IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 30 April 2024, accepted 6 May 2024, date of publication 13 May 2024, date of current version 20 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3399930

RESEARCH ARTICLE

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

AKISHIGE IKOMA[1], YUICHI OHSITA[2], (Member, IEEE), AND MASAYUKI MURATA[1], (Member, IEEE)
[1]Graduate School of Information Science and Technology, Osaka University, Suita 565-0871, Japan
[2]Cybermedia Center, Osaka University, Toyonaka 560-0043, Japan

Corresponding author: Akishige Ikoma (a-ikoma@ist.osaka-u.ac.jp)

This work was supported in part by the National Institute of Information and Communications Technology (NICT) under Grant JPJ012368C00101.

ABSTRACT A disaggregated data center (DDC) can efficiently use resources such as CPU and memory. In a DDC, because each resource is independent and connected by a network, communication between resources is required for task execution. Communication delays can be an overhead for task execution, causing performance degradation. Because communication delays depend on the correspondence between resources on the network and the paths over which they communicate, an efficient resource allocation method is required to determine this relationship. Herein, we propose a resource allocation method called *RA-CNP* to execute many tasks simultaneously while satisfying performance requirements. This method models the impact of the network on the performance of tasks for the provided service. Furthermore, this method defines a resource allocation problem to avoid the allocation of resources that will be requested in the future. We evaluated the effectiveness of our method by simulating various DDC networks, assuming a DDC at the edge. The results demonstrated that *RA-CNP* could execute more tasks than conventional methods could, without violating performance requirements, based only on current network information in both networks configured by circuit and packet switches. *RA-CNP* could allocate resources in less than 10 s, even in a relatively large network configured with 64 switches; this capability demonstrates its practicality.

INDEX TERMS Disaggregated data center, optical network, resource allocation, resource disaggregation.

I. 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 small 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 [2]. 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 [2]. Nevertheless, a data center on the edge has fewer resources than those with larger cloud data centers. Therefore, optimal resource utilization for each task is key [3].

Flexible resource allocation via infrastructure virtualization and optimization of resource utilization have been considered potential ways to achieve this goal [4]. However, in traditional architectures where resources such as CPU and memory are aggregated on a server, per-resource flexible management is limited [5]. 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

The associate editor coordinating the review of this manuscript and approving it for publication was Maged Abdullah Esmail.

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 [7]. Owing to these advantages, resource disaggregation has been considered in various areas, such as serverless computing, big data processing, and database processing [8], [9], [10]. Therefore, we focused on a data center applying resource disaggregation (hereafter referred to as a disaggregated data center (DDC)), as shown in Fig. 1 and aim to configure a DDC that can execute many service tasks simultaneously.

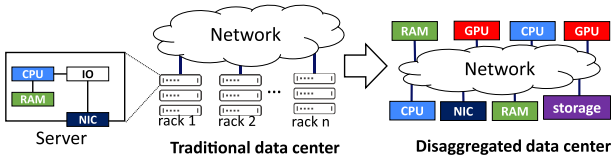


FIGURE 1. Traditional data center and disaggregated data center.

When a task for a service is executed in a 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 [7]. 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 [11], [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 [13], 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 [14]. The resource allocation method, which determines the CPUs and memories that execute tasks and the network resources that constitute the communication path, is also important for a DDC. In particular, network resource allocation has a significant impact on the performance of tasks [15]. 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 network resource 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 network resources are allocated regardless of the performance requirements of tasks to minimize performance degradation. Because available network resources 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 network resources 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 [14], [15], [16], [17], but these methods do not consider the impact of the network on the performance of each task and future tasks. Instead, they allocate network resources 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 DDC.

We have previously proposed a resource allocation method that considers the impact of the network on performance (hereafter called *RA-CNP*) [18]. In that study, we modeled 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 existed, we avoided allocating high-performance and low-latency network resources that may be requested in the future to execute more tasks. We defined the resource allocation cost in terms of resource importance; moreover, we formulated 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 DDC preserves those that can be used in future task requests, thereby executing more tasks.

This paper is an extension of a previous study [18], in which only packet switch networks were assumed, even though several DDCs composed of optical circuit switches have been proposed. Therefore, we extended the model

representing the network impact on performance to a general form to support both networks configured by circuit and packet switches. Then, we evaluated the effectiveness of *RA-CNP* by comparing it with other methods in networks configured by circuit and packet switches. By contrast, the previous study has only evaluated small networks and has not shown effectiveness in a DDC of the scale that we have envisioned. Therefore, we evaluated the effectiveness of *RA-CNP* in a larger network. Moreover, the effectiveness of *RA-CNP* has only been demonstrated in a solution derived via ant colony optimization, and the optimal solution to the defined resource allocation problem has not been verified. In this study, 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 study are as follows:

- We modeled the impact of computational and network resources 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 paper is organized as follows: Section II discusses related work. Section III provides an overview of the resource allocation method. Section IV discusses the simulations used to evaluate the effectiveness of *RA-CNP* and the computational time. Finally, Section V concludes this paper.

II. 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 [19]. 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.

A. 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 [20]. 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.

B. RESOURCE CONNECTION

In a DDC, performance degradation due to communication delays between resources is significant, and nanosecond resource communication is required [7]. 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 [11], [12], [21], [22]. Mishra et al. proposed a network architecture for a DDC using optical circuit switches [11]. 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 [22]. 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 [12], [21]. Terzenidis et al. proposed a network for a DDC configured by optical packet switches [21]. 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 resource 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 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 resources are finite, regardless of network architecture. Individual execution resources can only handle a limited number of tasks. Then, available network resources are not also infinite due to bandwidth constraints or dedicated network resources (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. 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.

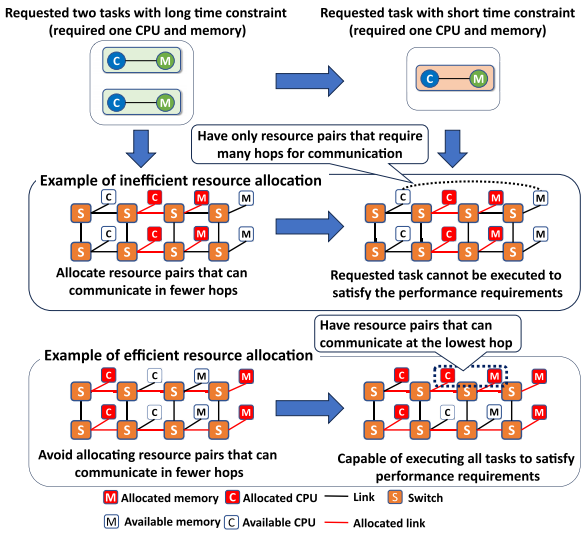


FIGURE 2. Example of inefficient resource allocation and efficient resource allocation.

C. 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 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 [16].

Several resource allocation methods have been proposed for a DDC [14], [15], [16], [17]. The characteristics of each method, in terms of objective and approach, are shown in Table 1. We analyze whether existing studies are sufficient to execute many tasks while satisfying performance requirements based on Table 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 [15]. 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 [14]. The authors demonstrated that this method can improve resource utilization in a DDC configured with optical circuit switches. These methods [14], [15] 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 [16]. 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 [17]. 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

TABLE 1. Objective and approach of each resource allocation method for a DDC.

Author	Objective	Approach
Papaioannou et al. [15]	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. [14]	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. [16]	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. [17]	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. (This study)	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

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. Any resources are sufficient if the performance requirements of the task are satisfied. By avoiding unnecessarily allocating resources and network resources 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 study, we propose a resource allocation method for a DDC 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 resources, 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.

III. RESOURCE ALLOCATION CONSIDERING THE IMPACT OF THE NETWORK ON PERFORMANCE

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.

A. OVERVIEW OF A DISAGGREGATED DATA CENTER

We show the assumed DDC in this section.

1) THE COMPONENTS OF A DISAGGREGATED DATA CENTER

We assume a DDC in which the memory and computational resources (CPUs and GPUs) are connected by a network, as shown in Fig. 3. This DDC 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 DDC are as follows:

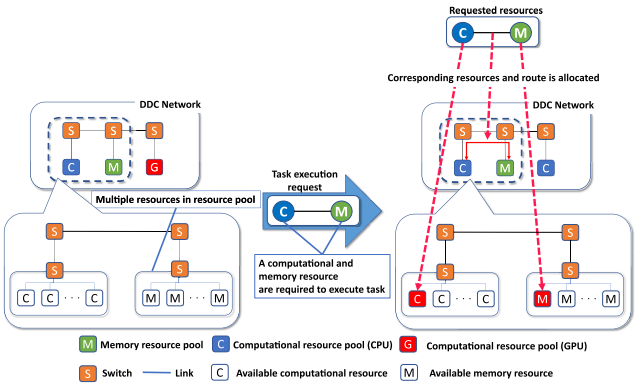


FIGURE 3. Overview of a disaggregated data center.

a: MEMORY RESOURCES

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

b: 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 DDC, 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 study, 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.

c: 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 DDC. 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.

2) EXECUTE TASK IN A DISAGGREGATED DATA CENTER

We assume that users request the execution of tasks at any time for services provided by a DDC. When a DDC 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. 3. Thereafter, it determines the communication path between the allocated resources. We treat this process as resource allocation.

Multiple types of processing with different resource requirements may be required to complete a task. In this study, 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.

B. MODELING A DDC AND A RESOURCE ALLOCATION REQUEST

The notations used for modeling a DDC and resource allocation request are listed in Table 2.

1) MODELING A DDC NETWORK

A DDC 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 $B > 0$.

TABLE 2. Notation of the DDC and resource request model.

Symbols	Definition
	DDC 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 DDC 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 $p \in N^p$

2) MODELING A 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. 4.

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.

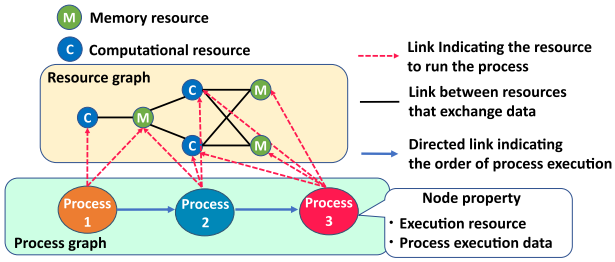


FIGURE 4. Example of resource and process graphs.

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 study, 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.

C. RELATIONSHIP BETWEEN RESOURCE ALLOCATION AND TASK EXECUTION TIME

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

1) MAPPING THE RESOURCES

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

2) MAPPING THE NETWORK RESOURCES

$\delta_{x,y}^E$ denotes the mapping between the resource graph links and paths in the DDC. $\delta_{x,y}^E = 1$ 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 DDC and $\delta_{x,y}^E = 0$ otherwise.

3) MODELING THE EXECUTION TIME OF A TASK

The execution time of task $t \in S$ is the sum of the times required to complete all processes in task t. In addition, the execution time for each process is 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 study, 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_t^e for task $t \in S$ is obtained as follows:

$$T_t^e = \max_{y \in P_t} \sum_{p \in y} \max_{c' \in c_p^v, m' \in m_p^v} \left(T_{c',p}^c + T_{c',m',p}^m \right). \quad (1)$$

where $T_{c',p}^c$ denotes the processing time of process $p \in N_t^p$ in the computational resource mapped to $c' \in N^v$, and $T_{c',m',p}^m$ 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 $c' \in N^v$.

a: PROCESSING TIME IN A COMPUTATIONAL RESOURCE

The processing time $T_{c',p}^c$ 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_p^c 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:

$$T_{c',p}^c = \sum\nolimits_{j \in N^s} \left(\delta_{c',j}^N \frac{\sigma_p^c}{F_j} \right). \quad (2)$$

$T_{c',p}^c$ 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 $j \in N^s$.

b: 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_c^I of the resource in computational resource pool $c \in N^c$, processing time T_m^R of the resource in memory resource pool $m \in N^m$, and communication delay $T_{c',m',p}^d$ 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 $p \in N^p$.

$$T_{c',m',p}^m = T_{c',m',p}^d + \sum_{j \in N^s} (\delta_{c',j}^N \cdot T_j^I + \delta_{m',j}^N \cdot T_j^R). \quad (3)$$

where c_j^s and m_j^s 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:

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

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. This function is based on the M/D/C queuing model. In the M/D/C queuing model, arrivals occur according to a Poisson process, and the system has C servers that can process an arrival at a fixed time D. In other words, 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 [23]. $T^R_n(\lambda, J, D)$ 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

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

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

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

D. 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 study, 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 III-C3.

1) RESOURCE ALLOCATION COSTS

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

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:

$$W^c_c = C^s_c \cdot K_c. \tag{4}$$

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:

$$W_m^m = M_m^s. \tag{5}$$

Network resources 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 resources which are possibly the shortest paths between critical resource pairs.

We define the possibility to be a network resource on the shortest path as the ratio of the number of shortest paths between resources to the number of shortest paths through that network resource. 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 resource 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 resource 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:

$$W_e^e = \begin{cases} \sum_{c \in N^c, m \in N^m} u_{c,m}(e) \cdot v_{c,m} & e \notin E^{alc} \\ \epsilon & e \in E^{alc} \end{cases}, \tag{6}$$

where E^{alc} denotes the set of network resources 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.

2) DEFINING THE 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 DDC and resource allocation request is given; this outputs the mapping δ^N, δ^E between the requested resource and allocated resource defined in Sections III-C1 and III-C2.

a: RESOURCE MAPPING CONSTRAINTS

A request graph node is mapped as a node, and a request graph link is mapped as a path in the DDC network as follows:

$$\forall i \in N^v, \sum_{j \in N^s} \delta_{i,j}^N = 1. \tag{7}$$

$$\forall x \in E^v, \forall k, s \in N^s, \sum_{y \in R_{k,s}} \delta_{x,y}^E = \delta_{n_x^{vs},k}^N \cdot \delta_{n_x^{vd},s}^N. \tag{8}$$

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:

$$\forall c \in N^c, C_c^s - \sum_{c' \in C^v} \delta_{c',c}^N \geq 0. \tag{9}$$

$$\forall m \in N^m, M_m^s - \sum_{m' \in M^v} \delta_{m',m}^N \geq 0. \tag{10}$$

b: TIME CONSTRAINTS

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

$$\forall t \in S, T_t^e \leq T_t^a \tag{11}$$

c: OBJECTIVE

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

$$\begin{aligned} minimize \ & \sum_{c \in N^c} \sum_{c' \in C^v} \delta_{c',c}^N(W_c^c) \\ & + \sum_{m \in N^m} \sum_{m' \in M^v} \delta_{m',m}^N(W_m^m) \\ & + \sum_{i,j \in N^s} \sum_{y \in R_{i,j}} \left[1_{\sum_{x \in E^v} \delta_{x,y}^E > 0} \left(\sum_{e \in y} W_e^e \right) \right], \end{aligned} \tag{12}$$

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 DDC 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 [24]. In this study, we solve this problem using ant colony optimization (ACO). ACO can respond flexibly to changes in the environment [25]. 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.

E. RESOURCE ALLOCATION BASED ON ANT COLONY OPTIMIZATION

We solve the resource allocation problem defined in Section III-D2 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 [24]. However, VNE-AC targets traditional architectures and does not target DDC. In traditional architectures, there is no need to consider performance degradation due to communication delays between resources, unlike DDC. 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 resource allocation to solve our resource allocation problem. Note that VNE-AC allocates network resources 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 resource 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 resource 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 3.

TABLE 3. 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 resource 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

1) 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:

$$p_c^c = \frac{(\tau_c)^\alpha \left(\frac{1}{(W_c^c)^\beta}\right)}{\sum_{x \in C^{cd}} \left[(\tau_x)^\alpha \frac{1}{(W_x^c)^\beta}\right]},$$

$$p_m^m = \frac{(\tau_m)^\alpha \left(\frac{1}{(W_m^m)^\beta}\right)}{\sum_{x \in M^{cd}} \left[(\tau_x)^\alpha \frac{1}{(W_x^m)^\beta}\right]}$$

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.

2) NETWORK RESOURCE 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 DDC. 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.

$$p_{e,n} = \frac{(\tau_e)^\alpha \frac{1}{(W_e^e)^\beta}}{\sum_{x \in E_n^{cd}} \left[(\tau_x)^\alpha \frac{1}{(W_x^e)^\beta}\right]}$$

Here, $\alpha > 0$ and $\beta > 0$ denote the relative importance of the pheromone and the cost, respectively. E_n^{cd} 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.

3) 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 resource allocation. This value is derived from the equation presented in Section III-C3. 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 DDC must be able to complete processing within an acceptable time.

4) 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 resource e as follows:

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

IV. EVALUATION

We evaluated *RA-CNP* by simulating the DDC networks. First, we discuss the effectiveness of the resource allocation problem defined in Section III-D2 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.

A. ENVIRONMENT

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

1) NETWORK

We assume a data center that is located near the users. In such data centers, on-site services such as industrial automation, environmental monitoring, and object recognition are provided [1], [26]. One of the early instances of an edge data center was one composed of approximately 10 servers to enable deployment even in a limited space. However, because of the increasing demand for edge services, the concept of edge data centers with multiple racks has been proposed [27]. We believe that data centers with more resources, including hundreds of servers, will be required at the edge in the future. In this study, we considered DDCs with 20–552 computational resources. This scope is similar to the number of servers in the data center we assumed.

Fig. 5 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. A more suitable network topology may exist; this will be a topic for future studies.

Optical packet switches and optical circuit switches have been proposed for resource disaggregation networks [14], [21]. In this study, we evaluated two cases: one configured with optical packet switches (packet switch network) and the other configured with optical circuit switches (circuit switch network).

We set the parameters of the DDC network as listed in Table 4. 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 [14]. In addition, we referred to the optical circuit switch proposed in [14] and the optical packet switch proposed in [21]. 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.

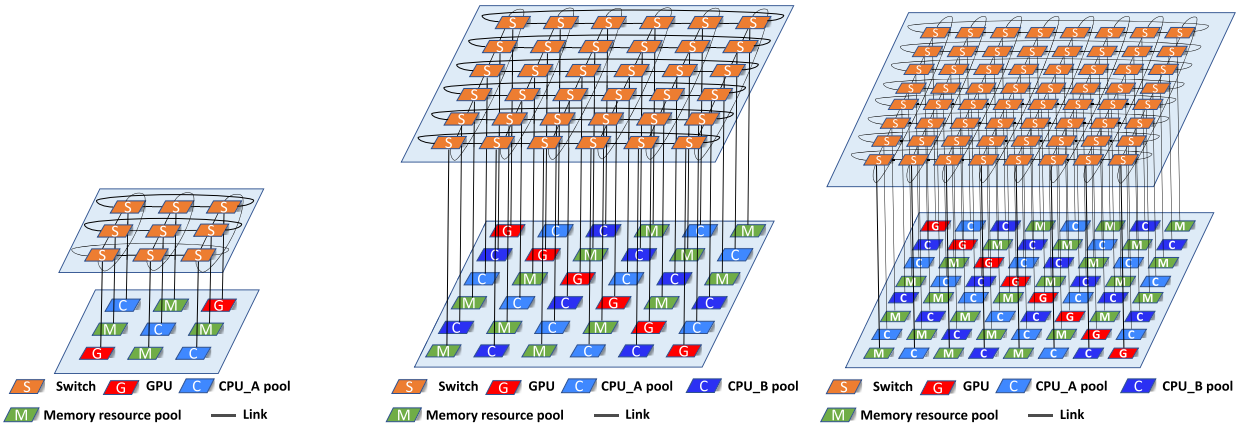
TABLE 4. Parameter settings for the DDC network.

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

2) RESOURCE REQUEST

Resources are requested when a task execution request arrives at the DDC. In this study, we assumed that tasks for two services are executed. One is image classification for face recognition using ResNet [28] (service 1) and the other is real-time object identification for automated driving using YOLO [29] (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 [2], making them prime candidates for being offered as services by a data center located at the edge [26].

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



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

FIGURE 5. Networks used in evaluation.

values obtained by running the task using an *Intel(R) Xeon(R) CPU E5-2687W*. The parameters for each task are shown in Table 5. We generate four types of resource requests for each task, with different acceptable times and required resources.

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

All resource requests have the same structure, shown in Fig. 4. We demonstrate the effectiveness of *RA-CNP* in various cases by changing the required resources and acceptable time. Table 6 shows the pattern of the number of resources required for evaluation. Acceptable time values are shown in Table 7.

TABLE 5. Parameter settings for a task for each service.

	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

3) 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 8. The two compared methods are described below.

a: 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.

b: 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. [14] and Amaral et al. [16]. 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:

$$W_{e,n}^e = \frac{\frac{\lambda_{e,n}^s}{N_e^{core}}}{\lambda^{max}} + \frac{D_{i,j}}{D^{max}}$$

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 DDC.

B. OPTIMAL SOLUTION OF RA-CNP IN A SMALL NETWORK

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 DDC, 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.

TABLE 6. Number of resources required for each request.

	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 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 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 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 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 7. Acceptable time for each request.

	Request 1	Request 2	Request 3	Request 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

TABLE 8. Parameter settings for ACO.

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

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

1) DDC NETWORK

We used a small DDC network, as shown in Fig. 5a, 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.

2) RESOURCE REQUEST

In this evaluation, the acceptable time for each request corresponded to Pattern 1 in Table 7. The number of computational and memory resources required for each request per process was set as listed in Pattern A of Table 6. We generated resource requests up to the number of tasks that could be executed in the DDC. The generated sequences of resource requests were the two patterns listed in Table 9, to be evaluated in various environments. The order in which the requests arrived in each case was uniformly random.

TABLE 9. Breakdown of generated requests in each pattern.

	Request 1	Request 2	Request 3	Request 4
Generated pattern 1	4	2	2	2
Generated pattern 2	2	2	4	2

3) METRICS

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

a: 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 9.

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 DDC, 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.

b: TOTAL ALLOCATION COST

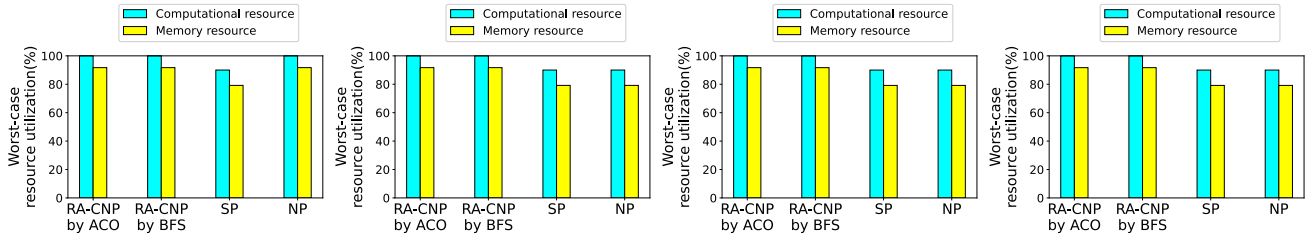
To compare the solutions obtained using *ACO* and *BFS*, we measured the total allocation cost. If the total cost of the solution obtained using *ACO* was the same as that obtained using *BFS*, we concluded that *ACO* derived the optimal solution for *RA-CNP*.

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.

4) RESULT

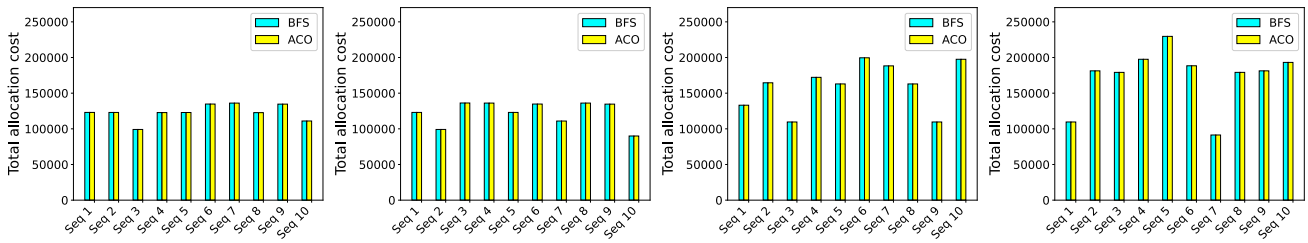
Fig. 6 shows the worst-case resource utilization according to 10 measurements in two cases: the packet and circuit switch networks. In this evaluation, when all generated requests were allocated, the resource utilization of computational resources was 100% and it was impossible to allocate more resources. Fig. 7 shows a comparison between the allocation costs of the solutions obtained using *ACO* and *BFS*.

As shown in Fig. 6, *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



(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 6. Worst-case resource utilization and blocked requests in each pattern.



(a) Generated pattern (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 7. Total allocated cost per request sequence in *BFS* and *RA-CNP*.

TABLE 10. 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 6)	Performance requirements (Table 7)	Generated requests	Required resources (Table 6)	Performance requirements (Table 7)
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

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. 7 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.

C. 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.

1) DDC NETWORK

We used two DDC networks of different scales: 6 × 6 and 8 × 8 2D torus networks, as shown in Fig. 5b and Fig. 5c, respectively. Each CPU pool had 16 computational resources, and each memory pool had 24 memory resources.

In this study, 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.

2) RESOURCE REQUEST

We continuously generated the requests listed in Table 5 for 300 min. The lifetime of each task was 90 min. In this evaluation, we set the probability that requests 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 10. Because the 8 × 8 2D torus network holds more resources, we allocated more resources to compare the methods.

3) 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.

4) 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. 8 illustrates the blocked requests for each allocation method.

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. 8a). 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. 8a and Fig. 8c, Request 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 DDC scale, *RA-CNP* was effective.

A comparison of Fig. 8a and Fig. 8b 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 resources 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. 8c and Fig. 8d 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.

D. COMPUTATIONAL TIME OF RA-CNP

For practical resource allocation, *RA-CNP* must allocate resources within a practical computational time at various DDC scales. We have presented an example solution based on ACO in Section III-E. 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 III-E, as the computational time depends on computational complexity.

1) COMPUTATIONAL COMPLEXITY OF RESOURCE ALLOCATION BASED ON ACO

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 and 3.

a: 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.

b: NETWORK RESOURCE SEARCH PHASE

In this study, 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 III-E, the network resource search is repeated a maximum of H times for each network resource in the resource graph. However, H is a constant parameter for ACO. Therefore, the computational complexity per agent in this phase is $O(|C^v||M^v|)$.

c: 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 $O(A^{deg})$.

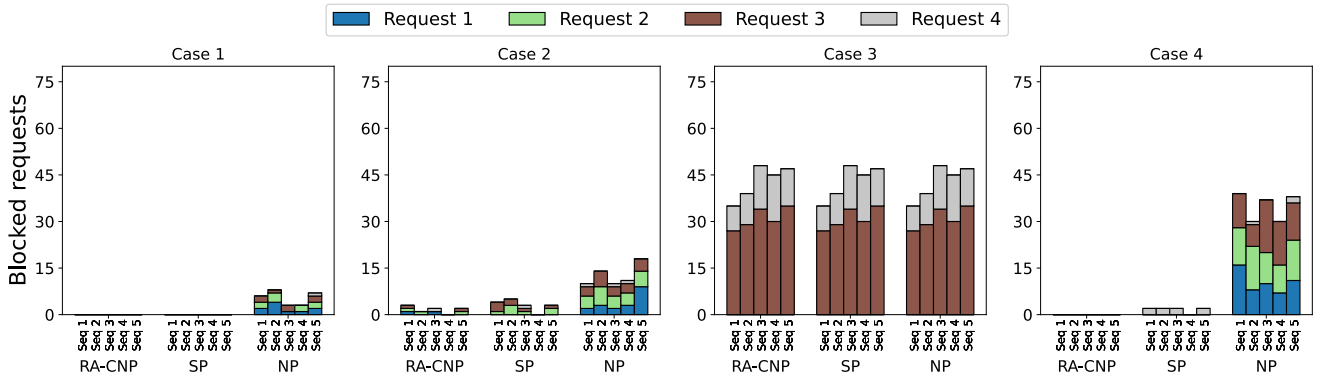
d: PHEROMONE UPDATE PHASE

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

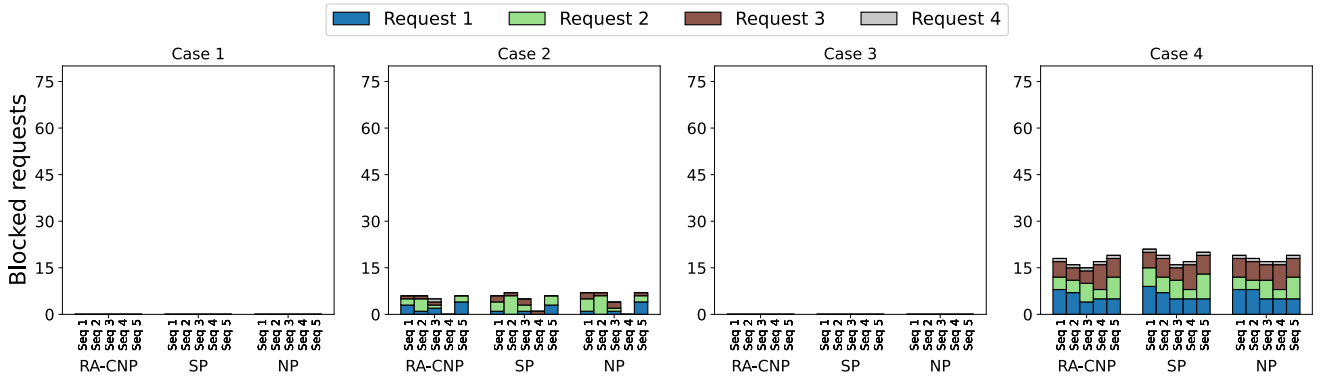
TABLE 11. Computational complexity in each phase.

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

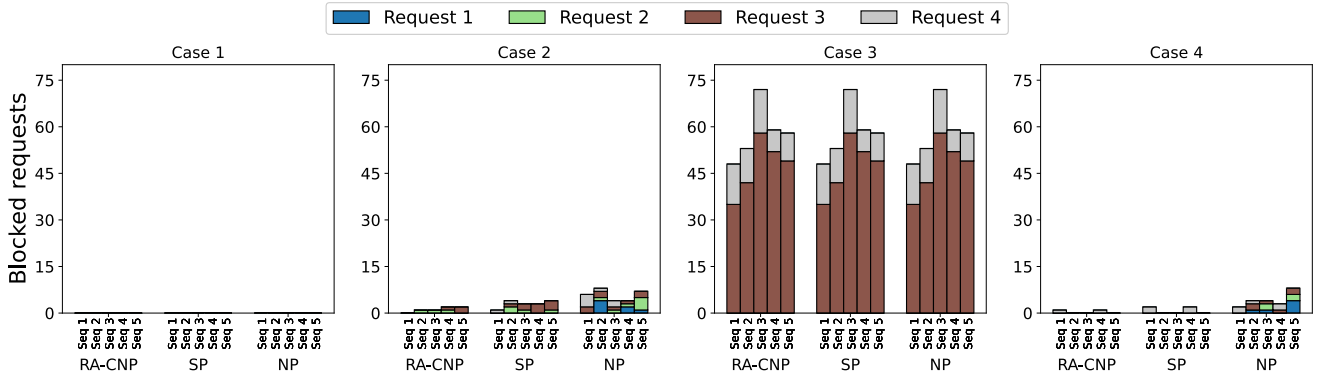
We summarize the computational complexity of each step in Table 11. This series of phases is iterated t^{itr} times. However, t^{itr} is a parameter for ACO. This value is constant and does not affect the computational time. R^s and E^s depend on the network scale, whereas C^v and M^v depend on the number of requested resources. A^{deg} increases as more tasks are allocated. We investigated whether computational time



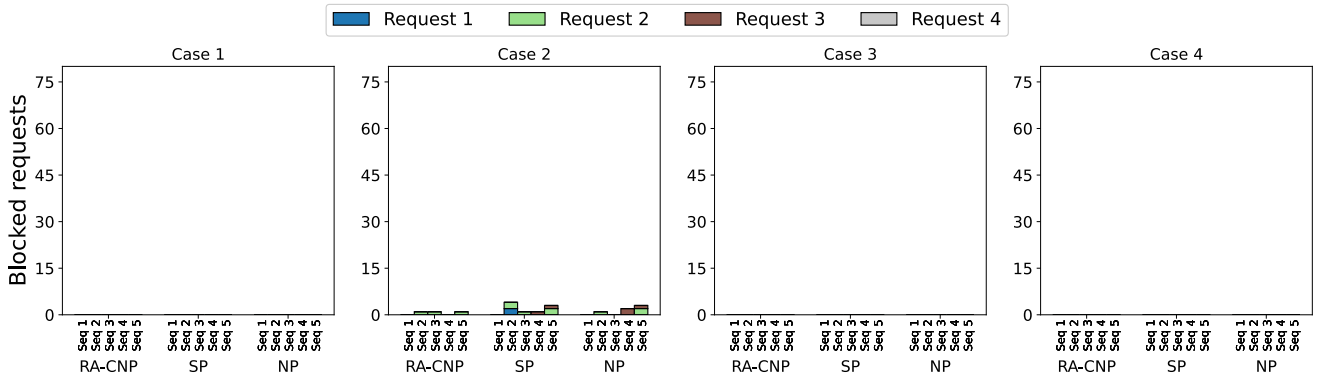
(a) Blocked requests for each case in the packet switch network (6 × 6 2D torus network)



(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 8. **Blocked requests.**

can be practical for the number of required resources, the number of accommodated requests, and the DDC network scale.

2) IMPACT OF THE NUMBER OF REQUIRED RESOURCES ON ALLOCATION TIME

a: ENVIRONMENT

We used the DDC network shown in Fig. 5b. The number of resources for requests 1, 2, and 3 shown in Table 6 was changed to five patterns, as shown in Table 12. In each measurement, 100 requests were randomly generated within 300 min.

TABLE 12. Required resources for requests 1, 2, and 3.

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

b: RESULT

Fig. 9 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. 9.

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 resource 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 resources 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.

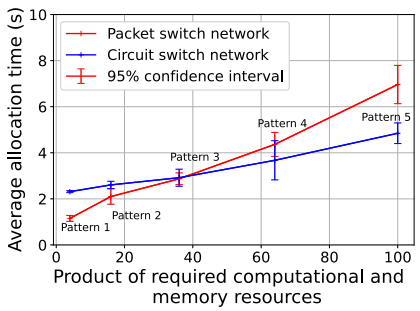


FIGURE 9. Required resources and allocation time.

3) IMPACT OF THE NUMBER OF ACCOMMODATED REQUESTS ON ALLOCATION TIME

a: ENVIRONMENT

We used the DDC network, as shown in Fig. 5b. In this evaluation, the number of resources required for each request was set according to Pattern 2 in Table 6. We changed the number of generated requests to investigate the effect of the number of accommodated requests. Table 13 lists the number of generated requests. Resource requests were generated randomly.

TABLE 13. Parameter setting for each type of generated requests.

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

b: RESULT

Fig. 10 shows the relationship between the number of accommodated requests and allocation time. The 95% confidence interval is included in Fig. 10.

Fig. 10 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.

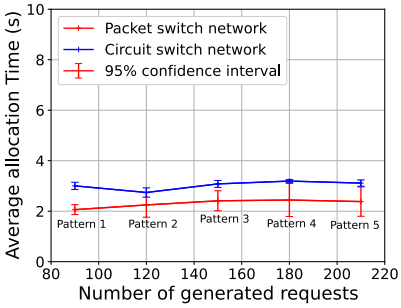


FIGURE 10. Number of requests and allocation time.

4) IMPACT OF DDC NETWORK SCALE ON ALLOCATION TIME

a: ENVIRONMENT

We used 5×5, 6×6, 7×7, and 8×8 2D torus networks. The parameters for each structure are listed in Table 14. In this evaluation, the number of resources required for each request was set according to Pattern B in Table 6. In each measurement, 100 requests were generated within 300 min.

b: RESULT

Fig. 11 shows the impact of the scale of the DDC network on the allocation time. The 95% confidence interval is included in Fig. 11.

TABLE 14. Parameter settings for each 2D torus 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

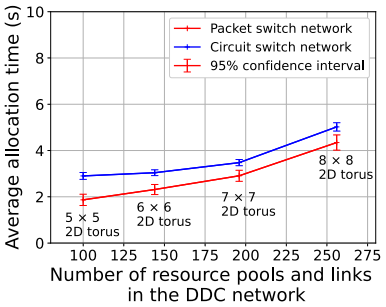


FIGURE 11. DDC network scale and allocation time.

Fig. 11 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 11. Because the scales of the 5 × 5 and 6 × 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 × 8 2D torus, the computational time is less than 6 s.

E. DISCUSSION ON LIMITATIONS AND FUTURE RESEARCH DIRECTIONS

We demonstrated *RA-CNP* can execute more tasks simultaneously than other resource allocation methods in both networks configured by circuit and packet switches. However, this study has some limitations.

First, *RA-CNP* assumes that service execution resources are allocated before task execution and that the service continues to use the allocated resources. *RA-CNP* provides enough computational time in such a situation. However, in a situation where execution resources for services are managed dynamically in real-time, as in the case of microservices, *RA-CNP* is not practical because it is excessively time-consuming. In such a situation, preparing candidate execution resources in advance is effective for real-time resource management. Execution resources can be allocated in real-time because the search for execution resources is no longer required at the time of a resource allocation request. To achieve this approach, a method for selecting candidate execution resources based on the prediction of future requested services is required. Proposals for such a method are future work.

Next, *RA-CNP* assumes that the execution information of each task, such as the resources required for task execution and the packet rate that occurs between resources, is known in advance. However, in cases such as task offloading from users, it is difficult to know which tasks will be offloaded in advance. In such a case, *RA-CNP* cannot achieve the objective because it cannot calculate the impact of the network on performance. Estimating execution information is required. The amount of transferred data and frequency of communication depend on the size of processed data and memory size. The clock count depends on the executable program and the performance of the device. Estimating execution information based on these factors is also a future work.

In future studies, we plan to introduce an optimal network architecture for a DDC. Currently, there have been several studies of network architecture in a DDC [11], [12], [21], [22]. However, there has been no comparative evaluation of whether the network is optimal through resource allocation. *RA-CNP* is a resource allocation method available for any network. Therefore, *RA-CNP* can allow us to evaluate the suitability of various DDC networks to execute many tasks. We plan to propose a network configuration method considering running tasks and network topology and evaluate the network by *RA-CNP*.

V. CONCLUSION

DDCs improve resource utilization and scaling flexibility. However, network resources 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 DDC 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.

We discussed the limitations of this study and suggested the need for real-time resource management methods and execution information estimation methods. In future studies, we plan to introduce a DDC network architecture considering network topology and running tasks by using *RA-CNP*.

REFERENCES

[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," *Comput. Netw.*, vol. 130, pp. 94–120, Jan. 2018.

[2] J. Pan and J. McElhannon, "Future edge cloud and edge computing for Internet of Things applications," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 439–449, Feb. 2018.

[3] 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 Comput. Commun. Rev.*, vol. 52, no. 1, pp. 28–33, Mar. 2022.
[4] 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. 27591–27610, 2022.
[5] M. Ewais and P. Chow, "Disaggregated memory in the datacenter: A survey," *IEEE Access*, vol. 11, pp. 20688–20712, 2023.
[6] S. Han, N. Egi, A. Panda, S. Ratnasamy, G. Shi, and S. Shenker, "Network support for resource disaggregation in next-generation datacenters," in *Proc. 12th ACM Workshop Hot Topics Netw.*, Nov. 2013, pp. 1–7.
[7] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker, "Network requirements for resource disaggregation," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*. Savannah, GA, USA: USENIX Association, 2016, pp. 249–264.
[8] X. Lu and A. Kashyap, "Towards offloadable and migratable microservices on disaggregated architectures: Vision, challenges, and research roadmap," in *Proc. 2nd Workshop Resource Disaggregation Serverless (WORDS)*. ACM, Apr. 2021, pp. 1–7. [Online]. Available: https://wuklab.github.io/words/words21-lu.pdf
[9] G. Vargas-Solar, M. S. Hassan, and A. Akoglu, "JITA4DS: Disaggregated execution of data science pipelines between the edge and the data centre," *J. Web Eng.*, vol. 21, no. 1, pp. 1–26, Nov. 2021.
[10] Q. Zhang, Y. Cai, S. G. Angel, V. Liu, A. Chen, and B. T. Loo, "Rethinking data management systems for disaggregated data centers," in *Proc. Conf. Innov. Data Syst. Res. (CIDR)*, Jan. 2020, pp. 1–8.
[11] V. Mishra, J. L. Benjamin, and G. Zervas, "MONet: Heterogeneous memory over optical network for large-scale data centre resource disaggregation," *J. Opt. Commun. Netw.*, vol. 13, no. 5, pp. 126–139, 2021, doi: 10.1364/JOCN.419145.
[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]," *J. Opt. Commun. Netw.*, vol. 14, no. 1, pp. A69–A80, Jan. 2022.
[13] A. Saljoghei, V. Mishra, M. Bielski, I. Syrigos, K. Katrinis, D. Syrivelis, A. Reale, D. N. Pnevmatikatos, D. Theodoropoulos, M. Enrico, N. Parsons, and G. Zervas, "DRedDbox: Demonstrating disaggregated memory in an optical data centre," in *Proc. Opt. Fiber Commun. Conf.*, Jan. 2018, Paper no. W1C.1.
[14] 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]," *J. Opt. Commun. Netw.*, vol. 10, no. 2, pp. A270–A285, Feb. 2018.
[15] A. D. Papaioannou, R. Nejabati, and D. Simeonidou, "The benefits of a disaggregated data centre: A resource allocation approach," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–7.
[16] 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 data-centers," *J. Cloud Comput.*, vol. 10, no. 1, pp. 1–20, Mar. 2021.
[17] C. Guo, X. Wang, G. Shen, S. Bose, J. Xu, and M. Zukerman, "Exploring the benefits of resource disaggregation for service reliability in data centers," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1651–1666, Feb. 2023, doi: 10.1109/TCC.2022.3151923.
[18] A. Ikoma, Y. Ohsita, and M. Murata, "Disaggregated micro data center: Resource allocation considering impact of network on performance," in *Proc. IEEE 20th Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2023, pp. 360–365.
[19] 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.
[20] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang, "LegoOS: A disseminated, distributed OS for hardware resource disaggregation," in *Proc. 13th USENIX Symp. Operating Syst. Design Implement. (OSDI)*. Carlsbad, CA, USA: USENIX Association, Oct. 2018, pp. 69–87. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/shan
[21] 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," *J. Opt. Commun. Netw.*, vol. 10, no. 7, pp. 102–116, Jul. 2018.
[22] 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 *Proc. Opt. Fiber Commun. Conf. Exhib. (OFC)*, Mar. 2022, pp. 1–3.
[23] T. Kimura, "Approximations for multi-server queues: System interpolations," *Queueing Syst.*, vol. 17, nos. 3–4, pp. 347–382, Sep. 1994.
[24] I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann, "VNE-AC: Virtual network embedding algorithm based on ant colony metaheuristic," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2011, pp. 1–6.
[25] M. Dorigo and T. Stützle, *Ant Colony Optimization: Overview and Recent Advances*. Boston, MA, USA: Springer, 2010, pp. 227–263, doi: 10.1007/978-1-4419-1665-5_8.
[26] V. Bahl. *Emergence of Micro Datacenter (Cloudlets/Edges) for Mobile Computing*. Accessed: Mar. 13, 2024. [Online]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/11/Micro-Data-Centers-mDCs-for-Mobile-Computing-1.pdf
[27] *Vapor Edge Module (VEM-20) Specifications*. Accessed: Mar. 13, 2024. [Online]. Available: https://www.vapor.io/technology/datasheet/#VEM20
[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[29] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.

AKISHIGE IKOMA received the M.E. degree in information science and technology from Osaka University, Japan, in 2022, where he is currently pursuing the Ph.D. degree with the Graduate School of Information Science and Technology.

His research interest includes network architecture for a disaggregated data center.

YUICHI OHSITA (Member, IEEE) received the M.E. and Ph.D. degrees in information science and technology from Osaka University, Japan, in 2005 and 2008, respectively.

From April 2006 to March 2012, he was an Assistant Professor with the Graduate School of Economics, Osaka University. In April 2012, he was with the Graduate School of Information Science and Technology, Osaka University, where he has been an Associate Professor with the Institute for Open and Transdisciplinary Research Initiatives, since January 2019. In August 2023, he was an Associate Professor with the Cybermedia Center, Osaka University. His research interests include traffic engineering, traffic prediction, and network security.

Dr. Ohsita is a member of IEICE and the Association for Computing Machinery (ACM).

MASAYUKI MURATA (Member, IEEE) received the M.E. and D.E. degrees in information and computer science from Osaka University, Japan, in 1984 and 1988, respectively.

In April 1984, he joined with Tokyo Research Laboratory, IBM Japan, as a Researcher. From September 1987 to January 1989, he was an Assistant Professor with the Computation Center, Osaka University. In February 1989, he joined with the Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University, where he became a Professor with the Graduate School of Engineering Science, in April 1999. He has been with the Graduate School of Information Science and Technology, since April 2004. His research interests include information network architecture, performance modeling, and evaluation.

Prof. Murata is a member of ACM and IEICE.