Dynamic Resource Allocation Considering Workload Changes in a Disaggregated Data Center

Akishige Ikoma
Graduate School of Information Science and Technology
Osaka University, Suita, Japan
a-ikoma@ist.osaka-u.ac.jp

Yuichi Ohsita
D3 Center
Osaka University, Toyonaka, Japan
yuichi.ohsita.cmc@osaka-u.ac.jp

Masayuki Murata
Graduate School of Information Science and Technology
Osaka University, Suita, Japan
murata@ist.osaka-u.ac.jp

***Abstract*—Disaggregated data centers (DDC) enable efficient resource utilization. However, a DDC faces performance degradation due to resource communication latency. Furthermore, workload changes in DDC affect the execution time of service tasks. Therefore, to continually satisfy the performance requirements for service tasks, a dynamic resource management method is required that considers workload changes and the impact of the network on performance. We propose a dynamic resource allocation method considering the workload changes in a DDC (*DRA-CWC*). We formulate the impact of workload changes on performance and define resource allocation costs based on service demand and resource importance for future service execution. Based on these, we preserve the required resources to satisfy the performance requirements for the future workload increase. By comparing conventional resource allocation methods for a DDC, we demonstrate that *DRA-CWC* can satisfy service performance requirements for a longer time.**

***Index Terms*—Disaggregated data center, resource allocation**

I. Introduction

In recent years, cloud computing has problem of high communication delays. Edge computing can be the solution to this problem. In edge computing, small-scale data centers are located at the edge. However, such data centers have fewer resources such as central processing units (CPUs) than cloud data centers. Therefore, efficient resource usage is required.

One approach for efficient resource usage is resource disaggregation [1]. In resource disaggregation, each resource aggregated into servers is decoupled and connected via a network. Because each resource is independent, resources can be flexibly used by allocating only the required resources to each task. This improves resource utilization [2]. We configure edge data centers applying resource disaggregation (hereafter referred to as a disaggregated data center (DDC)).

Communication between execution resources occurs whenever data are exchanged, which is handled by the motherboard in traditional architectures. Because the frequency of such communication is high, communication delays have a significant impact on service performance [3]. Communication between allocated resources must be sufficiently low latency to satisfy performance requirements. Therefore, resource allocation for a DDC is required.

Several resource allocation methods for a DDC have been proposed [2], [4], [5]. However, these methods aim to satisfy the performance requirements of the requested tasks in the resource usage situation at that time. If the demand for services increases, the task execution requests increase and the workload on the computing resources also increases. If the number of allocated resources is small relative to service demand, the performance will be degraded by waiting time for task execution. A dynamic resource allocation method that considers workload changes is required.

In this paper, we propose a dynamic resource allocation method considering workload changes in a DDC (hereafter called *DRA-CWC*). *DRA-CWC* aims to continually satisfy the performance requirements for service tasks in a DDC where workloads change dynamically. *DRA-CWC* predicts the demand for each execution service. Based on this prediction, we estimate the communication latency between allocated resources and the task execution waiting time in the future to determine additional resource allocation and reallocation of paths between resources. Furthermore,

This work was supported by JSPS KAKENHI Grant Number JP24K14914

we formulate the potential for additional allocation for each service and the potential as additional allocated resources for each resource. Based on this, we preserve the required resources to satisfy the performance requirements for the future workload increase. By comparing conventional resource allocation methods for a DDC, we demonstrate *DRA-CWC* can satisfy service performance requirements for longer periods.

II. Disaggregated data center

A. Overview of a disaggregated data center

A DDC consists of computational and memory resources interconnected by a network. Computational resources, which include CPUs and GPUs, are designated for task execution and have a small cache. Memory resources, which include RAM, are responsible for storing processed data. Data in memory are managed via paging. When the data required for tasks do not exist in the cache and a page fault occurs, computational resources obtain the data from the memory resource via a network. A DDC network employs packet switches. A packet switch aggregates multiple same-type resources and the set of resources connected to the same switch is called a resource pool.

For dynamic resource management, the following three processes run:

- Monitor: Measure arrival rate of execution requests for service tasks as service demand, and traffic on each network link at constant intervals. Then, send monitoring information to the Predictor and Allocator.
- Predictor: Predict future service execution requests based on information from the Monitor. Then, send prediction results to Allocator.
- Allocator: Determine whether to allocate execution resources and reallocate paths for each service based on prediction results. Then, determine execution resources and routes between resources.

The decision to allocate resources is based on the waiting time for execution requests and the communication delay threshold between execution resources. If the threshold is exceeded, additional resources are allocated to satisfy service performance requirements. To set the threshold, the allocator has a standard value for service execution request arrival rate and traffic and sets the threshold based on that. The notations for the information held by each process are listed in Table I.

TABLE I: The information held by each process

Symbols	Definition
	Allocator holding information
N^s, E^s	Set of nodes and links in a DDC network
L_n	Number of available resources in the resource pool corresponding to node $n \in N^s$
C^s, M^s	Set of available computational and memory resources
F_c	Clock frequency of computational resource $c \in N^s$
T_n^H	Delay to send the packet to the next node in node $n \in N^s$
T_e^P	Propagation delay in link $e \in E^s$
R^s	Set of paths between resources in the DDC network
B	Network bandwidth
P	page size
$\lambda_{e,n}$	Traffic from adjacent node n on network link $e \in E^s$
S	Set of executing services
T_s^t	Acceptable time of service $s \in S$
N_s^v, E_s^v	Set of nodes and links in the resource graph of service $s \in S$
δ_n^R	DDC resource corresponding to resource graph node $n \in N^v$
δ_n^P	Paths in the DDC network corresponding to resource graph link $e \in E^v$
C_s^v, M_s^v	Set of resource graph nodes corresponding to the execution computational and memory resources for service $s \in S$
N_s^μ, E_s^μ	Set of nodes and links in the μ-service graph of service $s \in S^e$
R_μ^v	Set of resource graph paths corresponding to μ-service $\mu \in N^\mu$
Λ_μ^c	Number of clocks required to execute μ-service $\mu \in N^\mu$
Λ_μ^f	Number of page faults that occur during the execution of μ-service $\mu \in N^\mu$
Λ_μ^p	Number of pages read during execution of μ-service $\mu \in N^\mu$
I_μ, O_μ	Size of input and output data for μ-service μ
λ^P	Standard value of traffic
λ^R	Standard value of arrival rate of service execution request
	Monitor holding information
$U_{\mu,t}$	Arrival rate of execution requests for μ-service $\mu \in N^\mu$ at time t
$\lambda_{\mu,t}^w, \lambda_{\mu,t}^r$	Number of packets from/to memory during the execution of μ-service $\mu \in N^\mu$ at time t
	Predictor holding information
$\hat{U}_{s,t+\Delta t}$	Arrival rate of service execution request in service $s \in S$ from t to Δt
T^{dur}	prediction period

B. Execution service

We assume an event-driven service consisting of multiple μ-services. Each μ-service in a service sends an execution request and processing results to the next μ-service after the end of processing. Requests for each μ-service are queued and wait until a pair of computational and memory resources to execute the μ-service is available. An example of the service is shown in Fig. 1.

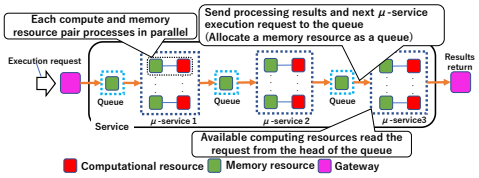


Fig. 1: Example of the assumed service

C. Modeling of execution service

We model an execution service using two directed graphs, where one indicates the relationships between the allocated resources (resource graph) and the other indicates the relationships between the μ-services (μ-service graph). An example of service modeling is shown in Fig. 2. These graphs are configured when deploying a new service and are updated when resources are additionally allocated. This graph is held in the allocator.

Resource graph of the service $s \in S$ is represented by $G_s^v(N_s^v, E_s^v)$. For each node $n \in N_s^v$, we define

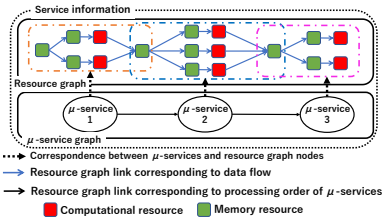


Fig. 2: Example of resource and μ-service graph

the correspondence with allocated resource $\delta_n^R \in N^s$. The directed link $e \in E_s^v$ represents the relationship between communicating resources. In the resource graph, the path from the start to end represents the flow of data during the execution of the service. For each link $e \in E_s^v$, we define the correspondence with allocated path $\delta_e^P \subset E^s$.

μ-service graph of the service $s \in S$ is represented by $G_s^\mu(N_s^\mu, E_s^\mu)$. Each node $\mu \in N_s^\mu$ corresponds to a μ-service comprising the service, and the nodes are connected in the order of their execution. For each node $\mu \in N_s^\mu$, we define the number of page faults Λ_μ^f, the number of pages transmitted per page fault Λ_μ^p, the clock counts required to execute the μ-service Λ_μ^c, traffic read from and written to memory during execution $\lambda_\mu^r, \lambda_\mu^w$, input data size I_μ and output data size O_μ. Furthermore, we define the set of paths R_μ^v in the resource graph from the resource that receives the processing request for the μ-service μ to the resource that forwards the processing result as the correspondence between resource graph and μ-service graph.

III. Dynamic resource allocation considering workload changes

First, we formulate the impact of allocated resources and workload change on execution time. Next, we define an optimization problem to determine the allocated resources to preserve the required resources to satisfy the performance requirements for the future workload increase.

A. The impact of allocated resources and workloads change on execution time

The execution time of μ-service μ is the sum of the communication delay of the path corresponding to the link and the processing time of the computational resource on the resource graph path $y \in R_\mu^v$. Links on the path y are classified into the following three links: link y_1 for transferring input data to memory resources, link y_2 used for communication between execution resources, and link y_3 for transferring results to the next μ-service. Therefore, the execution time of μ-service μ is the sum of the latency $T^O(y_1, I_\mu)$ to transfer input data I_μ on the path corresponding to y_1 and processing time $T^E(\mu, y_2)$ on computational and memory resources using the path corresponding to link y_2, latency $T^O(y_3, O_\mu)$ to transfer output data I_μ on the path corresponding to y_3. Furthermore, the transfer time $T^O(e_s^{start}, I_{\mu^{start}})$ of a request to the μ-service queue is incurred as a delay before execution. The execution time $T(s)$ of the service s is obtained as follows:

$$T(s) = T^O(e^{start}, I_{\mu^{start}}) + \sum_{\mu \in N_s^\mu} \max_{y \in R_\mu^v} \{T^O(y_1, I_\mu) + T^E(\mu, y_2) + T^O(y_3, O_\mu)\} \quad (1)$$

where e_s^{start} is the link whose source node is the starting point of the resource graph, and μ^{start} is the first executed μ-service.

a) Data transfer latency: The transfer time for the data of size D is the sum of the time required to obtain the head of the data $\frac{D}{B}$ and the transmission delay. Note that we assume that the transfer delay is a latency threshold $\mathbb{L}_{e'}$ to allocate resources to satisfy performance requirements even when the communication delay between resources is at the threshold. That is,

$$T^O(e', D) = \frac{D}{B} + \mathbb{L}_{e'} \ . \quad (2)$$

The latency threshold $\mathbb{L}_{e'}$ on the path between resources corresponding to the resource graph e' is the sum of the communication delay $T^L(e, \lambda^P, n_e^s)$ when the traffic on a link on a path corresponding to the resource graph e' is the standard value λ^P. That is,

$$\mathbb{L}_{e'} = \sum_{e \in \delta_{e'}^P} T^L(e, \lambda^P, n_e^s) \ . \quad (3)$$

The latency $T^L(e, \lambda, n)$ of transferring data from node n on a DDC network link $e \in E^s$ with traffic λ is the sum of the propagation delay T_e^p of link e and the processing delay $T^R(\ lambda, J, T_n^H)$ of the switch to which link e connects. That is,

$$T^L(e, \lambda, n) = T_e^p + T^R(\lambda, J, T_n^H) \quad (4)$$

where J is the number of links used for transferring data. $T^R(\lambda, J, T_n^H)$ is based on the M/D/C queuing model. In the M/D/C queuing model, we assume buffering as a situation where packets arriving according to the Poisson process are waiting to be processed until one of the J links that can process them in a fixed time T_n^H is ready to forward them.

However, obtaining an accurate response time using the M/D/C queuing model is difficult. We use the approximation [6]. $T^R(\lambda, J, T_n^H)$ is obtained as follows:

$$T^R(\lambda, J, D) = D + \frac{\left\{1+f^Q(\lambda,J,D)g^Q(\lambda,J,D)\right\}h^Q(\lambda,J,D)}{2} \tag{5}$$

where,

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

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

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

b) Processing time of a μ-service: Processing time of a μ-service is the sum of the communication delay $\Lambda_\mu^f \cdot T^O(e', P\cdot\Lambda_\mu^p)$ of the path between the execution resources and the execution time $\frac{\Lambda_\mu^c}{F_c}$ in computational resource, the waiting time threshold $\mathbb{W}_\mu$ for μ-service μ. The execution time in the computational resource of the μ-service μ is derived by dividing the clock counts Λ_μ^c required for execution by the clock frequency F_c of the computational resource c. Communication delay is derived by multiplying the data transfer time $T_e^O(P\cdot\Lambda_\mu^p)$ per page fault on the path corresponding to link e' by the number of page faults Λ_μ^f. $\mathbb{W}_\mu$ are used to allocate resources to satisfy performance requirements even when the waiting time for μ-service is at the threshold. The processing time $T^E(\mu, e')$ of μ-service μ in the computational and memory resource pair corresponding to the adjacent node of the resource graph link $e' \in E^v$ is obtained as follows:

$$T^E(\mu, e') = \frac{\Lambda_\mu^c}{F_{\delta^R_{n^s_{e'}}}} + \Lambda_\mu^f \cdot T^O(e', P\cdot\Lambda_\mu^p) + \mathbb{W}_\mu \tag{6}$$

$\mathbb{W}_\mu$ is the waiting time $T^Q(\lambda^R, C_\mu, T_\mu^A)$ when the request arrival rate is the standard value λ^R, the number of computational resources for the μ-service μ is C_μ and the average processing time in each computational resource is T_μ^A. That is,

$$\mathbb{W}_\mu = T^Q(\lambda^R, C_\mu, T_\mu^A) \tag{7}$$

$T^Q(\lambda^R, C_\mu, T_\mu^A)$ is derived by M/M/C queuing model. $T^Q(\lambda^R, C_\mu, T_\mu^A)$ is obtained as follows:

$$T^Q(\lambda, J, D) = \left(\frac{J}{D}-\lambda\right)^{-1}\left(1+\left(1-\frac{\lambda D}{J}\right)\left(\frac{J!}{(\lambda D)^J}\right)\sum_{k=0}^{J-1}\frac{(\lambda D)^k}{k!}\right)^{-1} \tag{8}$$

B. Allocation cost

We define the allocation costs for computational and memory resources and network links based on the importance for future service execution. First, based on future service demand, we define the potential for additional resource allocation for each service. Next, we define the potential to be an allocated resource for each resource and network link. Finally, based on them, we define the allocation cost.

1) Potential for additional allocation of execution resources: The greater the predicted demand relative to the service's execution computational resources, the more likely it is that additional execution resources are allocated. Therefore, the potential for additional allocation of execution resources $\eta_{s,t}$ for service s at the time t is obtained as follows:

$$\eta_{s,t} = \frac{\max\limits_{0\le\Delta t\le T^{dur}} \widehat{U}_{s,t+\Delta t}}{|C_s^v|} \tag{9}$$

2) Potential to be future service execution resources:

a) Computational resource: The additionally allocated compute resources communicate with the memory resources corresponding to the queues. To satisfy performance requirements, computational resources must be able to communicate to memory resources with low latency. Computational resources close to the executing memory resource are important. Computational resources with high clock frequency and many available resources in the corresponding pool are also important because they can handle tasks with various requirements. Therefore, the potential to be future service execution resources $\gamma_{s,c}$ for computational resource $c \in N^s$ is obtained as follows:

$$\gamma_{s,c} = \frac{1}{\sum_{a\in M_s^v} H(\delta_a^R, c)} \cdot F_c \cdot L_c \tag{10}$$

b) Memory resource: The additionally allocated memory resources communicate with a memory resource corresponding to the queue or newly additionally allocated computational resource. Memory resources close to the executing memory resource and available computational resources are important. Furthermore, memory resources with many available resources in the corresponding pool are also important. Therefore, the potential to be future service

execution resources $\zeta_{s,m}$ for memory resource $m \in N^s$ is obtained as follows:

$$\zeta_{s,m} = \left\{ \frac{1}{\sum_{a \in M_s^v} H(m, \delta_a^R)} + \frac{1}{\sum_{b \in C^s} H(m,b)} \right\} \cdot L_m \tag{11}$$

3) Network link: In communication between resources, the smaller the hop count and traffic volume, the smaller the delay and the less likely to cause congestion. Therefore, the lower-traffic network links on the shortest paths between resources are important. We define the possibility to be a network link on the shortest path as the ratio of the number of shortest paths between resources to the number of shortest paths through that network link. The possibility $\theta(a,b,e)$ that a link $e \in E^s$ is on the shortest path between resource $a, b \in N^s$ is obtained as follows:,

$$\theta(a,b,e) = \frac{\phi(a,b|e)}{\phi(a,b) \cdot H(a,b)} \tag{12}$$

where $\phi(a,b)$ denotes the number of shortest paths between resources $a, b \in N^s$, and $\phi(a,b|e)$ denotes the number of shortest paths between resources $a, b \in N^s$ passing through network link e. $H(a,b)$ is the shortest hops between resources $a, b \in N^s$. Resource pairs that possibly communicate when allocating additional resources are (1) executing memory resources and available resources, (2) available computational and memory resources and (3) executing resources. Let R^p be the set of these resource pairs, the potential to be a network link on the path for future service $\kappa_{s,e}$ is obtained as follows:

$$\kappa_{s,e} = \frac{\sum_{a,b \in R^p} \theta(a,b,e)}{\lambda_{n_e^s,e} + \lambda_{n_e^d,e}} \tag{13}$$

4) Defining the allocation cost: We avoid allocating resources with high potential to be resources for services for which demand will increase in the future. Therefore, the allocation costs $\mathcal{C}_{c,t}$, $\mathcal{M}_{m,t}$, and $\mathcal{E}_{e,t}$ of the computational resource $c \in N^s$, memory resource $m \in N^s$, and network link $e \in E^s$ at time t are obtained as follows:

$$\begin{aligned} \mathcal{C}_{c,t} &= \textstyle\sum_{s \in S} \{\eta_{s,t} \cdot \gamma_{s,c}\} \\ \mathcal{M}_{m,t} &= \textstyle\sum_{s \in S} \{\eta_{s,t} \cdot \zeta_{s,m}\} \\ \mathcal{E}_{e,t} &= \textstyle\sum_{s \in S} \{\eta_{s,t} \cdot \kappa_{s,e}\} \end{aligned} \tag{14}$$

C. Resource allocation problem

We define a resource allocation problem to preserve the required resources for future workload increases while satisfying the performance requirements.

a) Allocated resource constraints: One computational resource C_s^v, memory resource M_s^v, and edge E_s^v in the resource graph of service s correspond to a resource.

$$\begin{aligned} &\forall s \in S, \forall c \in C_s^v \quad \delta_c^R \in C^s \\ &\forall s \in S, \forall m \in M_s^v \quad \delta_m^R \in M^s \\ &\forall s \in S, \forall e \in E_s^v \quad \delta_e^P \in R^s_{\delta^R_{n_e^s}, \delta^R_{n_e^d}} \end{aligned} \tag{15}$$

b) Latency threshold: The latency of the path between resources corresponding to the link $e' \in E_s^v$ in the resource graph of the service $s \in S$ must be less than or equal to the threshold $\mathbb{L}_{e'}$.

$$\forall s \in S, \forall e' \in E_s^v \quad \textstyle\sum_{e \in \delta^P_{e'}} T^L(e, \lambda_{n_e^s,e}, n_e^s) \leq \mathbb{L}_{e'} \tag{16}$$

c) Waiting time threshold: The waiting time of each μ-service of the executing service s must be less than or equal to the threshold.

$$\forall s \in S, \forall \mu \in N_s^\mu \quad T^Q(U_{\mu,t}, C_\mu, T_\mu^A) \leq \mathbb{W}_\mu \tag{17}$$

d) Performance requirement: All services must be executed within the acceptable time.

$$\forall s \in S \quad T(s) \leq T_s^t \tag{18}$$

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

$$minimize \quad \sum_{c \in C_s^v} \mathcal{C}_{\delta_c^R,t} + \sum_{m \in M_s^v} \mathcal{M}_{\delta_m^R,t} + \textstyle\sum_{e' \in E_s^v} \sum_{e \in \delta^P_{e'}} \mathcal{E}_{e,t} \tag{19}$$

This problem is a nonlinear integer programming problem and NP-hard. To solve such problems, the method using ant colony optimization is already proposed [5]. In this paper, we derive this problem using the same approach. However, any method that can find a solution can be used.

IV. Evaluation

By simulating a DDC where workloads change dynamically, we demonstrate that DRA-CWC can continually satisfy service performance requirements for longer periods.

A. Environment

1) Network: Each resource is alternately deployed to form a 3x3 2D torus network. Each packet switch is connected to a computational or memory resource pool, which holds 24 and 20 resources, respectively. The clock frequency of each computational resource is 3.4 GHz. Each switch pair is connected by three network links and the propagation delay per link is 0.025 μs. The bandwidth is 50 Gbps.

2) Execution service: We assume that the following three types of services are provided: (1)Service 1: Semantic segmentation [7], (2) Service 2: 3D Object Detection [8] and Service 3: Combination of semantic segmentation [7] and bird's eye view perception [9]. All of these are inference processes using the AI model. Each service task is composed of three types of μ-services: (1) pre-processing of input data, (2) inference for input data, and (3)post-processing of processing results. The parameter settings for each service are shown in Table II.

TABLE II: Parameter settings for each service

	Semantic segmentation	3D Object Detection	bird's eye view perception
Acceptable time(ms)	100	125	220
Arrival rate standard value	0.0067	0.0033	0.0034
Traffic standard value	150	90	100
μ-service 1			
Clock count	68199102	49289698	181852
Traffic to/from memory(/ms)	2.58/4.38	1.74/0.0	0.72/2.61
Number of page faults	1977.88	144.56	31.26
Number of pages per page fault	2.78	2.58	2.36
μ-service 2			
Clock count	33095402	52219370	147762478
Traffic to/from memory(/ms)	3.4/3.99	0.77/5.43	8.36/10.36
Number of page faults	1112.48	6832.98	7532.65
Number of pages per page fault	1.4	1.86	2.75
μ-service 3			
Clock count	79239	59155594	399211
Traffic to/from memory(/ms)	1.39/0.0	0.81/0.0	3.72/3.13
Number of page faults	66.56	244.36	699.43
Number of pages per page fault	2.09	1.31	3.82

3) Shifting demand for each service: To simulate dynamic service demand changes, we generate service execution requests based on a dataset showing vehicle positions per second in Cologne, Germany for 24 hours [10]. We assume a DDC provides service within 1km range. Thus, the city of Cologne is divided into a grid every 2km, and the vehicles within the grid are considered as service users. We generate requests for as many vehicles as there are in the range. We evaluate in two ranges: the highest number of vehicles (High workload) and the number of vehicles as much as the overall average number (Average workload). The evaluation is based on the number of vehicles during a 15-hour period from 5:00 a.m. to 8:00 p.m. Note that the service used by each vehicle is determined at random.

4) Demand prediction for services: In this paper, We constructed a prediction model using the Temporal Fusion Transformer model. Based on this, we predict future demand for 60 seconds based on changes in demand over the past 120 seconds The mean absolute errors for the high workload and average workload are 6.97 and 2.97, respectively.

B. Comparative methods

We compared *DRA-CWC* with two resource allocation methods that do not consider workload changes. Note that the determination of resource allocation due to load changes is the same as for *DRA-CWC*.

a) Resource allocation considering the impact of the network on the performance (RA-CNP): RA-CNP avoids the allocation of high-performance resources and low-latency paths as much as possible while satisfying the performance requirements and corresponds to the resource allocation policy proposed by Ikoma et al. [5].

b) Resource allocation by considering network performance (NP): NP allocates paths with low traffic and short lengths between computational and memory resources. NP corresponds to the resource allocation policy proposed by Zervas et al. [2] and Amaral et al. [4].

C. Metrics

In this evaluation, we measure the time that the performance requirements for each service could not be satisfied. The higher this value, the lower the response to workload changes.

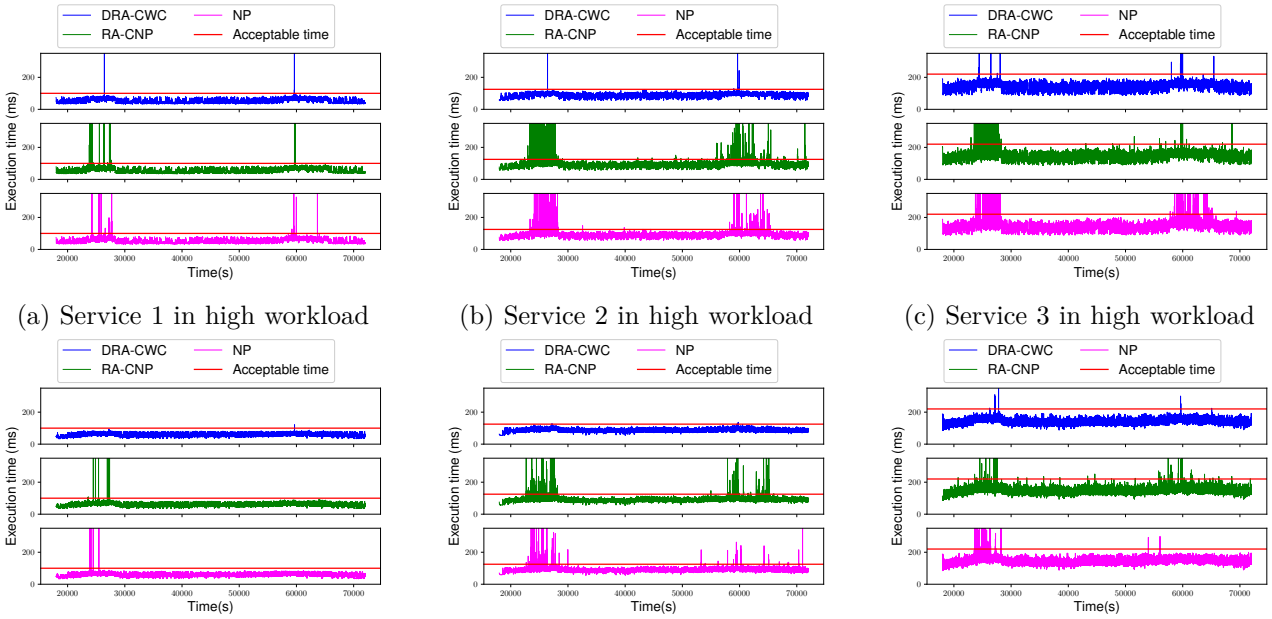
D. Result

The execution delay per second for each service is shown in Fig. 3. The red line in the figure is the acceptable time for service. If it is above the red line, the service does not satisfy the performance requirements.

The results show that, regardless of the environment or service, the *DRA-CWC* can satisfy the performance requirements of services for more time than the comparative methods. This means *DRA-CWC* enables resource management to respond more flexibly to workload changes than conventional methods. Furthermore, even in high workload, the period during which the acceptable time for service could not be satisfied in *DRA-CWC* was only 51 seconds out of the 15 hours evaluated. On the other hand, *RA-CNP* and *NP* were 2105 and 1311 seconds, respectively. Resource allocation by DRA-CWC is very effective.

V. Conclusion

We proposed a dynamic resource allocation method considering the workload changes in a DDC (*DRA-CWC*). DRA-CWC predicts the demand for each execution service. Based on this prediction, we formulated the impact of workload changes on performance. Furthermore, we formulated the potential for additional allocation for each service and the potential as an additional resource for each resource. Based on



(a) Service 1 in high workload (b) Service 2 in high workload (c) Service 3 in high workload

(d) Service 1 in average workload (e) Service 2 in average workload (f) Service 3 in average workload

Fig. 3: Service execution delay per second for each environment and service.

these, *DRA-CWC* preserve the required resources to satisfy the performance requirements for the future workload increase while satisfying the performance requirements. By comparing conventional resource allocation methods for a DDC, we demonstrated *DRA-CWC* can satisfy service performance requirements for a longer time.

We plan to consider communications between various types of resources. In this paper, we assumed services composed of multiple μ-services. In fact, there are various forms of service execution. It is necessary to generalize about the formulation of execution performance and resource allocation costs, regardless of the execution resource type.

References

[1] 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*, ser. HotNets-XII. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: https://doi.org/10.1145/2535771.2535778

[2] 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.

[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 *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 249–264. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/gao

[4] M. Amaral, J. Polo, D. Carrera, N. Gonzalez, C.-C. Yang, A. Morari, B. D'Amora, A. Youssef, and M. Steinder, "Dr-maestro: orchestrating disaggregated resources on virtualized data-centers," *Journal of Cloud Computing*, vol. 10, pp. 1–20, mar 2021.

[5] A. Ikoma, Y. Ohsita, and M. Murata, "Resource allocation considering impact of network on performance in a disaggregated data center," *IEEE Access*, vol. 12, pp. 67 600–67 618, 2024.

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

[7] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, "Segformer: Simple and efficient design for semantic segmentation with transformers," in *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2021.

[8] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors*, 2018.

[9] Z. Li, W. Wang, H. Li, E. Xie, C. Sima, T. Lu, Y. Qiao, and J. Dai, "Bevformer: Learning birdfs-eye-view representation from multi-camera images via spatiotemporal transformers," *arXiv preprint arXiv:2203.17270*, 2022.

[10] S. Uppoor, O. Trullols-Cruces, M. Fiore, and J. M. Barcelo-Ordinas, "Generation and analysis of a large-scale urban vehicular mobility dataset," *IEEE Transactions on Mobile Computing*, vol. 13, no. 5, pp. 1061–1075, 2014.