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

Akishige Ikoma, Yuichi Ohsita, Masayuki Murata
Graduate School of Information Science and Technology, Osaka University, Osaka, Japan
Email: {a-ikoma, y-ohsita, murata}@ist.osaka-u.ac.jp

***Abstract*—In a disaggregated micro data center (μDDC), network resources have a large impact on the performance of applications. Thus, we propose a resource allocation method for μDDC that allows it to efficiently utilize network resources. In this method, we model the impact of the allocated resources on the performance of the application. Then, we allow multiple applications to share links if all of the processes are expected to be completed within the acceptable time. To accommodate more applications, if multiple candidate resources exist, this method avoids allocating those that will be requested by applications in the future based on their importance. In this paper, we evaluate our method by simulating the allocation of continually generated resource requests and demonstrate that it can accommodate more applications without affecting the required performance.**

***Index Terms*—Micro data center, Resource disaggregation, Resource allocation, Multi-core optical fiber**

I. Introduction

In recent years, many services have come to be provided through cloud computing. However, some problems come with cloud-based services, such as latency and increased network traffic. Edge computing is one technology to address these problems [1]. This approach deploys small data centers (hereafter called micro data centers [μDCs]) near users.

Because a μDC has limited resources compared with larger data centers, efficient resource utilization is important. One approach is resource disaggregation [2], [3]. In this approach, a μDC is constructed of resources connected by a network. We refer to these as disaggregated μDCs (μDDCs). Unlike a traditional DC where several resources are closely connected in the servers, a μDDC is composed of resource units, and each resource is independent. Therefore, the resources in a μDDC can be easily upgraded and flexibly used by allocating the required amount of resources to each task [4].

In a μDDC, resource allocation has a large impact on the performance of the application, so how they are allocated needs to be taken into account when trying to achieve a required level of performance. Network resources in particular have a large impact on performance [4]. Inefficient allocation of network resources may delay the retrieval of required data and increase the time required to complete a process. Thus, network resources need to be considered when allocating resources to each application.

Several resource allocation methods for a DDC have been proposed [5], [6]. Zervas et al. proposed a method that allows multiple applications to use the same link, but they considered only the bandwidths and path lengths between the allocated resources and not their impact on the performance of the application. Amaral et al. proposed an allocation method based on network capacity and task completion time. This method allocates network resources to run at higher performance while minimizing performance interference with other applications. These methods preferentially use high-performance network resources, which may lead to resource depletion.

In this paper, we propose a Resource Allocation Considering impact of Network on Performance (*RA-CNP*). We model the impact of the allocated resources on the application's performance. Then, we assume that the μDDC network is configured with packet switches and allows multiple applications to use the same link if all processes for them are expected to be completed within the acceptable time. If multiple candidate resources exist, we avoid selecting those that will be requested in the future. Thus, we define the cost for each resource and select the candidate whose cost is the smallest.

Our main contributions are as follows:

- We model the impact of the network on the application performance in a μDDC.
- We propose a resource allocation method *RA-CNP* based on our model and the allocation cost defined by the importance of resources. It can allocate resources considering future requests by preserving important resources.
- We demonstrate that *RA-CNP* can accommodate more applications to satisfy the required performance.

The rest of this paper is organized as follows. Section II gives our resource allocation method. Section III evaluate *RA-CNP* by simulating. Finally, Section IV concludes this paper.

II. Resource allocation considering Impact of Network on Performance

In this section, we describe the process for modeling a μDDC and a resource allocation request. Then, we formulate the resource allocation problem and describe how to solve it.

A. Overview of a disaggregated micro datacenter network

A μDDC network is composed of resources such as the CPU, memory, switches, and optical fibers. Each computational resource has a small cache. When the required data to execute a process don't exist in the cache and a page fault occurs, the computational resource obtains the data from the memory resource. Each switch has a buffer that can cut through switching. If the next port is available, then the switch immediately relays the packet to it before the whole packet is received. If the next port is busy, the switch stores the packet in the buffer and waits for the next port to become available.

In this paper, a μDDC network is configured using a multicore optical fiber where each optical fiber has multiple cores. This allows for more flexible routing between resources because there are multiple links between the same node. In addition, we allow the construction of an aggregated virtual link from multiple optical fiber cores. Aggregating the links can reduce the delay; even if some of the fiber cores in the aggregated link are busy, the switch can still relay the packet without storing it in the buffer so long as it has at least one fiber core available. In this paper, we regard an optical fiber core or an aggregated virtual link as a link.

B. Modeling the disaggregated micro data center network

A μDDC network is represented as a graph $G^s(N^s, E^s)$, where N^s and E^s are the sets of nodes and links. We have three types of nodes: nodes with computational resources, memory resources, and switches. In this paper, each CPU core or the GPU is treated as a single computational resource. We divide the memory into blocks and treat each block as a memory resource. C^s and M^s are the sets of available computational resources and memory resources. Also, C_n^s and M_n^s are the sets of available computational resources and memory resources on node n. In addition, for each computational resource $c \in C^s$, we defined K_c as the floating-point operations per second, F_c as the clock frequency, and V_c as the page size. We also defined two properties of the switching capability: T_n^N as the processing delay when relaying a packet to the next node by cut-through and T_n^Q as the delay when sending the whole packet to the next node. If the node n does not have switching capability, then T_n^N and T_n^Q are infinite. For each link $e \in E^s$, we define N_e^o as the number of fiber cores and T_e^P as the propagation delay. We define $R_{i,j}$ as the set of configurable paths between nodes $i, j \in N^s$ on the μDDC. $r \in R_{i,j}$ is a set of links on the path r. The bandwidth of all links is B.

C. Modeling a resource allocation request

Before running an application, the required resources are requested. We model a resource request using two graphs, one indicating the relationships between the required resources (resource graph) and the other indicating the relationships between the processes that are needed to execute the application (process graph). An example of a request is shown in Figure1.

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

A process graph is given a directed graph structure $G^p(N^p, E^p)$, where N^p and E^p are the respective sets of

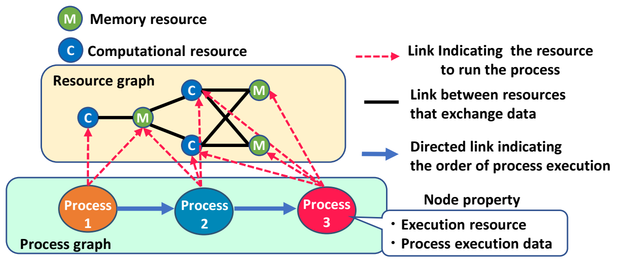


Fig. 1: An example of the resource and process graphs.

nodes and links. Each node $p \in N^p$ represents a process that is required to execute any task for the requested application. Associated with each node $p \in N^p$ are the number of page faults (σ_p^f), the number of pages transmitted per page fault (σ_p^n), and the clock counts needed to execute a process (σ_p^c). For a process corresponding to node $p \in N^p$, λ_p^r is the arrival rate of packets from the memory, and λ_p^w is the arrival rate of packets to the memory. These are obtained in advance by monitoring the application in a test environment. Each link $e \in E^p$ is a directed link indicating the order of a process. Each path from the first process to the final process gives the sequence of processes required to complete a particular task.

We define the set of application tasks as S. For the task $t \in S$, N_t^p is the set of processes required for the task t and the acceptable time T_t^a is defined. All tasks should be completed within the acceptable time. Also, we define the set of paths of the process graph in task $t \in S$ as P_t. A resource request has links indicating the resource to run the process in the process graph. c_p^v and m_p^v are the set of computational and memory resources to run the process $p \in N^p$.

D. Resource allocation problem

We first model the impact of resource allocation on the performance of the application. Then, we define allocation costs and the resource allocation problem.

1) Mapping the resources: $\delta_{i,j}^N$ denotes the mapping between the requested resources and those in the μDDC. $\delta_{i,j}^N = 1$ when the resource graph node $i \in N^v$ is mapped to the μDDC 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 μDDC. $\delta_{x,y}^E = 1$ when the resource graph link $x \in E^v$ is mapped to the path y between nodes in the μDDC and $\delta_{x,y}^E = 0$ otherwise.

3) Modeling the execution time of an application: The execution time of task $t \in S$ in a requested application is the sum of the times to complete all processes in task t. Also, the execution time for each process is the sum of the processing time in the computational resource and the communication delay to obtain the data from the memory. In this paper, we compare the worst execution time with the acceptable time to allocate resources that satisfy the requirements of the request. The worst execution time T_t^e for task $t \in S$ is

$$T_t^e = \max_{P \in P_t} \left(\sum\nolimits_{p \in P} \left(\max_{c' \in c_p^v} \max_{m' \in m_p^v} \left(T_{c',p}^c + T_{c',m',p}^d \right) \right) \right) . \tag{1}$$

where $T^c_{c',p}$ is the processing time of process $p \in N^p$ in computational resource mapped to $c' \in N^v$ and $T^d_{c',m',p}$ is the communication delay between the nodes mapped to $c', m' \in N^v$ in process p.

a) Processing time for the computational resource: The processing time $T^c_{c',p}$ for a process $p \in N^p$ in a computational resource mapped to $c' \in N^v$ is calculated by dividing the clock count σ^c_p to complete process p by the clock frequency $F_{c^s_j}$ in a computational resource $c^s_j \in C^s_j$ on node $j \in N^s$:

$$T^c_{c',p} = \sum_{j \in N^s} \left(\delta^N_{c',j} \frac{\sigma^c_p}{F_{c^s_j}} \right) \ . \tag{2}$$

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

b) Communication delay: Communication delay is the sum of the time to obtain the head of the page and the transmission delay. In process $p \in N^p$, communication delay $T^d_{c',m',p}$ to obtain the data from a memory mapped to $m' \in N^v$ to a computational resource mapped to $c' \in N^v$ is

$$T^d_{c',m',p} = \left\{ \left(\frac{\sum_{j \in N^s} \delta^N_{c',j} \cdot V_{c^s_j}}{B} \right) \sigma^n_p + T^l_{e^r_{c',m'},p} \right\} \sigma^f_p \ . \tag{3}$$

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

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

where n^{ss}_e is the source node of link $e \in E^s$ when reading data from memory. λ^s_{e,n^{ss}_e} is the current packet rate on link $e \in E^s$ occurring from node n^{ss}_e and λ^r_p is the packet rate occurring from memory to a computational resource in process $p \in N^p$. Also, $T^R_n(\lambda, J, D)$ is a function returning response time when buffering occurs in node $n \in N^s$ based on three arguments the packet rate λ, the number of fiber cores J, and processing speed D at the node. This function is based on M/D/C queuing model. However, it is difficult to obtain an accurate response time using the M/D/C queuing model. We use the approximation from [7]. $T^R_n(\lambda, J, D)$ is

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

where

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

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

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

4) Resource allocation costs: RA-CNP accommodate more applications by avoiding allocating important resources for future resource requests. Thus, we define resource allocation costs and allocate resources to minimize costs.

Computational resources that can execute applications whose acceptable processing times are small or that require large amounts of computational resources are important. Therefore, we define the cost as the product of available computational resources and FLOPS. Allocation cost W^c_c of computational resource $c \in C^s$ is

$$W^c_c = (|C^s_{Node_c}|) \cdot K_c . \tag{4}$$

where $Node_c$ is a node of the computational resource $c \in C^s$.

A memory with a large number of available memory blocks can execute applications that require a large amount of memory. Allocation cost W^m_m of memory resource $m \in M^s$ is

$$W^m_m = |M^s_{Node_m}| . \tag{5}$$

where $Node_m$ is a node with memory resource $m \in M^s$.

The link with a high potential to be paths between important resources are important. On the other hand, if a link has already been allocated as a path for any application, it cannot provide high-speed communication. Thus, the cost of an already allocated link is set to a minimal value ϵ. The allocation cost W^e_e of a link e is

$$W^e_e = \begin{cases} \sum_{c \in C^s, m \in M^s} \left(\frac{N^r_{c,m}(e)}{N^r_{c,m}} \right) \left(\frac{W^c_c \cdot W^m_m}{H_{c,m}} \right) & e \notin E^{alc} \\ \epsilon & e \in E^{alc} \end{cases} , \tag{6}$$

where E^{alc} is the set of already allocated links, $N^r_{c,m}$is the number of shortest paths between a resource c and m, and $N^r_{c,m}(e)$ is the number of shortest paths between resources c and m that pass through the link e. $H_{c,m}$ is the smallest hop count between resources c and m.

5) Defining the resource allocation problem:

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 μDDC network.

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

$$\forall x \in E^v, \forall s, t \in N^s, \quad \sum_{y \in R_{s,t}} \delta^E_{x,y} = \delta^N_{n^{vs}_x, s} \cdot \delta^N_{n^{vd}_x, t} \ . \tag{8}$$

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

Each requested resource must be allocated to one of the available resources in μDDC.

$$\forall c \in N^c, \quad |C^s_c| - \sum_{c' \in C^v} \delta^N_{c',c} \geq 0 \ . \tag{9}$$

$$\forall m \in N^m, \quad |M^s_m| - \sum_{m' \in M^v} \delta^N_{m',m} \geq 0 \ . \tag{10}$$

where N^c and N^m are the sets of nodes corresponding to the computational and memory resources in the μDDC network, respectively. C^v and M^v are the sets of nodes corresponding to the computational and memory resources in the resource graph, respectively.

b) Time constraints: All tasks in allocated applications must be executed within an acceptable time, therefore

$$\forall t \in S, \quad T^e_t \leq T^a_t \tag{11}$$

c) Objective: In this method, we allocate resources to minimize the costs of resources and links.

$$\begin{aligned} & minimize \quad \textstyle\sum_{c\in N^c}\sum_{c'\in C^v}\delta^N_{c',c}(W^c_{c^s_c})+ \\ & \textstyle\sum_{m\in N^m}\sum_{m'\in M^v}\delta^N_{m',m}(W^m_{m^s_m})+ \\ & \textstyle\sum_{i,j\in N^s}\sum_{y\in R_{i,j}}\left[1_{\sum_{x\in E^v}\delta^E_{x,y}>0}\left(\sum_{e\in y}W^e_e\right)\right], \end{aligned} \tag{12}$$

where $1_{\sum_{x\in E^v|}\delta^E_{x,y}>0}$ is 1 when $\sum_{x\in E^v}\delta^E_{x,y} > 0$ and 0 otherwise. c^s_c represents a computational resource in node $c \in N^c$, and m^s_j represents a memory resource on node $m \in N^m$.

E. Resource allocation based on ant colony optimization

The resource allocation problem defined in Section II-D5 is a nonlinear integer programming problem and NP-hard. Therefore, we solve this problem using one of the meta-heuristic methods, Ant Colony Optimization (ACO). ACO is a population-based metaheuristic where multiple agents probabilistically search for a solution and lead to the optimal solution. It can flexibly adapt to changes in the environment and is suitable for μDDC that can configure networks very flexibly. Our method is similar to the virtual network embedding method based on ACO (VNE-AC) [8], but we also use ACO for network resource allocation, unlike VNE-AC because the network significantly affects the application performance. In this method, each agent searches for resources and finds resource allocation that can finish the process within an acceptable time. Based on each agent's resource allocation, the pheromone is updated and the optimal resource is allocated. The processing steps for each agent are (1) Resource search phase, (2) Network resource search phase, and (3) Pheromone update phase. These steps are performed multiple times.

1) Resource search phase: In this phase, an agent probabilistically allocates resources corresponding to nodes in the resource graph from the available resources. We define the allocation probabilities p^c_c and p^m_m for computational resource $c \in C^s$ and memory resource $m \in M^s$ by

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

where α and β are the weight of pheromone and cost, respectively. C^{cd} and M^{cd} are the set of candidate computational and memory resources. τ_r is the pheromone of resource r.

2) Network resource search phase: In this phase, an agent searches paths between the resources selected in the resource search phase. To search them, the agent generates sub-agents. Each sub-agent probabilistically allocates paths corresponding to links in the resource graph from links in μDDC. The paths are searched starting from the source resource. First, the link from the source resource is selected. Then, the next link from the destination node of the first link is selected. This process is continued until a link to the destination resource is found. At each step of this process, the link $e \in E^s$ is selected based on the probabilities $p_{e,n}$ in node $n \in N^s$.

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

where α and β represent the relative importance of pheromone and cost, respectively. E^{cd}_n is the set of all candidate links adjacent to node n. τ_r is the pheromone of resource r.

3) Pheromone update phase: After finding the resources, the agent updates the pheromone. It is updated based on pheromone decrease rate ρ ($0 < \rho < 1$). The pheromone τ_r of any resource or link r is updated to $\rho\tau_r$. The pheromones of the resources and links of the best solution for each iteration are enhanced based on the pheromone increase rate ϕ and the resource allocation cost. Pheromone enhanced value h is

$$h = \frac{\phi}{\sum_{c\in C^b}W^c_c+\sum_{m\in M^b}W^m_m+\sum_{e\in E^b}W^e_e}$$

where C^b, M^b, and E^b are the sets of computational resources, memory resources, and links in the best solution.

III. Evaluation

We evaluate the performance of the *RA-CNP* by simulation. We set the parameters for the resource allocation to the values shown in Table I.

A. Simulation settings

1) μDDC network: We assume that the μDDC network is a 2D torus with a large number of routes between resources and flexible routing. In this paper, we use a 3×3 2D torus network as shown in Figure 2. Each CPU pool has 28 computational resources and each memory resource pool has 250 memory resources. Additionally, each optical fiber has 4 optical fiber cores. Parameter settings for the μDDC network are shown in Table II. We use these values to calculate the execution time of the application and resource allocation cost.

2) Resource request: We generate 4 types of requests. All requests have the same structure as shown in Figure 1 and each of them includes three processes; Process 1 selects the resource to execute the task, Process 2 loads the required data, and Process 3 executes the main process of the task. Considering the role of processes, we allocate the same memory resource to Processes 1 and 2 and the same computational resource

TABLE I: Parameter settings for *RA-CNP*

Parameters	Value
Number of agents and sub-agents	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

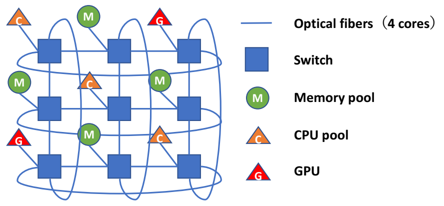


Fig. 2: A 3×3 two-dimensional torus network.

TABLE II: Parameter settings for the μDDC network

Parameters	Value
CPU FLOPS	13.619 GFLOPS
CPU clock speed per core	2.9GHz
GPU FLOPS	35.7 TFLOPS
GPU clock speed per core	1.7 GHz
Propagation delay	0.025 μs
Switch latency when buffering	3 μs
Cut-through latency	300 ns
Page size	4 KB
Bandwidth of each optical fiber core	50 Gbps

to Processes 2 and 3. Also, Processes 1, and 2 use small data and don't cause page faults. Table III shows parameter settings for the resource request. The parameters are set based on the average clock counts obtained by running the image classification application by machine learning models using *Intel(R) Xeon(R) CPU E5-2687W*. For request 1, we use ResNet [9], assuming that request 1 is related to the application that requires accurate prediction by using large machine learning models. For request 2, 3, and 4, we use YOLO [10], assuming that they are related to time-sensitive applications such as real-time identification of the objects. We generate requests at a rate of 20% per minute. The probabilities for the arriving requests are shown in Table IV. p_1, p_2, p_3, and p_4 are the probabilities that the generating request is request 1, 2, 3, and 4, respectively. We have 5 patterns for each request sequence to evaluate. The application lifetime is 60 minutes, making the total simulation time 300 minutes. Also, 3 cases determine the number of required resources:

- Case 1: Loose constraint request requires many resources
- Case 2: All requests require the same amount of resources
- Case 3: Strict constraint request requires many resources

TABLE III: Parameter settings for the resource requests

	Request 1	Request 2	Request 3	Request 4
Acceptable time	3000 ms	500 ms	150 ms	150 ms
Total required CPU (Case 1/ 2/ 3)	4/7/4	4/4/4	4/4/7	1/1/1
Total required GPU (Case 1/ 2/ 3)	0/0/0	0/0/0	0/0/0	1/1/1
Total required memory resources (Case 1/ 2/ 3)	4/7/4	4/4/4	4/4/7	3/3/3
Process 1				
Clock count	0.035	0.035	0.035	0.035
Packet rate to memory (/ms)	0.000495	0.003	0.0075	0.0075
Packet rate from memory (/ms)	0.000495	0.003	0.0075	0.0075
Required CPU (Case 1/ 2/ 3)	1/1/1	1/1/1	1/1/1	1/1/1
Required memory resources (Case 1/ 2/ 3)	1/1/1	1/1/1	1/1/1	1/1/1
Process 2				
Clock count	0.054	0.054	0.054	0.054
Packet rate to memory (/ms)	0	0	0	0
Packet rate from memory (/ms)	0.000495	0.000495	0.0075	0.0075
Required CPU or GPU (Case 1/ 2/ 3)	3/6/3	3/3/3	3/3/6	1/1/1 (GPU)
Required memory resources (Case 1/ 2/ 3)	1/1/1	1/1/1	1/1/1	1/1/1
Process 3				
Clock count	2371.33	1960.36	1960.36	1960.36
Packet rate to memory (/ms)	2.805	2.85	2.85	2.85
Packet rate from memory (/ms)	5.565	5.145	5.145	5.145
The number of page faults	67543.25	56661.29	56661.29	56661.29
The number of pages per page fault	5.27	4.84	4.84	4.84
Required CPU or GPU (Case 1/ 2/ 3)	3/6/3	3/3/3	3/3/6	1/1/1 (GPU)
Required memory resources (Case 1/ 2/ 3)	3/6/3	3/3/3	3/3/6	2/2/2

TABLE IV: Parameter settings for generating requests

Request sequence	p_1	p_2	p_3	p_4
Request sequence 1	0.75	0.05	0.05	0.15
Request sequence 2	0.35	0.15	0.35	0.15
Request sequence 3	0.05	0.05	0.75	0.15

B. Method comparison

We obtain the results in this section by using ACO, which is similar to our proposed method.

a) Resource allocation using the shortest path (SP): This method allocates resources based on the shortest path between resources. To do this, the link cost is defined by $W_e^e = 1$. This method is very simple and we evaluate whether a simple routing is sufficient for a μDDC resource allocation.

b) Resource allocation by considering network performance (NP): This method allocates paths based on whether there is a low traffic volume and short path lengths between the computational and memory resources. This method allocates resources focussing on performance and is similar to policies in [5], [6]. The cost of link $e \in E^s$ with node $n \in N^s$ as the source is defined by

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

where λ^{max} is the maximum traffic volume, $D_{i,j}$ is the shortest path length from node $i \in N^v$ to node $j \in N^v$, and D^{max} is the network diameter.

c) Resource allocation without link aggregation (w/o A): This method allows multiple applications to share the same link but doesn't aggregate links. We show the effectiveness of link aggregation by comparison with *w/o A*.

d) Resource allocation without link sharing (w/o S): In this method, each optical fiber core is limited to being used by a single application. We show the effectiveness of link sharing by comparison with *w/o S*.

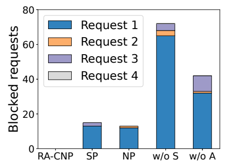
C. Metrics

We compare the number of applications whose requirements are satisfied. We use a simple scenario wherein requests whose corresponding resources cannot complete the required processes within the acceptable time are blocked, though we can accommodate them if the resulting performance degradation is acceptable. Then, we compare the number of blocked requests.

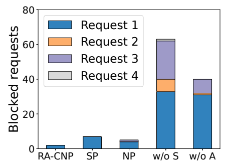
D. Results

Figure 3 shows the blocked requests for each case. The results shown in the graph are the sum of the number of blocks in the five patterns of request sequences.

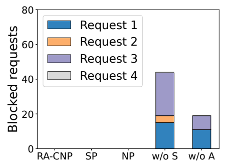
In *RA-CNP*, blocking occurred only in environments with a high arrival rate of requests with a high amount of required resources (Figure 3b, 3h, 3i). On the other hand, blocking occurred in SP and NP even in environments where blocking does not occur in *RA-CNP*. In particular, SP and NP have more than twice as much blocking compared to *RA-CNP* even when blocking occurs in *RA-CNP*. More blocking means that fewer applications can run simultaneously. This difference is due to the resources that are available when requests required



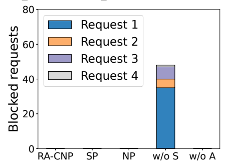
(a) Request sequence 1 (Case 1)



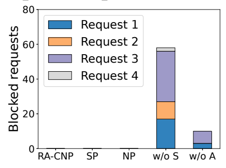
(b) Request sequence 2 (Case 1)



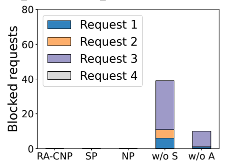
(c) Request sequence 3 (Case 1)



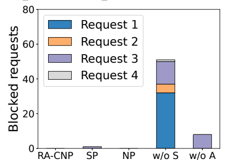
(d) Request sequence 1 (Case 2)



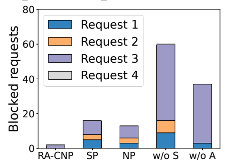
(e) Request sequence 2 (Case 2)



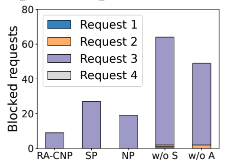
(f) Request sequence 3 (Case 2)



(g) Request sequence 1 (Case 3)



(h) Request sequence 2 (Case 3)



(i) Request sequence 3 (Case 3)

Fig. 3: Comparison of blocked requests for each case and request sequence

many resources arrive. NP allocates resources in a way that prioritizes achieving the best network performance. Also, SP does not consider performance and future requests. As a result, there are not enough paths with acceptably small execution times for requests, which requires many of them. On the other hand, because our method avoids using the resources that will be required by future requests, it can allocate more requests.

The results also indicate that more applications can be accommodated by link sharing because so many rejections occur in *w/o S*. By sharing links, we preserve the links that do not communicate frequently. In addition, the results indicate that link aggregation also increases the number of allocated resources. This is caused by the reduction in latency that results from aggregating links. As a result, even requests whose acceptable execution time is short can be accepted.

IV. Conclusion

We proposed a resource allocation method for a μDDC that achieved an efficient usage of network resources. In this method, we modeled the impact of the allocated resources on the time required to complete the processes involved in an application. Then, we allowed multiple applications to share links if all of the processes involved with them were expected to be completed within the acceptable time. By avoiding the use of important resources, this method preserves them so that they can be used by future applications, and more applications can be accommodated. In this paper, we evaluated *RA-CNP* by simulating the allocation of continually generated resource requests and demonstrated that it can accommodate more applications without affecting the required performance compared to the traditional resource allocation policy.

In future work, we plan to further investigate the structure of a μDDC, including the network topology and the location of resources, and to consider the resource allocation within it.

Acknowledgments

This work was partially supported by the National Institute of Information and Communications Technology (NICT).

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] 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.

[4] 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.

[5] 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.

[6] 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,” *Journal of Cloud Computing*, vol. 10, pp. 1–20, mar 2021.

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

[8] 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.

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

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