Master's Thesis

Title

Resource allocation method considering future application requests in a disaggregated micro data center

Supervisor

Professor Masayuki Murata

Author

Akishige Ikoma

February 3rd, 2022

Department of Information Networking

Graduate School of Information Science and Technology

Osaka University

Master's Thesis

Resource allocation method considering future application requests in a disaggregated micro data center

Akishige Ikoma

Abstract

Many services have become provided through the cloud. These services utilize large-scale computational resources, memory, and storage in large-scale data centers, and handle a large amount of data that cannot be handled by end user's devices. However, time-sensitive applications are difficult to provide through the cloud due to the large latency between the cloud data center and end device.

Edge computing is one technology for addressing this kind of problem. In this approach, many micro data centers are deployed near users. Each user has a micro data center that is close to him/her and has more resources than the end devices. Thus, time-sensitive services can also be provided by using micro data centers.

Efficient resource utilization is important in a micro data center because a micro data center has a limited amount of resources, compared with large data centers. One approach for achieving efficient resource usage is to disaggregate the resources. In a disaggregated micro data center, resources such as CPUs, GPUs, memories, and storages are connected via a network. We can flexibly construct a virtual computer by connecting the required resources.

In a disaggregated micro data center, resource allocation has a large impact on the performance of the application. Especially, network resources between the computation and memory resources are important. If some links become congested and cause a large delay between resources, it takes a time to complete the process using the resources. Therefore, we introduce a micro data center whose network is constructed of packet switches and multi-core optical fibers and propose a resource allocation method for the micro data center using multi-core fibers. The resource allocation method allocates the computation,

memory, and network resources to each application so that each process of the application will be completed within the requested time. In this data center, we allocate optical fiber cores to the communication between resources as network resources. We accommodate more applications by allowing multiple applications to share the same optical fiber core. In addition, we also allow communication in an application to use multiple optical fiber cores between the same nodes to reduce the delay if required.

The resources for an application are allocated when the request to start the application arrives. However, the allocated resources may affect the allocation of the resources to the request that will arrive in the future. Even if there are enough computation and memory resources, the network resources required to connect them may be already used by the previous requests. Therefore, our resource allocation method avoids using the resources that may be required for the accommodation of future applications. In this method, we define an allocation cost for each resource based on the importance of the resource and allocate resources based on the costs.

We evaluate our resource allocation method by simulation. The results show that our method can allocate requested resources even in the case that the method that allocates resources without considering future requests cannot allocate more than 10% of requested resources. we also demonstrate that we can accommodate more applications by allowing multiple applications to use the same optical fiber core and allowing each communication to use multiple optical fibers if necessary. By allowing them, we can reduce the number of requests whose requirements cannot be satisfied by 2/3.

Keywords

Disaggregation
Remote memory
Micro data center
Multicore fiber
Resource allocation

Contents

List of Figures

List of Tables

1 Introduction

Many services have become provided through the cloud. These services utilize large-scale computational resources, memory, and storage in large-scale data centers, and handle a large amount of data that cannot be handled by end-user devices. However, time-sensitive applications are difficult to provide through the cloud due to the large latency between cloud data center and end device.

Edge computing is one technology for addressing this kind of problem [1]. In this approach, many small data centers (hereafter we call micro data centers (μDCs)) are deployed near users. Each user has a μDC that is close to him/her and has more resources than the end devices. Thus, time-sensitive services can also be provided by using micro data centers. By deploying the process to compress the data in μDCs, we can also reduce the amount of traffic to the cloud data centers.

Efficient resource utilization is important in a μDC because a μDC has a limited amount of resources, compared with large data centers. One approach to achieving efficient resource utilization is to disaggregate resources [2]. In this approach, each μDC is constructed of resources connected by a network as shown in Figure 1. We call these μDCs disaggregated micro data centers (μDDCs). Unlike traditional data centers where each server has its own CPUs, RAM, and storage, we can flexibly use the resources in μDDCs by allocating the needed amount of resources to each task [3]. In addition, the resources in μDDCs can be easily upgraded because each resource is independent. That is, we can timely update CPU, GPU, and memory to follow their evolution.

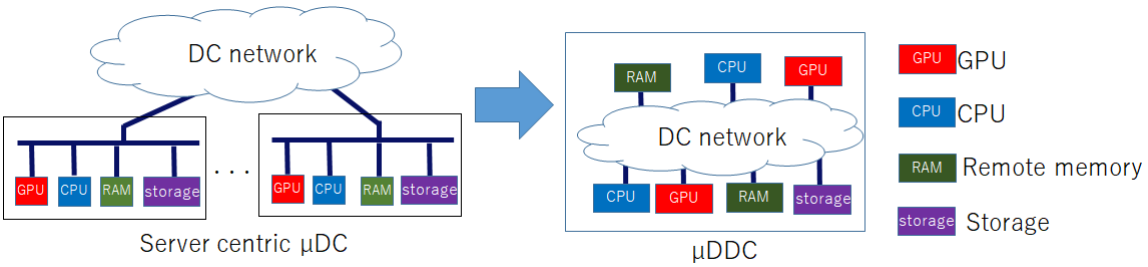


Figure 1: Differences between server centric and disaggregated data centers

In a disaggregated micro data center, resource allocation has a large impact on the performance of the application. Especially, network resources between the computation

and memory resources have a large impact [3–6].

We evaluated the impact of network performance on the execution performance of AI applications provided by μDDC [4]. The results indicate that latency has a large impact on the performance of the application. Therefore, several architectures to construct disaggregated data centers have been proposed. Zervas et al. used optical circuit switches in a data center in addition to the electrical packet switches [5]. By configuring the optical circuit switches, the resources can communicate with low latency.

In a disaggregated data center, resource allocation is also important. The inappropriate allocation may cause a large delay between resources and degrades the performance of the application. Therefore, several resource allocation methods have been proposed. Papaioannou et al. proposed a resource allocation method based on latency and bandwidth [7]. This method allocates resources and paths between them so that enough CPU and memory resources are allocated and latencies between them are less than the predefined target value. The above researches assume that the paths between the resources are reserved and cannot be used by the other applications. However, the network resources should also be used efficiently in a μDDC. μDDC can accommodate more applications by sharing network resources among multiple applications.

Therefore, we introduce a μDDC architecture and propose a resource allocation method for the architecture that allows the network resources shared among multiple applications. In this architecture, the network is constructed of packet switches and multi-core optical fibers. The resource allocation method allocates the computation, memory, and network resources to each application so that each process of the application will be completed within the requested time. In this data center, we allocate optical fiber cores to the communication between resources. We accommodate more applications by allowing multiple applications to share the same optical fiber core. In addition, we also allow communication in an application to use multiple optical fiber cores between the same nodes to reduce the delay if required.

The resources for an application are allocated when the request to start the application arrives. However, the allocated resources may affect the allocation of the resources to the request that will arrive in the future; even if there are enough computation and memory resources, the network resources required to connect them may be already used by the

previous requests. Therefore, our resource allocation method avoids using the resources that may be required for the accommodation of future applications. To achieve this, we define an allocation cost for each resource including computation, memory, and network resources based on the importance of the resource, and allocate resources based on the costs.

In this thesis, we evaluate our resource allocation method by simulation and demonstrate that our method accommodates more applications satisfying the requirements.

The rest of this thesis is organized as follows. Chapter 2 describes related work. Chapter 3 describes the μDDC assumed in this thesis, and Chapter 4 proposes a resource allocation method considering future application requests. Chapter 5 evaluates our resource allocation method by simulation. Finally, Chapter 6 concludes this thesis.

2 Related work

2.1 Recent research on resource disaggregation

A disaggregated data center (DDC) is a data center constructed of resource units such as CPU, GPU, and memory that are connected to each other via a network. Resource disaggregation improves resource utilization and scaling flexibility [8].

In a DDC, the network performance has a large impact on the performance of applications. In particular, the communication delay between the CPU and remote memory may degrade the performance. Gao et al. investigated the network requirements for a DDC and concluded that high bandwidth communication is necessary to reduce the impact of communication delays [3]. Liu et al. also discussed the requirements for the optical transmission technology in a DDC and pointed out that high-bandwidth communication devices that support more than 1 Tbps communication are necessary for a large-scale DDC that runs large-scale parallel computing applications [6]. We also investigated the impact of the network performance on the performance of the application, focusing on the AI application to be hosted in edge μDDC [4]. As a result, we showed that the latency has a large impact on the application performance while higher bandwidth than 50 Gbps is not necessary for the applications such as image recognition that require frequent communication with small data size between CPUs and memories.

Several approaches that reduce the delay between resources in a DDC have been proposed [9–11]. Gu et al. reduced processing delays during communication by using Remote Direct Memory Access (RDMA), which allows direct access to remote memory without OS processing [9]. Sidler et al. reduced the overhead of copying and processing of data from remote memories by direct access to data in remote memory through a programmable NIC [10]. Lee et al. reduced the overhead of memory management by centrally managing remote memory by programmable network switches [11].

Architectures in a DDC have also been proposed. Zervas et al. proposed network architecture for a DDC that uses optical circuit switches in addition to electrical packet switches [5]. By configuring the optical circuit switches, the resources can communicate with low latency.

Especially the DDC architecture focusing on the AI applications have been proposed

in recent years. Kwon et al. proposed a ring network in which computing devices and memories are connected alternately [12]. In this architecture, computing devices can access both of the neighbor memories with low latency. They also proposed another architecture that connects the GPU and memory with a high bandwidth link whose capacity is more than 100 Gbps [13]. They also have solved the problem of performance limited by the bandwidth inside the memory by providing a computing unit inside the memory although there is a problem of high power usage. Zhu et al. construct a low-latency remote memory by using an optical interconnect to connect the computing device to the memory [14]. Optical interconnects enable low latency communication.

The above researches assume that the paths between the resources are reserved and cannot be used by the other applications. However, the network resources should also be used efficiently in a μDDC. A μDDC can accommodate more applications by sharing network resources among multiple applications. Therefore, in this thesis, we introduce a μDDC architecture and propose a resource allocation method for the architecture that allows the network resources shared among multiple applications.

2.2 Resource allocation method for disaggregated data center

In a disaggregated data center, resource allocation is also important. The inappropriate allocation may cause a large delay between resources and degrades the performance of the application. Therefore, several resource allocation methods have been proposed [6, 7]

Papaioannou et al. proposed a resource allocation method based on latency and bandwidth [7]. This method allocates resources and the paths between them so that enough CPU and memory resources are allocated and latencies between them are less than the predefined target value.

Zervas et al. proposed a resource allocation method that minimizes the cost based on the residual amount of resources, the bandwidth, and length of the path between resources [6]. This method allocates resources and the paths between them so that enough CPU and memory resources are allocated and latencies between them are less than the predefined target value.

In this thesis, we consider a μDDC where network resources can be shared among multiple applications to achieve efficient network resources. In this environment, commu-

nication between a pair of resources has an impact on the communication of another pair of resources that share the same network resources; if a pair frequently exchanges a large amount of data, the delay on the links passed by the data increases. The above existing methods did not consider such impacts. Some of them assume that the links between the resources are reserved and cannot be used by other applications. The others consider only the available bandwidths even if they allow links shared by multiple applications. In this thesis, we model the delay within a network in a μDDC by using the queueing theory and propose a method to allocate the resources considering the time required to complete the processes in each application. In addition, the resources for an application are allocated when the request to start the application arrives. However, the allocated resources may affect the allocation of the resources to the request that will arrive in the future; even if there are enough computation and memory resources, the network resources required to connect them may be already used by the previous requests. This impact was not considered in the existing methods. Therefore, we propose an approach to avoiding using the resources that may be required for the accommodation of the future application. In this thesis, we achieve this by defining an allocation cost for each resource including computation, memory, and network resources based on the importance of the resource, and allocating resources based on the costs.

3 Disaggregated micro data center

The overview of a μDDC in this thesis is shown in Figure 2.

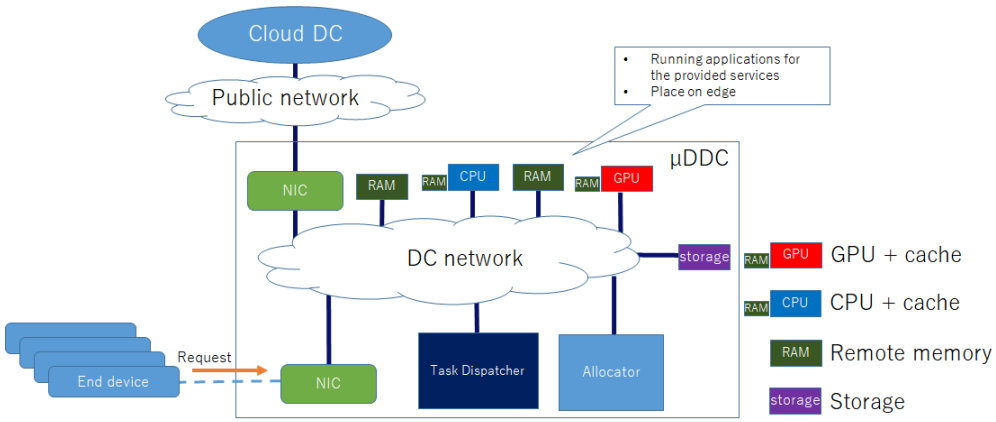


Figure 2: The overview of a disaggregated micro data center in this thesis

In this μDDC, CPUs, GPUs, remote memory, storage, NICs, task dispatchers, and allocators are connected by a DC network. The NICs are used to communicate with end devices and the cloud to provide services to users.

3.1 Components of a μDDC

3.1.1 Resources

In the μDDC, resources such as the CPU or remote memory and modules such as the task dispatcher or allocator are connected by a network.

Computing resource (CPU, GPU) Each CPU has multiple CPU cores. Each CPU core is allocated to an application. That is, we regard a CPU core as the unit of computing resource. In this thesis, we assume that each CPU core has a small amount of local memory as cache memory and uses a paging system. That is, if required data cannot be found in the local memory, the CPU obtains the page that contains the required data from the remote memory. When the cache becomes full, some pages are moved to the remote memory.

Each GPU is allocated to an application. That is, we regard a GPU as the unit of computing resource. GPUs are another type of computation resource. In the μDDC, if an

application requires GPUs, GPUs are allocated to the application. We regard one GPU as the unit of computation resources.

Remote memory Remote memory is a device that stores the data required by the computational resources. In this thesis, we divide each memory into blocks and allocate each block to an application. That is, we regard each block as a unit of memory.

Storage Storage stores the data needed to deploy the application. For example, for an AI application, the machine learning model is copied from storage to remote memory when the application is deployed.

NIC NICs are used to communicate with the external network. When providing a service, it sends the processing request from the end device to the task dispatcher and returns the processing result to the end device. Also, if the process needs to work with the cloud, it communicates with the cloud through the NIC.

Task dispatcher A task dispatcher is deployed for each service and relays the requests for the service to the corresponding computational resources of the service. A task dispatcher is allocated from the computational resources in the μDDC and is dynamically configured as the service is provided.

Allocator The allocator has information about the μDDC network, and each time a service deployment request arrives, it allocates resources to perform the task dispatch process and resources to execute the application corresponding to the requested service.

3.1.2 μDDC network

The μDDC network is a network that connects resources in a μDDC. This network is constructed of switches and multi-core optical fibers. A multi-core optical fiber has multiple optical fiber cores as shown in Figure 3. Each optical fiber core is regarded as a link. That is, we have multiple links between the switches connected by a multi-core optical fiber. In this thesis, each optical fiber core is allocated to an application but can be shared

by multiple applications. In addition, if required, multiple fiber cores between the same nodes can be allocated to an application.

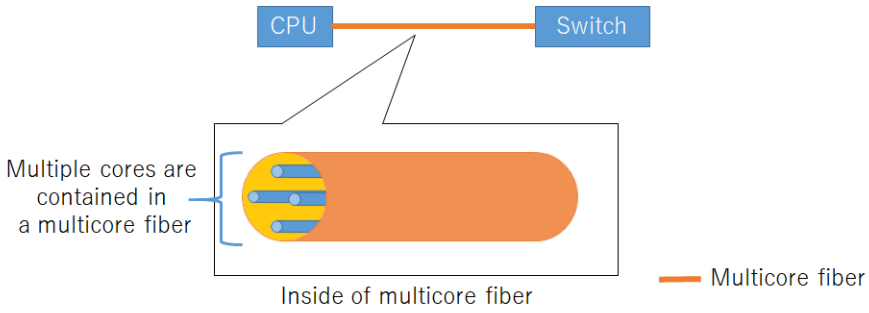


Figure 3: The overview of multicore fiber

Switch The switch relays the packets based on the configured path for each resource pair allocated to each application. Each switch is connected to the next switches by multi-core optical fibers. Each optical fiber core is connected to each port of each switch. That is, a switch relays packet by relaying it to the optical fiber core connected to the next switch on the path.

Each switch has a buffer and can cut-through switching. That is, if the next port is available, the switch immediately relays the packet to the next port before receiving the whole packet. If the next port is busy, the switch stores the packet in the buffer and wait for the next port to become ready.

In this thesis, we allow multiple fiber cores allocated to an application. In this case, if one of the fiber cores is available, the packets are relayed to the available fiber core. Thus, the delay in the switch can be reduced by allocating multiple fiber cores.

3.2 Allocation request

The allocation request is sent to the allocator when deploying a new service, or when the number of users increases and more resources such as CPU and remote memory are required to provide the service. The requests include the information of the required resources, the process to complete each task in the application, and the acceptable process time of the application. The allocator allocates the resources so that the process of the application can be completed within the acceptable process time. Then the program and

data required in this application are deployed to the allocated resources and the service is started.

3.3 Service delivery form in μDDC

Once the application is started, the process of the application is performed as follows.

Services are provided after the application is allocated. The flow of service is :

1. A task request from the end device arrives. The request is forwarded to the task dispatcher.

2. The task dispatcher finds available computational resources that can perform the requested task and sends the command to the computational resources.

3. The computational resources perform the task by cooperating with the other resources allocated to the application.

4. After completing the task, the computational resource sends back the results to the end device via the NIC.

In this thesis, we assume that the deployed applications remain unless the applications become no longer used. That is, the resources are allocated only before the application is used for the first time. By doing so, each application can perform tasks immediately.

4 Resource allocation method

4.1 Model of disaggregated micro data center

In this subsection, we model a μDDC network and an allocation request. Table 1 shows notations used in this model and Table 2 shows parameters used in this model

Table 1: Notations in μDDC network and allocation requests

μDDC network	
$G^s(N^s, E^s)$	Set of nodes and links in the μDDC network
C_n^s	Set of available computational resources located at node $n \in N^s$
M_n^s	Set of available remote memories located at node $n \in N^s$
Allocation requests	
a	Requested application
S_a	Set of tasks in application a
$G^v(N^v, E^v)$	Set of nodes and links in a resource graph
$G^p(N^p, E^p)$	Set of nodes and links in a process graph
c_p^p	Index of the node corresponding to a computational resource to execute the process p on a resource graph
m_p^p	Index of the node corresponding to a remote memory to execute the process p on a resource graph

4.1.1 Model of disaggregated micro datacenter network

A μDDC network is represented as a graph $G^s(N^s, E^s)$. N^s and E^s are the set of nodes and the set of links in the μDDC network, respectively.

Node In this model, we have three types of nodes, nodes with computational resources, nodes with memory resources, and switches. C_n^s is the set of available computational resources on n and M_n^s is the set of available memory resources on n. In addition, we define the property of each resource. For each computational resource $c \in C_n^s$, we define the performance value K_c defined by the number of floating-point operations per second, and clock frequency F_c. We also define the page size of the computational resource c as P_c.

We also define the property of switching capability. We define two properties; the processing delay T_n^N to relay a packet to the next node by cut-through, and the delay T_n^Q to send the whole packet to the next node. If the node n does not has switching capability, we set $T_n^N = \infty$ and $T_n^Q = \infty$.

Table 2: Parameters in μDDC network and allocation requests

μDDC network	
T_n^Q	The delay to send the whole packet to the next node in node $n \in N^s$
T_n^N	the processing delay to relay a packet to the next node by cut-through in node $n \in N^s$
B	Network bandwidth
K_c	FLOPS of computational resource c
F_c	Clock frequency of computational resource c
T_e^P	Propagation delay of link $e \in E^s$
$\lambda_{e,n}^s$	The arrival rate of packets passing through link $e \in E^s$ from node $n \in N^s$
N_e^{core}	The number of fiber cores contained by link $e \in E^s$
W_r	Cost of a resource r.
W_e	Cost of a link $e \in E^s$
P_c	Page size of computational resource c
Allocation requests	
$T_{a,t}$	Time constraints of task $t \in S_a$
λ_p^r	Arrival rate of packets from remote memory
λ_p^w	Arrival rate of packets to remote memory
$\sigma_{c,p}^c$	The clock counts to execute process p in computational resource c
σ_p^{pf}	The number of page faults to execute process p
σ_p^{pn}	The number of pages transmitted per page fault to execute process p

Link In this thesis, we allocate optical fiber cores to the communication between the resources allocated to applications. We also allow the communication between a resource pair in an application to use multiple fiber cores. In this model, we regard all candidates of links between nodes, including each available optical fiber core and available group of multiple optical fiber core, as links.

Each link $e \in E^s$ has the following properties, the number of fiber cores N_e^{core} and the propagation delay of e by T_e^P. All fiber cores have the same bandwidths B.

4.1.2 Model of allocation request

An allocation request is given in two graphs showing the relationship between the computational resources (hereafter we call a resource graph) and showing the relationship between the processes to execute a task of requested application (hereafter we call process graph), as shown in Figure 4.

A resource graph is given in a graph structure $G^v(N^v, E^v)$ where N^v and E^v are the set of nodes and the set of links. In this graph, each node corresponds to the required

computational or memory resources. The links are added between the nodes that require the exchange of information.

A process graph is given in a directed graph structure $G^p(N^p, E^p)$, where N^p and E^p are the set of nodes and the set of links. Each node $p \in N^p$ represents a process that is required to execute a task and has the property, the number of page faults, and the number of pages transmitted per page fault that are obtained in advance by monitoring the application in a test environment. By using them, we obtain the arrival rate of the packets from the memory λ_p^r and the arrival rate of packets to the memory λ_p^w in the process corresponding to node $p \in N^p$. In addition, the resource request includes the information on the mapping between the resource graph and the process graph. c_p^v and m_p^v indicate the pointer to the computational and memory resources in the resource graph corresponding to the node $p \in N^p$. Each link $e \in E^p$ is a directed link indicating the order of the process. Each path from the first process to the final process indicates the processes required to complete the task. We denote the set of all possible paths by S_a. We call each $t \in S_a$ a task. For each task t, the information of the acceptable execution time is defined. That is, the resources should be allocated so that all tasks can be completed within the acceptable execution time.

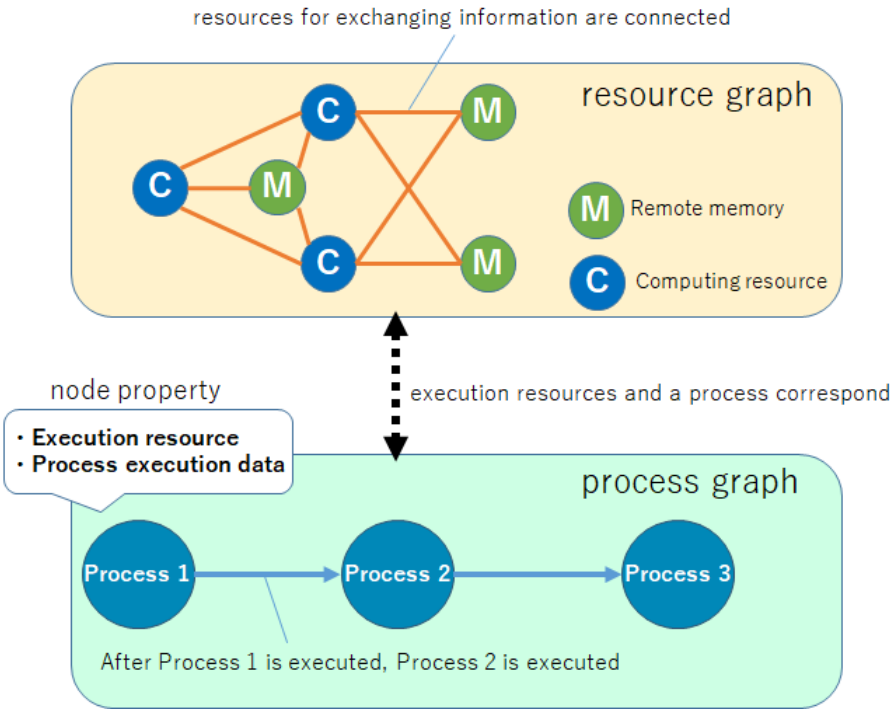


Figure 4: The overview of resource graph and process graph

4.2 Resource allocation probelem

We propose an approach to avoiding using the resources that may be required for the accommodation of future applications. In this thesis, we achieve this by defining an allocation cost for each resource including computation, memory, and network resources based on the importance of the resource. In this subsection, we define the costs of resources and formulate the problem to allocate resources based on the costs.

4.2.1 Variables

Mapping of computational or memory resources $\delta^N_{i,j}$ denotes the binary variable indicating the mapping between the requested resources and the resources in the μDDC. $\delta^N_{i,j} = 1$ when the resource graph node N^v_i is mapped in μDDC network node N^s_j, $\delta^N_{i,j} = 0$ otherwise.

Mapping of links $\delta^E_{i,j}$ denotes the binary variable indicating the mapping between the links in the requested resource graph and the links in the μDDC. $\delta^E_{i,j} = 1$ when the resource

graph link E_i^v is mapped to μDDC network link E_j^s, $\delta_{i,j}^E = 0$ otherwise.

4.2.2 Costs

In this thesis, we define the costs of memory, links, and computational resources such as CPUs and GPUs.We set the costs of resources that are required by future requests to large values.

Cost of a computational resource High-performance computational resources are important because they can execute applications whose acceptable process time is small or that require large computational resources. In this thesis, the performance of a computational resource c is defined as the product of the number of available cores $|C^s_{Node_c}|$ and the performance of the core K_c. Therefore, the cost of the computational resource c is

$$W_c = (|C^s_{Node_c}|) \cdot K_c \tag{1}$$

where, $Node_c$ is a node with computational resource c in μDDC network.

Cost of a remote memoy A remote memory that has a large number of available memory blocks can accommodate a resource request that requires a large amount of memory. Thus, we set the cost of remote memory m based on the number of available remote blocks in remote memory $|M^s_{Node_m}|$.

$$W_m = |M^s_{Node_m}| \tag{2}$$

where $Node_m$ is a node with remote memory m in μDDC network.

Cost of a link When a request of application that requires large computational and memory resources arrives, it requires the computational and memory resources whose costs defined above are large. In addition, the network resources between the resources are also required. Thus, we set the costs of the links between the computational and memory resources with large costs to large values. On the other hand, a link that has already been allocated as a path for any applications is not able to provide high-speed communication. Thus, the cost of already allocated links is set minimal value ϵ. We define the cost of the link e by

$$W_e = \begin{cases} \sum_{c \in C^{cand}, m \in M^{cand}} \left\{ \left(\frac{N^{path}_{c,m}(e)}{N^{path}_{c,m}} \right) \left(\frac{W_c \cdot W_m}{shortestHop(c,m)} \right) \right\} & e \notin E^{alloc} \\ \epsilon & e \in E^{alloc} \end{cases} \tag{3}$$

where C^{cand} and M^{cand} are the set of available computational resources and remote memories in the μDDC network, respectively. E^{alloc} is the set of links that have already been used in other applications and $N^{path}_{c,m}$ and $N^{path}_{c,m}(e)$ represent the number of shortest paths between resources c and m, and the number of shortest paths that pass through the link e. $shortestHop(c,m)$ is the shortest hop count between resource c and m.

4.2.3 Predicted execution time

The predicted execution time of any task $t \in S_a$ of the requested application a can be calculated as a sum of the predicted time T^{exe}_p to complete the processes corresponding to the nodes $p \in N^p_t$ in a process graph of task t. Therefore, predicted execution time T^{task}_t of task t is

$$T^{task}_t = \sum_{p \in N^p_t} T^{exe}_p \tag{4}$$

T^{exe}_p is the sum of the processing time in the computational resource and communication delay to obtain the data from a remote memory. That is

$$T^{exe}_p = \sum_{i=0}^{|N^s|-1} \left\{ \delta^N_{c^p_p,i} \cdot T^{calc}_{c^s_i,p} + \delta^N_{m^v_p,i} \cdot T^{delay}_{c^v_p,m^v_p,p} \right\}$$

where T^{calc} is the processing time in the computational resource and T^{delay} is the communication delay to obtain the data from a remote memory and c^s_i is a computational resource in C^s_i.

The processing time in computational resource The processing time $T^{calc}_{c,p}$ for process p in computational resource c calculated by dividing the expected clock count $L_{c,p}$ required to complete the process p by the clock frequency F_c of the computational resource c.

$$T^{calc}_{c,p} = \frac{\sigma^c_{c,p}}{F_c}$$

Communication delay The delay required to obtain the data of one page is the sum of the time required to obtain the head of the page from the remote memory m and the transmission delay. That is,

$$T^{delay}_{c,m,p} = \left(\sum_{i=0}^{|N^s|-1} \delta^N_{c,i} \cdot \frac{\sigma^{pn}_p \cdot P_{c^s_i}}{B} + T^{latency}_{E^r_{c,m},p} \right) \cdot \sigma^{pf}_p$$

where $E^r_{c,m}$ is the link between node c and m in the resource graph and $T^{latency}$ is the sum of the propagation delay and the response time and c^s_i is a computational resource in C^s_i.

$T^latency_{e,p}$ can be obtained by the sum of the propagation delay T^P and the response time T^R.

$$T^{latency}_{e,p} = \sum_{i=0}^{|E^s|-1} \delta^E_{e,i} \left(T^p_i + T^R_{N^s_{e,i}}(\lambda^s_{i,N^s_{e,i}} + \lambda^r_p, N^{core}_i, T^Q_{N^s_{e,i}}) \right)$$

where $N^s_{e,i}$ is the source node of link i when reading data from remote memory and T^R is response time when buffering occurs.

We model T^R by the queuing theory. The processing speed at each switch is fixed. By allocating multiple optical fiber cores, each packet can be relayed to any of the fiber cores if it is available. In this thesis, assuming that packets arrive at each switch according to the Poisson process, we model the response time by the M/D/C queuing model. However, it is difficult to obtain an accurate response time in the M/D/C queuing model. accurate derivation of the response time in the M/D/C queueing model is difficult. Thus, we use the approximation [15]. The approximated response time is

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

where

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

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

$$h^Q(\lambda, c, T_n^Q) = \frac{T_n^Q}{c\left(1-\frac{\lambda T_n^Q}{c}\right)} \left\{ \frac{\left(\lambda T_n^Q\right)^c}{c!\left(1-\frac{\lambda T_n^Q}{c}\right)} \left[\sum_{i=0}^{c-1} \frac{(\lambda T_n^Q)^c}{i!} + \frac{\left(\lambda T_n^Q\right)^c}{\left(1-\frac{\lambda T_n^Q}{c}\right) c!} \right]^{-1} \right\}$$

where the packet arrival rate is λ, the number of optical fiber cores is c, and the processing time of a node is T_n^Q.

4.2.4 Definition of the resource allocation problem

Resource mapping constraints A request graph node and a μDDC network node have a one-to-one relationship. That is

$$\begin{aligned} &\textstyle\sum_{i=0}^{m} \delta_{i,j}^N \leq 1 \\ &\textstyle\sum_{j=0}^{n} \delta_{i,j}^N = 1 \end{aligned} \tag{5}$$

Each request must be allocated to one of the available resources

$$|M_j^s| + |C_j^s| - \textstyle\sum_{i=0}^{m} \delta_{i,j}^N \geq 0 \tag{6}$$

Time constraints for requested applications All tasks in a requested application must be executed within an acceptable time.

$$T_t^{task} \leq T_{a,t} \quad \forall t \in S_a \tag{7}$$

Objective In this method, we allocate resources to minimize the allocation cost of resources and links.

$$minimize \quad \textstyle\sum_{i=0}^{|N^s|-1} \sum_{j=0}^{|N^v|-1} \delta_{i,j}^N (W_{c_j^s} + W_{m_j^s}) + \sum_{y=0}^{|E^s|-1} \left[1_{\sum_{y=0}^{|E^v|-1} \delta_{x,y}^E > 0} W_{e_y^s} \right] \tag{8}$$

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. c_j^s represents available computational resource in C_j^s and m_j^s represents available remote memory in M_j^s .

We mesmerize the resource allocation problem as follows.

Constraints:

$$\forall i \in \{0,1,...,|N^v|-1\}, \forall j \in \{0,1,...,|N^e|-1\} \mid \delta^N_{i,j} \in \{0,1\} \tag{9}$$

$$\forall x \in \{0,1,...,|E^v|-1\}, \forall y \in \{0,1,...,|E^s|-1\} \mid \delta^E_{x,y} \in \{0,1\} \tag{10}$$

$$\sum_{i=0}^{m} \delta^N_{i,j} \leq 1 \tag{11}$$

$$\sum_{j=0}^{n} \delta^N_{i,j} = 1 \tag{12}$$

$$|M^s_j| + |C^s_j| - \sum_{i=0}^{m} \delta^N_{i,j} \geq 0 \tag{13}$$

$$T^{task}_t \leq T_{a,t} \quad \forall t \in S_a \tag{14}$$

Objective:

$$minimize \quad \sum_{i=0}^{|N^s|-1} \sum_{j=0}^{|N^v|-1} \delta^N_{i,j}(W_{c^s_j} + W_{m^s_j}) + \sum_{y=0}^{|E^s|-1} \left[1_{\sum_{y=0}^{|E^v|-1} \delta^E_{x,y} > 0} W_{e^s_y} \right] \tag{15}$$

We allocate resources by solving the above. However, this problem is a nonlinear integer programming problem and NP-hard. One way to solve such a problem is a meta-heuristic method. In this thesis, we use Ant Colony Optimization [16].

4.3 Resource allocation method based on ant colony optimization

We allocate resources by solving a resource allocation problem. We use ant colony optimization to solve the problem. Our method is similar to the virtual network embedding method based on the ant colony optimization (VNE-AE) [16], but is different from the VNE-AE in the following points; (1) the cost defined in this thesis considers the future allocation request and is different from the VNE-AE and (2) our method searches the links so as to minimize the costs because the selection of the used links affects the cost significantly, though the VNE-AE searches only the locations of virtual machines and uses the shortest paths. We summarize the notations and parameters used in our method based on the ant colony optimization as shown in Table 3 and Table 4, respectively.

Our method finds suitable resource allocation by using agents. Each agent performs the following steps.

Table 3: Notations used in our resource allocation method based on ant colony optimization

C_i^{calc}	Set of computational resources that are candidates for allocation in agent i
C_i^{mem}	Set of remote memory that are candidates for allocation in agent i
$C_{i,n}^{link}$	Set of links that are candidates for allocation at agent i in node n
R^{best}	Set of resources of the best solution for each generation
E^{best}	Set of links of the best solution for each generation

Table 4: Parameters in our resource allocation method based on ant colony optimization

N^{ant}	The number of agents
N^{itr}	The number of agent generations
p_r	Probability that resource r is allocated
p_e	Probability that link e is allocated
τ	Pheromones associated with resources and links
α	Pheromone weight
β	Cost weight
ρ	Pheromone decrease rate
φ	Pheromone increase rate

- Resource search phase: The agent searches the available resources to be allocated to the node in the requested resource graph. After finding resources for all nodes, the agent goes to the Link search phase.

- Link search phase: The agent searches the links to be allocated to the links in the requested resource graph. To search the links, the agent generates N^{ant} sub-agents that search the links. After finding all required links, the agent goes to the Update phase.

- Update phase: The agent updates the pheromone based on the find resources and links.

After continuing the above steps N^{itr} times, we find the best allocation whose costs are

the smallest.

4.3.1 Resource search phase

In the resource search phase, an agent searches the computational and memory resources. The agent selects one of the nodes in the requested resource graph whose corresponding resources are not allocated. Then, the agent selects the resources to be allocated for the selected node based on the probabilities p_c for the computational resource and p_m for the memory. We set p_c and p_m by

$$p_c = \frac{\tau_c^\alpha \left(\frac{1}{W_c^\beta}\right)}{\sum_{c \in C_i^{calc}} [\tau_c^\alpha \left(\frac{1}{W_c^\beta}\right)]} \tag{16}$$

$$p_m = \frac{\tau_m^\alpha \left(\frac{1}{W_m^\beta}\right)}{\sum_{m \in C_i^{mem}} [\tau_m^\alpha \left(\frac{1}{W_m^\beta}\right)]} \tag{17}$$

where α and β are the relative importance of pheromone and cost, respectively. In the resource search phase, the above steps to select resources are repeated until resources for all nodes are found.

4.3.2 Link search phase

In the link search phase, the agent searches the links between resources selected in the resource search phase. To search the links, the agent generates N^{ant} sub-agents.

Each sub-agent selects one of the links in the requested resource graph whose corresponding paths in the μDDC have not been found. Then it searches the paths in μDDC for the selected link. The paths are searched from the source resource; the link from the source resource is first selected, and then the next link from the destination of the first link is selected. This process is continued until the link to the destination resource is found. At each step, the link is selected based on the probabilities $p_{e,n}$ defined for all candidates links from the node n.

$$p_{e,n} = \frac{\tau_e^\alpha \left(\frac{1}{W_e^\beta}\right)}{\sum_{e \in C_{i,n}^{link}} \left[\tau_e^\alpha \left(\frac{1}{W_e^\beta}\right)\right]} \tag{18}$$

where α and β represent the relative importance of pheromone and cost, respectively.

4.3.3 Upadate pheromone

After finding the resources, the agent updates pheromone. The pheromone is updated based on pheromone decrease rate ρ $(0 < \rho < 1)$. The pheromones of the resources and links of the best solution for each generation are enhanced based on the pheromone increase rate φ. The pheromone τ_r of any resource r in the μDDC network is updated by

$$\tau_r = \begin{cases} \rho\tau_r + \dfrac{\phi}{\sum_{x \in R^{best}} W_x + \sum_{e \in E^{best}} W_e} & r \in R^{best} \\ \rho\tau_r & r \notin R^{best} \end{cases} \tag{19}$$

The pheromone τ_e of any link e in the μDDC network is also updated by

$$\tau_r = \begin{cases} \rho\tau_e + \dfrac{\phi}{\sum_{x \in R^{best}} W_x + \sum_{e \in E^{best}} W_e} & e \in E^{best} \\ \rho\tau_e & e \notin E^{best} \end{cases} \tag{20}$$

5 Evaluation

In this section, we evaluate our method by simulation.

5.1 Comparison with optimum allocation

In this subsection, we compare the resource allocation by our method with the optimum resource allocation.

μDDC network We use a small μDDC network shown in Figure 5. Each computational resource has 5 cores and each memory has 250 memory blocks. Each optical fiber has one optical fiber core. The other parameters are set to the values shown in Table 5.

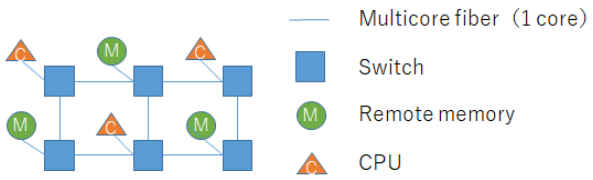


Figure 5: 2×3 grid μDDC network

Table 5: Parameters setting of μDDC network

Parameters	Value
CPU FLOPS	13.619GFLOPS
Propagation delay	$0.025\mu s$
Switch latency when buffering	$3\ \mu s$
Cut-through latency	$300ns$
Page size	4KB
Bandwidth of each core	50Gbps

Resource request In this evaluation, we generate three types of resource requests shown in Table 6. We generate the sequence of the resource requests by generating the requests at each time slot based on the probabilities shown in Table 7.

Table 6: Parameters setting of resource requests for comparison with optimum allocation

	Request 1	Request 2	Request 3
Time constraint	3000ms	500ms	250ms
Required computing resources	3	5	5
Required remote memories	3	5	5
Process 1			
Clock count	0.035	0.035	0.035
The arrival rate of the packets to remote memory per 1ms	0.00066	0.004	0.01
The arrival rate of the packets from remote memory per 1ms	0.00066	0.004	0.01
Process 2			
Clock count	0.054	0.054	0.054
The arrival rate of the packets to remote memory per 1ms	0	0	0
The arrival rate of the packets from remote memory per 1ms	0.00066	0.004	0.01
Process 3			
Clock count	2371.33	1960.36	1960.36
The arrival rate of the packets to remote memory per 1ms	3.74	3.8	3.8
The arrival rate of the packets from remote memory per 1ms	7.42	6.86	6.86
The number of page faults	67543.25	56661.29	56661.29
The number of pages transferred per page fault	5.27	4.84	4.84

Table 7: Parameters setting for generating resource requests

Request sequence	p_1	p_2	p_3
Request sequence 1	0.8	0.1	0.1
Request sequence 2	0.4	0.2	0.4
Request sequence 3	0.1	0.1	0.8

Resource allocation methods In this subsection, we run our resource allocation method by setting the parameter to the values shown in Table 8. We also use a method to find the optimum solution in addition to our resource allocation method based on the ant colony optimization. In this method, we find the optimum solution to the problem formulated in Section 4.2.4 by brute force search.

Table 8: Parameter settings of proposed resource allocation method

Parameters	Value
Number of agents searching for resources	20
Number of agents searching for routes	20
Number of agent generations	20
Pheromone decay rate	0.1
Pheromone enhancement rate	100
Pheromone weight	2
Allocation cost weight	1
Initial pheromone value	1000

5.1.1 Results

Table 9 shows the average number of rejections for each request sequence. Table 10 shows the maximum and minimum amount of allocated computational resources for each request sequence

Table 9: Average number of rejections for each request sequence

Request sequence	Optimal solution of our resource allocation problem	Solution derived by ant colony optimization	Theoretical optimal solution
Request sequence 1	0	0	0
Request sequence 2	0	0	0
Request sequence 3	0	0	0

Table 10: Maximum and minimum amount of allocated computational resources for each request sequence

Request sequence	Optimal solution of our resource allocation problem		Solution derived by ant colony optimization		Theoretical optimal solution	
	max value	min value	max value	min value	max value	min value
Request sequence 1	15	13	15	13	15	13
Request sequence 2	15	13	15	13	15	13
Request sequence 3	15	13	15	13	15	13

The results indicate that all methods allocate almost all computational resources without any rejection. That is, our method allocates the resources effectively even though it uses only the information of the current request. These results also indicate that the method based on the ant colony optimization achieves similar results to the optimal solution.

5.2 Evaluation of resource allocation method considering future application requests

We evaluate the proposed method by comparing our method with the method that allocates resources to achieve the best performance of the requested application without considering future application requests.

5.2.1 Evaluation environment

μDDC network We use a 3×3 two-dimensional torus network as shown in Figure 6. Each computational resource has 18 cores and each remote memory has 250 memory blocks. Each optical fiber has 4 optical fiber cores. The other parameters are shown in Table 5.

Resource request Similar to Section 5.1, we generate three types of resource requests but we change the resources required by each application. In this subsection, we simulate three cases of the required resources:

- Environment 1: the case that all applications require the same amount of resources

- Environment 2: the case that request 1 requires a large amount of resources.
- Environment 3: the case that request 3 has a large amount of resources.

We generate the sequence of the resource requests by generating the requests every minute based on the probabilities shown in Table 7.

In all cases, all applications end 60 minutes after the beginning of the application. We evaluated the cases that the probability of a request arriving per minute is 15% and 30%.

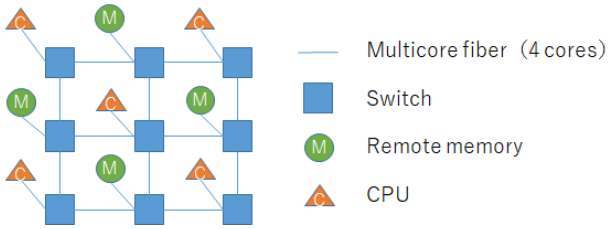


Figure 6: 3×3 two-dimensional torus μDDC network

Table 11: Parameters setting of resource requests

	Request 1	Request 2	Request 3
Time constraint	3000ms	500ms	150ms
Required computational resources(Environment 1/ 2/ 3)	4/7/4	4/4/4	4/4/7
Required remote memories(Environment 1/ 2/ 3)	4/7/4	4/4/4	4/4/7
Process 1			
Clock count	0.035	0.035	0.035
The arrival rate of the packets to remote memory per 1ms	0.000495	0.003	0.0075
The arrival rate of the packets from remote memory per 1ms	0.000495	0.003	0.0075
Process 2			
Clock count	0.054	0.054	0.054
The arrival rate of the packets to remote memory per 1ms	0	0	0
The arrival rate of the packets from remote memory per 1ms	0.000495	0.000495	0.0075
Process 3			
Clock count	2371.33	1960.36	1960.36
The arrival rate of the packets to remote memory per 1ms	2.805	2.85	2.85
The arrival rate of the packets from remote memory per 1ms	5.565	5.145	5.145
The number of page faults	67543.25	56661.29	56661.29
The number of pages transferred per page fault	5.27	4.84	4.84

5.2.2 Compared method

In this evaluation, we simulate this method by changing the cost of our method as follows.

The computational resources with higher FLOPS values have higher performance. Therefore, this method uses the cost of the computational resources defined by

$$W_c = \frac{1}{K_c}$$

In this evaluation, all remote memory has the same access speed. Therefore, this method uses the cost of remote memory defined by

$$W_m = 1$$

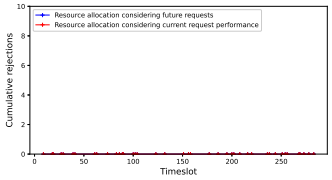
The latency on the link with low utilization is small. Therefore, this method uses the cost defined by

$$W_{e,n} = \frac{\lambda^s_{e,n}}{N^{core}_e}$$

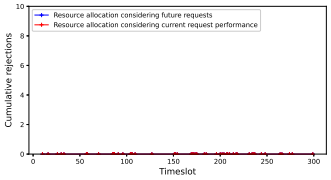
We also obtain the results for this method by using ant colony optimization similar to our method.

5.2.3 Results

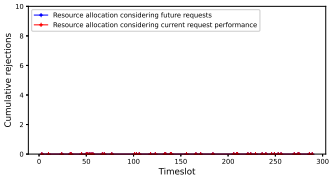
In this evaluation, the application requests whose requirements cannot be satisfied are rejected. Figures 7 shows the cumulative number of rejections for each environment and Table 12 shows the number of rejections from the start to the end of the simulation in a case that the probability of a request arriving per minute is 15%. Figures 8 shows the cumulative number of rejections for each environment and Table 13 shows the number of rejections from the start to the end of the simulation in the case that the probability of a request arriving per minute is 30%.



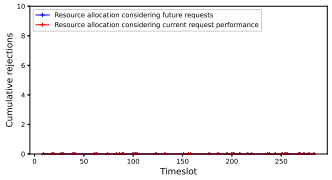
(a) Request sequence 1(Environment 1)



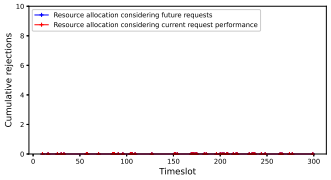
(b) Request sequence 2(Environment 1)



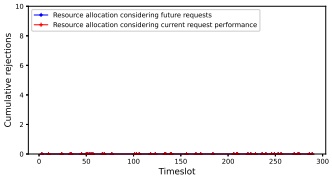
(c) Request sequence 3(Environment 1)



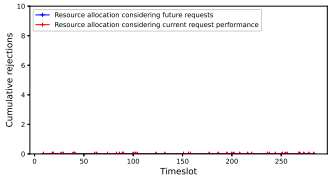
(d) Request sequence 1(Environment 2)



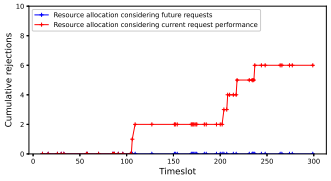
(e) Request sequence 2(Environment 2)



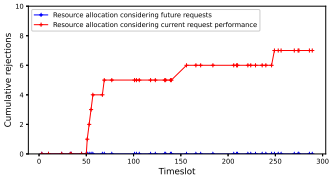
(f) Request sequence 3(Environment 2)



(g) Request sequence 1(Environment 3)

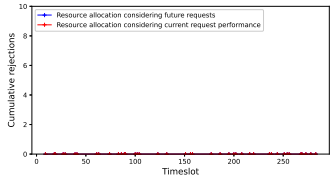


(h) Request sequence 2(Environment 3)

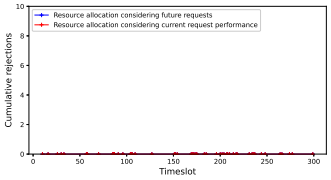


(i) Request sequence 3(Environment 3)

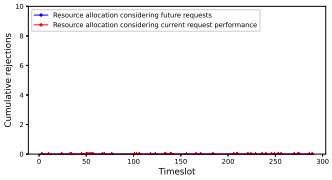
Figure 7: The cumulative number of rejections of our method and compared method (Request arrival rate per minute: 15%)



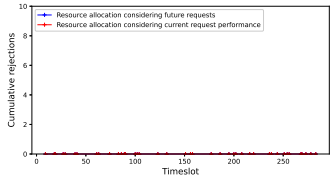
(a) Request sequence 1(Environment 1)



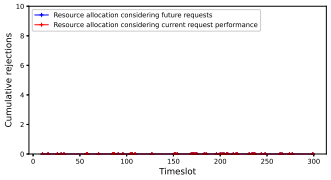
(b) Request sequence 2(Environment 1)



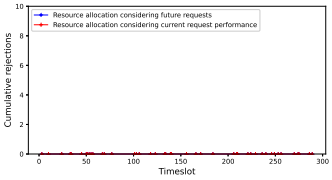
(c) Request sequence 3(Environment 1)



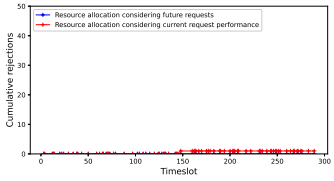
(d) Request sequence 1(Environment 2)



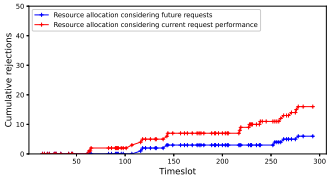
(e) Request sequence 2(Environment 2)



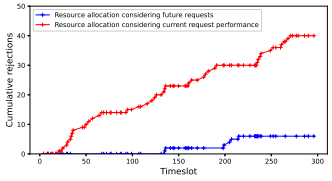
(f) Request sequence 3(Environment 2)



(g) Request sequence 1(Environment 3)



(h) Request sequence 2(Environment 3)



(i) Request sequence 3(Environment 3)

Figure 8: The cumulative number of rejections of our method and compared method (Request arrival rate per minute: 30%)

Table 12: The number of rejections of our method and compared method from the start to the end of the simulation (Request arrival rate per minute: 15%)

	Allocation considering future requests			Allocation considering only request performance		
	Request1	Request2	Request3	Request1	Request2	Request3
Environment 1						
Request sequence1	0	0	0	0	0	0
Request sequence2	0	0	0	0	0	0
Request sequence3	0	0	0	0	0	0
Environment 2						
Request sequence1	0	0	0	0	0	0
Request sequence2	0	0	0	0	0	0
Request sequence3	0	0	0	0	0	0
Environment 3						
Request sequence1	0	0	0	0	0	0
Request sequence2	0	0	0	0	0	6
Request sequence3	0	0	0	0	0	7

Table 13: The number of rejections of our method and compared method from the start to the end of the simulation (Request arrival rate per minute: 30%)

	Allocation considering future requests			Allocation considering only request performance		
	Request1	Request2	Request3	Request1	Request2	Request3
Environment 1						
Request sequence1	0	0	0	0	0	0
Request sequence2	0	0	0	0	0	0
Request sequence3	0	0	0	0	0	0
Environment 2						
Request sequence1	0	0	0	0	0	0
Request sequence2	0	0	0	0	0	0
Request sequence3	0	0	0	0	0	0
Environment 3						
Request sequence1	0	0	0	0	0	1
Request sequence2	0	0	6	0	0	16
Request sequence3	0	0	6	0	0	40

The proposed method did not cause any rejection in all environments in the case where the request arrival rate per minute is 15%. The method without considering future requests

could not allocate resources to all requests when request 3 requires a large amount of resources, regardless of the request arrival rate per minute. The rejection of the compared method was equal to or greater than that of the proposed method in all cases. That is, our method can allocate resources even in the case that the method that allocates resources without considering future requests cannot allocate resources. This difference is caused by the available resources when request 3 arrives. The method without considering future requests allocates resources to achieve the best performance for the requested application. As a result, enough resources for request 3 that requires a large amount of resources and whose acceptable execution time is small do not exist. On the other hand, our method avoids using the resources that are required by future requests. As a result, our method can allocate request 3.

The results also indicate that our method also rejects some requests when the application 3 requires a large amount of resources and arrives frequently. This is because the utilization of the resources is quite high and the application request can be accepted only when some applications end and free the resources. That is, μDDC should be deployed so that too high utilization of resources can be avoided.

5.3 Impact of sharing links

In this thesis, we allow multiple applications to share the same optical fiber core. In addition, we also allow one application to use multiple optical fibers if required. We evaluate the impact of these policies on applications accommodations.

5.3.1 Evaluation environment

μDDC network We use 3×3 two dimensional torus network similar to Section 5.2. We use the same parameter as Section 5.2.

Resource request We generate three types of requests similar to Section 5.2. Each type of request requires the resources as shown in Table 11. We generate the sequence of the resource requests by generating the requests every minute based on the probabilities shown in Table 7. In all cases, all applications end 60 minutes after the beginning of the application. We evaluated sharing links in the case that the probability of a request

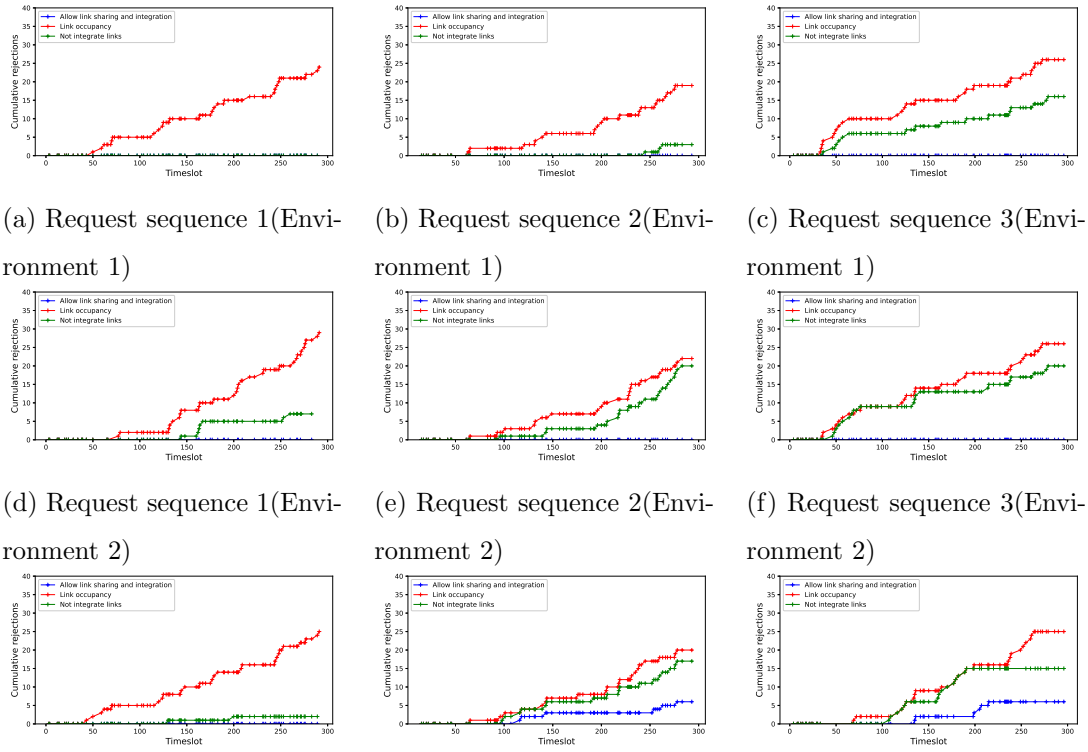
arriving per minute is 30%, where many requests arrive because we investigate how the link usage policy affects the capacity of the network.

Compared policies We compare the following policies.

- Link share and integration: In this policy, we allow multiple applications to share the same fiber core. In addition, we also allow applications to use integrated links constructed of multiple optical fiber cores.
- Link sharing without integration: In this policy, we allow multiple applications to share the same fiber core but we do not construct the integrated links.
- Exclusive: We allow only a single application to use an optical fiber core.

5.3.2 Results

Figures 9 shows the cumulative number of rejections for each environment. Table 14 shows the number of rejections from the start to the end of the simulation.



(a) Request sequence 1(Environment 1)

(b) Request sequence 2(Environment 1)

(c) Request sequence 3(Environment 1)

(d) Request sequence 1(Environment 2)

(e) Request sequence 2(Environment 2)

(f) Request sequence 3(Environment 2)

(g) Request sequence 1(Environment 3)

(h) Request sequence 2(Environment 3)

(i) Request sequence 3(Environment 3)

Figure 9: The cumulative number of rejections for each link usage policy

Table 14: The number of rejections for each link usage policy from the start to the end of the simulation

	Allowing for link sharing and integration			Prohibiting link sharing and integration			Prohibiting only link integration		
	Request1	Request2	Request3	Request1	Request2	Request3	Request1	Request2	Request3
Environment 1									
Request sequence1	0	0	0	22	1	1	0	0	0
Request sequence2	0	0	0	12	1	6	0	0	3
Request sequence3	0	0	0	0	3	23	0	0	16
Environment 2									
Request sequence1	0	0	0	27	0	2	7	0	0
Request sequence2	0	0	0	16	0	6	6	0	14
Request sequence3	0	0	0	0	4	22	1	1	18
Environment 3									
Request sequence1	0	0	0	19	2	4	0	0	2
Request sequence2	0	0	6	3	1	16	0	1	16
Request sequence3	0	0	6	0	1	24	0	0	15

The results indicate that link sharing increases the number of applications that can be allocated. That is, the effective use of the network resource is required to accommodate more applications. In addition, the results indicate that the integrated links also increase the number of accommodated applications. This is caused by the reduction of latency by integrating links. As a result, even the requests whose acceptable execution time is short can be accepted.

5.4 Impact of using multi-core fibers

In this thesis, we regard each core as an independent link. But multi-core fibers can be used as a link with large bandwidth. In this section, we investigate the impact of the usage of multi-core fibers.

5.5 Evaluation environment

μDDC network We use the same network structure as Section 5.2. But we change the number of optical fiber cores in each multi-core fiber and bandwidth of each optical fiber core as shown in Table 15. The total bandwidth of each multi-core optical fiber is the

same for all cases.

Resource request We generate the same resource requests as Section 5.3.

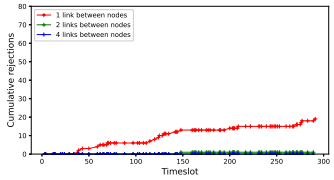
Compared environment The four cores of the multicore fiber of the evaluation network in Figure 6 are compared for the following cases:

- Environment in which all four cores can be controlled independently
- Environment in which two cores in a multicore fiber are bundled together and the number of links between each node is 2.
- Environment in which all cores are used together as a single high bandwidth link

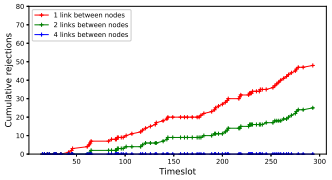
Table 15: Bandwidth and the number of links between each node in each evaluation environment

	Network with 4 links between nodes	Network with 2 links between nodes	Network with 1 link between nodes
The number of links between nodes	4	2	1
Bandwidth	50Gbps	100Gbps	200Gbps

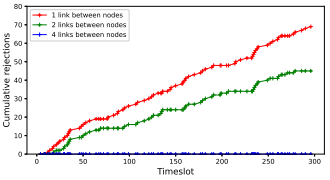
Figure 10 shows the cumulative number of rejections for each environment and Table 16 shows the number of rejections from the beginning to the end of the simulation.



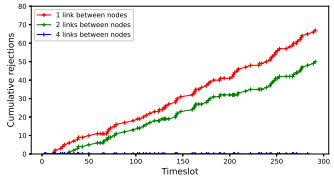
(a) Request sequence 1(Environment 1)



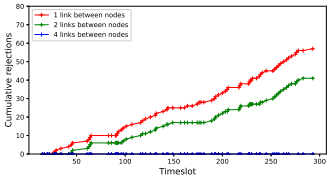
(b) Request sequence 2(Environment 1)



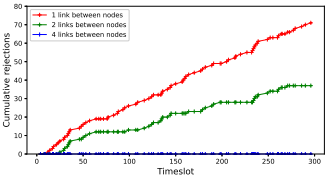
(c) Request sequence 3(Environment 1)



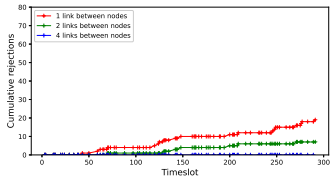
(d) Request sequence 1(Environment 2)



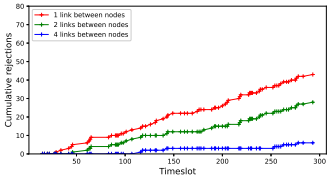
(e) Request sequence 2(Environment 2)



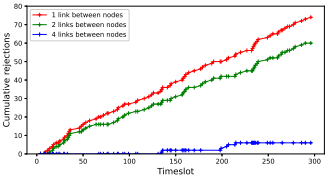
(f) Request sequence 3(Environment 2)



(g) Request sequence 1(Environment 3)



(h) Request sequence 2(Environment 3)



(i) Request sequence 3(Environment 3)

Figure 10: The cumulative number of rejections for each environment

Table 16: The number of rejections for each environment from the start to the end of the simulation

	4 links between nodes			2 links between nodes			1 links between nodes		
	Request1	Request2	Request3	Request1	Request2	Request3	Request1	Request2	Request3
Environment 1									
Request sequence1	0	0	0	0	0	1	10	2	7
Request sequence2	0	0	0	5	3	17	16	6	26
Request sequence3	0	0	0	1	0	44	1	4	64
Environment 2									
Request sequence1	0	0	0	46	0	4	62	0	5
Request sequence2	0	0	0	15	3	23	20	6	31
Request sequence3	0	0	0	1	3	46	2	3	66
Environment 3									
Request sequence1	0	0	0	0	0	7	10	2	7
Request sequence2	0	0	6	1	1	26	8	4	31
Request sequence3	0	0	6	1	1	58	0	2	72

The results indicate that a large number of optical fiber cores increases the number of accommodated applications even if the bandwidth of each fiber core is small. Even if a bandwidth of a link becomes large, if each optical fiber has only one core, all applications that use the fiber share the same link. On the other hand, if we have multiple fiber cores, an application that requires low latency communication can use a different link from the other applications that do not require low latency communication.

6 Conclusion

We proposed a resource allocation method that avoids the use of resources that may be required for the accommodation of future applications. In this method, we defined an allocation cost for each resource based on the importance of the resource and allocate resources based on the costs.

We evaluated our resource allocation method by simulation. The results showed that our method can allocate requested resources even in the case that the method that allocates resources without considering future requests cannot allocate more than 10% of requested resources. We also demonstrated that we can accommodate more applications by allowing multiple applications to use the same optical fiber core and allowing each communication to use multiple optical fibers if necessary. By allowing them, we could reduce the number of requests whose requirements cannot be satisfied by 2/3.

In our evaluation, we used a small 2-D torus network in evaluation. Evaluation of our method in a large μDDC is one of our future work. In addition, the network topology affects the performance of μDDC. Thus, we will also investigate the structure of μDDC including the network topology and the location of resources, considering the resource allocation in our future work.

Acknowledgments

We would like to thank Professor Masayuki Murata of the Graduate School of Information Science and Technology at Osaka University. He took time for me and gave me good advice and suggestions for my research. It has been a great help to me in my research. I would like to thank him again.

I would also like to express my sincere appreciation to Associate Professor Yuichi Ohsita of the Graduate School of Information Science and Technology, Osaka University. He advised me not only on my research but also on many other aspects as a researcher. I could not have completed this thesis without his advice.

I would also like to show great appreciation to Associate Professor Shin ' ichi Arakawa of Graduate School of Information Science and Assistant Professor Daichi Kominami of the Graduate School of Information Science and Technology, Osaka University and Assistant Professor Tatsuya Otoshi of the Graduate School of Economics, Osaka University. Their support in conducting my research daily was essential to me. I received a lot of support in various aspects of my research.

Finally, I would like to thank all the members of the Advanced Network Architecture Research Laboratory at the Graduate School of Information Science and Technology, Osaka University, for creating a comfortable daily research environment and for their discussions and advice.

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," *Computer Networks*, vol. 130, pp. 94–120, Jan. 2018.

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

[3] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker, "Network requirements for resource disaggregation," in *Proceedings of 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 249–264.

[4] A. Ikoma, Y. Ohsita, and M. Murata, "Impact of remote memory and network performance on execution performance of disaggregated micro data centers," in *Proceedings of 2021 International Conference on Emerging Technologies for Communications*, Dec. 2021, pp. C2–2.

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

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

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

[8] 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. [Online]. Available: http://www.osapublishing.org/optica/abstract.cfm?URI=optica-5-11-1354

[9] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient memory disaggregation with infiniswap," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, Mar. 2017, pp. 649–667.

[10] D. Sidler, Z. Wang, M. Chiosa, A. Kulkarni, and G. Alonso, "Strom: Smart remote memory," in *Proceedings of the Fifteenth European Conference on Computer Systems*, ser. EuroSys '20. New York, NY, USA: Association for Computing Machinery, 2020.

[11] S.-s. Lee, Y. Yu, Y. Tang, A. Khandelwal, L. Zhong, and A. Bhattacharjee, "Mind: In-network memory management for disaggregated data centers," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, ser. SOSP '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 488–504. [Online]. Available: https://doi.org/10.1145/3477132.3483561

[12] Y. Kwon and M. Rhu, "A disaggregated memory system for deep learning," *IEEE Micro*, vol. 39, no. 5, pp. 82–90, 2019.

[13] Y. Kwon, Y. Lee, and M. Rhu, "Tensordimm: A practical near-memory processing architecture for embeddings and tensor operations in deep learning," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 740?753. [Online]. Available: https://doi.org/10.1145/3352460.3358284

[14] Z. Zhu, G. Di Guglielmo, Q. Cheng, M. Glick, J. Kwon, H. Guan, L. P. Carloni, and K. Bergman, "Photonic switched optically connected memory: An approach to address memory challenges in deep learning," *Journal of Lightwave Technology*, vol. 38, no. 10, pp. 2815–2825, 2020.

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

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