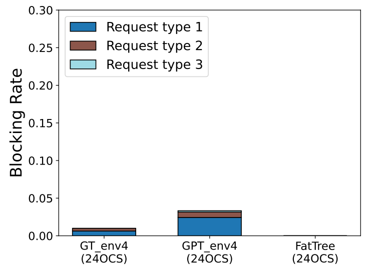
(d) Environment 2 (Case 2).



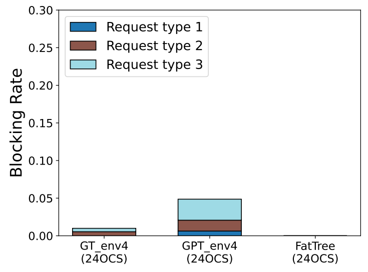
(e) Environment 3 (Case 1).



(f) Environment 3 (Case 2).



(g) Environment 4 (Case 1).



(h) Environment 4 (Case 2).

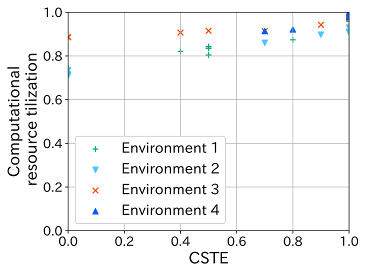
Figure 3.5: Blocking rate in each optical network topology.

Across both environments, the generated topologies (GT_env1, GT_env2, GT_env3, GT_env4) and the fat-tree configuration exhibited lower blocking rates compared to other topologies. This difference was particularly pronounced in Environments 1 and 2, where communication demands between resources were greater, with these topologies reducing blocking by over 50%. GT_env1, GT_env2, GT_env3, and GT_env4 achieved a CSTE of 1 in their respective generative environments

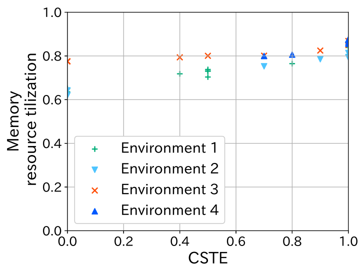
(Table 3.9). By designing a topology based on CSTE, we can create configurations that support a higher concurrency of task execution. In addition, when the CSTE values are identical, the blocking rates are similar, regardless of the number of OCS involved (e.g., GT_env1 (16OCS + 1PS) and GT_env2 (24OCS + 1PS), as depicted in Figure 3.5a to 3.5d). This observation suggests that the simultaneous execution of numerous tasks is feasible, even within smaller-scale networks. Furthermore, in Environments 1 and 2, GT_env3 and GT_env4 experience more blockings than other topologies that incorporate OPS. This result indicates that the combined use of PS and OCS facilitates the establishment of flexible optical paths, thereby enabling a higher number of tasks to be executed concurrently. In Case 1, the blocking rate for Request 1, which allowed the longest acceptable time, was relatively high. This phenomenon is attributed to the depletion of available optical fibers and resources. Therefore, factors beyond communication delays, e.g. the number of resource pairs to establish routes, are important for the execution of numerous tasks.

Relationship between CSTE and resource utilization

Figure 3.6 shows the correlation between CSTE and computational and memory resource utilization. Across each environment, a higher CSTE correlates with increased utilization regardless of the resource type. Indicating that topologies with a CSTE closer to 1 enable more resource pairs to communicate simultaneously. The ability for many resource pairs to engage in simultaneous communication is essential for the simultaneous execution of numerous tasks. Therefore, making CSTE a valuable metric for evaluating a μDDC network topology's efficiency in this regard. The network topology where CSTE is 0 achieves a resource utilization up to about 0.9. When CSTE is 0, this only means resource allocation to use all resources while satisfying the performance requirements is impossible in the network topology. If the number of execution tasks is small, it may be possible to allocate resources to satisfy the performance requirements. Several requests can be allocated. However, it is noteworthy that not all computational and memory resources were successfully allocated even in scenarios with a high CSTE. This discrepancy may be attributed to the resource allocation method employed. Developing more efficient resource allocation methods tailored to topologies generated based on CSTE could potentially improve these results.



(a) Computational resource utilization.



(b) Memory resource utilization.

Figure 3.6: Relationship between CSTE and computational and memory resource utilization.

3.5 Discussion

3.5.1 Comparison of number of tasks that can be executed simultaneously in traditional data center and micro disaggregated data center

To demonstrate that a μDDC can execute more tasks than a traditional data center, we compare μDDC and traditional data centers on the number of tasks that can be executed simultaneously.

Settings of micro disaggregated data center

We set the bandwidth and propagation delay to three patterns each. The bandwidth is set to 10Gbps, 100Gbps, and 1Tbps. Propagation delays are set to 0.025μs, 0.05μs, and 0.5μs, assuming 5m, 10m, and 100m lengths for each optical fiber.

Each parameter for network topology is the same as the environment 3 in Section 3.4. To accommodate tasks with strict performance requirements, we use the μDDC with the optimal network topology in Case 2 shown in Section 3.4. The parameters for the network performance are the same as those in Table 3.4.

In this μDDC, the resource allocation method is the same as that shown in Section 3.4.4.

Settings of traditional data center

We assume a traditional data center consists of 240 servers that have computational and memory resources. Each server has 28 computational resources and 32 memory resources. The total number of computational resources and total number of memory resources are 6720 and 7680, respectively. This number is equal to the total number of resources that the μDDC has.

Each resource can only exchange data with resources in the same server. The latency for exchanges between computational and memory resources within the same server is negligible compared to communication delay between resources in a μDDC. Delays in data transfer between computational and memory resources are ignored.

In this traditional data center, we preferentially allocate resources in servers with fewer available resources for requested tasks. All resources allocated for each task must be in the same server.

Execution task

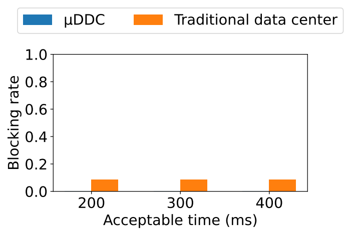
We execute the service task shown in Table 3.5. In this evaluation, we compare three patterns with different performance requirements for the requested tasks. Therefore, we set the acceptable time for tasks to 400 ms, 300 ms, and 200 ms, which correspond to request type 1, 2, and 3 in Table 3.6. Note that the performance requirements of tasks requested in each pattern are all the same.

Metric

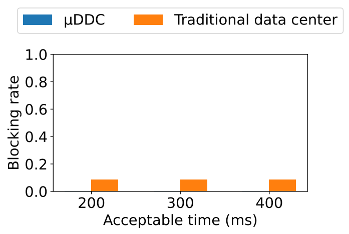
We measure the blocking rate to evaluate whether many tasks are executed simultaneously. In this evaluation, if a task cannot be completed within an acceptable time, the request is blocked. The blocking rate represents the ratio of blocked requests to the total number of generated requests. The smaller the blocking rate, the more tasks can be executed simultaneously.

Result

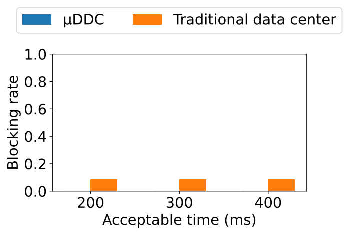
Fig. 3.7 shows the blocking rate of traditional data center and μDDC in each network performance.



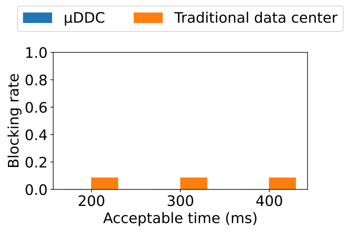
(a) Propagation delay: 0.025μs, bandwidth: 10Gbps.



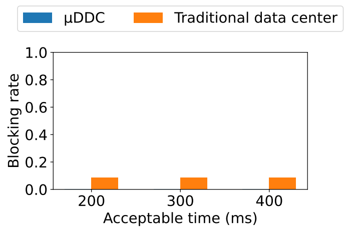
(b) Propagation delay: 0.025μs, bandwidth: 100Gbps.



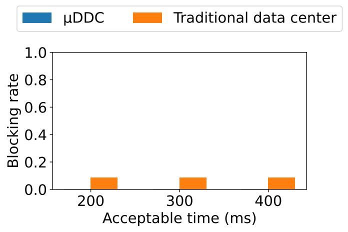
(c) Propagation delay: 0.025μs, bandwidth: 1Tbps.



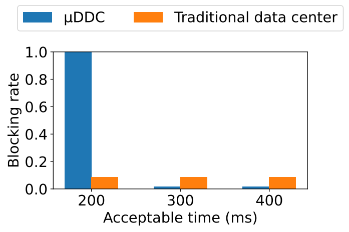
(d) Propagation delay: 0.05μs, bandwidth: 10Gbps.



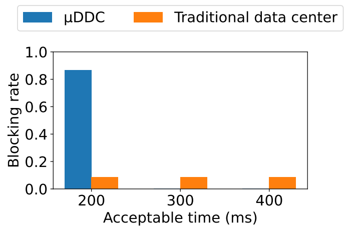
(e) Propagation delay: 0.05μs, bandwidth: 100Gbps.



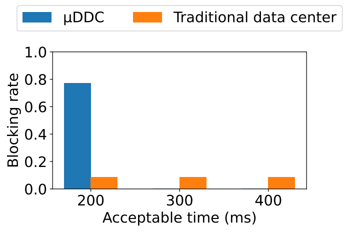
(f) Propagation delay: 0.05μs, bandwidth: 1Tbps.



(g) Propagation delay: 0.5μs, bandwidth: 10Gbps.



(h) Propagation delay: 0.5μs, bandwidth: 100Gbps.



(i) Propagation delay: 0.5μs, bandwidth: 1Tbps.

Figure 3.7: Blocking rate of traditional data center and micro disaggregated data center in each network performance.

The μDDC exhibited lower blocking rates compared to traditional data centers except for the case of allocating tasks with the acceptable time of 200 ms in the μDDC with a propagation delay of 0.5μs. Especially, in μDDC where the propagation delay was 0.05 μs or less and bandwidth was 100 Gbps or higher, the blocking rate of the μDDC was 0. In contrast, the blocking rate of the traditional data center was 8.4% in all cases. By configuring an appropriate μDDC network architecture, it is possible to execute more tasks simultaneously than traditional data centers.

3.5.2 Discussion on effective use of optical circuit switches and packet switches

Previous research has proposed networks using both OCS and PS, yet a quantitative demonstration of their efficacy remains unexplored. Through the evaluation and generation of topologies based on CSTE, we shed light on the optimized use of both OPS and OCS. In Environment 1, where the deployment of OPS is necessary, we generate multiple topologies featuring different numbers of OCS and OPS. Each parameter for network topology generation is the same as that in Section 3.4. We compared CSTE for these topologies. Table 3.11 shows the correlation between CSTE and the distribution of switch types across different cases. Note that the total number of switches used in any topology was 17.

Table 3.11: Relationship between CSTE and number of each switch type.

OCS	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPS	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
CSTE (Case1)	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
CSTE (Case2)	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0

A topology exhibiting a CSTE greater than zero is unachievable when relying solely on either OCS or OPS. This underscores the challenge of designing a suitable μDDC topology using only one type of switch. OCS facilitates high-speed communication. However, it faces limitations owing to the potential depletion of available optical fibers as communication demands among resource pairs increase. Conversely, the OPS provides greater flexibility in resource allocation. However, its higher switching delay compared to OCS reduces the number of resource pairs that can satisfy task performance requirements. A mixed-switch configuration has emerged as a practical solution.

Furthermore, topologies predominantly composed of OPS failed to achieve a CSTE greater than zero. Therefore, it reflects the limitations imposed on performance-satisfying communications. Communication tends to be localized because only proximate resources can satisfy the performance requirements. Complicating the establishment of optical paths. This issue is mitigated in topologies with a higher proportion of OCS. Based on these considerations, it is advisable to rely primarily on OCS and use PS as necessary.

3.5.3 Discussion on number of optical fiber cables on physical topology design

In the topologies generated, connecting nodes with numerous optical fibers is crucial for establishing optical paths for the simultaneous execution of tasks. This increases the complexity of the topology. A viable solution to mitigate this complexity is the use of multi-core optical fiber (MCF) technology. Proposals for DDC architectures using MCF have been made, highlighting MCF's capability of MCF to aggregate multiple optical fiber cores within a single cable. This integration does not require stringent wavelength control or additional components, as noted in previous studies [51]. Utilizing MCF allows for a greater number of communication paths per cable, potentially simplifying the overall network architecture. We explored topologies using a single-core optical fiber, a 4-core MCF, and an 8-core MCF. The CSTE and requisite number of cables for each physical topology variant are listed (Table 3.12). Each parameter for network topology generation is the same as in Section 3.4.

Table 3.12: Relationship between CSTE and number of cables for each optical network topology.

	Case 1			Case 2		
	1 core	4 cores	8 cores	1 core	4 cores	8 cores
Environment 1(16 OCS + 1PS)						
Cables	1244	318	150	1284	312	118
CSTE	1	1	0.5	1	1	0.5
Environment 3(16 OCS)						
Cables	1156	308	114	1170	278	106
CSTE	1	1	1	1	1	1

Employing MCF with an increased count of optical fiber cores effectively reduces the number of cables required for the optical network topology. However, the CSTE for the optical network topology utilizing an 8-core MCF drops 0.5 in Environment 2. This value is half of the CSTE observed in other physical topology configurations. The 8-core MCF network topology demands eight ports per cable connection. This limits the possible number of switch connections compared to the other cases. It is crucial to balance the aggregation of optical fibers to avoid restricting switch interconnectivity.

3.6 Conclusion

We proposed a physical topology evaluation metric named CSTE, designed to account for resource allocation flexibility and the performance requirements of execution tasks. CSTE quantifies the ability of resources to satisfy performance requirements without exhausting the optical path during communication between target resources. In addition, we defined a physical topology design problem based on CSTE. We generated optimal topologies for the simultaneous execution of multiple tasks. Simulations results demonstrated that topologies optimized based on CSTE could significantly reduce task blocking, reducing it by more than 50% compared to conventional topologies.

Chapter 4

Resource Aware Deep Learning Model Partitioning and Allocation to Execute Many Deep Learning Tasks

4.1 Introduction

In recent years, several deep learning (DL) services such as computer vision, natural language processing, and streaming video processing have emerged [52]. Accordingly, the DL models used by DL services are rapidly evolving. Large language models with up to 100 billion parameters [53] and vision models with over 10 billion parameters [54] have emerged. In this regard, DL services are diversifying in terms of both service characteristics and DL model scale [55].

Service providers construct graphics processing unit (GPU) clusters to deliver DL services. In GPU clusters, multiple GPUs are interconnected through a network. Through cooperation among multiple GPUs, GPU clusters enable the execution of DL services using a large-scale DL model that cannot be executed on a single GPU. The scale of GPU clusters varies from large-scale GPU clusters provided by large artificial intelligence (AI) companies to accommodate any DL model to limited-scale GPU clusters [56]. Currently, owing to the fast release cycle of GPU products, enterprise-provided GPU clusters are equipped with GPUs with various performances [57]. A GPU cluster is typically multitenant [58] and multiple services are executed simultaneously. In particular, clusters in edge infrastructures such as micro data centers are small and do not have plentiful GPUs, the efficient use of GPUs to deliver many service tasks is required. Resource disaggregation is effective for the flexible cooperation of multiple GPUs [59]. By resource disaggregation, any GPUs in the cluster, regardless of the rack or server, can work together to process by abstracting the GPUs. Therefore, we target GPU clusters with heterogeneous GPUs in a micro data center applying resource disaggregation (micro disaggregated data center (μDDC)).

When allocating the execution resources for DL services in a cluster with heterogeneous GPUs, a GPU with sufficient performance is allocated to satisfy the following three performance requirements:

- Throughput requirement: Sufficient throughput to complete all requested DL service tasks.
- Execution time requirement: Completion of service tasks within an acceptable time for the service users.
- Memory requirement: Sufficient memory capacity of GPU to execute the DL model.

However, if the memory and computing capacity of the GPUs in the cluster are insufficient for the DL model size and computational complexity, DL services cannot satisfy the listed performance requirements. Pipeline parallelism is used to address this issue [60]. In pipeline parallelism, a DL model is partitioned and a GPU is allocated for each partition. Hereinafter, this partitioning is referred to as the pipeline stage. Throughput increases because the input data for DL service can be processed in parallel at each pipeline stage. Furthermore, larger DL models can be executed by the cooperation of multiple GPUs. Therefore, pipeline parallelism is an important technique to satisfy the throughput and memory requirements of services that provide inference for streaming data and inference using a large-scale DL model.

Although pipeline parallelism is an important technique, communication delays between pipeline stages can cause an increase in execution time [61]. Furthermore, when the execution resource for the next pipeline stage is processed, the input data from the previous stage wait until the resource becomes available. Therefore, if the execution time of one pipeline stage is excessively large, the overall execution time will be larger, even if the execution time of the other pipeline stages is smaller. Therefore, efficient model partitioning and allocation methods are required for pipeline parallelism. DL model partitioning and allocation methods for pipeline parallelism have been previously proposed [60–63]. The objective of these methods is to run large-scale DL models, maximize the throughput, or both. These methods improve DL service performance and enable it to operate in environments where GPU performance is constrained.

Conventional methods aim to maximize the performance of DL services requested at that time. However, in GPU clusters where multiple DL services are executed simultaneously, resource management capable of simultaneously executing the maximum number of services while satisfying the performance requirements is required. To execute many service tasks, the required resources must be available when a service is requested. Therefore, considering not only the services requested at that time but also the execution of services requested in the future is required. Owing to these differences, the following points must be considered: (1) a specific and number of GPUs currently available in GPU clusters, and (2) the specific GPUs allocated for requested services. For instance, if a sufficient number of GPUs with large memory and/or high computing capacity are available, DL service tasks can be executed with fewer pipeline stages, even on larger-scale DL models. Therefore, the performance requirements of the requested service can be satisfied using fewer GPUs. The lack of consideration for a specific and number of GPUs currently available in GPU clusters can lead to excess GPU allocations for each service. Consequently, GPUs for executing future services may be rapidly depleted, and the number of services that can be executed simultaneously is constrained.

Owing to these problems, conventional methods are constrained by the number of services that can be executed simultaneously. Model partitioning and allocation method to execute several DL services are required. We previously proposed a resource allocation method to execute more services simultaneously while satisfying performance requirements [19]. However, this method does not target DL service task execution using pipeline parallelism in GPU clusters. In this chapter, we comprehensively consider resource allocation and model partitioning to execute a many DL services in a GPU cluster.

We propose resource aware model partitioning and allocation(RAMPA). RAMPA aims to minimize the allocation of important resources for future DL service task execution to avoid inhibiting future DL service task execution. We define the resource allocation cost for GPU and network links in terms of the resource importance. Furthermore, we formulate the impact of model partitioning, allocated GPUs, and paths on execution time and throughput. Then, to comprehensively consider resource allocation and model partitioning, we define an optimization problem to minimize the resource allocation costs while satisfying the service performance requirements. By optimizing the model partitioning strategy and resource allocation based on the optimization problem, we preserve the required GPUs to satisfy the performance requirements of future services. This allows more DL services to operate simultaneously. By comparing other model partitioning and allocation methods, we demonstrate that RAMPA can run more DL services while satisfying performance requirements. Furthermore, we compare the execution performances of the services allocated by RAMPA with those allocated by the conventional method. Finally, we investigate whether RAMPA could allocate DL services within a practical computation time.

The main contributions of this chapter are as follows:

- We formulated the impact of model partitioning, allocated GPUs, and paths on execution time and throughput.

- We defined an optimization problem to execute more DL services simultaneously.

- We demonstrated that RAMPA can run more DL services while satisfying the performance requirements.

The remainder of this chapter is organized as follows: Section 4.2 discusses the related work. Section 4.3 provides an overview of clusters with heterogeneous GPUs. Section 4.4 provides an overview of RAMPA. Section 4.5 validates that more DL services can be executed by RAMPA and discusses DL service task execution performance and computational time. Finally, Section 4.6 concludes the chapter.

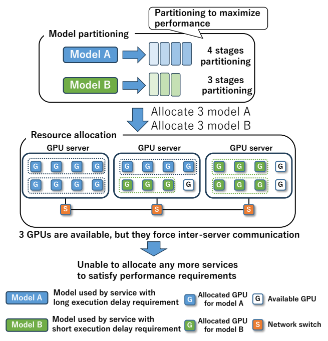
4.2 Related Work

In GPU clusters, when the memory and computing capacity of the GPUs are insufficient for the DL model size and computational complexity, pipeline parallelism is used. In pipeline parallelism, a DL model is partitioned into multiple pipeline stages and each pipeline stage is allocated to a GPU. Larger size DL models can be executed by the cooperation of multiple GPUs. Each pipeline stage is processed in parallel. The throughput increases because the amount of data that can be processed simultaneously increases. However, the execution time and throughput change depending on the process of the DL model that is executed by the GPU at each pipeline stage. An appropriate model partitioning and allocation method is required to exploit pipeline parallelism.

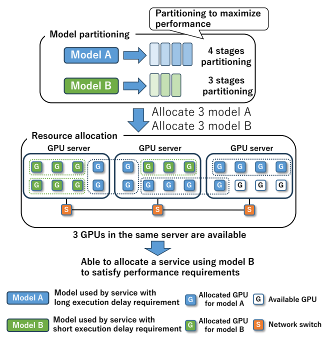
Several model partitioning and allocation methods are proposed [60–62]. Huang et al. proposed a pipeline parallelism method, GPipe, to achieve the fast training of large models [60]. Gpipe maximizes the efficiency of the pipeline parallelism by minimizing the variance in the estimated computational cost of each pipeline stage. Narayanan et al. proposed a pipeline parallelism method, PipeDream, to minimize the large model training time [62]. PipeDream partitions the DL model to minimize the maximum execution time for each pipeline stage by estimating the execution time of each pipeline stage and the communication time between the stages based on the DL model. Zhuohan et al. proposed a model partitioning method, AlpaServe, to execute the maximum number of DL services to satisfy the execution time requirements of a requested set of services [61]. AlpaServe partitions DL models to minimize the maximum execution time for each pipeline stage and selects a combination that maximizes the number of services that satisfy the performance requirements of services. These methods achieve a high-performance DL model execution. However, they do not consider the impact of the performance of allocated resources on service task execution performance because they are targeted for execution on homogeneous architectures. Therefore, these methods cannot achieve proper model partitioning and resource allocation to satisfy the performance requirements of clusters with heterogeneous GPUs.

A method that considers the impact of the performance of allocated resources on the service task execution performance was proposed [63]. Hu et al. proposed a pipeline parallelism method, PipeEdge, for fast inference of large DL models in a heterogeneous device-connected environment [63]. This method maximizes the throughput by model partitioning and resource allocation to minimize the maximum execution time for each pipeline stage, considering the resource performance and communication delays.

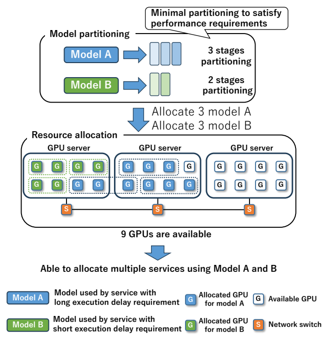
Conventional methods aim to maximize DL service performance requested at a given time and do not consider the resources allocated to future services. Consequently, the number of DL services that can be executed is limited. Fig. 4.1 shows three examples of DL model partitioning and allocation in GPU cluster comprising three GPU pools with eight GPUs. In each example, three services with long execution time requirements using model_A and three services with short execution time requirements using model_B are allocated. In Fig. 4.1a, a DL model is partitioned to maximize throughput and allocate GPUs within the same GPU pool to minimize execution time. Consequently, GPUs for newly allocated tasks are forced to communicate with longer delays. In Fig. 4.1b, allocation of GPUs in pools with numerous available resources is avoided. Under this policy, GPUs on different pools can be used for services that use model_A, which accepts longer delays. Consequently, some GPUs are available for running services with shorter execution time requirements. In Fig. 4.1c, in addition to the resource allocation policy in Fig. 4.1b, the number of model partitions should be the lowest possible to reduce allocated GPUs. In this policy, the performance is not maximized; however, the performance requirements can be satisfied. Consequently, all GPUs on one pool are available. Therefore, additional services can be provided. In this manner, model partitioning and resource allocation must be performed considering the resources to be used for services and the resource allocation situation to execute more DL services.


Partitioning to maximize performance
Model partitioning
Model A
4 stages partitioning
Model B
3 stages partitioning
Allocate 3 model A
Allocate 3 model B
Resource allocation
GPU server
GPU server
GPU server
3 GPUs are available, but they force inter-server communication
Unable to allocate any more services to satisfy performance requirements
Model used by service with long execution delay requirement
Allocated GPU for model A
Available GPU
Model used by service with short execution delay requirement
Allocated GPU for model B
Network switch

(a) Policy that maximizes performance.


Partitioning to maximize performance
Model partitioning
Model A
4 stages partitioning
Model B
3 stages partitioning
Allocate 3 model A
Allocate 3 model B
Resource allocation
GPU server
GPU server
GPU server
3 GPUs in the same server are available
Able to allocate a service using model B to satisfy performance requirements
Model used by service with long execution delay requirement
Allocated GPU for model A
Available GPU
Model used by service with short execution delay requirement
Allocated GPU for model B
Network switch

(b) Policy that avoids allocation of GPUs in pools with numerous available resources.


Minimal partitioning to satisfy performance requirements
Model partitioning
Model A
3 stages partitioning
Model B
2 stages partitioning
Allocate 3 model A
Allocate 3 model B
Resource allocation
GPU server
GPU server
GPU server
9 GPUs are available
Able to allocate multiple services using Model A and B
Model used by service with long execution delay requirement
Allocated GPU for model A
Available GPU
Model used by service with short execution delay requirement
Allocated GPU for model B
Network switch

(c) Policy that minimizes the number of allocated GPUs.

Figure 4.1: Example of deep learning model partitioning and allocation.

4.3 Cluster with heterogeneous GPUs

4.3.1 Overview of cluster with heterogeneous GPUs

In this chapter, we assume that DL services are executed in a GPU cluster with multiple types of GPUs. For the flexible cooperation of multiple GPUs, we assume GPU clusters in a μDDC. Any GPUs in a μ center, regardless of the rack or server, can work together to process by abstracting the GPUs. In this μDDC, multiple same-type GPUs is aggregated as a GPU pool by a network switch. Network switches are connected to form a network between GPUs.

Fig. 4.2 shows DL service task execution process. We assume that resource allocation requests for tasks are sent to a cluster at any time. After a resource allocation request for DL service task is sent to the cluster, the corresponding DL model is partitioned into multiple pipeline stages. Subsequently, the execution GPU for each pipeline stage and the communication paths between GPUs are allocated. After resource allocation, the allocated GPUs load the corresponding model partition into the memory of the GPU from the storage devices in the GPU cluster. After completing these processes, the DL service is executed. In this chapter, we assume DL services that perform inference processing on the input data. After allocation, the data for inference are sent to the allocated service task. Note that each GPU does not communicate with any storage device during execution. Therefore, communication with a storage device does not affect the service task execution performance.

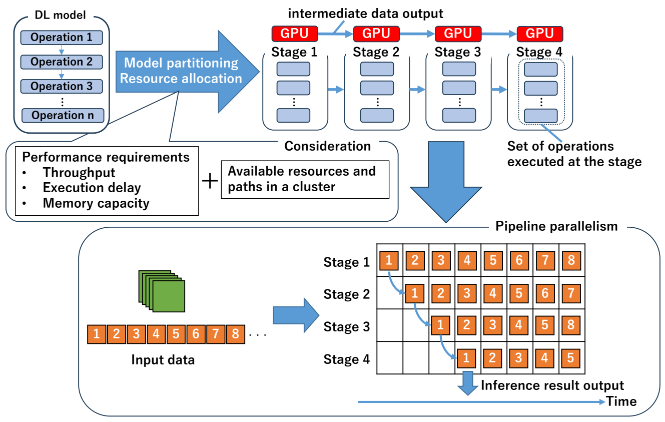


Figure 4.2: Overview of task processing.

4.3.2 Information considered for model partitioning and allocation

The notations for the information considered for DL model partitioning and allocation are listed in Table 4.1.

Table 4.1: Notation of the GPU cluster and task execution.

Symbols	Definition
	Notation of GPU cluster
G	Set of available GPUs
L	Set of network links
S	Set of switches
f_g	Performance metric (FLOPS) of GPU $g \in G$
m_g	Memory capacity of GPU $g \in G$
R	Set of available paths between GPUs
b_l	Bandwidth of network link $l \in L$
t_e^p	Bandwidth of propagation delay $e \in L$
t_s^s	Switching delay of switch $s \in S$
	Notation of DL task execution information
K	Set of execution service
γ_k	Throughput requirement of service $k \in K$
δ_k	Acceptable time of service $k \in K$
a_k	DL model used for service $k \in K$
V_a	Set of operations of DL model a
E_a	Set of edges indicating the correspondence of operations in DL model a
$R^v_{a,v,v'}$	Set of paths between operations $v, v' \in V_a$ in DL model a
w_v	Memory consumption for operation $v \in V_a$
$t^g_{v,g}$	Computation time for operation $v \in V_a$ on GPU $g \in G$
d_e	Output data size between operations corresponding to edge $e \in E_a$
v_e^s	Source node in operation graph edge e
v_e^t	Target node in operation graph edge e
	Notation of mapping of tasks to GPU clusters
$\chi(v,g)$	Mapping of operation v to GPU g
$v(e,r)$	Mapping of edge indicating the correspondence of operations e to path r

GPU cluster information

We represent the sets of available GPUs, network links, and switches as G, L, and S, respectively. For each GPU $g \in G$, we define the floating-point operations per second (FLOPS) f_g and GPU memory capacity m_g. We define the set of paths that can be established between any GPU pair as R. Each path is a subset of the set L of network links. For each network link $l \in L$, we define the bandwidth b_l and the propagation delay t_l^p. For each switch $s \in S$, we define the switching processing time t_s^s.

Task execution information

In this chapter, we represent a set of execution services as K. For each service, we denote a_k using the DL model used for each service $k \in K$. As the execution information of a service

$k \in K$, operation graph $G(V_{a_k}, E_{a_k})$ representing the relationship between the operations required to execute DL model a_k, throughput requirement γ_k, and execution time requirement δ_k is provided. An operation corresponds to the layer of the DL model and can be a single pipeline stage in pipeline parallelism. In this chapter, we only target inferences using the DL model in DL service task. Therefore, the operation graph is constructed without considering training processes such as back propagation. Model partitioning and allocation for training is future work.

In the operation graph of DL model a, each node $v \in V_a$ corresponds to the operation, and each edge $e \in E_a$ represents the relationships between operations. For each operation $v \in V_a$, we define the amount of memory consumed to execute the operation as w_v. In addition, we define $t^g_{v,g}$ as the execution time for executing operation $v \in V_a$ on GPU $g \in G$. For each operation graph edge $e \in E_a$, the intermediate data size transferred between the corresponding operations is defined as d_e. These are set by prior profiles. Furthermore, for the operation graph edge e, we define the source node v^s_e and target node v^t_e. We define the set of paths between operations $v, v' \in V_a$ as $R^v_{a,v,v'}$. This is a subset of the set of operation graph edges.

4.3.3 Mapping task to GPU clusters

To represent DL model partitioning and allocation, we map the nodes and edges of the operation graph to the GPUs and paths in the cluster, respectively. Fig. 4.3 shows the mapping of the operation graph and the GPU cluster when the DL model is partitioned by four pipeline stages. Each GPU executes all the operations mapped to the GPU. The number of GPUs to which the operation graph nodes are mapped corresponds to the number of pipeline stages. In addition, communication occurs between GPUs to send data to the next pipeline stage. To determine the path in this communication, the mapping between an operation graph edge and a path between GPUs to which the operations are mapped.

$\chi(v, g)$ denotes the mapping between an operation and a GPU. When an operation $v \in V_a$ of model a is executed on GPU $g \in G$, $\chi(v, g) = 1$ and $\chi(v, g) = 0$ otherwise.

$\upsilon(e, r)$ denotes the mapping between an operation graph edge and the path between GPUs. When an operation graph edge $e \in E_a$ of model a is mapped to a path $r \in R_{g^1,g^2}$ between GPU pairs $g^1, g^2 \in G$, $\upsilon(e,r) = 1$ and $\upsilon(e,r) = 0$ otherwise.

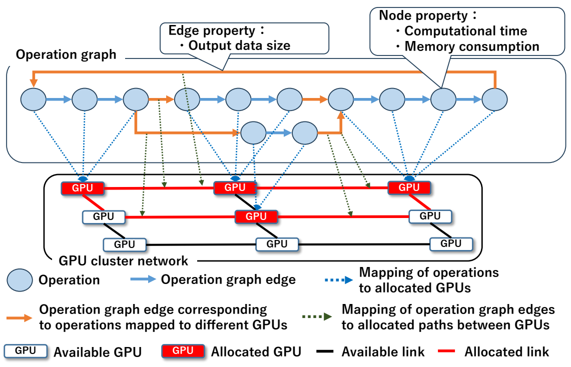


Figure 4.3: Example of mapping of operation graph to GPU cluster.

4.4 Resource aware model partitioning and allocation (RAMPA)

In this chapter, we propose RAMPA to run several DL services while satisfying performance requirements. RAMPA determines (1) the number of pipeline stages, (2) GPUs executing each pipeline stage and the path between GPUs, and (3) operations corresponding to each pipeline stage. First, we formulate the impact of model partitioning, allocated GPUs, and paths on execution time and throughput. Next, we define the allocation costs for GPUs and network links in the cluster to avoid the allocation of resources required for future requested services. Finally, we define an optimization problem to determine the model partitioning and allocation that can minimize the allocation cost while satisfying the performance requirements to execute numerous DL services simultaneously.

4.4.1 Impact of model partitioning and resource allocation on execution time and throughput

We formulate the impact of model partitioning and resource allocation on throughput and execution time.

Throughput of task

In pipeline parallelism, when the execution GPU for the next pipeline stage is processed, the input data from the previous stage wait until the GPU becomes available. Therefore, data are sent to the next stage in a cycle that does not cause data conflicts. We call this cycle the pipeline cycle. Because the pipeline stages can be processed in parallel, the throughput is the inverse of the pipeline cycle. Throughput P_k of service $k \in K$ is obtained as follows:

$$P_k = \frac{1}{T_k^u} \tag{4.1}$$

where T_k^u denotes the pipeline cycle of service k.

Execution time of task

The execution time of the service is the time from the data input to the first pipeline stage until the completion of execution in the last pipeline stage. Because data are sent to the next stage of every pipeline cycle, the execution time of the service is the sum of the pipeline cycle and communication delay between the pipeline stages in the flow of input data. In pipeline parallelism, execution time depends on the processing of the pipeline stage with the highest latency. Therefore, the execution time of a service is the maximum value of the execution time when the input data passes through the corresponding pipeline stage in each path of the operation graph. Execution time T_k^r of service $k \in K$ is obtained as follows:

$$T_k^r = \max_{y \in P_{v_k^f, v_k^e}} \left\{ T_k^u + \sum_{e \in y} \sum_{r \in R} \upsilon(e, r) \left(T_k^u + T_{a_k, e}^c \right) \right\} \tag{4.2}$$

where T_k^u denotes the pipeline cycle of service k and $T_{a_k,e}^c$ denotes the communication delay in the path mapped to operation graph edge e. v_k^f and v_k^e represent operations that receive data first and output data last, respectively.

Pipeline cycle

In pipeline parallelism, the input data from the previous stage are processed until a GPU is available. In addition, communication between pipeline stages and pipeline stage processing can overlap [63]. Therefore, to complete processing without data conflicts in pipeline parallelism, where data transition is only forward, the pipeline cycle is the maximum execution time of each pipeline stage and the communication delay between stages. However, if the operation graph has backward edges and the operations are repeated, the data sent from a later stage to the previous stage must also be considered. In this case, the pipeline cycle is the maximum execution time of the set of pipeline stages to be repeated, in addition to the time mentioned above. Therefore, pipeline cycle T^u_k for service $k \in K$ is obtained as follows:

$$\begin{aligned} T^u_k = \max_{e \in E} \sum_{r \in R} & \upsilon(e, r) \cdot \\ & \begin{cases} \max\left(T^e_{a_k, v^s_e},\, T^c_e\right) & e\ is\ forward \\ \max\left(T^b_{a_k, v^t_e, v^s_e},\, T^c_e\right) & e\ is\ backward \end{cases} \end{aligned} \tag{4.3}$$

where $T^e_{a_k, v^s_e}$ denotes the execution time in a pipeline stage corresponding to operation v^s_e and T^c_e denotes the communication delay in the path mapped to the operation graph edge e. $T^b_{a_k, v^t_e, v^s_e}$ denotes the execution time from the pipeline stage corresponding to operation v^t_e to the pipeline stage corresponding to operation v^s_e.

Execution time in pipeline stage

The execution time of the pipeline stage corresponding to operation v is the sum of the execution latencies of all operations mapped GPUs that operation v is mapped to. Execution time $T^e_{a,v}$ of the pipeline stage corresponding to operation v for model a is obtained as follows:

$$T^e_{a,v} = \sum_{g \in G} \chi(v, g) \left(\sum_{v' \in V_a} \chi(v', g) t^g_{v', g} \right) \tag{4.4}$$

Communication delay between pipeline stages

The communication delay in the path mapped to operation graph edge e is the sum of the time required to obtain the head of the intermediate output data corresponding to operation graph edge e and the transmission delay. The transmission delay is the sum of propagation delay t_l^p of each network link $l \in r$ and switching delay t_s^s of switch $s \in S$ on path $r \in R$ to which operation graph edge e is mapped. Communication delay T_e^c required to transfer data on the path that operation graph edge e is mapped to is obtained as follows:

$$T_{a,e}^c = \sum_{r \in R} \upsilon(e,r) \cdot \sum_{l \in r} \left(T_l^p + T_{n_{l,r}^s}^s + \frac{d_e}{b_l} \right) \tag{4.5}$$

where $\frac{d_e}{b_l}$ denotes the time required to obtain the head of intermediate output data. $n_{l,r}^s$ denotes the source switch when passing through link l on path r.

Execution time from one pipeline stage to another pipeline stage

The execution time from the pipeline stages corresponding to operation v to the pipeline stage corresponding to operation v' is the sum of the execution time in the pipeline stages and the communication delay between the pipeline stages. Therefore, the execution time from the pipeline stages corresponding to operation v to the pipeline stage corresponding to operation v' for DL model a is obtained as follows:

$$\begin{aligned} &T_{a,v,v'}^b = \\ &\max_{y \in P_{v.v'}} \left\{ T_{a,v}^e + \sum_{e \in y} \sum_{r \in R} \upsilon(e,r) \left(T_{a,v_e^t}^e + T_{a,e}^c \right) \right\} \end{aligned} \tag{4.6}$$

4.4.2 Model partitioning and resource allocation problem

We aim to execute several services simultaneously in a cluster with heterogeneous GPUs. To achieve this objective, we avoid allocating important GPUs and network links that may be required for future tasks. This policy is similar to that of RA-CNP shown in Chapter 2. However, this method does not target DL service task execution using pipeline parallelism in GPU clusters. In this chapter, we define an optimization problem for model partitioning and resource allocation to execute numerous services simultaneously. We define the allocation costs for GPUs and network links based on the importance of future service task execution and minimize the costs of the allocated GPUs and network links. Thereafter, we define an optimization problem to determine the optimal model partitioning and resource allocation to execute numerous DL services while satisfying the performance requirements of the services based on the allocation cost and the impact of model partitioning, allocated GPUs, and paths on the execution time and throughput.

Allocation cost

We define the allocation costs for GPUs and network links in the GPU cluster.

GPU allocation cost GPUs with higher computational and memory capacities are more capable of satisfying performance requirements. Furthermore, GPUs in the pools with more available GPUs have more GPUs in close proximity. This implies that low latency communication between pipelines is probable. Therefore, GPUs with high computing and memory capacities, and several available GPUs in the corresponding pool are important. We define GPU allocation cost as the product of these factors. GPU allocation cost C_g^g for GPU $g \in G$ is obtained as follows:

$$C_g^g = f_g \cdot m_g \cdot q_g \tag{4.7}$$

where q_g denotes the number of available GPUs in the pool with GPU g.

Network link allocation cost Network links used as paths between important GPUs are essential. Furthermore, the path length must be short for low latency communication. Therefore, the network links, which may be the shortest paths between important GPU pairs, are essential. This policy is similar to that in our previously proposed resource allocation method [19]. We set the network link allocation cost using the same policy as that used in this chapter.

First, for network link $l \in L$, we define the potential to be on the shortest path for the GPU pair $g^1, g^2 \in G$. The larger the proportion of the number of shortest paths through link l to the number of shortest paths for GPU pair g^1, g^2, the more likely is the network link to be on the path of the GPU pair. Therefore, the potential for link $l \in L$ to be the link on the shortest path for the GPU pair g^1, g^2 is obtained as follows:

$$u_{g^1,g^2}(l) = \frac{N^r_{g^1,g^2}(l)}{N^r_{g^1,g^2}} \tag{4.8}$$

where $N^r_{g^1,g^2}(l)$ denotes the number of shortest paths through link l and $N^r_{g^1,g^2}$ denotes the number of shortest paths for GPU pair g^1, g^2. The importance of a GPU pair is the sum of the costs of the two GPUs divided by the shortest number of hops between them because more distant GPUs have a larger communication latency. The network link allocation cost C^l_l for network link $l \in L$ is obtained as follows:

$$C^l_l = \sum_{g^1,g^2 \in G} u_{g^1,g^2}(l) \cdot \frac{C_{g^1} + C_{g^2}}{H_{g^1,g^2}} \tag{4.9}$$

where H_{g^1,g^2} denotes the shortest hop between GPU pair $g^1, g^2 \in G$.

Optimization problem

We define an optimization problem that outputs mappings between operations and executing GPUs, and between operation graph edges and paths based on information about the GPU cluster and DL service task execution.

Objective

The objective is to minimize the sum of the allocation costs of GPUs and network links allocated to the DL service, that is,

$$\begin{aligned} minimize \quad & \textstyle\sum_{g \in G} 1_{\sum_{v \in V_{a_k}} \chi(v,g) > 0} C^g_g + \\ & \textstyle\sum_{r \in R} \sum_{e \in E_{a_k}} \upsilon(e,r) \sum_{l \in r} C^l_l \end{aligned} \tag{4.10}$$

where $1_{\sum_{v \in V_{a_k}} \chi(v,g) > 0}$is 1 when $\sum_{v \in V_{a_k}} \chi(v,g) > 0$ and 0 otherwise.

Constraints

Mapping constraint These operations must be mapped to an available GPU. The operation graph edge must be mapped to the path between the two GPUs mapped to operations corresponding to the adjacent nodes of that operation graph edge, that is,

$$\forall v \in V_{a_k}, \ \sum_{g \in G} \chi(v, g) = 1 \tag{4.11}$$

$$\begin{aligned} &\forall e \in E_{a_k}, \\ &\sum_{g^1, g^2 \in G} \ \sum_{r \in R_{g^1, g^2}} \upsilon(e, r) = \chi(v_e^s, g^1) \cdot \chi(v_e^t, g^2) \end{aligned} \tag{4.12}$$

where v_e^s and v_e^t denote the source and target nodes of operation graph edge $e \in E_{a_k}$, respectively. R_{g^1, g^2} denotes the set of paths between GPUs $g1, g2 \in G$.

Throughput requirement The throughput of the allocated DL service $k \in K$ must be larger than the throughput requirement, that is,

$$\forall k \in K, \ \gamma_k \leq P_k \tag{4.13}$$

Execution time requirement The execution time of the allocated DL service $k \in K$ must be smaller than the execution time requirement, that is,

$$\forall k \in K, \ \delta_k \geq T_k^r \tag{4.14}$$

Memory requirement The total memory consumption of the operations mapped to the GPU and the data size input to the pipeline stage corresponding to that GPU must be less than or equal to the GPU memory capacity, that is,

$$\begin{aligned} &\forall k \in K, \forall g \in G, \\ &\sum_{v \in V_{a_k}} \chi(v, g) w_v + \sum_{e \in E_{a_k}} \overline{\chi}(v_e^s, g) \chi(v_e^t, g) d_e \leq M_g \end{aligned} \tag{4.15}$$

where $\overline{\chi}(v, g)$ denotes the inversion of $\chi(v, g)$. If $\chi(v, g)$=1, the value is 0. If $\chi(v, g)$=0, the value is 1.

Solving this optimization problem minimizes the allocation of important GPUs and paths for service task execution while satisfying the performance requirements.

4.4.3 Model partitioning and allocation based on ant colony optimization

To derive the optimization problem defined in Section 4.4.2, we searched for mappings between the operation graph nodes and GPU, operation graph edges, and paths. However, such mappings are a binomial combinational optimization problem, and resource allocation based on the binomial combinational optimization problem is NP-hard [34]. To address such problems, metaheuristic methods have been used to address these problems. In this chapter, we solve this problem using ant colony optimization (ACO).

ACO is a population-based metaheuristic method in which multiple agents probabilistically search for solutions. ACO is a flexible method that can adapt to changes in the environment [35] and can flexibly search for solutions even if the resource utilization status in a cluster changes. In ACO, the pheromone values are first assigned to GPUs and network links. The higher the pheromone values of the GPU and network link, the more likely they are to be selected by the agent. After multiple agents probabilistically search for a solution based on pheromones, an optimal solution is selected from the searched solutions. Finally, the pheromone value in the optimal solution is increased. This process is repeated several times.

VNE-AC was proposed for resource mapping using ACO [34]. However, this method allocates only the shortest routing paths. In this chapter, because network link allocation costs are not directly related to communication delays, the performance requirements may not be satisfied because of communication delays between GPUs if a path is allocated according to the shortest path problem. We arranged and used VNE-AC to use ACO to select network links. However, any method can be used as long as the solution can be derived.

In deriving the solution using ACO, we change the number of pipeline stages from one to the number of operations and determine the lowest cost solution for each number of pipeline stages.

Thereafter, we derive an optimal solution by selecting the lowest cost among these solutions. To search for a solution, the following steps are performed: (1) GPU search, (2) network link search, (3) Performance requirement check, and (4) Pheromone update. If the allocation cost exceeds the current minimum allocation cost, then the process is rejected to avoid unnecessary processes. The notations used for ACO are listed in Table 4.2.

Table 4.2: Notation of model partitioning and allocation based on ant colony optimization.

Symbols	Definition
τ_r	Pheromone of GPU or link r
α	Pheromone weight
β	Resource allocation cost weight
ρ	Pheromone decrease rate
ϕ	Pheromone increase rate
G^b	Set of GPUs in current best solution
L^b	Set of links in current best solution

GPU search

During the GPU search, the agent probabilistically selects the GPU corresponding to each pipeline stage from the available GPUs. The objective is to minimize the allocation cost; therefore, the allocation probability of GPUs with a low cost is set high. We define GPU $g \in G$ allocation probability p_g^g as follows:

$$p_g^g = \frac{(\tau_g)^\alpha \left(\frac{1}{(C_g^g)^\beta}\right)}{\sum_{x \in G}\left[(\tau_x)^\alpha \frac{1}{(C_x^g)^\beta}\right]},$$

Network link search

In a network link search, the agent generates sub-agents to explore the paths between the GPUs selected in the GPU search. Each sub-agent probabilistically selects a network link from the source GPU. Next, the sub-agent probabilistically selects the next network link from the destination node of the first link. This process is repeated until the destination GPU is reached. We define network link $l \in L$ and allocation probability $p_{l,n}^{l}$ as follows:

$$p_{l,n}^{l} = \frac{(\tau_l)^{\alpha} \frac{1}{(C_l^l)^{\beta}}}{\sum_{x \in L} \left[(\tau_x)^{\alpha} \frac{1}{(C_x^l)^{\beta}} \right]}$$

Performance requirement check

In this phase, we check whether the performance requirements are satisfied when the DL service is executed by the selected GPUs and paths. First, we calculate the throughput, execution time, and memory consumption for each combination of operations executed at each pipeline stage. Then, we check whether a combination exists that satisfies the performance requirements. If no combination satisfies the performance requirements, the process is rejected.

Pheromone update

After the performance requirement check, pheromones of all GPUs and network links decay based on the pheromone reduction rate rho. However, only the pheromones of the GPU and network link in the optimal solution for each iteration are augmented based on the pheromone increase rate phi and the allocation cost value. The pheromone enhancement value h is obtained as follows:

$$h = \frac{\phi}{\sum_{g \in G^b} C_g^g + \sum_{l \in L^b} C_l^l}$$

The pheromones τ_g, τ_l for GPU $g \in G$ and network link $l \in L$ are updated as follows:

$$\tau_g = \rho \tau_g + h, \;\; \tau_l = \rho \tau_l + h,$$

4.5 Evaluation

We evaluate RAMPA through simulations of a cluster with heterogeneous GPUs and DL service allocation.

4.5.1 Environment

We describe the cluster with heterogeneous GPUs, execution services, and comparative methods used to evaluate RAMPA.

Cluster with heterogeneous GPUs

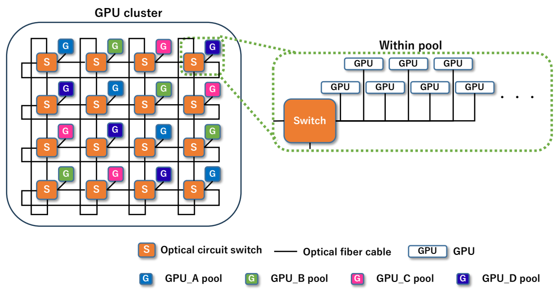
We assume a GPU cluster comprising 16 GPU pools. Each pool has 40 GPUs. We evaluate RAMPA in the following three GPU clusters with different GPU and network performances:

- Base cluster: Neutral GPU cluster for comparison.
- High-bandwidth cluster: GPU cluster with high network bandwidth.
- High-performance cluster: GPU cluster with many high-performance GPUs.

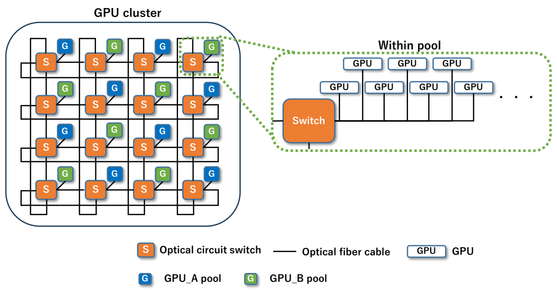
A high-bandwidth cluster differs from the base cluster only in terms of the bandwidth of each network link. In a high-performance cluster, relatively low-performance GPUs are removed from the base cluster. In all other aspects, the base and high-performance clusters were identical. Fig. 4.4a shows the base and high-bandwidth clusters and Fig. 4.4b illustrates a high-performance cluster. For stable and fast communication between GPUs, the same type GPUs in each pool are connected to an optical circuit switch. The optical circuit switches are connected to each other by an optical fiber to form an network. We assume that the network topology is a two-dimensional torus topology of 4×4 with 16 optical circuit switches. For flexible routing, an optical circuit switch pear is connected to four optical fibers. Connected GPUs are the following four types: NVIDIA L4 [64], NVIDIA V100 [65], NVIDIA A30 [66], and NVIDIA Tesla T4 [67]. Hereinafter, we refer to these as GPU_A, GPU_B, GPU_C, and GPU_D, respectively. In the high-performance cluster, eight pools with only GPU_A and GPU_B are connected. GPU_A and GPU_B performed better than GPU_C and GPU_D in terms of memory capacity and FLOPS.

The parameters of the GPU clusters used to estimate the performance and set the allocation costs are listed in Table 4.3. The bandwidth in the base and high-performance clusters is 10 Gbps and that in the high-bandwidth cluster is 100 Gbps. The network link length within a pool is 10 m, and that between pools is 20 m. The propagation delay of each network link is 0.05 μs and 0.1

μs. By referencing CALIENT’s optical circuit switch [50], we set the switching delay to 0.03 μs. GPU FLOPS, memory bandwidth, and memory capacity are based on the data sheet of each GPU.



(a) Network for base and high-bandwidth clusters.



(b) Network for high-performance cluster.

Figure 4.4: Networks used in evaluation.

Table 4.3: Parameter settings for GPU cluster.

Parameters	Value
GPU_A FLOPS	30.3 TFLOPS
GPU_A memory size	24GB
GPU_A memory bandwidth	300GB/s
GPU_B FLOPS	14 TFLOPS
GPU_B memory size	32GB
GPU_B memory bandwidth	900GB/s
GPU_C FLOPS	10.3 TFLOPS
GPU_C memory size	24GB
GPU_C memory bandwidth	933GB/s
GPU_D FLOPS	8.1 TFLOPS
GPU_D memory size	16GB
GPU_D memory bandwidth	320GB/s
Propagation delay (Switch - GPU)	0.05 μs
Propagation delay (Switch - Switch)	0.1 μs
Switch latency of the optical circuit switch	30 ns
The bandwidth (Base / high bandwidth)	10 Gbps/100Gbps

Execution task

We assume tasks for the following DL services with different characteristics in throughputs, execution latencies, and memory requirements:

- High-throughput service：Object recognition service for video streaming using YOLOS [68].

- Low-delay service： Image classification service using vision transformer [68].
- Huge model service： AI chat bot service using Gemma2 [68].

DL models used in this chapter were downloaded from the Hugging Face. The throughput, execution time requirements, and usage DL model for each service are listed in Table 4.5. Throughput refers to the number of transactions per second (tps), where one inference of an input datum is a transaction. The execution time is defined as the time required to complete a transaction. In the high-throughput, low-delay, and huge model services, one input data is set to one frame, one image, and 1000 characters of text, respectively. We set the proportion of the number of services to be executed in the following four environments to evaluate RAMPA:

- Same demand for all services：The proportion of the number of executed service is balanced.
- High demand for high-throughput services：The proportion of the number of high-throughput service is high.
- High demand for low-delay services： The proportion of the number of low-delay service is high.
- High demand for huge model services： The proportion of the number of huge model service is high.

To simulate each environment, we generate resource allocation request for each service with a certain probability. The probabilities of each service are listed in Table 4.4.

Table 4.4: Generation probability of resource allocation requests.

Service	High throughput	Low delay	Huge model
Same demand for all services	Same probability		
High demand for high-throughput service	0.8	0.1	0.1
High demand for low-delay service	0.1	0.8	0.1
High demand for huge model service	0.1	0.1	0.8

Table 4.5 lists the layer names corresponding to the operations of each DL model, corresponding floating-point operations (FLOPs), amount of memory consumed, and size of the output data for each operation. The FLOPs and output data sizes are set by profiling using calflops [69]. Memory

consumption is set to the parameter size of each operation multiplied by 1.2. The reason for multiplying by 1.2 is to consider the overhead of the consumed memory. In this evaluation, we set the execution time of the operation for each GPU type based on the values shown in 4.3. Execution time is the sum of FLOPs divided by GPU FLOPS and memory consumption divided by GPU memory bandwidth. Therefore, execution time $t^g_{v,g}$ of operation v in GPU g can be obtained as follows:

$$t^g_{v,g} = \frac{FLOPs(v)}{f_g} + \frac{d_v}{Memory\_Band(g)}$$

where $FLOPs(v)$ denotes the FLOPS of operation v and $Memory\_Band(g)$ is the memory bandwidth of GPU g. However, more appropriate execution time estimation methods may exist, which will be a topic for future studies.

Table 4.5: Deep learning model and performance requirements.

Service	High throughput	Low delay	Huge model
Model	yolos-base [68]	vit-huge [70]	gemma-2-27b [71]
Execution time (s)	0.1	0.05	10
Throughput (tps)	60	30	0.1

Table 4.6: Parameter settings for each deep learning model.

Operation	FLOPs	memory consumption	output data size
yolos-base [68]			
Embedings	3.9G	192.55MB	109.60MB
YolosLayer (× 12)	48.17 G	32.46MB	9.96MB
LayerNorm	13.06 M	0	9.96MB
YolosPooler	1.18 M	2.7MB	3KB
vit-huge [70]			
ViTEmbeddings	385.68 M	4.94MB	1.25MB
ViTLayer (× 32)	10.11 G	90.08MB	1.25MB
LayerNorm	1.64 M	11.72KB	1.25MB
ViTPooler	3.28 M	7.51MB	5.00KB
gemma-2-27b [71]			
Embedding	0	5.28GB	5.98MB
GemmaDecoderLayer (× 46)	1.15 T	2.52GB	5.98MB
GemmaRMSNorm	0	20.4KB	5.98MB
Linear	2.36 T	5.28GB	332.03MB

Parameter settings for ant colony optimization

We use ACO for GPU and path allocation. Parameters for ACO are listed in Table 4.7.

Table 4.7: Parameter settings for ant colony optimization.

Parameters	Value
Number of agents	20
Number of agent generations	20
Pheromone decrease rate	0.1
Pheromone increase rate	100
Pheromone weight	2
Allocation cost weight	1
Initial pheromone value	1000

Comparative methods

RAMPA can optimize model partitioning and allocation by considering both the importance of allocated resources and number of allocated resources. To demonstrate their effectiveness, we compared them using the following two methods.

PipeEdge (PE) PipeEdge [63] is a model partitioning and allocation method to maximize throughout by considering the performance of allocated resources. This method minimizes the maximum execution time of each pipeline stage by preferentially selecting high-performance GPUs and low-latency paths. The number of pipeline stages is fixed for each model in advance. In comparison with this method, we demonstrate the effectiveness of the objective setting of RAMPA.

No considering allocated resource number(NCAR) NCAR is a model partitioning and allocation method that minimizes the allocation of important GPUs and paths to execute tasks while satisfying the service performance requirements. In this method, the number of pipeline stages is fixed for each model in advance. This method is similar to RAMPA in terms of its objective. However, it differs in the absence of consideration of the number of allocated resources. In comparison with this method, we demonstrate the effectiveness of model partitioning and allocation by considering the number of resources allocated.

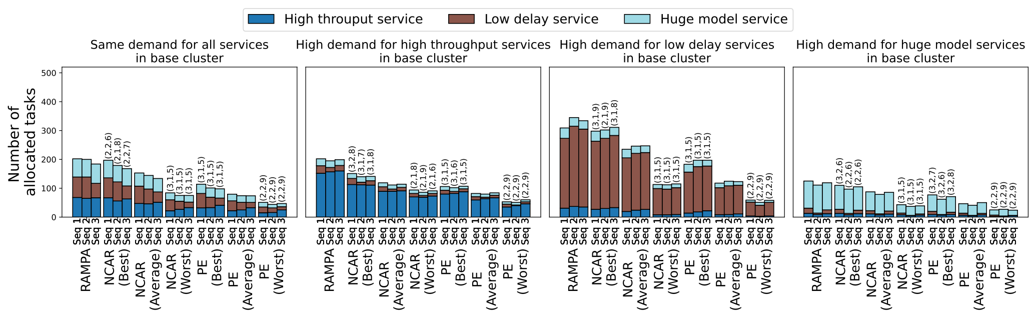
In the comparison methods, the number of pipeline stages is fixed for each DL model. We evaluate the comparative methods for all possible pipeline stage number patterns to demonstrate the effectiveness of optimizing model partitioning and allocation, including the number of allocated resources. Therefore, we set the number of pipeline stages to range from the number of pipeline stages that can be executed on any GPU in this evaluation environment to the number of pipeline stages that can be executed only on a GPU with high-performance (GPU_A and GPU_B). The high-throughput service ranges from 2 to 3, the low-delay service ranges from 1 to 2, and the huge model service ranges from 5 to 9. We evaluate all the combinations of the number of pipeline stages for the three types of services for each comparison method. Hereinafter, in the comparison method PE, when the number of pipeline stages for high-throughput, low-delay, and huge model services are 2, 1, and 6, respectively, it is denoted as PE(2,1,6).

Metric

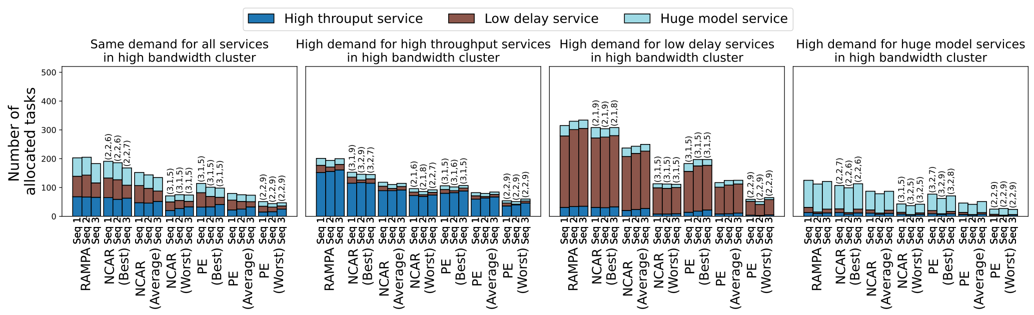
We measure the number of allocated tasks to evaluate the ability of RAMPA to execute more DL services. In the evaluation, we continue to generate resource allocation requests until resource allocation that can satisfy performance requirements fails. The evaluation terminates when the allocation of a service that satisfies the performance requirements fails.

4.5.2 Number of allocated tasks

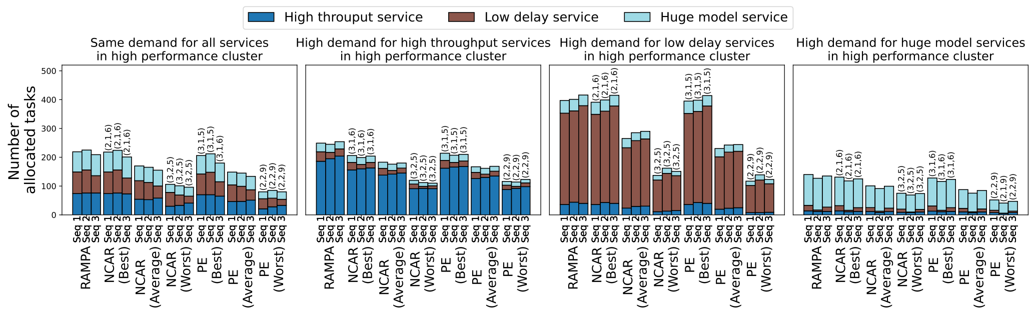
In Fig. 4.5, we show the number of services successfully allocated that satisfy the performance requirements for RAMPA and the comparative methods for all combinations of the three different GPU clusters and four different proportions of executed services. We measure 20 different combinations of pipeline stages using comparative methods. The results of the comparative methods are shown in Fig. 4.5; we show the best case, worst case, and average of all the combinations. The numbers above the bars represent the corresponding number of pipeline stages.



(a) Base cluster.



(b) High-bandwidth cluster.



(c) High-performance cluster.

Figure 4.5: Number of allocated tasks.

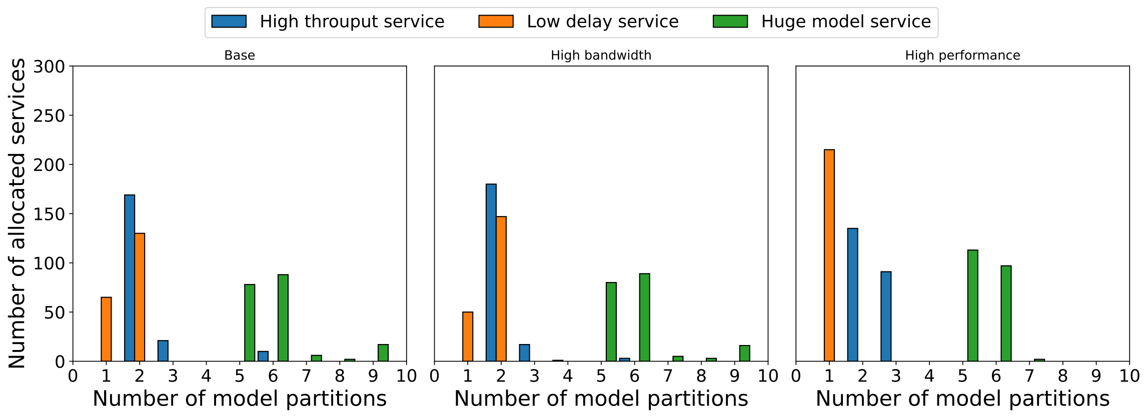
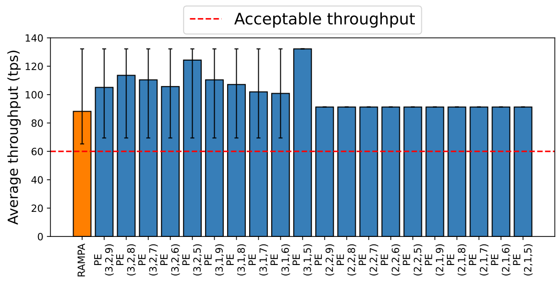


Figure 4.6: Distribution of number of pipeline stages of allocated tasks in RAMPA.

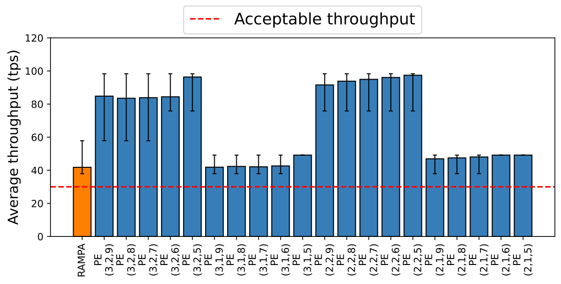
First, we compare NCAR and PE, which differ only in terms of their objectives. NCAR tends to be able to allocate more services. This is because it preserves the required GPUs to execute the future requested services by avoiding the allocation of resources used by other services. The effectiveness of considering the resource utilization of other services to allocate more services was demonstrated. Next, we compare RAMPA and NCAR. In all environments, RAMPA allocates more services to satisfy the performance requirements and improves the number of executed tasks by up to 30%. This result indicates that optimizing the selection of allocated resources is not sufficient for the efficient use of resources. To execute more services, considering the number of resources allocated is necessary. To verify the model partitioning that was performed by RAMPA, we show the distribution of the number of pipeline stages of allocated tasks in the case of the same demand for all services in Fig. 4.6. As shown in Fig. 4.6, RAMPA partitions the DL model based on the number of pipeline stages in multiple patterns. This implies that the suitable model partitioning strategy changes depending on the resource allocation situation. For the execution of numerous services, the optimization of the resources used and their number is effective based on the current resource allocation situation.

4.5.3 Comparison of task execution performance

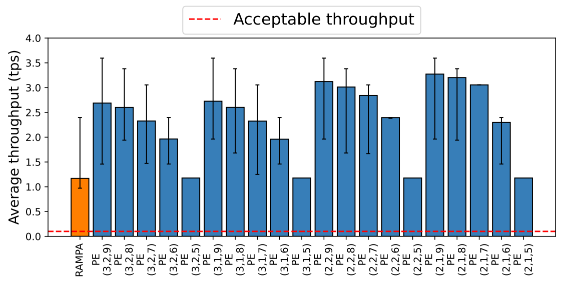
Unlike conventional methods, RAMPA does not aim to maximize performance. Therefore, it may be inferior to conventional methods in terms of service task execution performance. We compare the service throughput and execution time of RAMPA and PE and discuss the limitations of RAMPA on execution performance. Fig. 4.7 and 4.8 show the average values of throughput and execution time for each service in the case of the same demand for all services in the base cluster. Error bars represent the maximum and minimum values. The red lines indicate the performance requirements. The orange bars in the figure represent RAMPA and the other bars represent PE.



(a) High-throughput service.

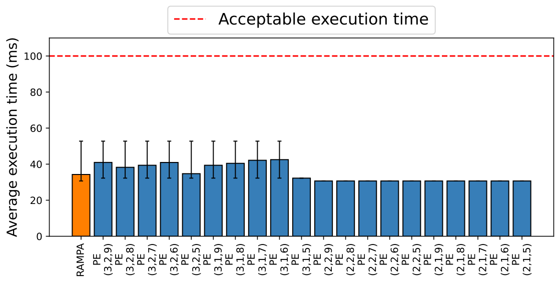


(b) Low-delay service.

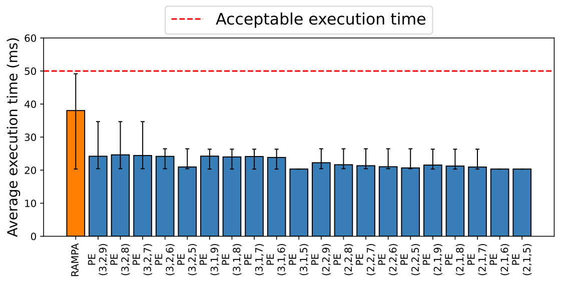


(c) Huge model service.

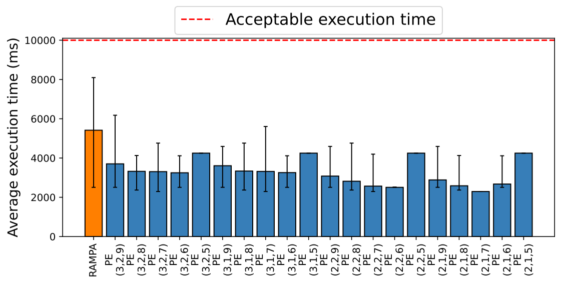
Figure 4.7: Throughput of tasks allocated in each method.



(a) High-throughput service.



(b) Low-delay service.



(c) Huge model service.

Figure 4.8: Execution time of tasks allocated in each method.

RAMPA had a smaller average and minimum throughput than PE for all combinations of pipeline stage numbers. Because PE aims to maximize throughput, this result is similar to that of Hu et al. [63]. In execution time, for services other than high-throughput services, PE was superior. By contrast, in the high-throughput service, PE had a lower execution time than RAMPA only when the number of pipeline stages was two. This is because the execution time is affected by the communication delay between pipeline stages. If excess pipeline stages exist, then the communication delay overhead will increase. When the model was properly partitioned, RAMPA exhibited a lower service task execution performance than PE. However, the performance requirements were satisfied. RAMPA is effective when performance requirements are properly set and performance maximization is not required.

4.5.4 Computational time for model partitioning and allocation by RAMPA

Unlike conventional methods, RAMPA considers the resource allocation and optimizes the number of resources allocated. Because of these processes, more computational time is required than in conventional methods. The relationship between computational complexity and computation time is discussed as a limitation of RAMPA.

Computational complexity of RAMPA

We verified the computational complexity of RAMPA by clarifying the computational complexity of each of the processes described in Section 4.4.3.

GPU search In GPU search, the GPU corresponding to each pipeline stage is selected from the available GPUs in the GPU cluster. Therefore, the GPU selection is repeated for the number of pipeline stages, and all available GPUs in the cluster are checked in each iteration. When the number of pipeline stages is N^p and the number of available GPUs is $|G|$, the computational complexity is $O(N^p|G|)$.

Network link search In this phase, we probabilistically select the transit link from the source GPU to the destination GPU. This process is repeated $N^p - 1$, because it is performed to establish the path between GPUs when the number of pipeline stages is N^p. In addition, in the worst-case scenario, a path through all the nodes is established. Therefore, the computational complexity is $O(N^p + |G|)$.

Performance requirement check In this phase, we check the performance requirements for each combination of operations executed for each pipeline stage. When the number of pipeline stages is N^p and that of operations is $|V_a|$, the computational complexity is $O(|V_a|N^p)$.

Pheromone update After GPU and path selection, the pheromones are updated for all GPUs and network links. Therefore, the computational complexity is $O(|G| + |L|)$.

The maximum number of pipeline stages is equal to the number of operations in the model. Therefore, the above process is repeated from 1 to $|V_a|$ at the maximum. Thus, the computational complexity of RAMPA is $O(|V_a|^2 \cdot (|G| + |L| + |S|) + |V_a|^3)$. From this perspective, the computational complexity of RAMPA depends on the scale of the cluster and DL model.

Relationship between number of operations and average computational time

We investigate the relationship between the computational complexity and computation time of RAMPA to discuss the practicality of RAMPA. In this chapter, we assume clusters of scales as shown in Fig. 4.4. Therefore, we measure the computational time for the cluster, as shown in Fig. 4.4. Because $|G| + |L| + |S|$ is constant within the same cluster, the computational complexity is $O(|V_a|^3)$. The relationship between the number of operations and the average computational time for each model is shown in Fig. 4.9. Error bars represent 95% confidence interval.

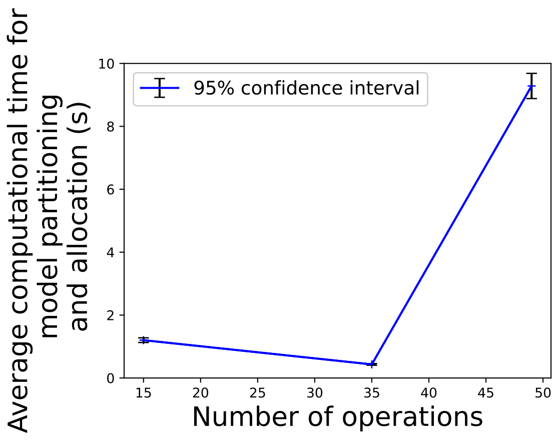


Figure 4.9: Relationship between number of operations and average computational time for model partitioning and allocation by RAMPA.

In the huge model service with 49 model operations, the computation time is approximately 9 s. For a high-throughput service with 15 model operations, the computation time is approximately 1 s. This result matches the computational complexity $O(|V_a|^3)$. However, the low-delay service had a shorter computation time than the high-throughput service, even though the number of operations was larger. This is because the number of pipeline stages required to satisfy performance requirements is significantly constrained. Once the number of pipeline stages exceeds a certain level, it becomes impossible to satisfy the execution time requirements and the processes are quickly terminated.

The results also demonstrate that, in the cluster assumed in this chapter, processing is completed within 10 s, even for large models. This is an acceptable time for the allocation of services after resource allocation. One solution is to avoid the search process by reserving the execution resources. Estimating the resources that can satisfy the performance requirements of future requested services based on predictions of future resource allocation situations in the cluster is necessary. Proposals for these methods will be investigated in future studies.

4.6 Conclusion

We proposed RAMPA to execute more DL services while satisfying the performance requirements in clusters of heterogeneous GPUs of a disaggregated data center. RAMPA minimizes the allocation of important resources for future DL service task execution to avoid inhibiting future DL service task execution. We defined the resource allocation cost for GPU and network links in terms of resource importance. Furthermore, we formulated the impact of model partitioning, allocated GPUs, and paths on the execution time and throughput. To comprehensively consider resource allocation and model partitioning, we defined an optimization problem to minimize resource allocation costs while satisfying the service performance requirements. We evaluated the effectiveness of RAMPA by simulating the execution of DL services. The results demonstrated that more service tasks can be executed while satisfying the performance requirements compared to the conventional method. By RAMPA, we achieved efficient GPU utilization to deliver many DL service tasks in clusters with heterogeneous GPUs.

Chapter 5

Conclusion

μDDCs can efficiently use resources such as CPUs, GPUs, and memory. From this advantage, it is effective for providing many services in an environment where resources are limited. On the other hand, because network and resource allocation have a significant impact on the execution of service tasks, efficient network topology and resource allocation for a μDDC are required. In this thesis, we proposed resource allocation methods and optical network topology to execute many tasks simultaneously in a μDDC.

First, we proposed a resource allocation method that considers the impact of the network on performance. We call this method RA-CNP. In this method, we model the impact of the network on the performance of tasks and verify whether the resource allocation can satisfy performance requirements. Furthermore, we define resource allocation costs for each resource based on whether it is necessary for future task execution. By defining a resource allocation problem to minimize resource allocation costs while satisfying the performance requirements, we achieve resource utilization where the required resources are available when a task is requested. We evaluated the effectiveness of RA-CNP by simulating μDDC networks. We demonstrated that RA-CNP can reduce task blocking to 0 even in environments where task blocking occurs in conventional methods.

Next, we proposed an optical network topology evaluation metric called the capability of simultaneous task execution (*CSTE*) and an optical network topology design based on CSTE. CSTE represents the ratio of resources that could be used as a resource communicating with other resources without violating the performance requirements in a situation where tasks up to the maximum number of executable tasks are executed. μDDC with high CSTE can have a large number of resource pairs capable of communicating satisfying task performance requirements. Furthermore, we formulate an optical network topology design problem aimed at generating an optical network topology capable of maximizing task execution based on CSTE. By solving this optimization problem, we generate optical network topologies capable of executing many tasks simultaneously. We evaluated optical network topologies generated based on CSTE. The results showed that an optimal network topology based on CSTE reduces task blocks by over 50% compared to conventional optical network topologies.

Finally, we extended RA-CNP to estimate the suitable resources for each task. We focused on the deep learning-based tasks that can be partitioned and executed in parallel and proposed resource aware model partitioning and allocation (RAMPA) to execute many tasks while satisfying the performance requirements. First, we extended the model of the impact of the network on the performance to consider the impact of model partitioning on performance. We also extended the resource allocation problem defined in RA-CNP to determine the combination of model partitioning and resource allocation that minimizes the resource allocation costs while satisfying the service performance requirements. We demonstrated that RAMPA can execute more tasks in any environment and improve the number of executed tasks by up to 30% compared to conventional methods.

By resource allocation methods and optical network topology proposed in this thesis, we can configure a μDDC capable of executing many tasks simultaneously. In this μDDC, resources are allocated to preserve the resources required for future tasks while satisfying the performance requirements of services. Furthermore, an optical network topology with a large number of resource pairs capable of communicating satisfying task performance requirements is configured to connect resources and enables flexible resource allocation to execute many tasks simultaneously. By this μDDC, we can maximize the utilization of the limited resources at the edge and provide more edge services.

In this thesis, we assumed that the execution information of each service task, such as the resource type required for task execution and the traffic that occurs between resources, is profiled in advance. However, in cases such as task offloading, it is difficult to know this information. Therefore, it is not possible to accurately estimate the impact of resource allocation on performance. Task execution depends on the program of the task, processed data, and resource performance. By extending our resource allocation method to estimate task execution information based on program and resource performance, we can also handle tasks that are difficult to profile in advance. Such an extension is our future work.

Bibliography

[1] K. Bilal, O. Khalid, A. Erbad, and S. U. Khan, “Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers,” *Computer Networks*, vol. 130, pp. 94–120, Jan. 2018.

[2] “Vapor edge module,” online: https://www.vapor.io/technology/datasheet/. Accessed 18 November 2024.

[3] “Zella dc: Micro data centres,” online: https://www.zelladc.com/. Accessed 18 November 2024.

[4] “Container data center solutions,” https://www.kstar.com/solution/detail/data-center-container.html, (Accessed 21 November, 2024).

[5] T. H. T. Le, N. H. Tran, T. LeAnh, T. Z. Oo, K. Kim, S. Ren, and C. S. Hong, “Auction mechanism for dynamic bandwidth allocation in multi-tenant edge computing,” *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 15 162–15 176, 2020.

[6] S. Han, N. Egi, A. Panda, S. Ratnasamy, G. Shi, and S. Shenker, “Network support for resource disaggregation in next-generation datacenters,” in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, Nov. 2013, pp. 1–7.

[7] G. Zervas, H. Yuan, A. Saljoghei, Q. Chen, and V. Mishra, “Optically disaggregated data centers with minimal remote memory latency: Technologies, architectures, and resource allocation [invited],” *Journal of Optical Communications and Networking*, vol. 10, no. 2, pp. A270–A285, 2018.

[8] R. Lin, Y. Cheng, M. D. Andrade, L. Wosinska, and J. Chen, “Disaggregated data centers: Challenges and trade-offs,” *IEEE Communications Magazine*, vol. 58, no. 2, pp. 20–26, 2020.

[9] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker, “Network requirements for resource disaggregation,” in *Proceedings of 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16).* Savannah, GA: USENIX Association, Nov. 2016, pp. 249–264. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/gao

[10] V. Mishra, J. L. Benjamin, and G. Zervas, “Monet: Heterogeneous memory over optical network for large-scale data centre resource disaggregation,” *Research Article Journal of Optical Communications and Networking*, vol. 1, 2021. [Online]. Available: http://dx.doi.org/10.1364/ao.XX.XXXXXX

[11] N. Terzenidis, M. Moralis-Pegios, G. Mourgias-Alexandris, T. Alexoudi, K. Vyrsokinos, and N. Pleros, “High-port and low-latency optical switches for disaggregated data centers: The hipoλaos switch architecture,” *Journal of Optical Communications and Networking*, vol. 10, no. 7, pp. 102–116, 2018.

[12] X. Guo, X. Xue, F. Yan, B. Pan, G. Exarchakos, and N. Calabretta, “Dacon: a reconfigurable application-centric optical network for disaggregated data center infrastructures [invited],” *Journal of Optical Communications and Networking, Vol. 14, Issue 1, pp. A69-A80*, vol. 14, pp. A69–A80, 1 2022.

[13] A. D. Papaioannou, R. Nejabati, and D. Simeonidou, “The benefits of a disaggregated data centre: A resource allocation approach,” in *Proceedings of 2016 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2016, pp. 1–7.

[14] M. Amaral, J. Polo, D. Carrera, N. Gonzalez, C.-C. Yang, A. Morari, B. D’Amora, A. Youssef, and M. Steinder, “Drmaestro: orchestrating disaggregated resources on virtualized datacenters,” *Journal of Cloud Computing*, vol. 10, pp. 1–20, mar 2021.

[15] O. O. Ajibola, T. E. H. El-Gorashi, and J. M. H. Elmirghani, “Network topologies for composable data centers,” *IEEE Access*, vol. 9, pp. 120 955–120 984, 2021.

[16] A. Ikoma, Y. Ohsita, and M. Murata, “Impact of remote memory and network performance on execution performance of disaggregated micro data centers,” in *Proceedings of 2021 International Conference on Emerging Technologies for Communications*, Dec. 2021.

[17] ——, “Resource allocation method considering future resource requests in a disaggregated micro data center,” in *Technical Reports of IEICE (IN2021-36)*, ser. IN2021-36, vol. 121, no. 434, Online, March 2022, pp. 31–36, thu, Mar 10, 2022 - Fri, Mar 11 : Online (NS, IN).

[18] ——, “Disaggregated micro data center: Resource allocation considering impact of network on performance,” in *Proceedings of 2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)*, 2023, pp. 360–365.

[19] ——, “Resource allocation considering impact of network on performance in a disaggregated data center,” *IEEE Access*, vol. 12, pp. 67 600–67 618, 2024.

[20] ——, “Optical network topology design to execute many tasks simultaneously in a disaggregated data center,” *Journal of Optical Communications and Networking*, vol. 16, no. 7, pp. 764–780, 2024.

[21] J. Pan and J. McElhannon, “Future edge cloud and edge computing for internet of things applications,” *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 439–449, 2018.

[22] A. Y. Ding, E. Peltonen, T. Meuser, A. Aral, C. Becker, S. Dustdar, T. Hiessl, D. Kranzlmüller, M. Liyanage, S. Maghsudi, N. Mohan, J. Ott, J. S. Rellermeyer, S. Schulte, H. Schulzrinne, G. Solmaz, S. Tarkoma, B. Varghese, and L. Wolf, “Roadmap for edge ai: A dagstuhl perspective,” *ACM SIGCOMM Computer Communication Review*, vol. 52, no. 1, pp. 28–33, mar 2022.

[23] L. A. Haibeh, M. C. E. Yagoub, and A. Jarray, “A survey on mobile edge computing infrastructure: Design, resource management, and optimization approaches,” *IEEE Access*, vol. 10, pp. 27 591–27 610, 2022.

[24] M. Ewais and P. Chow, “Disaggregated memory in the datacenter: A survey,” *IEEE Access*, vol. 11, pp. 20 688–20 712, 2023.

[25] X. Lu and A. Kashyap, “Towards offloadable and migratable microservices on disaggregated architectures: Vision, challenges, and research roadmap,” in *Proceedings of the Second Workshop On Resource Disaggregation and Serverless (WORDS 2021), co-located with ASPLOS 2021*, ser. WORDS ’21. ACM, Apr. 2021, pp. 1–7, vision Paper. [Online]. Available: https://wuklab.github.io/words/words21-lu.pdf

[26] G. Vargas-Solar, M. Hassan, and A. Akoglu, “Jita4ds: Disaggregated execution of data science pipelines between the edge and the data centre,” *Journal of Web Engineering*, vol. 21, no. 1, pp. 1–26, Nov. 2021.

[27] Q. Zhang, Y. Cai, S. G. Angel, V. Liu, A. Chen, and B. T. Loo, “Rethinking data management systems for disaggregated data centers,” in *Proceedings of Conference on Innovative Data Systems Research (CIDR)*, jan 2020, pp. 1–8.

[28] A. Saljoghei, M. Enrico, D. Syrivelis, K. Katrinis, A. Reale, M. Bielski, I. Syriogs, D. Pnevmatikatos, D. Theodoropoulos, N. Parsons, G. Zervas, and V. Mishra, “dredbox: Demonstrating disaggregated memory in an optical data centre,” in *Proceedings of Optical Fiber Communication Conference*, 01 2018, p. W1C.1.

[29] C. Guo, X. Wang, G. Shen, S. K. Bose, J. Xu, and M. Zukerman, “Exploring the benefits of resource disaggregation for service reliability in data centers,” *IEEE Transactions on Cloud Computing*, vol. 11, no. 2, pp. 1651–1666, Feb. 2023.

[30] Q. Cheng, M. Bahadori, M. Glick, S. Rumley, and K. Bergman, “Recent advances in optical technologies for data centers: a review,” *Optica*, vol. 5, no. 11, pp. 1354–1370, Nov 2018.

[31] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang, “LegoOS: A disseminated, distributed OS for hardware resource disaggregation,” in *Proceedings of 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX

Association, Oct. 2018, pp. 69–87. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/shan

[32] S. Yan, Z. Zhu, M. S. Glick, Z. Wu, and K. Bergman, "Accelerating distributed machine learning in disaggregated architectures with flexible optically interconnected computing resources," in *Proceedings of 2022 Optical Fiber Communications Conference and Exhibition (OFC)*, 2022, pp. 1–3.

[33] T. Kimura, "Approximations for multi-server queues: System interpolations," *Queueing Systems*, vol. 17, pp. 347–382, 1994.

[34] I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann, "VNE-AC: Virtual network embedding algorithm based on ant colony metaheuristic," in *Proceedings of 2011 IEEE International Conference on Communications (ICC)*, Jun. 2011, pp. 1–6.

[35] M. Dorigo and T. Stützle, *Ant Colony Optimization: Overview and Recent Advances*. Boston, MA: Springer US, 2010, pp. 227–263. [Online]. Available: https://doi.org/10.1007/978-1-4419-1665-5_8

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[37] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[38] V. Bahl, "Emergence of micro datacenter (cloudlets/edges) for mobile computing," online: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/11/Micro-Data-Centers-mDCs-for-Mobile-Computing-1.pdf. Accessed 18 November 2024.

[39] "Container micro data center," https://attom.tech/agilecub-container-micro-data-center/, (Accessed 21 November, 2024).

[40] K. Ishii, R. Matsumoto, T. Inoue, and S. Namiki, “Disaggregated optical-layer switching for optically composable disaggregated computing [invited],” *Journal of Optical Communications and Networking*, vol. 15, pp. A11–A25, 1 2023.

[41] Y. Shan, W. Lin, Z. Guo, and Y. Zhang, “Towards a fully disaggregated and programmable data center,” *APSys 2022 - Proceedings of the 13th ACM SIGOPS Asia-Pacific Workshop on Systems*, vol. 22, pp. 18–28, 8 2022. [Online]. Available: https://dl.acm.org/doi/10.1145/3546591.3547527

[42] Y. Mori and K.-I. Sato, “High-port-count optical circuit switches for intra-datacenter networks [invited tutorial],” *Journal of Optical Communications and Networking*, vol. 13, no. 8, pp. D43–D52, 2021.

[43] X. Guo, F. Yan, X. Xue, G. Exarchakos, and N. Calabretta, “Performance assessment of a novel rack-scale disaggregated data center with fast optical switch,” in *Proceedings of 2019 Optical Fiber Communications Conference and Exhibition (OFC)*, 2019, pp. 1–3.

[44] F. Yan, W. Miao, O. Raz, and N. Calabretta, “Opsquare: A flat dcn architecture based on flow-controlled optical packet switches,” *Journal of Optical Communications and Networking*, vol. 9, no. 4, pp. 291–303, 2017.

[45] N. Calabretta, X. Guo, G. Exarchakos, X. Xue, and B. Pan, “Optical switching for memory-disaggregated datacenters,” in *Proceedings of 2021 Optical Fiber Communications Conference and Exhibition (OFC)*, 2021, pp. 1–3.

[46] R. Luo, R. Matzner, A. Ottino, G. Zervas, and P. Bayvel, “Exploring the relationship among traffic, topology, and throughput: towards a traffic-optimal optical network topology design,” *J. Opt. Commun. Netw.*, vol. 15, no. 5, pp. B1–B10, May 2023. [Online]. Available: https://opg.optica.org/jocn/abstract.cfm?URI=jocn-15-5-B1

[47] J. Mata, I. de Miguel, R. J. Duran, N. Merayo, S. K. Singh, A. Jukan, and M. Chamania, “Artificial intelligence (ai) methods in optical networks: A comprehensive

survey," *Optical Switching and Networking*, vol. 28, pp. 43–57, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S157342771730231X

[48] R. Morais, C. Pavan, A. Pinto, and C. Requejo, "Genetic algorithm for the topological design of survivable optical transport networks," *Journal of Optical Communications and Networking*, vol. 3, no. 1, pp. 17–26, 2011.

[49] "S5860-20sq, 24-port ethernet l3 switch," https://www.fs.com/products/108710.html?attribute=65771&id=1676385, (Accessed 21 November, 2024).

[50] "Calient ' s optical circuit switch," https://www.calient.net/wp-content/uploads/2022/06/Datasheet_Calients-Optical-Circuit-Switches.pdf, (Accessed 21 November, 2024).

[51] A. Saljoghei, H. Yuan, V. Mishra, M. Enrico, N. Parsons, C. Kochis, P. D. Dobbelaere, D. Theodoropoulos, D. Pnevmatikatos, D. Syrivelis, A. Reale, T. Hayashi, T. Nakanishi, and G. Zervas, "Mcf-smf hybrid low-latency circuit-switched optical network for disaggregated data centers," *Journal of Lightwave Technology*, vol. 37, pp. 4017–4029, 8 2019.

[52] S. Ahmad, I. Shakeel, S. Mehfuz, and J. Ahmad, "Deep learning models for cloud, edge, fog, and iot computing paradigms: Survey, recent advances, and future directions," *Computer Science Review*, vol. 49, p. 100568, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013723000357

[53] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck,

J. Dean, S. Petrov, and N. Fiedel, "Palm: scaling language modeling with pathways," *J. Mach. Learn. Res.*, vol. 24, no. 1, mar 2024.

[54] M. Dehghani, J. Djolonga, B. Mustafa, P. Padlewski, J. Heek, J. Gilmer, A. P. Steiner, M. Caron, R. Geirhos, I. Alabdulmohsin, R. Jenatton, L. Beyer, M. Tschannen, A. Arnab, X. Wang, C. Riquelme Ruiz, M. Minderer, J. Puigcerver, U. Evci, M. Kumar, S. V. Steenkiste, G. F. Elsayed, A. Mahendran, F. Yu, A. Oliver, F. Huot, J. Bastings, M. Collier, A. A. Gritsenko, V. Birodkar, C. N. Vasconcelos, Y. Tay, T. Mensink, A. Kolesnikov, F. Pavetic, D. Tran, T. Kipf, M. Lucic, X. Zhai, D. Keysers, J. J. Harmsen, and N. Houlsby, "Scaling vision transformers to 22 billion parameters," in *Proceedings of the 40th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol. 202. PMLR, 23–29 Jul 2023, pp. 7480–7512. [Online]. Available: https://proceedings.mlr.press/v202/dehghani23a.html

[55] B. He, X. Zheng, Y. Chen, W. Li, Y. Zhou, X. Long, P. Zhang, X. Lu, L. Jiang, Q. Liu, D. Cai, and X. Zhang, "Dxpu: Large-scale disaggregated gpu pools in the datacenter," *ACM Trans. Archit. Code Optim.*, vol. 20, no. 4, dec 2023. [Online]. Available: https://doi.org/10.1145/3617995

[56] Z. Chen, W. Quan, M. Wen, J. Fang, J. Yu, C. Zhang, and L. Luo, "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, 2020.

[57] X. Zhang, "Mixtran: an efficient and fair scheduler for mixed deep learning workloads in heterogeneous gpu environments," *Cluster Computing*, vol. 27, pp. 1–10, 08 2023.

[58] Z. Ye, P. Sun, W. Gao, T. Zhang, X. Wang, S. Yan, and Y. Luo, "Astraea: A fair deep learning scheduler for multi-tenant gpu clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 2781–2793, 2022.

[59] X. Jin, Z. Bai, Z. Zhang, Y. Zhu, Y. Zhong, and X. Liu, "Distmind: Efficient resource disaggregation for deep learning workloads," *IEEE/ACM Transactions on Networking*, vol. 32, no. 3, pp. 2422–2437, 2024.

[60] Y. Huang, Y. Cheng, A. Bapna, O. Firat, M. X. Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen, *GPipe: efficient training of giant neural networks using pipeline parallelism.* Red Hook, NY, USA: Curran Associates Inc., 2019.

[61] Z. Li, L. Zheng, Y. Zhong, V. Liu, Y. Sheng, X. Jin, Y. Huang, Z. Chen, H. Zhang, J. E. Gonzalez, and I. Stoica, "AlpaServe: Statistical multiplexing with model parallelism for deep learning serving," in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23).* Boston, MA: USENIX Association, Jul. 2023, pp. 663–679. [Online]. Available: https://www.usenix.org/conference/osdi23/presentation/li-zhouhan

[62] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream: generalized pipeline parallelism for dnn training," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, ser. SOSP '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 1–15. [Online]. Available: https://doi.org/10.1145/3341301.3359646

[63] Y. Hu, C. Imes, X. Zhao, S. Kundu, P. A. Beerel, S. P. Crago, and J. P. Walters, "PipeEdge: Pipeline parallelism for large-scale model inference on heterogeneous edge devices," in *Proceedings of 2022 25th Euromicro Conference on Digital System Design (DSD)*, 2022, pp. 298–307.

[64] "Nvidia l4 tensor core gpu," https://www.nvidia.com/en-us/data-center/l4/, (Accessed 21 November, 2024).

[65] "Nvidia v100 tensor core gpu," https://www.nvidia.com/en-us/data-center/v100/, (Accessed 21 November, 2024).

[66] "Nvidia a30 tensor core gpu," https://www.nvidia.com/ja-jp/data-center/products/a30-gpu/, (Accessed 21 November, 2024).

[67] "Nvidia t4," https://www.nvidia.com/en-us/data-center/tesla-t4/, (Accessed 21 November, 2024).

[68] Y. Fang, B. Liao, X. Wang, J. Fang, J. Qi, R. Wu, J. Niu, and W. Liu, "You only look at one sequence: Rethinking transformer in vision through object detection," *CoRR*, vol. abs/2106.00666, 2021. [Online]. Available: https://arxiv.org/abs/2106.00666

[69] xiaoju ye. (2023) calflops: a flops and params calculate tool for neural networks in pytorch framework. [Online]. Available: https://github.com/MrYxJ/calculate-flops.pytorch

[70] B. Wu, C. Xu, X. Dai, A. Wan, P. Zhang, Z. Yan, M. Tomizuka, J. Gonzalez, K. Keutzer, and P. Vajda, "Visual transformers: Token-based image representation and processing for computer vision," 2020.

[71] G. Team, M. Riviere, S. Pathak, P. G. Sessa, C. Hardin, S. Bhupatiraju, L. Hussenot, T. Mesnard, B. Shahriari, A. Rame, J. Ferret, P. Liu, P. Tafti, A. Friesen, M. Casbon, S. Ramos, R. Kumar, C. L. Lan, S. Jerome, A. Tsitsulin, N. Vieillard, P. Stanczyk, S. Girgin, N. Momchev, M. Hoffman, S. Thakoor, J.-B. Grill, B. Neyshabur, O. Bachem, A. Walton, A. Severyn, A. Parrish, A. Ahmad, A. Hutchison, A. Abdagic, A. Carl, A. Shen, A. Brock, A. Coenen, A. Laforge, A. Paterson, B. Bastian, B. Piot, B. Wu, B. Royal, C. Chen, C. Kumar, C. Perry, C. Welty, C. A. Choquette-Choo, D. Sinopalnikov, D. Weinberger, D. Vijaykumar, D. Rogozińska, D. Herbison, E. Bandy, E. Wang, E. Noland, E. Moreira, E. Senter, E. Eltyshev, F. Visin, G. Rasskin, G. Wei, G. Cameron, G. Martins, H. Hashemi, H. Klimczak-Plucińska, H. Batra, H. Dhand, I. Nardini, J. Mein, J. Zhou, J. Svensson, J. Stanway, J. Chan, J. P. Zhou, J. Carrasqueira, J. Iljazi, J. Becker, J. Fernandez, J. van Amersfoort, J. Gordon, J. Lipschultz, J. Newlan, J. yeong Ji, K. Mohamed, K. Badola, K. Black, K. Millican, K. McDonell, K. Nguyen, K. Sodhia, K. Greene, L. L. Sjoesund, L. Usui, L. Sifre, L. Heuermann, L. Lago, L. McNealus, L. B. Soares, L. Kilpatrick, L. Dixon, L. Martins, M. Reid, M. Singh, M. Iverson, M. Gorner, M. Velloso, M. Wirth, M. Davidow, M. Miller, M. Rahtz, M. Watson, M. Risdal, M. Kazemi, M. Moynihan, M. Zhang, M. Kahng, M. Park, M. Rahman, M. Khatwani, N. Dao, N. Bardoliwalla, N. Devanathan, N. Dumai,

N. Chauhan, O. Wahltinez, P. Botarda, P. Barnes, P. Barham, P. Michel, P. Jin, P. Georgiev, P. Culliton, P. Kuppala, R. Comanescu, R. Merhej, R. Jana, R. A. Rokni, R. Agarwal, R. Mullins, S. Saadat, S. M. Carthy, S. Cogan, S. Perrin, S. M. R. Arnold, S. Krause, S. Dai, S. Garg, S. Sheth, S. Ronstrom, S. Chan, T. Jordan, T. Yu, T. Eccles, T. Hennigan, T. Kocisky, T. Doshi, V. Jain, V. Yadav, V. Meshram, V. Dharmadhikari, W. Barkley, W. Wei, W. Ye, W. Han, W. Kwon, X. Xu, Z. Shen, Z. Gong, Z. Wei, V. Cotruta, P. Kirk, A. Rao, M. Giang, L. Peran, T. Warkentin, E. Collins, J. Barral, Z. Ghahramani, R. Hadsell, D. Sculley, J. Banks, A. Dragan, S. Petrov, O. Vinyals, J. Dean, D. Hassabis, K. Kavukcuoglu, C. Farabet, E. Buchatskaya, S. Borgeaud, N. Fiedel, A. Joulin, K. Kenealy, R. Dadashi, and A. Andreev, “Gemma 2: Improving open language models at a practical size,” 2024. [Online]. Available: https://arxiv.org/abs/2408.00118