Algorithm for Server Allocation in Delay-Sensitive Internet-of-Things Monitoring Services

Shoya Imanaka[†], Akio Kawabata[‡], Bijoy Chand Chatterjee[§], and Eiji Oki[†]
†Kyoto University, Kyoto, Japan ‡Toyohashi University of Technology, Aichi, Japan
§South Asian University, Delhi, India

***Abstract*—This paper proposes a polynomial-time algorithm for a server allocation problem in delay-sensitive Internet-of-Things (IoT) monitoring services. The server allocation problem determines the appropriate servers to which the database and application are allocated to minimize the maximum delay between the latest update of reference data and the start of application processing for monitoring data. The proposed algorithm comprises two components. The initial phase involves the selection of usage servers for both the database and the application. Subsequently, the second phase entails matching each usage server and its corresponding IoT device. The proposed algorithm obtains an optimal allocation solution in polynomial time. Numerical results show that the proposed algorithm obtains the optimal solution faster than an integer linear programming approach.**

***Index Terms*—server allocation problem, Internet of things, monitoring service, polynomial-time algorithm**

I. Introduction

The Internet of Things (IoT) has recently become widespread in various areas and fields [1]. A typical example of an IoT-applied service is a monitoring service. The monitoring service employs multi-access edge computing (MEC) technology, with telecommunication providers, in charge of network design, deploying a group of MEC servers in a distributed rather than a centralized manner to shorten the overall transmission distance between the IoT device and the server.

A monitoring service uses two types of data (reference data and monitoring data) to determine the next actions to be taken. Reference data is association data that characterizes and categorizes specific data and is usually stored as a database and utilized in application processing. Monitoring data is collected by sensors and used in application analysis and processing. For example, camera sensors are used to detect particular people and vehicles for smart cities [2], or detect some problems when monitoring the real-time status of factory production lines for smart factories [3]. In monitoring systems where decisions should be based on the latest data, such as cases where detecting criminals by surveillance cameras or automated driving usage situations, the application needs to process the latest reference data as there is a change in data.

When the IoT-based service is provided over a large network, the reference data is usually located at the center of the network, applications are situated at the network edges [4], [5], and IoT-sensor devices are widely distributed as network terminal endpoints. The data is exchanged between database (reference data) and application, and between the IoT device and application. The application must receive reference data and monitoring data to quickly detect and process the latest data and accelerate application processing. Application processing can be accelerated by improving the server's computing performance and resources. On the other hand, to reduce transmission delays, it is necessary to shorten the transmission distance between the database server storing reference data and the application server, and between the IoT device and the application server. The magnitude of data transmission delays is directly proportional to the extent of network coverage.

In the operation of the monitoring service, the latest monitoring data needs to be analyzed with low latency based on the latest reference data. For example, monitoring services for criminal detection needs to quickly and accurately analyze facial image data (monitoring data) observed by security camera sensors based on the latest criminal database (reference data). Reference data is sent to the application when it is updated, and an IoT device located at a terminal endpoint (TE) sends monitoring data to the application (APL) server on a regular basis. Even if the transmission delay between a TE and an APL server is so small, when the delay between the database (DB) server and an APL server is extremely large, the monitoring service might not be able to accurately and properly analyze the new monitoring data based on the latest reference data. In this case, application processing of the monitoring data is rewound when the updated reference data is received at APL after receiving monitoring data and restarted so that the analysis is based on the latest reference data. In the opposite case, the monitoring service will not be able to quickly analyze the latest monitoring data. Let T_{te} denote the transmission delay between a TE and an APL server, and T_{db} denote the transmission delay between the DB server and an APL server. For a given network configuration, there are some transmission paths for reference data from the DB server to multiple APL servers, and for each APL server, there are transmission paths for monitoring data from multiple devices to the APL server. For all data transmission delays, let $T_{\mathrm{te}}^{\max}$ and $T_{\mathrm{db}}^{\max}$ denote the maximum values of T_{te} and T_{db}, respectively. To reduce the overall data transmission delay in monitoring service and to analyze any latest monitoring data quickly and properly, $\max(T_{\mathrm{te}}^{\max}, T_{\mathrm{db}}^{\max})$ needs to be reduced [6], [7].

For reducing $\max(T_{\mathrm{te}}^{\max}, T_{\mathrm{db}}^{\max})$, it is necessary to appropri-

This work was supported in part by the Japan Society for the Promotion of Science (JSPS) KAKENHI, Japan, under Grant Numbers 21H03426 and 23H03382.

979-8-3503-7099-7/24/$31.00 ©2024 IEEE

ately select the server to which DB and APL are allocated and the server to which TE connects, TE is allocated, taking into account the data transmission path. In other words, a server allocation of DB, APL, and TE on a network appropriate for delay-sensitive monitoring service is required.

Kawabata *et al.* in [6], [7] formulated the server allocation problem as an integer linear programming (ILP) problem to minimize $\max(T_{\text{te}}^{\max}, T_{\text{db}}^{\max})$. They solved the ILP problem using an ILP solver to find each corresponding server to which DB and APL are allocated and TE is connected. However, the work using the ILP approach did not analyze the computational time complexity required to solve the server allocation problem for monitoring service. It is desirable to provide more efficient algorithm to solve the server allocation problem for a service provider instead of solving the ILP problem, and to clarify the computational time complexity.

A question arises: *is there any efficient algorithm to solve the server allocation problem for DB, APL, and TE in the monitoring service, instead of solving an ILP problem?*

This paper proposes a polynomial-time algorithm to solve the server allocation problem for DB, APL, and TE in the delay-sensitive monitoring service. The proposed algorithm comprises two components. The initial phase involves the selection of usage servers for both the database and the application. Subsequently, the second phase entails matching each usage server and its corresponding IoT device. The proposed algorithm obtains an optimal solution for the server allocation problem in polynomial time. Numerical results show that the proposed algorithm outperforms the ILP approach in terms of computation time under various experimental conditions.

II. Problem Description

A. Network model and notations

The network is represented as an undirected graph $G(V, E)$, where V and E are, respectively, the set of nodes and undirected edges. $V_{\text{T}} \subset V$ denotes the set of TEs, and $V_{\text{S}} \subset V$ denotes the set of servers, where $V = V_{\text{T}} \cup V_{\text{S}}$ and $V_{\text{T}} \cap V_{\text{S}} = \emptyset$. $E_{\text{T}} \subset E$ denotes the set of edges between each pair of a TE and a server, and $E_{\text{S}} \subset E$ denotes the set of edges between each pair of servers, where $E = E_{\text{T}} \cup E_{\text{S}}$ and $E_{\text{T}} \cap E_{\text{S}} = \emptyset$. $(t, i) \in E_{\text{T}}$ denotes an edge between TE $t \in V_{\text{T}}$ and server $i \in V_{\text{S}}$. d_{ti} denotes the delay of edge $(t, i) \in E_{\text{T}}$. $(i, j) \in E_{\text{S}}$ expresses an edge between server $i \in V_{\text{S}}$ and server $j \in V_{\text{S}} \setminus \{i\}$. d_{ij} expresses the delay of edge $(i, j) \in E_{\text{S}}$. The maximum number of TEs that server $i \in V_{\text{S}}$ can accommodate is denoted by M_i. We define decision variables x_{kl}, y_i, and z_i. x_{kl} is a binary variable that is for $(k, l) \in E$, where $x_{kl} = 1$ if edge $(k, l) \in E$ is selected, and $x_{kl} = 0$ otherwise. y_i is a binary variable that is for server $i \in V_{\text{S}}$, where $y_i = 1$ if server $i \in V_{\text{S}}$ is selected and APL is allocated to it, and $y_i = 0$ otherwise. z_i is a binary variable that is for server $i \in V_{\text{S}}$, where $z_i = 1$ if DB is allocated to server $i \in V_{\text{S}}$, and $z_i = 0$ otherwise. The maximum delay of an edge between TE $t \in V_{\text{T}}$ and server $i \in V_{\text{S}}$ which accommodates TE is indicated by $T_{\text{te}}^{\max}$, and the maximum delay of an edge between server $i \in V_{\text{S}}$ to which APL is allocated and server j to which DB is allocated is indicated by $T_{\text{db}}^{\max}$. The maximum value of $T_{\text{te}}^{\max}$ and $T_{\text{db}}^{\max}$ is denoted as T_{delay}; $T_{\text{delay}} = \max(T_{\text{te}}^{\max}, T_{\text{db}}^{\max})$.

B. Examples

This section presents examples of server allocation for DB, APL, and TE that are addressed in this study. Fig. 1 shows an example of network. Fig. 2(a) and Fig. 2(b) show network setup configuration based on the network in Fig. 1. In Fig. 2(a), TEs 1 and 2 are connected to server 1, TE 3 is connected to server 3, APL is allocated to servers 1 and 3, and DB is allocated to server 2. In this configuration scenario, individual TE is connected to the nearest servers from each. In this configuration, $T_{\text{te}}^{\max}$ is 10, $T_{\text{db}}^{\max}$ is 13, so T_{delay} is 13. On the other hand, in Fig. 2(b), TE 1 is connected to server 2, TEs 2 and 3 are connected to server 3, APL is allocated to servers 2 and 3, and DB is allocated to server 2. In this configuration setting, the individual TE is not selecting the nearest server from itself, but is selecting the server to be connected so as to minimize T_{delay}. In this configuration, $T_{\text{te}}^{\max}$ is 12, $T_{\text{db}}^{\max}$ is 6, and thus T_{delay} is 12, less than Fig. 2(a) case. These network configuration examples show that each IoT device's selection of the nearest server from itself is not necessarily the best strategy for reducing the delay T_{delay} for data analysis. In this paper, we assume the network configuration model shown in Fig. 2(b).

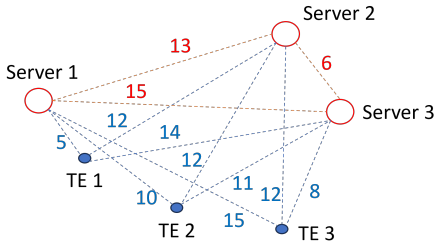
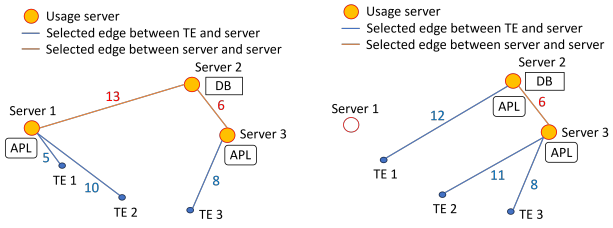


Fig. 1. Example of network. A number attached to each edge indicates the edge delay.



(a) Example 1. (b) Example 2.

Fig. 2. Example of network configurations. A number attached to each edge indicates the edge delay.

C. Problem formulation

We formulate the optimization problem to minimize T_{delay} as an integer linear programming (ILP) problem as follows:

$$\text{Objective} \quad \min \quad T_{\text{delay}} \tag{1a}$$

$$\text{s.t.} \quad \sum_{i \in V_{\text{S}}} x_{ti} = 1, \ \forall t \in V_{\text{T}} \tag{1b}$$

$$\sum_{t \in V_{\text{T}}} x_{ti} \leq M_i, \ \forall i \in V_{\text{S}} \tag{1c}$$

$$\sum_{j \in V_S} z_j = 1 \tag{1d}$$

$$y_i \geq x_{ti},\ \forall i \in V_S, (t,i) \in E_T \tag{1e}$$

$$d_{ti} x_{ti} \leq T_{te}^{max},\ \forall (t,i) \in E_T \tag{1f}$$

$$d_{ij} x_{ij} \leq T_{db}^{max},\ \forall (i,j) \in E_S \tag{1g}$$

$$T_{te}^{max} \leq T_{delay},\ \forall (t,i) \in E_T \tag{1h}$$

$$T_{db}^{max} \leq T_{delay},\ \forall (i,j) \in E_S \tag{1i}$$

$$x_{ij} \geq z_j + y_i - 1,\ \forall (i,j) \in E_S \tag{1j}$$

$$x_{ij} \leq y_i,\ \forall (i,j) \in E_S \tag{1k}$$

$$x_{ij} \leq z_j,\ \forall (i,j) \in E_S \tag{1l}$$

$$x_{kl} \in \{0,1\},\ \forall (k,l) \in E \tag{1m}$$

$$y_i, z_i \in \{0,1\},\ \forall i \in V_S. \tag{1n}$$

Equation (1a) expresses the objective function to minimize T_{delay}. Equation (1b) represents that there is one server to which each TE $t \in V_T$ is connected. Equation (1c) represents that the maximum number of TEs that can be connected to server $i \in V_S$ is at most M_i. Equation (1d) represents that the number of servers to which DB is allocated is one. Equation (1e) indicates that APL is allocated to a server, if at least one TE is connected to the server. Equation (1f) and (1g) represent that T_{te}^{max} and T_{db}^{max} are the maximum delay between TE-server edges and between server-server edges, respectively. Equations (1h)-(1i) represent that T_{delay} is the greater value of T_{te}^{max} and T_{db}^{max}. Equations (1j)-(1l) indicate that $y_i \cdot z_j = x_{ij}, \forall (i,j) \in E_S$. Equations (1m)-(1n) express that x_{kl}, y_i, and z_i are 0-1 binary variables.

III. Proposed server allocation algorithm

A. Algorithm description

We present an algorithm for the server allocation problem. The proposed algorithm is inspired by a basic idea of the server-user matching algorithm presented in [8], [9]. The proposed algorithm aims to minimize the value $T_{delay} = \max(T_{te}^{max}, T_{db}^{max})$. It consists of two phases: usage server selection and TE-sever matching, where each TE is connected to one server to which APL is allocated. We set $T_{delay} = \infty$ as an initial setting (line 1).

First, we start the usage-server-selection phase for DB and APL. We select the server j to which DB is allocated (line 2). We sort server $k \in V_S$ in non-decreasing order of $d_{jk}, (j,k) \in E_S \cup \{(j,j)\}$, where $d_{jj} = 0$; we create set $\{usage\}$ as an empty set (line 3). By adding a server to $\{usage\}$ one by one based on the sorting result, we repeatedly update $\{usage\} \subseteq V_S$ (line 4). T_{db}^{usage} denotes the maximum delay of edges between DB-allocated server j and server $i \in \{usage\}$. If T_{db}^{usage} is larger than or equal to T_{delay}, we select another server $j \in V_S$ to which DB is allocated (lines 5–7). If $\sum_{i \in \{usage\}} M_i$ is less than $|V_T|$, we check another set of usage servers with larger T_{db}^{usage} from server j (lines 8–10).

Next, we move to the TE-server matching phase. We firstly define set V_S' with a set of M_i copies of server $i \in V_S$ including the original one, and $|V_S'|$ is equal to $\sum_{i \in V_S} M_i$. E_T' is the set of edges between TE $t \in V_T$ and server $i_m \in V_S'$, where $i \in V_S, m \in [1, M_i]$. Delay d_{ti_m}, $(t,i_m) \in E_T'$, is the same value as the original d_{ti}, $(t,i) \in E_T$. We initialize the bipartite graph $G'(V', E_T'')$, where $V' = V_T \cup V_S'$ and $E_T'' = \{\emptyset\}$, respectively (line 11). For each set of usage servers, we sort $(t,i) \in E_T : i \in \{usage\}$ in non-decreasing order of d_{ti} (line 12). We introduce counter r, which is initialized as zero (line 13). In addition, we introduce a flag, $flag\_match$, which is initialized as false (line 13). This flag indicates whether the first matching for all TEs has already been found for a set of usage servers. If $d_{ti_m} \geq T_{delay}$, (t,i_m) is the edge to be added, we check another set of usage servers (lines 15–17). We set D_T to the delay of the first edge in the list sorted in non-decreasing order of d_{ti} (line 18). We add the top-listed edge, which has the smallest delay in the list, to E_T'' and remove it from the list and increment r if E_T'' does not include any edge connecting TE $t \in V_T$ (line 19). We also add the edges that have the same delay value as D_T to E_T'' and remove them from the list (line 20). We stop examining a set of usage servers if a matching for all TEs has already been found for the set of usage servers or if the delay of an edge we try to add $d_{ti_m} \geq T_{delay}$. In line 20, each time a new edge (t,i_m) is added to E_T'', increment r if E_T'' does not include any edge connecting t. If r is equal to $|V_T|$ (line 21), E_T'' includes any edges connecting every $t \in V_T$; we run the Hopcroft-Karp algorithm [10], which obtains the maximum matching in a given bipartite graph (line 22). We investigate whether all TEs can be matched to servers using the Hopcroft-Karp algorithm in the context of $G'(V', E_T'')$ (line 22–23). If the number of matching pairs between TEs and servers is equal to $|V_T|$ (line 23), we set $flag\_match$ = true, calculate T_{te}^{usage} for $\{usage\}$, and get x_{ti}, y_i, z_j (lines 24–25). T_{te}^{usage} denotes the maximum delay of edges between TE $t \in V_T$ and server $i \in \{usage\}$ established by TE-server matching. We then calculate $T_{delay}^{usage} = \max(T_{te}^{usage}, T_{db}^{usage})$ (line 26). If T_{delay}^{usage} is smaller than the previous T_{delay}, we update T_{delay} with that T_{delay}^{usage} and allocation x_{ti}, y_i, z_j (lines 27–29). If $flag\_match$ = true, the first matching for all TEs has already been found for a set of usage servers, we do not investigate the case any further and investigate the next set of usage servers (lines 32–34). If we have checked all $\{usage\}$ for all DB-allocated server $j \in V_S$, we finish the proposed algorithm.

B. Computational time complexity

We analyze the computational time complexity of the proposed algorithm considering the usage server selection phase and the TE-server matching phase.

First, we consider the usage servers selection phase. We select and investigate each set of usage servers for each server to which DB is allocated. It takes $|V_S|$ times to select each DB-allocated server. For each case that DB is allocated to server j, the number of servers to be sorted (line 3) is $|V_S|$. The computational time complexity of sorting servers in non-decreasing order of $d_{jk}, (j,k) \in E_S \cup \{(j,j)\}$ for case of server j is $O(|V_S| \log |V_S|)$; for sorting servers (lines 2 and 3), the total computational time complexity is $O(|V_S|^2 \log |V_S|)$. It takes at most $|V_S|$ times to select and investigate all the sets of usage servers for each DB-allocated server. The number of sets of usage servers is at most $(|V_S|)(|V_S|)$. Thus, it takes $O(|V_S|^2)$ to select and investigate all the sets of usage servers.

Second, we consider the TE-server matching phase. We make M_i copies of servers $i \in V_S$ to initialize $G(V', E')$.

Algorithm 1: Proposed algorithm

Input: $G(V, E)$, d_{ti}, $(t, i) \in E_{\mathrm{T}}$, d_{ij}, $(i, j) \in E_{\mathrm{S}}$, M_i, $i \in V_{\mathrm{S}}$
Output: T_{delay}, x_{ti}, y_i, z_j, $t \in V_{\mathrm{T}}$, $i, j \in V_{\mathrm{S}}$
1: $T_{\mathrm{delay}} = \infty$
Usage server selection (lines 2–10)
2: **for** $j \in V_{\mathrm{S}}$ to which DB is allocated
3: Sort server $k \in V_{\mathrm{S}}$ in non-decreasing order of d_{jk}, $(j, k) \in E_{\mathrm{S}} \cup \{(j, j)\}$, where $d_{jj} = 0$; create set $\{usage\}$ as an empty set
4: **for** each set of usage servers selected, $\{usage\}$, updated based on the sorting result
5: **If** $T_{\mathrm{db}}^{\mathrm{usage}} \geq T_{\mathrm{delay}}$
6: **break**
7: **end if**
8: **If** $\sum_{i \in \{usage\}} M_i < |V_{\mathrm{T}}|$
9: **continue**
10: **end if**
TE-server matching (lines 11–35)
11: Initialize the bipartite graph $G'(V', E_{\mathrm{T}}'')$.
12: Sort $(t, i) \in E_{\mathrm{T}} : i \in \{usage\}$ in non-decreasing order of d_{ti}
13: $r = 0$ and $flag\_match$ = false
14: **for** each edge on the top of list
15: **If** $d_{ti_m} \geq T_{\mathrm{delay}}$
16: **break**
17: **end if**
18: Set D_{T} to the delay of top-listed edge
19: Add the edge (t, i_1) to E_{T}'' and remove it from the list. Increment r if E_{T}'' does not include any edge connecting t.
20: Add the edges that have the same delay as D_{T} to E_{T}'' and remove them from the list. Each time a new edge is added to E_{T}'', increment r if E_{T}'' does not include any edge connecting t.
21: **If** r is equal to $|V_{\mathrm{T}}|$
22: Run the Hopcroft-Karp algorithm
23: **If** the number of matching pairs is equal to $|V_{\mathrm{T}}|$
24: $flag\_match$ = true
25: Calculate $T_{\mathrm{te}}^{\mathrm{usage}}$ and get allocation x_{ti}, y_i, z_j
26: Calculate $T_{\mathrm{delay}}^{\mathrm{usage}} = \max(T_{\mathrm{te}}^{\mathrm{usage}}, T_{\mathrm{db}}^{\mathrm{usage}})$
27: **If** $T_{\mathrm{delay}}^{\mathrm{usage}}$ is smaller than the previous value
28: Update T_{delay} and allocation x_{ti}, y_i, z_j
29: **end if**
30: **end if**
31: **end if**
32: **If** $flag\_match$ = true
33: **break**
34: **end if**
35: **end for**
36: **end for**
37: **end for**

The computational time complexity of making server copies is $O(|V_{\mathrm{S}}'|)$; for making server copies (lines 2, 4, and 11), the total computational time complexity is $O(|V_{\mathrm{S}}|^2|V_{\mathrm{S}}'|)$. The number of edges $(t, i), i \in \{usage\}$ is at most $|V_{\mathrm{T}}||V_{\mathrm{S}}|$. The computational time complexity of sorting $(t, i) \in E_{\mathrm{T}} : i \in \{usage\}$ in non-decreasing order of d_{ti} is $O(|V_{\mathrm{T}}||V_{\mathrm{S}}| \log |V_{\mathrm{T}}||V_{\mathrm{S}}|)$; for sorting edges (lines 2, 4, and 12), the total computational time complexity is $O(|V_{\mathrm{S}}|^2|V_{\mathrm{T}}||V_{\mathrm{S}}| \log |V_{\mathrm{T}}||V_{\mathrm{S}}|)$. After that, we add the edges to $G'(V', E_{\mathrm{T}}'')$. We consider $G'(V', E_{\mathrm{T}}'')$, which has TE and server sides. For adding edges to E_{T}'' (lines 2, 4, and 19–20), the total computational time complexity is $O(|V_{\mathrm{S}}|^2|V_{\mathrm{T}}||V_{\mathrm{S}}'|)$. When we get to a situation where there is at least one edge out of every TE on $G'(V', E_{\mathrm{T}}'')$, we run the Hopcroft-Karp algorithm. In each case using the Hopcroft-Karp algorithm, the maximum number of vertices in $G'(V', E_{\mathrm{T}}'')$ is $|V'| = |V_{\mathrm{T}}| + |V_{\mathrm{S}}'| \leq 2|V_{\mathrm{S}}'|$; it takes $O(|E_{\mathrm{T}}''|\sqrt{|V_{\mathrm{S}}'|})$. $|E_{\mathrm{T}}''|$ is at most $|V_{\mathrm{S}}'||V_{\mathrm{T}}|$; it takes $O(|V_{\mathrm{S}}'||V_{\mathrm{T}}|\sqrt{|V_{\mathrm{S}}'|})$ in the Hopcroft-Karp algorithm. We use the Hopcroft-Karp algorithm only when $r = |V_{\mathrm{T}}|$ for each set of usage servers; for the Hopcroft-Karp algorithm (lines 2, 4, and 22), the total computational time complexity is $O(|V_{\mathrm{S}}|^2|V_{\mathrm{S}}'||V_{\mathrm{T}}|\sqrt{|V_{\mathrm{S}}'|})$. The comparison between new and previous values takes $O(1)$. With $|V_{\mathrm{S}}'| = |\sum_{i \in V_{\mathrm{S}}} M_i|$, the total computational time complexity for the Hopcroft-Karp algorithm is represented as $O(|V_{\mathrm{S}}|^2|V_{\mathrm{T}}|(\sum_{i \in V_{\mathrm{S}}} M_i)^{1.5})$.

We compare $O(|V_{\mathrm{S}}|^2|V_{\mathrm{T}}|(\sum_{i \in V_{\mathrm{S}}} M_i)^{1.5})$ and $O(|V_{\mathrm{S}}|^2|V_{\mathrm{T}}||V_{\mathrm{S}}| \log |V_{\mathrm{T}}||V_{\mathrm{S}}|)$. We investigate a ratio of $(\sum_{i \in V_{\mathrm{S}}} M_i)^{1.5}/(|V_{\mathrm{S}}| \log |V_{\mathrm{T}}||V_{\mathrm{S}}|)$. Given that $\sum_{i \in V_{\mathrm{S}}} M_i$ is equal to $|V_{\mathrm{S}}|M_i$ and M_i is expressed as $\gamma|V_{\mathrm{T}}|$, where γ is a positive parameter, the ratio is represented as $(\sqrt{|V_{\mathrm{S}}||V_{\mathrm{T}}||V_{\mathrm{T}}|\gamma^{1.5}})/(\log |V_{\mathrm{T}}||V_{\mathrm{S}}|)$; $O(|V_{\mathrm{S}}|^2|V_{\mathrm{T}}|(\sum_{i \in V_{\mathrm{S}}} M_i)^{1.5})$ is larger in terms of computational time complexity.

Therefore, the overall computational time complexity of the proposed algorithm is $O(|V_{\mathrm{S}}|^2|V_{\mathrm{T}}|(\sum_{i \in V_{\mathrm{S}}} M_i)^{1.5})$. Thus, the proposed algorithm is a polynomial-time algorithm.

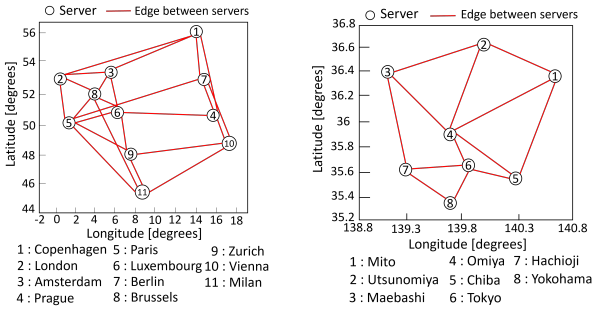
Property 1. *The proposed algorithm obtains an optimal server allocation for DB, APL, and TE which minimizes* $T_{\mathrm{delay}} = \max(T_{\mathrm{te}}^{\max}, T_{\mathrm{db}}^{\max})$ *with the computational time complexity of* $O(|V_S|^2|V_T|(\sum_{i \in V_S} M_i)^{1.5})$.

IV. Numerical Results

We evaluate the proposed algorithm and ILP approach with two networks: COST 239 [11], JPN Kanto region [12] networks. We assume that the distributed servers are located according to each network model, as shown in Fig. 3(a), Fig. 3(b). There are 11 servers in COST239, eight servers in JPN Kanto region. In each network, each pair of servers can logically communicate with each other by using one or more edges on the shortest path route. For example, London is connected to Berlin via Paris in COST 239. TEs are uniformly distributed with respect to latitude and longitude on the area corresponding to a given network. For server-server and TE-server distances, spherical trigonometry is used to calculate the distance between two points on earth based on latitude and longitude. The distance of d_{AB} between point A (longitude x_A, latitude y_A) and point B (longitude x_B, latitude y_B) is expressed by: $d_{AB} = R\cos^{-1}(\sin y_A \sin y_B + \cos y_A \cos y_B \cos(x_B - x_A))$, where R denotes the equatorial radius. The transmission delay is mainly proportional to the corresponding edge distance.

We compare the proposed algorithm with the ILP approach of (1a)–(1n); the ILP problem is solved by Solving Constraint Integer Programs (SCIP) [13], which is a non-commercial solver. The proposed algorithm is coded by using the C++ language. The computer is configured with Intel(R) Core(TM) i7-1360P 2.20GHz 16GB memory. We note that both optimal values of the objective function obtained by the proposed algorithm and the ILP are identical, as proved by Property 1.

Tables I and II show the computation times by the proposed algorithm and ILP approach using the combined patterns of



(a) Location of servers in COST 239.

(b) Location of servers in JPN Kanto.

Fig. 3. Locations of servers in COST 239 and JPN Kanto.

number of TEs and a ratio, $\alpha = |V_T|/(\sum_{i \in V_S} M_i)$, on COST 239, JPN Kanto region, respectively. M_i is a given parameter, and α is initially set up in the network design phase. We assume that $M_i, \forall i \in V_S$, is the same value. For every TE to be connected to any of the servers in a given network, $\alpha \leq 1$ must be satisfied.

TABLE I
AVERAGE COMPUTATION TIMES [S] BY ILP APPROACH AND PROPOSED ALGORITHM IN COST 239 NETWORK.

Ratio	Number of TEs							
	100		300		500		1000	
(α)	ILP	Prop.	ILP	Prop.	ILP	Prop.	ILP	Prop.
0.91	3.98	0.010	58.9	0.045	75.3	0.11	1526.7	0.41
0.70	2.70	0.010	7.96	0.054	63.1	0.14	360.3	0.50
0.45	1.96	0.028	4.31	0.24	4.78	0.64	30.9	2.79
0.30	2.22	0.53	2.72	0.42	11.0	1.13	42.7	4.93

TABLE II
AVERAGE COMPUTATION TIMES [S] BY ILP APPROACH AND PROPOSED ALGORITHM IN JPN KANTO REGION NETWORK.

Ratio	Number of TEs							
	100		300		500		1000	
(α)	ILP	Prop.	ILP	Prop.	ILP	Prop.	ILP	Prop.
0.83	0.95	0.008	2.46	0.032	5.72	0.083	17.07	0.34
0.70	0.53	0.012	1.61	0.047	4.47	0.14	18.07	0.52
0.45	0.34	0.016	1.34	0.12	3.77	0.33	9.54	1.45
0.30	0.31	0.029	1.20	0.21	2.58	0.60	5.33	2.65

Tables I and II show that the proposed algorithm solves the server allocation problem faster than the ILP approach under all experimental conditions. Specifically, for COST 239 when $|V_T| = 1000$ with $\alpha = 0.91$, the proposed algorithm solves the server allocation problem around 3724 times faster than the ILP approach, and with $\alpha = 0.70$, 721 times faster. For JPN Kanto region when $|V_T| = 1000$ and $\alpha = 0.83$, the proposed algorithm solves the problem 50 times faster than the ILP approach, and with $\alpha = 0.70$, 35 times faster. For both networks, when $|V_T|$ and α are the same, it takes more computation time in most cases to solve the server allocation problem in COST 239 than in JPN Kanto region. This is related to the fact that there are more servers in COST 239 than in JPN Kanto region.

We observe from Table I and II that the computation time of the proposed algorithm increases as α decreases. This is because the smaller α is (the larger server capacity $M_i = |V_T|/\alpha|V_S|, \forall i \in V_S$, is), the greater the number of cases where $\sum_{i \in \{usage\}} M_i < |V_T|$ (line 8) is false; we investigate the more sets of usage servers in the proposed algorithm. As for the specific relationship between values of α and M_i, in COST 239, when $\alpha = 0.91$, $M_i = |V_T|/10$; when $\alpha = 0.45$, $M_i = |V_T|/5$. Given $|\{usage\}|$ denotes the number of usage servers to be examined, $\alpha|V_S| \leq |\{usage\}| \leq |V_S|$ holds so that servers in $\{usage\}$ can accommodate all TEs. This means when α becomes small (M_i becomes relatively large), we might investigate more sets of usage servers. In addition, the increase in M_i leads to the overall increase in the number of edges added to the bipartite graph $G'(V', E''_T)$, which results in increase in computation time.

V. CONCLUSION

This paper proposes an algorithm to solve the server allocation problem for the database, application, and IoT devices in delay-sensitive monitoring services. The proposed algorithm comprises two components. The initial phase involves the selection of usage servers for both the database and the application. Subsequently, the second phase entails matching each usage server and its corresponding IoT device. The proposed algorithm obtains an optimal server allocation for the database, application, and IoT devices in polynomial time. Numerical results show that the proposed algorithm solves the server allocation problem for the database, application, and IoT devices faster than ILP.

REFERENCES

[1] P. D. Baruah, S. Dhir, and M. Hooda, "Impact of IoT in current era," in *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, 2019, pp. 334–339.
[2] H. Rajab and T. Cinkelr, "IoT based smart cities," in *2018 Int. Symp. Netw., Comput. Commun. (ISNCC)*, 2018, pp. 1–4.
[3] M. Soori, B. Arezoo, and R. Dastres, "Internet of things for smart factories in industry 4.0, a review," *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 192–204, 2023.
[4] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the internet of things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.
[5] S. Singh, "Optimize cloud computations using edge computing," in *2017 International Conference on Big Data, IoT and Data Science (BID)*. IEEE, 2017, pp. 49–53.
[6] A. Kawabata, T. Tojo, B. Chatterjee, and E. Oki, "A network design scheme in delay sensitive monitoring services," *IEICE Trans. Commun.*, vol. E106-B, no. 10, pp. 903–914, Oct. 2023.
[7] A. Kawabata, T. Tojo, B. C. Chatterjee, and E. Oki, "An optimal allocation scheme of database and applications for delay sensitive IoT services," in *2021 IEEE Global Commun. Conf. (GLOBECOM)*, 2021, pp. 1–6.
[8] T. Sawa, F. He, A. Kawabata, and E. Oki, "Polynomial-time algorithm for distributed server allocation problem," in *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*, 2019, pp. 1–3.
[9] ——, "Algorithms for distributed server allocation problem," *IEICE Trans. Commun.*, vol. E103-B, no. 11, pp. 1341–1352, Nov. 2020.
[10] J. Katrenic and G. Semanišin, "A generalization of Hopcroft-Karp algorithm for semi-matchings and covers in bipartite graphs," *Computing Research Repository - CORR*, 03 2011.
[11] M. O'Mahony, "Results from the COST 239 project. ultra-high capacity optical transmission networks," in *Proc. European Conf. Opt. Commun.*, vol. 2, 1996, pp. 11–18 vol.2.
[12] "Japan Photonic Network Model," https://www.ieice.org/cs/pn/jpn/jpnm.html, accessed on 2023-08-20.
[13] "Solving Constraint Integer Programs (SCIP)," https://www.scipopt.org/, accessed on 2023-08-20.