PAPER *Special Section on Network Virtualization and Network Softwarization for Diverse 5G Services*

Program File Placement Problem for Machine-to-Machine Service Network Platform

Takehiro SATO†a), ***Member and*** **Eiji OKI**†, ***Fellow***

SUMMARY The Machine-to-Machine (M2M) service network platform accommodates M2M communications traffic efficiently by using tree-structured networks and the computation resources deployed on network nodes. In the M2M service network platform, program files required for controlling devices are placed on network nodes, which have different amounts of computation resources according to their position in the hierarchy. The program files must be dynamically repositioned in response to service quality requests from each device, such as computation power, link bandwidth, and latency. This paper proposes a Program File Placement (PFP) method for the M2M service network platform. First, the PFP problem is formulated in the Mixed-Integer Linear Programming (MILP) approach. We prove that the decision version of the PFP problem is NP-complete. Next, we present heuristic algorithms that attain sub-optimal but attractive solutions. Evaluations show that the heuristic algorithm based on the number of devices that share a program file reduces the total number of placed program files compared to the algorithm that moves program files based on their position.

key words: *IoT, M2M, placement problem, optimization, NP-completeness, heuristic algorithm*

1. Introduction

As symbolized by the buzz phrase “Internet of Things (IoT),” Internet-based technologies that can connect and control every device in our lives, such as smartphones, home appliances, industrial machines, and vehicles, are becoming popular. The number of IoT devices was estimated to be 8.4 billion in 2017 with an increase to 20.4 billion by 2020 [1]. The popularity of IoT services, in which devices and computers communicate, calculate, and make decisions without human intervention, means there will be a rapid growth in Machine-to-Machine (M2M) communications traffic [2]. Data from a lot of sensors will be accumulated and analyzed by using the computation resources deployed in the network, such as cloud datacenters and edge computers. By placing and triggering controllers according to the analysis results, highly accurate and efficient device control can be achieved. Unfortunately, due to the continuous increase in Internet traffic, network equipment and datacenters are consuming more and more electrical power [3]. Therefore, more efficient use of network and computation resources in the provision of IoT services is required.

The M2M service network platform was presented in [4] as an architecture suitable for accommodating and controlling various types of connected devices. This network platform forms a tree-structured network that aggregates traffic from a lot of edge nodes to a solitary center cloud and computation resources deployed on every node in the network. The program files required to control the connected devices are placed on the network nodes according to the service quality requirements from each device, such as computation power, link bandwidth, and latency. To lower the power consumption, program files must be dynamically repositioned to meet the change in service requirements imposed by the devices. In [4], network and computation resources of the M2M service network platform are controlled in an integrated manner by using network virtualization techniques.

This paper proposes a Program File Placement (PFP) method for the M2M service network platform. First, to determine the optimal placement of program files, we formulate the PFP problem as a Mixed-Integer Linear Programming (MILP) problem. It is proved that the decision version of the PFP problem is NP-complete. For the case that an optimum placement cannot be obtained within a practical time by the MILP model, heuristic algorithms that offer sub-optimal solutions are presented.

The rest of this paper is organized as follows. The related works are described in Sect. 2. The detailed architecture of the M2M service network platform is introduced in Sect. 3. The PFP problem is modeled by MILP in Sect. 4. Sect. 5 shows that the PFP decision problem is NP-complete. Heuristic algorithms are presented in Sect. 6. The performance of the heuristic algorithms is evaluated in Sect. 7. Finally, we conclude this paper in Sect. 8.

2. Related Works

As a research topic similar to the PFP problem, the replica placement problem in Content Delivery Networks (CDN) should be mentioned [5]. The replica placement problem exists on a network consisting of origin servers, which hold the original contents, and surrogate servers, which can cache replicas of the contents. The objective of the replica placement problem is to obtain the placement of replicas that maximizes the Quality of Service (QoS) metrics of clients, such as latency and availability.

The M2M service network platform, on the other hand, must respond to device requests by setting program files that can realize the tasks requested. Unlike CDN, the time taken to finish the overall task has to be guaranteed for every

Manuscript received March 11, 2018.
Manuscript revised July 6, 2018.
Manuscript publicized September 20, 2018.

†The authors are with the Graduate School of Informatics, Kyoto University, Kyoto-shi, 606-8501 Japan.
a) E-mail: takehiro.sato@i.kyoto-u.ac.jp

DOI: 10.1587/transcom.2018NVP0001

device. In addition, nodes that a certain device can access is limited. Every program file that the device requests must be placed onto any one of these nodes. The rearrangement of program files to satisfy the service quality request of a specific device may impact other devices by varying the task execution time or making program files inaccessible. As program file rearrangements often occur, the whole the platform is likely to become unstable. Thus the PFP problem in the M2M service network platform requires a different solution than the techniques developed to address the replica placement problem in CDN.

There are several existing works on the resource allocation problem in a multi-stage computing environment for offloading tasks of mobile/wearable applications [6]–[8]. Z. Cheng et al. [6] present a three-layer computing architecture for code offloading of wearable devices. The architecture consists of a wearable device layer, a mobile device layer, and a remote cloud layer. Two categories of computing tasks that compose a wearable application are considered; *w-tasks* that are dependent on local functions of the wearable device (e.g., input devices, sensors, and displays) and *non-w-tasks* that are independent of the local functions and able to be offloaded to the upper layers. The authors formulate an optimization problem and present a heuristic approach based on genetic algorithm to determine the assignment and scheduling of non-w-tasks that maximizes the number of w-tasks executed within a given delay time after their previous w-task. F. Berg et al. [7] present a code offloading system for mobile applications in a multi-tier computing environment named Code Bubbling Offload System (CoBOS). When a mobile device, which is considered as the bottom tier, executes a mobile application and reaches an offloadable part, the mobile device creates an offload request message and send it to a manager of the nearest tier. The manager, which is deployed in every tier, locally decides whether the code should be executed in the tier or not, and, only in the former case, the manager responses an offering message to the mobile device. The manager forwards the offload request message to the upper tier regardless of the decision. The mobile device receives the offering messages from multiple tiers, and decides whether the device continues the execution of the application part locally or offloads the application part to one of the offering tiers. All of the decisions in the tier managers and the mobile device are based on the calculation of utility value, which takes into account energy consumption, execution time, and monetary cost. The overall objective of CoBOS is to minimize the utility value. L. Tong et al. [8] present a hierarchical multi-tier edge cloud computing architecture for the remote execution of mobile applications. The mobile users connect with the servers in the bottom tier and send the workloads of mobile application to the servers. The workloads that cannot be handled in the server due to the lack of computational capacity are offloaded to servers in the higher tiers. The authors formulate an optimization problem and present a heuristic approach based on simulated annealing to determine the workload placement on the servers that minimizes the computation delay and the communication delay.

In contrast with these existing resource allocation problems, each of which determines the placement of offloadable tasks originally executed in the mobile/wearable device, the PFP problem presented in this paper determines the placement of program files that are provided by a center cloud and required to execute a task of each device. These program files are able to be shared by multiple devices as long as enough computation resources are reserved on a node. The objective of PFP problem is to reduce the total number of placed program files while satisfying the service quality of devices such as latency requirements.

3. Architecture of M2M Service Network Platform

Figure 1 overviews the M2M service network platform [4]. This network platform has a tree-structured topology. Devices (i.e., sensors and actuators) are connected to edge nodes, and their traffic streams are aggregated and passed to a solitary center cloud. Computation resources (i.e., CPU cores, memory) are deployed not only the edge nodes and the center cloud, but also intermediate nodes, such as in-company datacenters, regional datacenters, and Central Office Re-architected as a Data center (CORD™) [9]. The amount of computation resources and link bandwidth capacity increases the higher the node is located in the hierarchy (i.e. closer to the center cloud).

One of the features of M2M service network platform is that a device can use only the computation resources of nodes that lie on the path between the edge node hosting the device and the center cloud. Every program file that the device requests is placed onto any one of these nodes. Therefore, the device does not need to search all nodes in the platform for the program files required to execute its task. The device simply sends queries and sensor data in the upstream direction to discover the desired program files.

Program files are basically placed on the center cloud, which has huge computation resources, and moved to or replicated at downstream nodes dynamically according to device status. For example, a program file may be moved or replicated to a downstream node when a certain device requires low latency or uses the program file frequently. In contrast, a program file may be moved or aggregated to upstream nodes if running the program file requires greater computation resources or is used by multiple devices. The placement of program files should be determined so as to utilize network and computation resources efficiently while satisfying the service quality requests of all devices.

4. Modeling the Program File Placement Problem

In this section, the Program File Placement (PFP) problem in the M2M service network platform is modeled using Mixed-Integer Linear Programming (MILP). We define the PFP problem as the problem to find the placement of program files on the nodes, which minimizes the number of placed files, while satisfying program file requirements and service

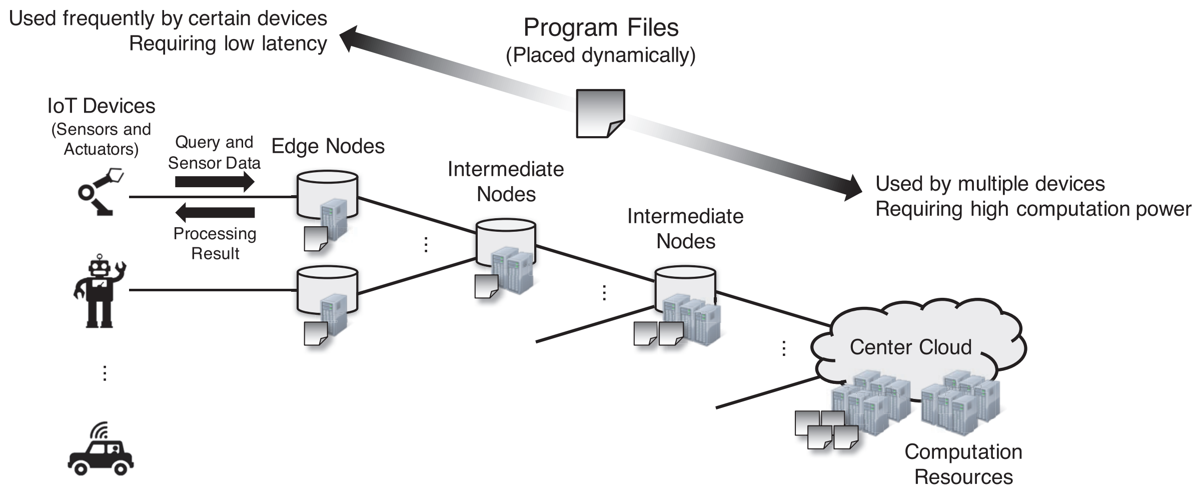

Fig. 1 Overview of M2M service network platform.

quality requests of all devices in the M2M service network platform.

We assume virtual machine (VM) image files as the program files handled in the M2M service network platform. Each of VM image files consists of a set of codes that provides a certain function required to complete a task. A dedicated VM instance is launched for every device that requires the function by using computation resources deployed on the node. The size of VM image file is relatively large (e.g., several gigabytes or more), so it is important to promote sharing the file by multiple devices and reduce the total number of placed files.

4.1 Conditions of Computation Model

In the computation model discussed in this Section, all nodes including edge nodes and the center cloud have computation resources to place and execute program files. When a node receives a query from a device, the node executes the requested program file if the program file is held by the node and the node has enough computation resources available. If the desired program file is not on the node or the node has insufficient free computation resources, the node transfers the query to the next upstream node. The center cloud has huge computation resources and holds all types of program files. When a node finishes executing a program file or receives a processing result from an upstream node, the node transfers the processing result to the next downstream node towards the device. The nodes do not transfer a query received from one downstream node to a different downstream node.

The model is intended for the static scenario, i.e., point of attachment, a set of requested program files, and latency requirement of each device are known in advance. Each edge node serves a single device. Each device has both sensors and actuators, and operates in a stand-alone manner. A task in a device may consist of multiple program files that should be executed in order. A program file held by a node can be shared by multiple devices if the node has sufficient computation resources to support all of those devices. Each program file process is assumed to be completed in a single node. Interactions between multiple program files placed in a single node or multiple nodes are not considered. Background traffic other than queries and sensor data from the devices and processing results from the nodes are not considered. Queueing delay and packet losses in the nodes are assumed to be zero.

Note that we introduce the condition that each edge node serves a single device in order to keep the model simple. The model can be extended to deal with the problem where multiple devices are attached to an edge node by the following procedure.

1. Add node stages to the edge-side of the platform.
2. Set the computational resource amount of added nodes to zero.
3. Rearrange the attachment point of each device so that a single device is attached to each added node.
4. Attach a dummy device which requests no program files to each vacant node.

4.2 Computation Model by Mixed-Integer Linear Programming

Table 1 defines the symbols used to model the PFP problem.

Figure 2 shows the model of the M2M service network platform as represented by undirected graph $G = (V, E)$. For the sake of simplicity, a perfect binary tree is considered below. The term “node $(m, n) \in V$” includes all edge nodes, intermediate nodes, and the center cloud. m $(1 \leq m \leq M)$ is the number of stages counted from the edge node. n $(1 \leq n \leq 2^{M-m})$ is the index number of nodes belonging to the same stage. The center cloud is located on the Mth stage, and represented as node $(M, 1)$. In this model, each node other than the center cloud has a single upstream link. Therefore, links are represented as $(m, n) \in E$ $((M, 1) \notin E)$.

The devices connected to this platform are represented by $u \in U$, where U is a set of u and $|U| > 1$. In this model, only one device is connected with each edge node. That

Table 1 Definition of symbols for PFP problem in M2M service network platform.

Sets

V	Set of nodes
V_u	Set of nodes ($V_u \subset V$) available to device $u \in U$
E	Set of links
E_u	Set of links ($E_u \subset E$) available to device $u \in U$
U	Set of devices
F	Set of types of program files
R_u	Set of types of program files ($R_u \subseteq F$) that device $u \in U$ requests

Given Parameters

$c_{u,f}$	Amount of computation resources that device $u \in U$ uses to execute program file $f \in R_u$
$b_{u,f}$	Bandwidth required at links between device $u \in U$ and the node holding program file $f \in R_u$
$d^{\text{prop}}_{m,n}$	One-way propagation delay time on the upstream link $(m,n) \in E$ of node (m,n)
$d^{\text{exe}}_{u,f,m,n}$	Processing time of program file $f \in R_u$ requested by device $u \in U$ at node $(m,n) \in V$
$d^{\text{after}}_{u,f}$	Post processing time of processing result of program file $f \in R_u$ at device $u \in U$
$C_{m,n}$	Amount of available computation resources at node $(m,n) \in V$
$B_{m,n}$	Bandwidth capacity of the upstream link $(m,n) \in E$ of node (m,n)
D_u	Latency requirement as regards completing the overall task at device $u \in U$
M	Number of node stages at M2M service network platform

Decision Variables

$x_{u,f,m,n}$	1 if program file of type $f \in R_u$ requested by device $u \in U$ is executed on node $(m,n) \in V$; 0 otherwise
$y_{f,m,n}$	1 if program file of type $f \in F$ is held by node $(m,n) \in V$; 0 otherwise
$l_{u,f,m,n}$	1 if traffic between device $u \in U$ and the node where program file $f \in R_u$ is placed goes through link $(m,n) \in E$; 0 otherwise
$b^{\max}_{u,m,n}$	Bandwidth that device $u \in U$ has to reserve on upstream link $(m,n) \in E$ of node (m,n)

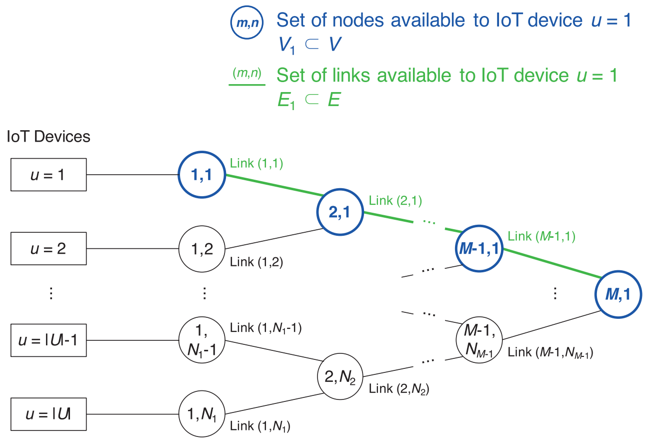

Fig. 2 Graph model of M2M service network platform.

means, device u is connected with edge node $(1,n)$ where $n = u$ at the first stage of the platform. Nodes available to device u are limited to those located on the path between edge node $(1,u)$ and the center cloud $(M,1)$. $V_u \subset V$ is the set of nodes available to device u. $E_u \subset E$ is the set of links that connect adjacent nodes in V_u. The traffic from device u must go through the links in $E_u \subset E$. Note that the link that connects device u to edge node $(1,u)$ is not in E or E_u.

The types of program files provided by this platform are represented by $f \in F$, where F is a set of f. The set of program types forming the task requested by device u is represented by $R_u \subseteq F$, and the ith program file that u requires is represented by $r_{u,i} \in R_u$, where $1 \leq i \leq |R_u|$. $|R_u|$ is the number of program files that compose the task requested by device u. Each program file in R_u is placed on node $(m,n) \in V_u$. The computation resources that device u uses to execute program file $r_{u,i} \in R_u$ are represented by $c_{u,i}$. As mentioned in Sect. 4.1, a program file placed on a node can be shared by multiple devices, but sufficient computation resources must be reserved for every device. The amount of bandwidth required at links between device u and the node where program file $r_{u,i} \in R_u$ is placed is represented by $b_{u,i}$. In other words, at least $b_{u,i}$ of bandwidth has to be reserved on these links to transfer a query, sensor data, and a processing result. The program files are executed in sequence. To prevent interruption of the task of device u due to insufficient bandwidth, the maximum bandwidth that goes through the link, $\max_{f \in R_u}\{l_{u,f,m,n} b_{u,f}\}$, is reserved for device u at each link $(m,n) \in E_u$ in advance. The time required to execute the task of device u should be equal to or less than the latency requirement D_u. The required time consists of the sum of propagation delay time on the links used, the sum of processing time at each node, and the sum of post processing time at device u.

Figure 3 shows an example of the allocation of computation resources and link bandwidth for a task requested by device $u = 1$. The corresponding task sequence is shown in Fig. 4. In Figs. 3 and 4, the task requested by device $u = 1$ consists of three program files: $r_{1,1}$, $r_{1,2}$ and $r_{1,|R_1|}$. These program files are held by nodes $(1,1)$, $(M,1)$, and $(2,1)$, respectively. The computation resources required by the program files are taken to be $c_{1,1} = 2$, $c_{1,2} = 4$, and $c_{1,|R_1|} = 2$. The bandwidths that the program files require are taken to be $b_{1,1} = 3$, $b_{1,2} = 2$, and $b_{1,|R_1|} = 1$. The required computation resources and bandwidth are reserved at every node and link, respectively. Note that the bandwidth of $\max_{f \in R_u} b_{u,f}$ is always required at the link between device u and edge node $(1,u)$.

As shown in Fig. 4, we consider a sequential program access in this computation model. Several types of M2M applications inevitably require such a sequential program access and do not allow to shorten the procedure anymore. For example, the remote control application of an industrial robot based on the robot operating system (ROS) framework is reported in [10]. In this application, the robot has to access three programs placed on cloud or edge compute nodes, namely the surface detection, the process path planning, and

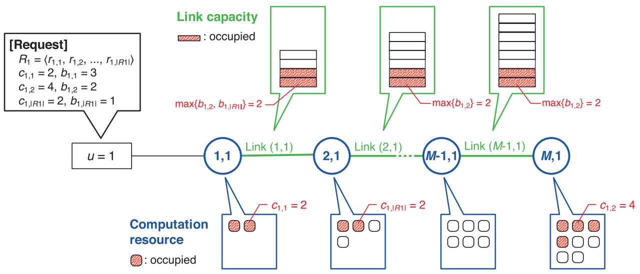

Fig. 3 Example of resource allocation for a device.

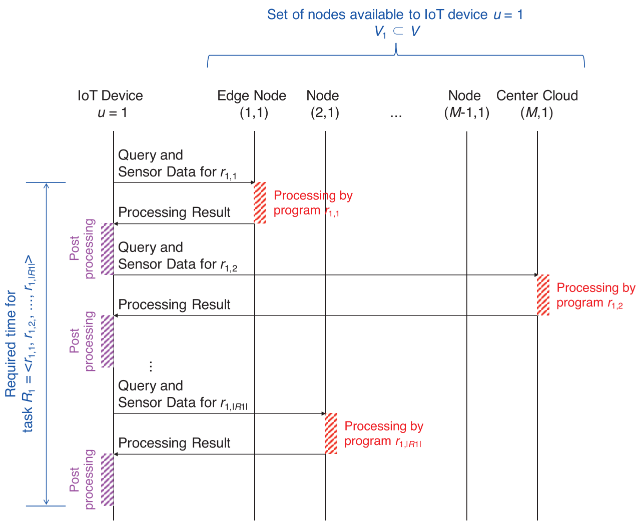
Set of nodes available to IoT device u = 1
$V_1 \subset V$
IoT Device u = 1
Edge Node (1,1)
Node (2,1)
...
Node (M-1,1)
Center Cloud (M,1)
Query and Sensor Data for $r_{1,1}$
Processing by program $r_{1,1}$
Processing Result
Post processing
Query and Sensor Data for $r_{1,2}$
Processing by program $r_{1,2}$
Processing Result
Post processing
Query and Sensor Data for $r_{1,|R1|}$
Processing by program $r_{1,|R1|}$
Processing Result
Post processing
Required time for task $R_1 = <r_{1,1}, r_{1,2}, \ldots, r_{1,|R1|}>$

Fig. 4 Example of task sequence for a device.

the motion planning, in a sequential manner to complete a surface blending task. The process path planning and the motion planning require the results of their previous program as an input, so it is impossible to exchange the order of program executions or shorten the procedure.

The PFP problem is formulated by Mixed-Integer Linear Programming (MILP) as follows.

Objective:

$$\min \left(\sum_{f\in F} \sum_{(m,n)\in V} (M-m+1)\, y_{f,m,n} + \epsilon \sum_{u\in U} \sum_{(m,n)\in E} b^{\max}_{u,m,n} \right) \qquad (1)$$

Equation (1) is the objective function of this problem. In contrast with the objectives in existing works [6]–[8] presented in Sect. 2, this objective function aims to minimize the number of program files placed on the nodes.

The first term of Eq. (1), excluding $(M-m+1)$, represents the number of program files placed on nodes. As mentioned in Sect. 3, the M2M service network platform has a policy that program files are basically placed on the center cloud and can be moved or replicated to downstream nodes if necessary [4]. To place as many program files as possible on cloud-side nodes, the first term of Eq. (1) should be multiplied by a weight that decreases according to the number of stages m. In Eq. (1), $(M-m+1)$, which indicates the number of stages counted from the center cloud, is used for the purpose.

There is a possibility that multiple feasible solutions which provide the same optimal value exist on the MILP model. In order to choose an effective solution from them, the second term is added in Eq. (1). The second term, excluding ϵ, represents the sum of bandwidth reserved on the links. Although this term does not relate to the original objective of the PFP problem, it helps to obtain a solution with the best link bandwidth utilization efficiency among the feasible solutions which provide the same value in the first term. In order to prioritize the first term over the second term, the latter is multiplied by a small number, ϵ, so that the value of second term becomes less than one. ϵ is given by Eq. (2).

$$\epsilon = \frac{1}{1 + \sum_{(m,n)\in E} B_{m,n}} \qquad (2)$$

Subject to:

$$\sum_{(m,n)\in V_u} x_{u,f,m,n} = 1, \quad \forall u \in U, \forall f \in R_u \qquad (3)$$

$$x_{u,f,m,n} \le y_{f,m,n}, \quad \forall u \in U, \forall f \in R_u, \forall (m,n) \in V_u \qquad (4)$$

$$\sum_{u\in U} \sum_{f\in F} c_{u,f}\, x_{u,f,m,n} \le C_{m,n}, \quad \forall (m,n) \in V \qquad (5)$$

$$l_{u,f,m-1,i} \ge x_{u,f,m,n}, \quad \forall u \in U, \forall f \in R_u, \forall (m,n) \in V_u, \forall (m-1,i) \in V_u, m \ne 1 \qquad (6)$$

$$1 - x_{u,f,m,n} \ge l_{u,f,m,n}, \quad \forall u \in U, \forall f \in R_u, \forall (m,n) \in V_u \qquad (7)$$

$$l_{u,f,m-1,i} \ge l_{u,f,m,n}, \quad \forall u \in U, \forall f \in R_u, \forall (m,n) \in V_u, \forall (m-1,i) \in V_u, m \ne 1 \qquad (8)$$

$$l_{u,f,m,n} = 0, \quad \forall u \in U, \forall f \in F : f \notin R_u, \forall (m,n) \in V_u \qquad (9)$$

$$\sum_{u\in U} \max_f \{l_{u,f,m,n} b_{u,f}\} \le B_{m,n}, \quad \forall (m,n) \in E \qquad (10)$$

$$\sum_{(m,n)\in E_u} \sum_{f\in R_u} d^{\text{prop}}_{m,n} l_{u,f,m,n} \times 2 + \sum_{(m,n)\in V_u} \sum_{f\in R_u} d^{\text{exe}}_{u,f,m,n} x_{u,f,m,n} + \sum_{f\in R_u} d^{\text{after}}_{u,f} \le D_u, \quad \forall u \in U \qquad (11)$$

Equations (3)–(10) are the constraints of this problem.

Equation (3) means that a program file of type $f \in R_u$ is executed at one of the nodes $(m,n) \in V_u$ that are available to device u. Equation (4) means that a program file of type

$f \in R_u$ is placed on node $(m,n) \in V_u$ when the program file is executed at the node. Equation (5) means that the total required amount of computation resources to execute program files at node $(m,n) \in V$ is equal to or less than the available computation resources $C_{m,n}$. Equation (6) means that when a program file of type $f \in R_u$ is executed on node $(m,n) \in V_u$, the bandwidth required to transfer the traffic for the program file is reserved at the link between nodes (m,n) and $(m-1,i) \in V_u$. Equation (7) means that when a program file of type $f \in R_u$ is executed on node $(m,n) \in V_u$, no bandwidth is reserved for the program file at the links upstream of node (m,n). Equation (8) means that when the bandwidth required to transfer traffic for a program file of type $f \in R_u$ is reserved at the upstream link of node $(m,n) \in V_u$, the same amount of bandwidth is reserved for the program file at the link between nodes (m,n) and $(m-1,i) \in V_u$. Equation (9) means that no bandwidth is reserved for a program file of type f at links upstream of node in V_u when device u does not request the program file. Equation (10) means that the sum of bandwidth reserved at link $(m,n) \in E$ is equal to or less than the bandwidth capacity of link $B_{m,n}$. Equation (11) means that the time needed to execute the task of device u is equal to or less than the latency requirement D_u.

Equation (10) includes a max function. Equation (10) can be recast in linear form by using $b^{\max}_{u,m,n}$ to yield Eqs. (12) and (13).

$$l_{u,f,m,n} b_{u,f} \leq b^{\max}_{u,m,n}, \quad \forall u \in U, \forall f \in R_u, \forall (m,n) \in E_u \tag{12}$$

$$\sum_{u \in U} b^{\max}_{u,m,n} \leq B_{m,n}, \quad \forall (m,n) \in E \tag{13}$$

5. NP-Completeness

In this Section, we prove that the decision version of PFP problem is NP-complete. From the PFP problem described in Sect. 4.2, we define the PFP decision problem as:

Problem Given a set of requested program files $R_u \subseteq F$ from each device $u \in U$, is it possible to place program files of F onto nodes $v \in V$, which satisfy the link bandwidth and latency requirements between each u and v, such that the total required computation resources on v does not exceed C_v?

Theorem *The PFP decision problem is NP-complete.*

Proof: First, we show that the PFP decision problem is in NP. For a given instance of the PFP decision problem, we can verify whether the program files in $R_u \subseteq F$ are placed onto nodes $v \in V$ that device $u \in U$ can access in polynomial time $O(|U|\,|V|\,|R_u|)$. We can also verify whether the total required computation resources in each node $v \in V$ do not exceed v's computation resources capacity C_v in polynomial time $O(|U|\,|V|\,|R_u|)$. Therefore, we can check the validity of an instance of the PFP decision problem in polynomial time.

Next, we show that the partition problem (PB), which is a known NP-complete problem [11], is reducible to the PFP decision problem. PB is defined as: is it possible to partition a given set G of positive integers into two subsets G_1 and G_2 such that the sums of numbers in the two subsets equal each other?

We construct an instance of the PFP decision problem from any instance of PB. An instance of PB consists of a set G of positive integers and the value of positive integer $g \in G$ is represented by I_g. An instance of the PFP decision problem is constructed with the following algorithm, which runs in polynomial time of $O(|G|)$.

1. We consider an M2M service network platform consisting of one edge node, i.e. node $(1,1)$, and a center cloud, i.e. node $(2,1)$. Device $u = 1$ connected to node $(1,1)$ can use nodes $(1,1)$ and $(2,1)$ and requests a set of program files R_1 with $|R_1| = |G|$. For each positive integer $g \in G$, there is a corresponding program file $r_{1,j} \in R_1$ with computation resource requirement of $c_{1,j} = I_g$.
2. The maximum computation resources that nodes $(1,1)$ and $(2,1)$ can provide are set to $C_{1,1} = C_{2,1} = \frac{\sum_{i=1}^{|G|} c_{1,i}}{2}$.
3. The bandwidth capacity of the link between nodes $(1,1)$ and $(2,1)$ is set to $B_{1,1} = \infty$, which means that the link can accommodate any traffic between the nodes.
4. The latency requirement of device $u = 1$ is set to $D_1 = \infty$, which means that the device does not care about the latency to finish the task of R_1.

Consider that a PB instance is a Yes instance. G can be partitioned into two subsets G_1 and G_2, and the sums of the numbers in the two subsets are $\frac{\sum_{g \in G} I_g}{2}$. Define a PFP instance from the PB instance by using the above described algorithm. In the PFP instance, each requested program file $r_{1,j} \in R_1$ requires computation resources with value $c_{1,j}$. The set of computation resource requirements $S = \{c_{1,1}, c_{1,2}, \cdots, c_{1,|G|}\}$ is able to be partitioned into two subsets S_1 and S_2, which refer to G_1 and G_2, respectively, and the total computation resource requirements in the two subsets are $\frac{\sum_{i=1}^{|G|} c_{1,i}}{2}$. By allocating S_1 and S_2 to nodes $(1,1)$ and $(2,1)$, respectively, device $u = 1$ can access the program files of R_1. As a result, it is possible to place program files of R_1 onto nodes $v \in V$, which satisfy the link bandwidth and latency requirements, with the total computation resource requirements on node v equal to $C_v = \frac{\sum_{i=1}^{|G|} c_{1,i}}{2}$. Therefore, the PFP instance is a Yes instance.

Conversely, consider a PFP instance is a Yes instance. In the Yes PFP instance, in order to provide a set of program files R_1 to device $u = 1$, for each program file $r_{1,j} \in R_1$, corresponding computation resource requirements $c_{1,j} \in S$ have to be allocated to nodes $(1,1)$ and $(2,1)$. Since the capacities of nodes $(1,1)$ and $(2,1)$ are $\frac{\sum_{i=1}^{|G|} c_{1,i}}{2}$ and $\frac{\sum_{i=1}^{|G|} c_{1,i}}{2}$, respectively, the set of computation resource requirements S can be partitioned into two subsets S_1 and S_2, where the total computation resource requirements of the two subsets equal $\frac{\sum_{i=1}^{|G|} c_{1,i}}{2}$. Referring to the partition of S, we are able to partition G into two subsets G_1 and G_2 such that the sums of numbers in the two subsets equal each other. Therefore, if the PFP instance is a Yes instance, then the PB instance is a Yes instance.

The above described algorithm transforms any PB instance into a PFP instance in polynomial time. This confirms that if a PB instance is a Yes instance, then the corresponding PFP instance is a Yes instance, and vise versa. This proves that PB, a known NP-complete problem, is polynomial time reducible to the PFP decision problem. Thus, the PFP decision problem is NP-complete. □

6. Heuristic Algorithms

For the case that the optimum placement cannot be obtained within a practical time by the MILP problem modeled in Sect. 4.2, this section presents heuristic algorithms for the PFP problem that output sub-optimal solutions. The heuristic algorithms presented in this section are designed focusing on the reduction of the number of placed program files, which is the original objective of the PFP problem. The link bandwidth utilization is not taken into account in the heuristic algorithms since it is additionally considered in the MILP formulation to choose a better solution from multiple solutions with the same performance in the program file placement.

6.1 Most Upstream Position First Algorithm

The Most Upstream Position First (MUPF) algorithm aims to find a feasible placement of program files with low computational time complexity. The basic idea of the MUPF algorithm is to move program files initially placed on the center cloud to downstream nodes by considering only their position.

6.1.1 Algorithm Description

The MUPF algorithm is illustrated in Algorithm 1.

Devices are sorted and processed sequentially in ascending order of D_u; a device with stricter latency requirement is processed before other devices. For each device, all requested program files are initially held only by the center cloud. The algorithm checks if all the constraints, i.e. computation resources, link bandwidth, and latency requirement, of the device are met. If the current placement of program files does not meet at least one constraint, a requested program file placed on the most upstream node is selected and moved to the next downstream node. If multiple program files are placed on the most upstream node, a program file with the smallest index number is selected. The movement of program files and the constraint checks are iterated until every constraint is met, or until all requested program files have been moved to the edge node. When the placement of program files for the device that meets every constraint is found, the algorithm moves on to the next device. When the placement of program files for a device does not meet any constraint even when all of the requested program files have been moved to the edge node, the algorithm judges that there is no feasible solution.

A program file is shared if multiple devices have decided to place the same program file on the same node. When the placement of program files has been successfully decided for all devices, the algorithm returns the entire placement of program files and the link bandwidth allocation as a solution.

6.1.2 Computational Time Complexity

Sorting $|U|$ devices in ascending order of D_u takes $O(|U| \log |U|)$. Each device requests up to $|F|$ program files. For each iteration for a device, a program file placed on the most upstream node and that has the smallest index number is selected in $O(1)$ and moved to the next downstream node. All of the requested program files are moved from the center cloud $(M, 1)$ to the edge node $(1, u)$ in the worst case. This movement of program files is performed in $O(M|U||F|)$. As a result, the computational time complexity of the MUPF algorithm is $O(|U|(\log |U| + M|F|))$. If the topology of M2M service network platform is a perfect binary tree, the number of node stages is $M = \log_2 |U| + 1$. In this case, the computational time complexity is given by $O(|U||F| \log |U|)$.

The above discussion indicates that the computational time of the MUPF algorithm grows by a factor of $|U| \log |U|$ against the number of devices $|U|$ at the worst case. The MUPF algorithm has a polynomial computational time, so practically it can be considered as an efficient algorithm [12].

6.2 Small Sharing Degree First Algorithm

The Small Sharing Degree First (SSDF) algorithm aims to reduce the total number of placed program files compared to the MUPF algorithm by increasing the program files that are shared by multiple devices. The basic idea of the SSDF algorithm is to move program files to downstream nodes based on the number of devices that share a program file.

6.2.1 Algorithm Description

The SSDF algorithm is illustrated in Algorithm 2. Different from the MUPF algorithm, the SSDF algorithm checks the number of devices sharing the same program file in each iteration, and selects and moves the requested program file shared by the least number of devices.

Devices are sorted and processed sequentially in ascending order of D_u; a device with stricter latency requirement is processed before other devices. For each device, all requested program files are initially held only by the center cloud and marked “movable”. The algorithm checks if all the constraints of computation resources, link bandwidth, and latency requirement of the device are met. If the current

Algorithm 1: Most Upstream Position First (MUPF)

```
Data: Network condition
      $\langle V, E, U, F, d^{\mathrm{prop}}_{m,n}, C_{m,n}, B_{m,n}, M \rangle$, and program file
      requests from devices
      $\langle R_u, c_{u,f}, b_{u,f}, d^{\mathrm{exe}}_{u,f,m,n}, d^{\mathrm{after}}_{u,f}, D_u \rangle$
Result: Node utilization $x_{u,f,m,n}$, placement of program files
        $y_{f,m,n}$, link utilization $l_{u,f,m,n}$, and link bandwidth
        allocation $b^{\max}_{u,m,n}$
begin
    for $u \in U$ in ascending order of $D_u$ do
        Place all program files in $R_u$ on the center cloud
          $(M, 1)$, and record the placement in $x_{u,f,m,n}$
        Utilize every link between the edge node $(1, u)$ and the
          center cloud $(M, 1)$ to access every program file in
          $R_u$, and record the link utilization in $l_{u,f,m,n}$
        Allocate bandwidth at each link $(m, n)$, and record the
          link bandwidth allocation in $b^{\max}_{u,m,n}$
        while The current placement of program files does not
          meet $C_{m,n}, \forall(m, n) \in V_u$ , $B_{m,n}, \forall(m, n) \in E_u$ ,
          and $D_u$ do
            if All of program files in $R_u$ are placed on the
              edge node $(1, u)$ then
                Error exit (feasible solution not found)
            Select a program file in $R_u$ placed on the most
              upstream node
            Move the selected program file to the next
              downstream node
            Update $x_{u,f,m,n}$, $l_{u,f,m,n}$, and $b^{\max}_{u,m,n}$
    Get $y_{f,m,n}$ from $x_{u,f,m,n}$
    if The placement of program files meets
      $C_{m,n}, \forall(m, n) \in V$ , $B_{m,n}, \forall(m, n) \in E$ , and
      $D_u, \forall u \in U$ then
        Return $x_{u,f,m,n}$, $y_{f,m,n}$, $l_{u,f,m,n}$, and $b^{\max}_{u,m,n}$
    else
        Error exit (feasible solution not found)
```

placement of program files does not meet at least one constraint, a requested program file shared by the least number of devices examined earlier and marked "movable" is selected. If the next downstream node has enough computation resources to accept the selected program file, the program file is moved to the node. Otherwise, the program file remains on the current node and is marked "immovable". The program file is also marked "immovable" when it reaches the edge node. The movement of program files and the constraint checks are iterated until every constraint is met, or until all requested program files have been marked "immovable". When the placement of program files for the device that meets every constraint is found, the algorithm moves on to the next device. When the placement of program files for a device does not meet any constraint even when all requested program files have been marked "immovable", the algorithm judges that there is no feasible solution.

A program file is shared if multiple devices have decided to place the same program file on the same node. When the placement of program files has been successfully decided for all devices, the algorithm returns the entire placement of program files and the link bandwidth allocation as a solution.

6.2.2 Computational Time Complexity

Sorting $|U|$ devices in ascending order of D_u takes $O(|U| \log |U|)$. Each device requests up to $|F|$ program files. For each iteration for a device, one of the program files shared by the least number of devices and marked "movable" is selected in $O(|F|)$ and moved to the next downstream node. In the same way as the MUPF algorithm, all of the requested program files are moved from the center cloud $(M, 1)$ to the edge node $(1, u)$ in the worst case. Therefore, the movement of program files is performed in $O(M|U||F| \times |F|)$. As a result, the computational time complexity of the SSDF algorithm is $O(|U|(\log |U| + M|F|) \times |F|)$. If the topology of M2M service network platform is a perfect binary tree, the computational time complexity is given by $O(|U||F|^2 \log |U|)$.

Although the complexity of the SSDF algorithm is higher than that of the MUPF algorithm, the SSDF algorithm is still a polynomial-time algorithm. From the perspective of the scalability against the number of devices $|U|$, the computational time of SSDF algorithm also grows by a factor of $|U| \log |U|$ at the worst case.

7. Evaluation

We evaluated the performance of our heuristic algorithms by comparisons against the optimal value obtained by using the MILP model presented in Sect. 4.2. The averages of objective value and number of placed program files were evaluated as the metrics of resource utilization efficiency.

7.1 Simulation Environment

The M2M service network platform with 4, 8, 16, and 32 devices is considered in this simulation. The number of steps of network topology, M, is 3, 4, 5, and 6, respectively. The number of program types, $|F|$, is set to 4 and 8.

A set of program files that each device u requests, R_u, is selected by the following procedure. First, the number of requested program files, $|R_u|$, is selected from uniform random numbers between 1 and $|F|$. Then $|R_u|$ types of program files are randomly chosen from the set of program types, F.

To simplify the discussion, $c_{u,f}$, $b_{u,f}$, $d^{\mathrm{prop}}_{m,n}$, $d^{\mathrm{exe}}_{u,f,m,n}$, and $d^{\mathrm{after}}_{u,f}$ are all set to 1. The amount of computation resources of node (m, n), $C_{m,n}$, is set to $\lceil \frac{|F|+1}{2} \rceil \times 2^{(m-1)}$ except for the center cloud $(M, 1)$. $C_{M,1}$ is set to a large enough number. The bandwidth capacity of link (m, n), $B_{m,n}$, is set to $2^{(m-1)}$; this is sufficient to accommodate all of the traffic from devices $u \in U$ in this simulation environment. This bandwidth setting helps us to evaluate the performance of algorithms focusing on the reduction of the number of placed program files, which is the original objective of the PFP problem.

The latency requirement from device u, D_u, is selected

Algorithm 2: Small Sharing Degree First (SSDF)

```
Data: Network condition
      $\langle V, E, U, F, d^{\mathrm{prop}}_{m,n}, C_{m,n}, B_{m,n}, M \rangle$, and program file
      requests from devices
      $\langle R_u, c_{u,f}, b_{u,f}, d^{\mathrm{exe}}_{u,f,m,n}, d^{\mathrm{after}}_{u,f}, D_u \rangle$
Result: Node utilization $x_{u,f,m,n}$, placement of program files
        $y_{f,m,n}$, link utilization $l_{u,f,m,n}$, and link bandwidth
        allocation $b^{\max}_{u,m,n}$
begin
    for $u \in U$ in ascending order of $D_u$ do
        Place all program files in $R_u$ on the center cloud
          $(M, 1)$, and record the placement in $x_{u,f,m,n}$
        Utilize every link between the edge node $(1, u)$ and the
          center cloud $(M, 1)$ to access every program file in
          $R_u$, and record the link utilization in $l_{u,f,m,n}$
        Allocate bandwidth at each link $(m, n)$, and record the
          link bandwidth allocation in $b^{\max}_{u,m,n}$
        Mark every program file in $R_u$ "movable"
        while The current placement of program files does not
          meet $C_{m,n}, \forall(m,n) \in V_u$ , $B_{m,n}, \forall(m,n) \in E_u$ ,
          and $D_u$ do
            if All of program files in $R_u$ are marked
              "immovable" then
                Error exit (feasible solution not found)
            Get the current position and the number of
              sharing devices of every program file in $R_u$
            Select a program file shared by the least number
              of devices and marked "movable"
            if The next downstream node has enough
              computation resources to place the selected
              program file additionally then
                Move the program file to the next
                  downstream node
                Update $x_{u,f,m,n}$, $l_{u,f,m,n}$, and $b^{\max}_{u,m,n}$
                if The program file reaches the edge node
                  $(1, u)$ then
                    Mark the program file "immovable"
            else
                Mark the program file "immovable"
    Get $y_{f,m,n}$ from $x_{u,f,m,n}$
    if The placement of program files meets
      $C_{m,n}, \forall(m,n) \in V$ , $B_{m,n}, \forall(m,n) \in E$ , and
      $D_u, \forall u \in U$ then
        Return $x_{u,f,m,n}$, $y_{f,m,n}$, $l_{u,f,m,n}$, and $b^{\max}_{u,m,n}$
    else
        Error exit (feasible solution not found)
```

from uniform random numbers between $2|R_u|$ and $2m|R_u|$. These numbers are the required times to finish the task when all of the program files in R_u are placed on edge node $(1, u)$ and center cloud $(M, 1)$, respectively.

We generated 1,000 patterns of simulation parameters for each set of $(M, |F|)$, and tried to solve the same set of parameters by using the MILP model and the heuristic algorithms. The MILP model is solved by using the CPLEX®Interactive Optimizer 12.7.1.0. We compared the results in case that the feasible solution was obtained by both MILP model and heuristic algorithms.

7.2 Simulation Results

Figures 5 and 6 show the average of objective values when the number of program types, $|F|$, is 4 and 8, respectively. Each of the figures shows the results for 4, 8, 16, and 32 devices. The objective values of the heuristic algorithms are calculated by using the objective function of the MILP model, i.e., Eq. (1). Therefore, a value related to the link bandwidth utilization is included in the objective values, but it is ignorably small (less than one). Note that, as mentioned in Sect. 4.2, the objective value becomes larger if program files are placed on nodes located more downstream. Compared to the optimal value, the objective value is 9%–38% and 9%–48% larger when using the MUPF algorithm and the SSDF algorithm, respectively.

Figures 7 and 8 show the average number of placed program files for 4 and 8 program types, respectively. Compared to the optimal value, the number of placed program files is 14%–61% and 8%–36% higher when using the MUPF algorithm and the SSDF algorithm, respectively. As Figs. 7 and 8 indicate, the SSDF algorithm yields solutions that require fewer program files than the MUPF algorithm. The number of placed program files 17% lower when the number of devices is 16 and the number of program types is 8. In the SSDF algorithm, program files shared by fewer devices are preferentially moved to downstream nodes. This feature

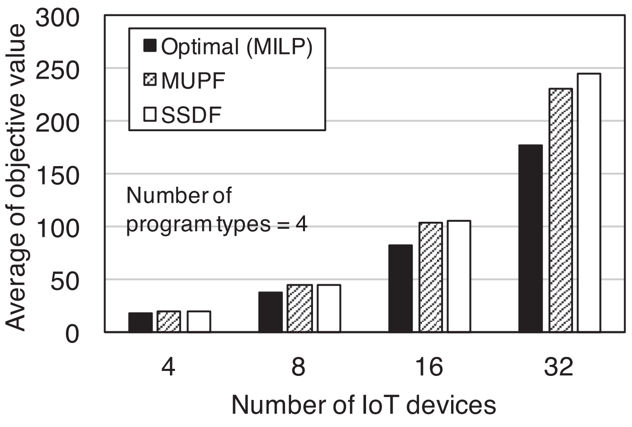

Fig. 5 Average of objective value when the number of program types, $|F|$, is 4.

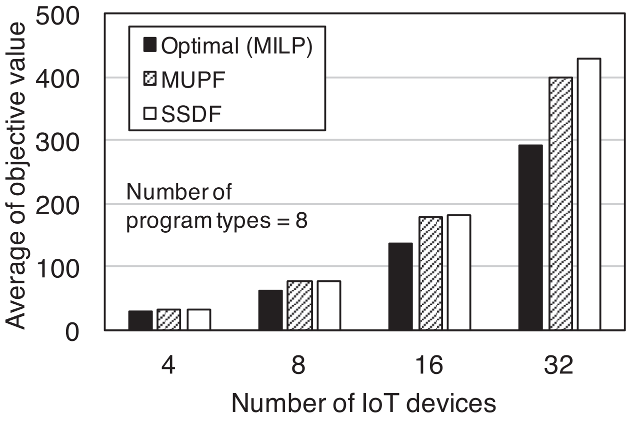

Fig. 6 Average of objective value when the number of program types, $|F|$, is 8.

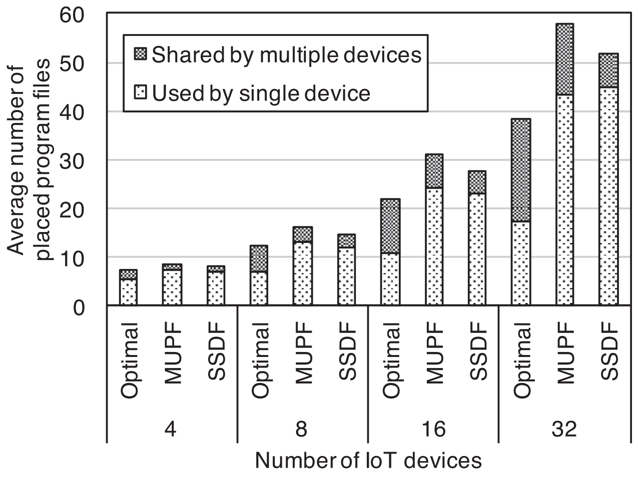

Fig. 7 Average number of placed program files when the number of program types, $|F|$, is 4.

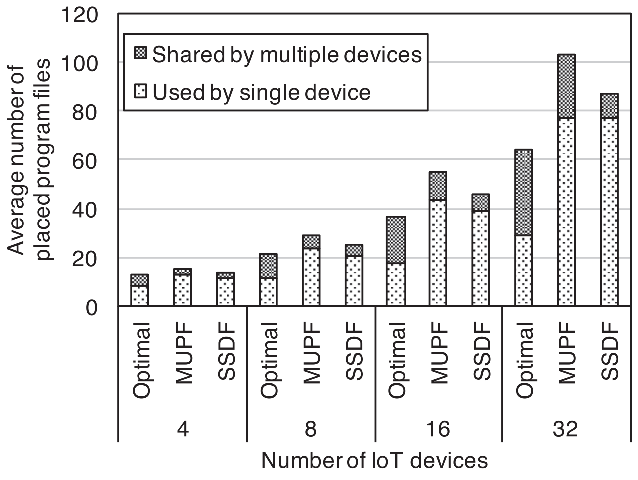

Fig. 8 Average number of placed program files when the number of program types, $|F|$, is 8.

leads an increase in objective value since more program files are placed on edge-side nodes compared to the MUPF algorithm. However, the total number of placed program files becomes smaller since the program files remaining on the upstream nodes are shared by many more devices. Figures 7 and 8 also show the number of program files shared by multiple devices and that used by a single device. The ratio of the program files shared by multiple devices to those used by a single device is larger in the optimal scenarios compared to the scenarios of applying heuristic algorithms. It can be observed that, when the number of devices is 8, 16, and 32, the number of shared program files obtained by using the SSDF algorithm is smaller than that obtained by the MUPF algorithm. This fact indicates that the SSDF algorithm enables program files to be shared by more devices compared to the MUPF algorithm.

The difference between the number of placed program files obtained by the heuristic algorithms and that obtained by the MILP model becomes bigger as the number of devices increases. This is because the number of node stages, M, increases according to the number of devices. Both heuristic algorithms determine the placement of program files for each individual device in a sequential manner. Therefore, when the number of node stages becomes large, the requested program files are likely to be scattered across multiple nodes; the number of program files that are shared by a large number of devices decreases. In the case of SSDF algorithm, the difference between the obtained value and the optimal value increases by approximately 8% when the number of devices doubles. However, as discussed in Sects. 6.1.2 and 6.2.2, both heuristic algorithms are polynomial-time algorithms. This fact indicates that these heuristic algorithms have a possibility of obtaining a solution for the PFP problem within a practical time even in the case of large number of devices.

Note that this simulation assumes that each link has enough bandwidth capacity to carry all traffic from the devices. If we consider the variability of link capacity $B_{m,n}$ and the required bandwidth from each device $c_{u,f}$, the number of placed program files might increase.

8. Conclusion

This paper proposed a Program File Placement (PFP) method for the Machine-to-Machine (M2M) service network platform providing program files and computation resources required to execute the tasks requested by connected IoT devices. To obtain the optimal placement of program files, a computation model of the PFP problem was introduced as a Mixed-Integer Linear Programming (MILP) problem. The PFP decision problem was proven to be NP-complete by reducing the partition problem (PB) to the PFP decision problem in polynomial time. Two heuristic algorithms, Most Upstream Position First (MUPF) algorithm and Small Sharing Degree First (SSDF) algorithm, were introduced. Simulations showed that the SSDF algorithm tends to place more program files on edge-side nodes, but reduces the number of placed program files compared to the MUPF algorithm. As future work, the proposed PFP algorithms could be expanded to online algorithms in order to support dynamic scenarios wherein the devices connect and disconnect to the platform dynamically or request multiple tasks. From the viewpoint of feasibility, methods to migrate and replicate program files between network nodes without disrupting current device tasks should be established.

References

[1] Gartner, "Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016," Press Release, Feb. 2017.

[2] Cisco Systems, Inc., "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021," White Paper, Feb. 2017.

[3] M. Pickavet, W. Vereecken, S. Demeyer, P. Audenaert, B. Vermeulen, C. Develder, D. Colle, B. Dhoedt, and P. Demeester, "Worldwide energy needs for ICT: The rise of power-aware networking," IEEE Advanced Networks and Telecommunication Systems (ANTS) 2008, pp.1–3, Dec. 2008.

[4] N. Yamanaka, N. Yoshikane, and T. Sato, "Optical M2M service platform and global robot control demonstration," Monthly OPTRONICS, vol.35, no.420, pp.75–81, Dec. 2016 (in Japanese).

[5] M.A. Salahuddin, J. Sahoo, R. Glitho, H. Elbiaze, and W. Ajib, "A survey on content placement algorithms for cloud-based content delivery networks," IEEE Access, vol.6, pp.91–114, Sept. 2017.

[6] Z. Cheng, P. Li, J. Wang, and S. Guo, "Just-in-time code offloading for wearable computing," IEEE Trans. Emerg. Topics Comput., vol.3, no.1, pp.74–83, March 2015.

[7] F. Berg, F. Dürr, and K. Rothermel, "Increasing the efficiency of code offloading in n-tier environments with code bubbling," Proc. 13th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MOBIQUITOUS 2016), pp.170–179, Nov. 2016.

[8] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," The 35th Annual IEEE International Conference on Computer Communications (IEEE INFOCOM 2016), April 2016.

[9] Open CORD – CORD, https://opencord.org, accessed March 11, 2018.

[10] N. Yoshikane, T. Sato, Y. Isaji, C. Shao, M. Tacca, S. Okamoto, T. Miyazawa, T. Oshima, C. Yokoyama, Y. Sumida, H. Sugiyama, M. Miyabe, T. Katagiri, N. Kakegawa, S. Matsumoto, Y. Ohara, I. Sato, A. Nakamura, S. Yoshida, K. Ishii, S. Kametani, J. Nicho, J. Meyer, S. Edwards, P. Evants, T. Tsuritani, H. Harai, M. Razo, D. Hicks, A. Fumagalli, and N. Yamanaka, "First demonstration of geographically unconstrained control of an industrial robot by jointly employing SDN-based optical transport networks and edge compute," The 21st OptoElectronics and Communications Conference/International Conference on Photonics in Switching 2016, no.PDP1-1, July 2016.

[11] R.M. Karp, "Reducibility among combinatorial problems," in Complexity of Computer Computations, R.E. Miller and J.W. Thatcher, eds., pp.85–104, Plenum Press, New York, 1972.

[12] J. Kleinberg and É. Tardos, Algorithm Design, Addison-Wesley, 2005.

Takehiro Sato received the B.E., M.E. and Ph.D. degrees in engineering from Keio University, Japan, in 2010, 2011 and 2016, respectively. He is currently an assistant professor in Graduate School of Informatics, Kyoto University, Japan. His research interests include communication protocols and network architectures for the next generation optical network. From 2011 to 2012, he was a research assistant in the Keio University Global COE Program, "High-level Global Cooperation for Leading-edge Platform on Access Spaces" by Ministry of Education, Culture, Sports, Science and Technology, Japan. From 2012 to 2015, he was a research fellow of Japan Society for the Promotion of Science. From 2016 to 2017, he was a research associate in Graduate School of Science and Technology, Keio University, Japan. Dr. Sato is a member of the IEEE.

Eiji Oki is a Professor at Kyoto University, Japan. He received the B.E. and M.E. degrees in instrumentation engineering and a Ph.D. degree in electrical engineering from Keio University, Yokohama, Japan, in 1991, 1993, and 1999, respectively. In 1993, he joined Nippon Telegraph and Telephone Corporation (NTT) Communication Switching Laboratories, Tokyo, Japan. He has been researching network design and control, traffic-control methods, and high-speed switching systems. From 2000 to 2001, he was a Visiting Scholar at the Polytechnic Institute of New York University, Brooklyn, New York, where he was involved in designing terabit switch/router systems. He was engaged in researching and developing high-speed optical IP backbone networks with NTT Laboratories. He was with The University of Electro-Communications, Tokyo, Japan from July 2008 to February 2017. He joined Kyoto University, Japan in March 2017. He is an IEEE Fellow.