Towards a Dataflow Platform in a Hierarchical Network: A Proposal for a Dataflow Component Management Method

SHINTARO ISIHARA1,a) TOYOKAZU AKIYAMA1,b)

Received: November 8, 2019, Accepted: June 1, 2020

Abstract: Massive dataflows are generated from IoT applications for controlling real-world environments. For maintaining the long-running IoT application costs, it becomes important to reduce network traffic and delay considering hierarchical network characteristics, e.g., leveraging edge resources. Research and development of such a dataflow platform are ongoing, and the platform enables application developers to build a new dataflow application by just combining reusable software components. However, the existing dataflow platform does not support component management, which includes application deployment and placement selection, assuming the hierarchical network environments. In this paper, to realize the dataflow platform in a hierarchical network, we proposed an approach to deploy components into heterogeneous environments using a major open-source product and to enable the large-scale deployment by considering network and resource conditions in the proposed platform. During the definition of the component placement problem, we also extracted parameters, which are dataflow platform parameters, dataflow application parameters and dataflow application component parameters, required to estimate component deployment budgets. Furthermore, we implemented a simple dataflow application based on two use cases and confirmed the validity of the proposed method.

Keywords: edge computing, cloud computing, internet of things

1. Introduction

Widespread Internet of Things (IoT) deployment causes the massive amount of stream data generated from sensing devices at various locations, which we call “dataflow”, and most of current IoT applications collect and analyze them in cloud environments. On the other hand, ongoing research and development of edge computing aim at reducing network traffic to the cloud and network delay by deploying calculation resources into the neighborhood of devices. Major edge computing projects, for example, Multi-access Edge Computing (MEC) [2], Fog Computing [3], Cloudlet [4] and so on, define similar network architectures considering two or three layer network hierarchies based on the distance from measurement and control environments. Considering such a “hierarchical network” is an important aspect for IoT application development. Dataflow platforms, e.g., Amazon Kinesis Data Streams [5], Azure Stream Analytics [6] and Google Cloud Dataflow [7], also play key roles in IoT application development. They enable an application developer to build a new application more easily by just combining multiple software components for processing dataflows and reduce the development time. The application developed by using these platforms is called a “dataflow application” [8]. Various cloud services provide dataflow processing functions, and IoT enabled dataflow platforms are also released for extending cloud functionalities to outside networks, e.g., Azure IoT Edge [9] and Google Cloud IoT Edge [10]. If traffic generated by IoT sensors or intermediate processes can be reduced before passing through gateways charged by communication carriers or cloud services, running costs of dataflow applications can be reduced (Fig. 1). Furthermore, since several types of dataflow applications obtain analyzed results for controlling field devices and require a shorter response time, introducing the intermediate layer is an important factor, and the communication delay is a key parameter to realize such a process automation. In that sense, locating dataflow application components into an appropriate network layer becomes an important point for developing IoT applications. Though the existing services possibly provide the deployment considering both delays and operational budgets, it is not available as open-source and cannot be leveraged. And since most of them are strongly coupled with a specific cloud service, it also cannot be used in the large-scale network across multi-domain networks. In order to prevent vendor lock-in and construct such a network, the flexibility for selecting and combining cloud services should be provided to the developers. We aim to realize a dataflow platform that can deploy IoT application components in appropriate locations considering both delays and operational budgets across multi-domain networks while using existing cloud services.

1 Graduate School of Frontier Informatics, Kyoto Sangyo University, Kyoto 603–8047, Japan
a) shintaro.stonefield@gmail.com
b) akiyama@cc.kyoto-su.ac.jp

This paper is an extension of our previous work [1].
Reference [8] proposed a dataflow platform considering the hierarchical network. It does not depend on a specific cloud service and proposes an interesting approach exploiting a pub/sub infrastructure for inter-component communication. However, it does not consider a management method of dataflow application components. And since the proposed inter-component communication method does not consider the status changes like CPU load and memory consumption at computing nodes, a messaging method supporting load balancing with such information should be considered. The above-mentioned issues can be solved by three functions, a component management method without vendor lock-in, flexible inter-component communication and a component placement method considering both delays and operation budgets. The orchestration function is also required to cooperate with those functions. While the existing dataflow platform [8] solves the inter-component communication issue partially, it still requires extensions. Our contributions are as follows:

- Introducing a dataflow orchestrator which cooperates a deployment layer selection function and functions for deploying and connecting the application components by leveraging existing middleware (Section 2.3)
- Proposing a component management platform in the large-scale network across multi-domain networks by using open-source software and container technology (Section 2.4)
- Consideration of a flexible inter-component messaging which supports load balancing by using label information in the computing resources management method (Section 2.5)
- Proposing a component placement method for considering both delays and operation budgets, and verifying its applicability with several use cases (Sections 3, 4)

In Section 2, the dataflow platform considering the hierarchical network and its advantages are explained. Then, an assumed environment and design of the platform, which are mainly component management, inter-component messaging and component deployment method, are described. After that, a detail of the component deployment method is described in Section 3, and its applicability targeting the several use cases is described in Section 4.

2. Dataflow Platform

In this section, the overview of a dataflow platform considering a hierarchical network is explained based on the proposal in Ref. [8]. In the following sections, firstly, the hierarchical network assumed in the dataflow platform will be described, and the merits of deploying the components in each network layer will be clarified. Then, dataflow application definitions and dataflow application components will be described using the example IoT applications for bus services, and the constraints for deploying the components will also be discussed. Reference [8] focuses on an inter-component routing method in a P2P overlay network and does not refer to dataflow application component management. In Section 2.3, the dataflow orchestrator is introduced for managing components, and in Section 2.4, we propose an implementation approach of component management using Kubernetes [11] which is widely used for constructing container environments and managing the containers. Furthermore, the relationship between the component management and the inter-component communication method using a pub/sub infrastructure as proposed in Ref. [8] will be described in Section 2.5.

2.1 A Hierarchical Network

In this paper, the three-layer network is assumed, and the name of three networks are “cloud network”, “edge network” and “device network”. “cloud network” connects the servers of cloud computing environments, and it is located at a distant data center from measurement and control environments. “edge network” connects servers deployed in a point of presence (POP) of a network service provider. “device network” connects targeted measurement and control IoT devices. Computing nodes in each layer have environments to run containers, and each dataflow application component is deployed as a container.

The components can be distributed geographically since the computing resources of device and edge networks are geographically distributed. Device and edge networks have a lower network delay compared with cloud networks, and bandwidth budgets also become lower if the traffic can be confined inside the network. On the other hand, since the devices have limited computational resources, it is necessary to consider which network is appropriate as a component deployment location according to processing elements and required resources. The advantages of distributing the components on a hierarchical network are described below.

Reduction of Communication Expenses

In many cases, enormous stream sensor data are analyzed and aggregated to the minimum necessary data. Such data reduction contributes to reducing communication expenses for transmitting them into cloud environments. For example, in fixed-rate contracts of mobile communications, a monthly fee is paid against the amount of communication data available per month. In this case, if the amount of communications exceeds the available data capacity, it is automatically charged, or the communication speed is limited. In cloud computing services like [9], [10], the amount of data transferred from and to the cloud environments are subject to charge. Though edge computing services are still in the proof of concept phase, the communication service providers may adopt the same fee structure, and a reduction of inter-network layer communication is expected to reduce the total budgets of IoT applications.

Low Delay Response

One IoT use case requires a high-speed response [12]. For example, in machine control, the real-world events must be detected.

![An example of traffic reduction in edge.](image-url)
as soon as possible, and a low network delay of less than 1 ms is required. In other cases, the low latency network produces better user experiences in VR/AR applications and improves the average viewing time of interactive digital signages. A dataflow platform can provide merits to the developers to realize a low delay response by deploying components near measurement and control devices according to needs of the above-mentioned dataflow applications. In order to leverage hierarchical network characteristics, dataflow application components should be placed into appropriate network layers based on the application requirements. How to describe dataflow application requirements and how to locate proper component placements are described in Section 3.

Here, we aim to construct the platform in a large-scale environment which includes many geographically distributed clusters. Although it seems difficult to find the best node from the whole computing resources because of the computational complexity, introducing the idea of network clusters as proposed in Ref. [13] simplifies the issue. Figure 2 shows an example of a cluster hierarchy in Japan. In this case, if the target application is deployed for a company providing a service to West Japan area, candidate computing resources of device and edge networks should be selected from the related clusters, i.e., “JP/West”. The cloud network can be located to “/JP” but cannot be “/US” for reducing the delay. In such cases, the number of target resources becomes smaller enough for calculating the optimal placement. In that platform, it is required to be able to manage multiple clusters distributed over a wide area integratively and manage computing resources by determining which clusters are present from the cluster id.

2.2 An Example Dataflow Application

In this section, how to construct a dataflow application is described using an example IoT application targeting bus services. The dataflow application can be represented as a dataflow graph as shown in Fig. 3. A dataflow graph can contain multiple dataflow applications. Each node $c_i$ represents a component of a dataflow application. The dataflow graph can represent a division of a dataflow by branch nodes and an integration of the multiple dataflows by synchronization nodes. Here, we assume that the dataflow graph is a directed acyclic graph (DAG) to avoid the calculation complexity. However, even if a user wants to include the loop, e.g., to improve the accuracy of results by reusing stored data in a DB like machine learning, it can be represented by adding a dataflow that has a DB component as an input node.

We consider the system proposed in Ref. [14], and target buses are operated by Kobe Minato Kanko Bus Inc. which is a bus company in Kobe City. The dataflow application described in Ref. [14] is represented as a dataflow graph shown in Fig. 3. The application is a passenger counter using the video stream obtained from a drive recorder in a bus. The counting procedure is divided into four components: (1) a bus stop detector to reduce false positives of the passenger counting, (2) a frame rate converter to reduce processing costs considering the computing resources, (3) a passengers’ movements extractor from the video images, and (4) a passengers’ statuses “getting on/off” estimator and a passenger counter. Here, the component (3) only transmits the results to the component (4) when the bus is located at a nearby bus stop for reducing the amount of network traffic. In this case, the outputs of components (2) and (3) can be reused by the other applications, for example, sudden movements of passengers could be used for detecting dangerous driving such as sudden braking or steering. Figure 3 includes both applications in one graph.

The target bus routes are set in Kobe City and Ashiya City. There are eleven point of presences (POPs) of NTT-West, which is a telecommunications company in the west area of Japan [15]. Mobile carriers, e.g., NTT DOCOMO, deploy their access points into NTT-West’s POPs, and the traffic passing through the access point is transferred to the central facility of each prefecture by using NTT-West’s lines [16]. Once the 5G service is ready, new services might be released by mobile carriers. While the mobile traffic is usually transferred over the virtual circuit, in the new services, it can directly go out to the servers located at NTT-West’s POPs, which is so-called “local breakout”. Such services increase edge computing possibilities. In the following sections, local breakout services are assumed to be ready at POPs in telecommunications companies.

2.3 Dataflow Orchestrator

Dataflow platform architecture discussed in this paper is shown in Fig. 4. The dataflow platform provides an execution environment of dataflow applications. As already discussed, it has
to manage application components and inter-component messaging. The existing component management platform provides flexible component customization functions as container images and start-up options for modifying its behavior to fit the environment. However, in order to handle dataflow graphs, an orchestration layer is required to combine the existing middleware, and we call a dataflow orchestrator. The main tasks of the dataflow orchestrator are (1) selecting a deployment network layer, (2) deploying/undeploying dataflow application components into/from the hierarchical network, and (3) connecting components flexibly by using a pub/sub infrastructure. Function (1) will be discussed later in Section 3. Here, implementing the above functions as an orchestrator requires heavy development tasks, e.g., integrating heterogeneous cloud APIs and supporting inter-cloud messaging. However, the implementation cost can be reduced by integrating the existing functions. In Section 2.4 and Section 2.5, how to realize the functions (2) and (3) by leveraging the existing function is described.

2.4 A Component Management Method

The network architecture covered by this research consists of three layers: cloud, edge and device networks, however, usually each network layer is provided by different resource providers, e.g., cloud service providers, telecommunication companies, bus companies and so on. Therefore, it is necessary to integrate and manage a hierarchical network over different management domain networks for deploying components to an arbitrary layer. In Ref. [17], MEC, Fog Computing, Cloudlet are compared as representative network architectures of edge computing. MEC, Fog Computing and Cloudlet target widely internet applications. However, MEC and Fog Computing are proceeded by enterprises, and there is a concern about lock-in by cloud businesses and network equipment vendors. In contrast, Cloudlet adopts an approach by utilizing open-source software, thus can realize component management over multiple edge/cloud networks while avoiding vendor lock-in. For that reason, we also adopt Cloudlet architecture. Here, virtual machine (VM) based Cloudlets [18] and OpenStack++ [19] which is extended to adapt Cloudlet are proposed for constructing Cloudlet environments across multiple clouds. On the other hand, when deploying the VMs on the end node devices, there is too much overhead, and the number of application components that can be deployed is limited [20]. In this paper, in order to reduce the size for deployments, we propose a component management method in a hierarchical network over different resource provider networks by leveraging container technologies. Moreover, since resource providers basically have different APIs to manage their resources, the dataflow platform must maintain the compatibility to each provider API. For example, Amazon Web Service [21], Google Cloud Platform [22] and Microsoft Azure [23] have different APIs, and the work of integrating them is complicated for dataflow platform administrators. To provide the integrated interface, we introduce the Kubernetes as an integration layer. Figure 5 shows an example deployment approach of Kubernetes over the multi-layer network. That approach can reduce the implementation costs of the dataflow orchestrator because it provides a single API. By leveraging the function to translate the provider API such as Rancher [24] and Virtual Kubelet [25], one can add the distributed computing resources to the underlying platform.

As discussed in Section 2.1, we need to consider network layers, i.e., cloud/edge/device, and network clusters, e.g., /JP/West, /JP/East, for deploying components into widely distributed resources. However, such resource attributes are not considered in the original Kubernetes. We propose to add those attributes to computing nodes as labels in Kubernetes. In Kubernetes, application components are registered as a container image, and its instance is deployed into a computing node. Here, Kubernetes provides a deploying method to decide the appropriate node by filtering the node which cannot satisfy the condition including labels. Based on the function, the dataflow orchestrator can deploy components into the placement decided by the method described in Section 3.

2.5 Inter-component Messaging Using a Distributed pub/sub Infrastructure

In this paper, it is assumed that components communicate with each other using a topic-based pub/sub infrastructure similar to [8]. Reference [8] proposed a load balancing method among the components subscribing the same topic. Each component has
an index range, and if one component publishes a message, it also specifies the index in addition to a topic. For example, a publisher can choose a scheduling strategy such as Round-Robin. Furthermore, deploying two components having the index range divided by two, computation offload and scale-out can be realized as shown in Fig.6. However, when a publisher specifies an index value, they cannot refer to destination component statuses, e.g., CPU load, memory consumption and so on. Here, a distributed pub/sub broker PIQT [26] is provided, that is compatible with standardized communication protocols including the MQTT protocol. PIQT provides more flexible inter-component messaging using the Suzaku [27] overlay network. While the details of the PIQT messaging functions are out-of-scope of this paper, our dataflow platform can be used with a topic-based pub/sub infrastructure as described in Ref. [8]. In order to support existing dataflow components, e.g., Node-RED [28], Fluentd [29], Apache Flink [30] and so on, a wrapper layer should be deployed together with them. In this case, each dataflow component subscribes and publishes to the wrapper layer instance instead of directly subscribing and publishing to the other components. Here, component statuses such as CPU load and memory usage are registered to the overlay network by this wrapper, and flexible messaging considering resource status can be realized. In the rest of this paper, components are assumed to communicate with the other components via topic-based pub/sub infrastructures, and the method to choose the network layer for each dataflow application component is focused on.

3. A Deployment Layer Selection Method

A dataflow application component included in a given dataflow graph should be deployed into a network layer in a hierarchical network. In this section, how to decide the network layer is described. If the deployment costs of a component can be defined for each network layer, it becomes a cost-minimizing problem. In the following, the problem of finding the cost-minimizing combination is formulated, and the deployment of example applications is examined with assumed parameters.

A dataflow graph can have branch and synchronization nodes. While our final goal is also realizing an arbitrary graph, most applications have a simple straight line dataflow graph without branch and synchronization, which we call a “dataflow path”. In some cases, the dataflow graph has branch or synchronization nodes, but the graph could be divided into several dataflow paths by using the depth-first search. Then, each dataflow path will be deployed considering the reuse of already deployed components with the same data source. In the following, only a “dataflow path” is considered for simplicity, and the consideration of complex dataflow graphs is a future subject.

A dataflow platform also allows a dataflow graph to expressely specify the target deployment layer because in some cases, deployable network layers are limited due to the requirements of the component, e.g., a sensor device is located in the device network, a large scale database with the past data is located in the cloud network and so on. In that sense, the automatic deployment of the whole dataflow graph is not assumed in our approach, and application developers can specify deployment layers of the whole or some part of the input graph.

The cost-minimization problem is formulated where all of the parameters are pre-extracted. When some parameters cannot be extracted, the target layer of the related components will be decided manually, and the algorithm just indicates a procedure to decide deployment layers. When the number of components which support the parameters described in Section 3.1 is increased, a fully automatic component deployment becomes possible.

3.1 A Cost Minimization Problem

Hereinafter, let \( N \in \mathbb{N} \) be the number of components of a dataflow application and \( M \in \mathbb{N} \) be the number of layers. A dataflow graph indicates an order of components \( c_i (i = 0, \ldots, N) \) as shown in the upper part of Fig.7. We assume that at least the data source component \( c_0 = c_i \) which means a sensor device and the output component \( c_N = c_e \) (\( N = 5 \) in Fig.7) are manually specified their deployment layers by application developers. The deployment layer of the other components \( (c_1, \ldots, c_{N-1}) \) will be decided by the algorithm described in the following section.

In order to decide the deployment layers of the components \( (c_1, \ldots, c_{N-1}) \), proper parameters should be extracted for the estimating deployment cost based on the assumption discussed in Section 2. Figure 7 shows possible candidate locations of the components for a given dataflow path in a hierarchical network. When a unit vector \( x_i \in \mathbb{R}^M \) is used to express the deployment location of a component \( c_i \), the decision variable matrix \( X \) can be represented as

![Fig. 6 A routing method using index assignment [8].](image1)

![Fig. 7 All combinations of the dataflow path deployment.](image2)
Each candidate location is considered to have a deployment cost. When the cost estimation method is defined, a deployment location can be decided based on the cost. The cost of each component is estimated from three metrics, the amount of data transfer, the communication/processing delay and the resource consumption considering the advantage of the hierarchical network. These metrics can be estimated using the parameters defined in Table 1, and the candidate deployment layers of \(c_i\) and \(c_{i-1}\) are represented as \(x_i\) and \(x_{i-1}\). The detail metrics definitions are described below.

### Data transfer

The cost metrics of data transfer related to the hierarchical network are estimated from inputs and outputs of components. The first component \(c_1\) of a dataflow path receives data from a sensor device and then sends processing results to the next component \(c_2\). Similar to \(c_1, c_2\) receives data from \(c_1\) and then processes and sends them to \(c_3\). It means that the amount of data received by a component \(c_i\) is equal to the amount of data outputted by \(c_{i-1}\), and the dataflow is continued to be processed until the last output component receives the data. Here, if the amount of \(c_i\)'s input data is defined as \(b_i\), the value is the same with the output data of the previous component \(c_{i-1}\) in the dataflow path. Then, the amount of \(c_{i-1}\)'s output data can be estimated by the amount of \(c_{i-1}\)'s input data \(b_{i-1}\) and \(r_{i-1}\) which is the input/output ratio of \(c_{i-1}\). Thus \(b_i\) can be represented as

\[
b_i = b_{i-1}r_{i-1} \quad (i = 1, \ldots, N). \tag{2}
\]

If a threshold is provided for considering the inter-layer communication budgets discussed in Section 2.1, it will be an indicator to calculate a cost metric for \(b_i\). The following matrix \(B_{\text{max}}\) indicates the upper bandwidth limit between \(x_{i-1}\) and \(x_i\) by calculating an inner product \(x_{i-1}^T \cdot B_{\text{max}} \cdot x_i\).

\[
B_{\text{max}} = \begin{bmatrix}
\infty & \text{(Dev \to Dev)} & \text{Dev \to Edge} & \text{Dev \to Cloud} \\
\text{Edge \to Dev} & \infty & \text{(Edge \to Edge)} & \text{Edge \to Cloud} \\
\text{Cloud \to Dev} & \text{Cloud \to Edge} & \infty & \text{(Cloud \to Cloud)}
\end{bmatrix}
\tag{3}
\]

The bandwidth of the same network is set to infinite for representing its lower budgets compared to the other communications. In some cases, \(B_{\text{max}}\) represents the upper limit of a physical network, and in the other cases, it shows an acceptable communication budget in a charged network so that we assume it is given by platform providers or application developers. Here, we introduce the cost function \(bw()\) for calculating the data transfer budgets as follows. It outputs the ratio of \(b_i\) (the bandwidth used by a component \(c_i\)) to the available bandwidth between inter-network specified by \(x_i\) and \(x_{i-1}\).

\[
bw(i, x_i, x_{i-1}) = \frac{b_i}{x_{i-1}^T \cdot B_{\text{max}} \cdot x_i} \tag{4}
\]

### Communication and processing delay

Inter-component communication and component processing delay become important factors for real-time applications. When we consider the communication delay between component \(c_{i-1}\) and \(c_i\) in the dataflow path, it does not depend on the type of the component but depends on the network layer where the component is deployed. We assume that the inter-layer delay is represented by a matrix \(D\), whose format is the same as \(B_{\text{max}}\)'s format. \(D\) indicates the communication delay between \(x_{i-1}\) and \(x_i\) by calculating an inner product \(x_{i-1}^T \cdot D \cdot x_i\). Hence, the processing delay depending on the type of component is given as \(d_i\), and the total acceptable delay is defined as \(d_{\text{max}}\). Then, the cost due to the communication and processing delay can be expressed as follows by introducing the cost function \(delay()\).

\[
delay(i, x_i, x_{i-1}) = \frac{x_{i-1}^T \cdot D \cdot x_i + d_i}{d_{\text{max}}} \tag{5}
\]

### Resource consumption

To reflect the advantage of a hierarchical network for communication budgets, consumed resources should also be taken into account. In the actual deployment phase, the component consumes resources in the deployed network layer, and the amount of consumed resources depends on the processing contents and the size of the input data. Here, we focus on "processor" and "memory" as resources of each network layer. The storage resource is also an important factor for the components with larger window size, for example, the components that calculate statistics of dataflows in a certain period require the storage for storing the dataflows during that period. However, we currently assume that the intermediate components are not required to have storage. Considering the storage resources is a part of future work. Furthermore, power consumption also becomes a critical factor for devices without power supply. In bus use cases, a battery of a bus can be used for sensor and actuator devices, though it is desirable to reduce power consumption. However, it is not so critical compared to the sensor network case. In this paper, we focus on the reduction of the load on a device network instead of power consumptions.

---

Table 1 Parameters used for estimating the deployment cost.

(a) Dataflow platform parameters

| Name | Description |
|------|-------------|
| \(M\) | Number of network layers |
| \(A_i\) | All available resources in each network |
| \(D\) | Communication delay between each network |

(b) Dataflow application parameters

| Name | Description |
|------|-------------|
| \(N\) | Number of components |
| \(c_i\) | \(i\)th Component \((i = 1, \ldots, N - 1)\) |
| \(a\) | Sensor device location |
| \(e\) | Last component location |
| \(b_i\) | Input data amount from \(c_{i-1}\) to \(c_i\) when the input data size of \(c_i\) is defined as \(b_i\) \((i = 1, \ldots, N)\). |
| \(B_{\text{max}}\) | Upper limitation of bandwidth between each network |
| \(d_{\text{max}}\) | Total acceptable delay for the entire dataflow application |

(c) Parameters related to dataflow application component \(c_i\)

| Name | Description |
|------|-------------|
| \(r_i\) | Output/input data ratio of \(c_i\) per unit time \((i = 1, \ldots, N)\) |
| \(p_i\) | Predicted resource consumption of \(c_i\) per unit time against input data \((i = 0, \ldots, N)\) |
| \(d_i\) | Processing delay of \(c_i\) \((i = 1, \ldots, N)\) |
While the resources consumed by the component are not constant, since it depends on the size of the input data, the average required resources can be predicted. In this platform, it is assumed that the computing resource usage of components, e.g., CPU and memory usage, is measured for several input data sizes in a measurement environment. Then, based on the results, the required resource estimation function \( \text{predictRequiredResources}(i) \) is derived. In the deployment phase, the required resources are predicted by using the function. On the other hand, the degree of resource consumption depends on the target layer \( x_i \). For example, the processor type such as CPU, the GPU and the operating frequency depend on the deployment layer. And the 1 GB memory consumption in the cloud network does not significantly affect the overall resources, but that negative effect becomes large in the device network since resources are not so much. Therefore, it is better to calculate them separately to be considered in the deployment layer selection. Our approach expresses the required resources of the component as the ratio to all resources in each layer so that even if the amount of resources used is the same, the influence according to the layer can be reflected. Let \( j \) \((1 \leq j \leq J \in \mathbb{N})\) be the number matched the number of types of resources such as CPU, memory, and storage, and then \( p_{ij} \) represents the required resources of a component \( c_i \) for the kind of resources \( j \) by introducing a function \( \text{predictRequiredResources}(i,j) \). The function predicts the amount of resource consumption of \( j \) for executing the component \( c_i \) from the kind of components, the input data amount and the deployed network layer.

\[
\rho_{ij} = \text{predictRequiredResources}(i,j) \quad (6)
\]

Here, let matrix \( A_j \) be all available resources of \( j \).

\[
A_j = \left( \begin{array}{ccc}
\alpha_{j,\text{device}} & \alpha_{j,\text{edge}} & \alpha_{j,\text{cloud}} \\
\end{array} \right) \quad (7)
\]

\( \alpha_{j,\text{device}}, \alpha_{j,\text{edge}} \) and \( \alpha_{j,\text{cloud}} \) represent all resources of \( j \) in each network layer. The platform manager can make the deployment layer selection method to be calculated with an arbitrary kind of resources. Here, in this platform, it is assumed that patterns of available resources such as “4 core/RAM 32 GB” for each network layer are presented by the dataflow platform provider. And the total resources reserved by the application developer in each layer are set to \( A_j \) as the maximum available resources. Then, the cost function \( \text{resource}(i,j) \) is introduced for calculating the ratio to an arbitrary kind of resources in the deployed layer.

\[
\text{resource}(i,j) = \frac{p_{ij}}{A_j \cdot x_i} \quad (8)
\]

### 3.2 Formulation

In the proposed method, the cost for each candidate deployment of a dataflow path is estimated by calculating each component deployment cost from \( c_i = c_0 \) to \( c_N = c_N \). Then, all candidates are examined, and the deployment path with a minimum total cost is found, which is a simple brute-force search. During the calculation, the conditions must be satisfied, and the paths exceeding the upper limit before reaching the final component are removed from the candidate lists. The objective function to be minimized and the constraints are shown below.

\[
\begin{align*}
\min & \sum_{i=1}^{N} f(i, x_i, x_{i-1}) \\
\text{s.t.} & \quad f(i, x_i, x_{i-1}) = a \cdot \text{bw}(i, x_i, x_{i-1}) \\
& \quad \quad + \beta \cdot \text{delay}(i, x_i, x_{i-1}) + \sum_{j=1}^{J} \gamma_j \cdot \text{resource}(i, x_i) \\
\end{align*}
\quad (9)
\]

\[
\begin{align*}
\text{Subject to} & \quad \text{bw}(i, x_i, x_{i-1}) \leq 1, \forall i \in \{1, \ldots, N\} \\
& \quad \text{delay}(i, X) = \sum_{k=1}^{i} \text{delay}(k, x_k, x_{k-1}) \leq 1, \forall i \in \{1, \ldots, N\} \\
& \quad \text{resource}(i, X) = \sum_{k=1}^{i} \text{resource}(k, x_k) x_k^T \cdot x_i \leq 1, \\
& \quad \forall i \in \{1, \ldots, N\}, \forall j \in \{1, \ldots, J\}
\end{align*}
\quad (10)
\]

Expression (9) indicates the objective function of the minimization problem. Eq. (10) represents the deployment cost of the proposed algorithm to two typical use cases and verify its applicability. In the next section, we apply the proposed algorithm to two typical use cases and verify its applicability.
Algorithm 1 Finding minimal cost dataflow path deployment (1)

1: function findMinimalCostDeployment(\(N, s, e, B_{\text{max}}, d_{\text{max}}\))
2: \(X \leftarrow M \times (N + 1)\) matrix is generated
3: \(X \leftarrow x_0 = s, x_N = e, \) others are initialized by zero
4: global \(\text{minCost} \leftarrow \infty\) (initialize with a sufficiently large number)
5: global \(\text{bestX} \leftarrow \text{NULL}\)
6: calculateDeploymentCost(X, 1, 0)
7: return \(\text{bestX}\)
8: end function
9: function calculateDeploymentCosts(X, i, cost)
10: if \(i\) is not \(N\) then
11: placements <- createNewPlacements
12: for \(x\) in placements do
13: if \(\text{satisfied}\), merge \(X, i, x\) is false then continue end if
14: calculateDeploymentCostsMerge(X, i, x, i+1, cost + \(f(i, x, x_{i-1})\))
15: end for
16: else
17: if \(\text{satisfied}\), \(X\) is false then return end if
18: cost <- cost + \(f(i, x, x_{i-1})\)
19: if \(\text{minCost} > \text{cost}\) then
20: \(\text{minCost} \leftarrow \text{cost}\)
21: \(\text{bestX} \leftarrow X\)
22: end if
23: return
24: end if
25: end function

Algorithm 2 Finding minimal cost dataflow path deployment (2)

1: function createNewPlacements
2: return The standard basis vectors in \(\mathbb{R}^M\)
3: end function
4: function merge(X, i, x)
5: return A matrix that the vector \(x\) is merged on the \(i\)th column of \(X\)
6: end function
7: » Although the cost calculation of \(\text{satisfied}\) can have further optimizations, the above-mentioned expression is used for readability
8: function satisfies(i, X)
9: if \(\text{not} b_{\text{true}}(i, x, x_{i-1}) \leq 1\) then return false end if
10: if \(\text{not} T_{\text{stop}}(i, X) \leq 1\) then return false end if
11: for \(j\) in \(J\) that the number of resource kinds do
12: if \(\text{not} T_{\text{resource}}(j, X) \leq 1\) then return false end if
13: end for
14: return \(\text{TRUE}\)
15: end function

4. Evaluation

Dataflow applications for bus services are our current target to consider the platform requirements. In this section, we focus on four dataflow paths just extracted by using a depth-first search in a dataflow graph. Then, we demonstrate how to find an appropriate components deployment for each dataflow path in our proposed method. We also implement sample applications to extract parameters. In the following, firstly the implemented applications and measurement results of parameters are explained.

Four components constituting the dataflow paths shown in Fig. 8 are implemented. \(c_1\) of (a, b) is a bus stop detection component, \(c_1\) of (c, d) is a frame rate converter component, \(c_3\) is a human skeleton detection component, and \(c_3\) is a component detecting passengers who are getting on/off the bus and dangerous driv-

Fig. 8 Measurement environments.

ing. Here, while [14] counts the number of people using the random forest regression, a detection process using a created model does not require a large amount of computational resources, thus a simple human count component is used for \(c_3\) in this paper. We assume that components communicate via an MQTT Broker Moquette [31] and are deployed as shown in Fig. 9. In the bus stop detection, whether the bus is near the bus stop is judged with a GPS and speed data are sent from bus sensors, and then it publishes the result as true/false with the “bus_stop” topic to the MQTT Broker using Mosquitto MQTT client library [32]. GPS data is matched with bus stops by using the search function of the Elasticsearch database [33], in which bus stop locations on the route of the target bus are stored. In the frame rate converter, MPEG data output from the camera are acquired and converted to JPEG using video4linux. Then, JPEG data are associated with a “skeleton” topic and published to the MQTT Broker using Mosquitto. The output rate of the camera is 30 fps and the frame rate converter reduces that into 1 fps. The human skeleton detection component subscribes to the “skeleton” topic using Paho MQTT client library [34] and receives image data. It extracts the skeleton from the received image utilizing OpenPose [35] and then publishes the extracted skeleton data to a “count” topic on the MQTT Broker using Paho. Here, when this component is used as \(c_3\) in (a, c), i.e., it is used for a passenger counter application, it subscribes to the “bus_stop” topic, and it publishes the results only if the data associated with the topic is “true”. Similar to the above component, the human count component subscribes to the “count” topic using Paho library and publishes the counted results to a “DB” topic on the MQTT Broker.

In this evaluation, the CPU usage and the memory usage are considered in a resource cost. It means the number of resource kinds \(J\) is 2. The CPU and memory usage are measured using top command during the application execution, and the traffic between each component is recorded using Wireshark. The specifi-
Algorithm 1 and Algorithm 2, in order to confirm the progress of network resource and the bandwidth are set to satisfy the consumption of the equipment used for the measurement are shown in Table 2. The CPU usage, the memory usage and the calculated input/output ratio based on the measured values of each component are shown in Table 3. For simplicity, the resource consumption is assumed to be constant regardless of the amount of input data and the class of assigned CPU core. It means that the processing delay $d_i$ is not affected by the network layer and $d_i$ is omitted in the deployment cost calculation. In the following, the proposed deployment method is applied to each use case using extracted parameters.

**USE CASE 1: Getting on/off passenger counter application**

The first application is a getting on passenger counter” shown in Fig. 8 (a), (c). The parameters of the platform and an application are shown in Eq. (14). We assume that deployment network layers of a sensor device and an output component are a device network and a cloud network respectively, and also, each network resource and the bandwidth are set to satisfy the condition “the device network < the edge network < the cloud network”. Mobile communications such as LTE are assumed for considering the upper limit bandwidth. Here, in this paper, the 5G feature is assumed, however, in the evaluation, in order to consider the bandwidth and budgets, LTE is referred. Most of the cheap SIMs provide a fixed charge menu, and one of the popular menus is 3 GB per month. \(B_{\text{max}}\) is calculated using that value. In the case (a), \(b_1\), which is the amount of input data to the first component \(c_1\), is 2.07 Mbps, and in the case (c), it is 0.01 Mbps. The result1 of path (a)

| \(c_i\) cockpit | \(c_2\) skeleton | \(c_3\) count | \(c_4\) DB |
|-----------------|-----------------|--------------|----------|
| \(r_1\)        | 0.01            |              |          |
| \(p_1,1\)      | 0.69            | 0            | 0        |
| \(p_2,2\) [MB] | 1340            | 0            | 0        |

The result1 of path (c)

| \(c_i\) cockpit | \(c_2\) skeleton | \(c_3\) count | \(c_4\) DB |
|-----------------|-----------------|--------------|----------|
| \(r_1\)        | 0.03            | 0.01         | 0.18     |
| \(p_1,1\)      | 0.1             | 0.4          | 0.1      |
| \(p_2,2\) [MB] | 0.5             | 214          | 21       |

The calculation, sample calculation results of intermediate components \(c_i\) are also described in the following. The part of results when priority parameters are set to \(\alpha = 1, \beta = 0, \gamma_1 = 0.5\) and \(\gamma_2 = 0.5\) is shown in Table 4 (a), (c). It shows the deployment cost of a dataflow path until the component indicated in the \(c_i\) column. The letters written in the “deployed layers” column represent the network layer (D: the device network, E: the edge network, and C: the cloud network) where the components \(c_1\), \(c_2\), \(c_3\) and \(c_4\) are deployed, and the “cost” column shows the estimated cost of the deployment. The proposed method excludes the combinations do not satisfy the constraints defined in Section 3, however, for the comparison, it is also shown in Table 4 with a \(\times\) mark.

The output data from \(c_1\) of a dataflow path (a) is only leveraged in \(c_2\), and its processing is considered in a dataflow path (c). Therefore, only \(c_1\) deployment cost is calculated in Table 4 (a). From the result, \(c_1\) of the path (a) cannot be deployed in the device network because of the lack of memory, but it can be deployed in the edge or cloud network. Then, the result of Table 4 (c) is explained. Since the data transfer usage is large until the human skeleton detection \(c_2\) is executed and exceeds the upper limit bandwidth, it is not deployed in the edge or the cloud network.

By \(c_2\) processing, the data transfer usage is reduced sufficiently, and \(c_3\) can be deployed to any network layer but if it is deployed outside the device network, the cost is increased due to data transfer usage. As a result, D-D-D-C deployment shows minimal cost. However, if \(c_1 \sim c_3\) are deployed in the device network, the CPU usage will be 80%, and it becomes difficult to deploy the other components to the same location. Therefore, increasing the priority of the cost for CPU usage, the result when the parameters are set to \(\alpha = 1, \beta = 0, \gamma_1 = 0.7, \gamma_2 = 0.5\) is shown in Table 4 (b). As a result, it shows D-D-C-C is better than D-D-D-C. From the above result, it is confirmed that the cost due to the resource consumption is emphasized and the number of components deployed in the device network can be reduced by tuning the priority.

**USE CASE 2: Dangerous driving detector application**

The second application is a “dangerous driving detector” shown in Fig 8 (b), (d). While the application also consists of
four components, $c_3$ and $c_4$ are different from the use case 1. They detect dangerous driving and notify to a driver. In this use case, deploying an output component to the device network and the response in a short time are required. In order to reflect the requirements, a communication delay for each network layer is newly defined, and the allowable maximum delay $d_{\text{max}}$ is set to 20 ms. The parameter added and changed for use case 2 is shown in the Eq. (15).

$$d_{\text{max}} = 20 \, [\text{ms}], \quad e = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad D = \begin{bmatrix} 0 & 1 & 10 \\ 1 & 0 & 7 \\ 10 & 7 & 0 \end{bmatrix} \, [\text{ms}]$$

(15)

The results with the priority parameters set to $\alpha = 1$, $\beta = 1$, $\gamma_1 = 1$, $\gamma_2 = 1$ are shown in Table 5 (a) and (b). The result of Table 5 (a) is similar to the result of the use case 1, but the cost when the component is deployed in the edge network is low by considering the delay. In Table 5 (b), most deployment candidates are excluded due to the upper limit bandwidth, and only two patterns D-D-E-D and D-D-C-D survived. D-D-D-D is calculated as the minimal cost deployment, it is rejected due to exceeding the resource cost. In many cases, it is difficult to deploy all components in the device network, thus the deployment cost is appropriately calculated considering both of the delay and the resources. In this case, while D-D-E-D is proposed for the deployment, from the above reason, it is considered that the resource shortage of the device network is not correctly expressed by the parameters. It is caused by the omission of differences of processor classes and dependency on the input data size.

From the above results, it is clarified that the cost metrics can be properly adjusted by the priority parameters of the proposed method in both use cases. On the other hand, the differences in the computing environments of each network layer are not reflected sufficiently. Therefore, the differences in hardware such as CPU and GPU in the network layer must be considered with the component processing time $d_i$ described in Section 3.1 including the consideration of power consumption. However, since it is difficult to measure changes in processing performance caused by hardware differences, further investigation and consideration are required.

5. Related Work

Reference [36] also considered to realize a dataflow platform by exploiting Node-RED [28]. Although Node-RED can easily define a dataflow application using GUI, it cannot handle the components over multiple nodes such as physical machines. For resolving the problem, Reference [36] extended the communication function using MQTT Broker for specifying the processing node. The dataflow graph defined by the user is divided into several sub dataflow graph, however, it is not discussed how to select the node to deploy the component.

References [9] and [10] can manage clouds and edges collectively and deploy a dataflow graph, however, a vendor lock-in cannot be avoided and a flexible service composition cannot be realized.

References [37] and [38] proposed component placement methods for a MEC environment. Reference [37] tries to solve a similar problem tackled in this paper. They consider a deployment problem of an application graph, which represents application requirements such as the CPU load and the network bandwidth, to a physical graph, which represents resource statuses of physical nodes and links. Their method minimizes the maximum resource utilization of physical nodes and links for load balancing. However, the communication delay is not considered to reduce computational complexity. It becomes critical for real-time applications. Our proposal considers the communication delay but focuses on inter-cluster communication and omits the trivial delay caused by the intra-cluster communication for simplifying the model. On the other hand, currently, since our algorithm does not cover arbitrary branch and synchronization nodes, it should be considered in the future extensions. Reference [38] considers the mobility of the users and proposed a component placement method minimizing the communication cost depending on the distance between users and edge servers. However, in their model, it is assumed that the execution period of applications and user mobility characteristics are known in advance, hence it is not practical. Even in the case of the scheduled transports, e.g., buses and trains, they could cause unpredictable delays. In order to assume a realistic preconditions, our proposal does not rely on prior knowledge and separates component placement and inter-component communication functions. It enables dynamic inter-component routing based on the monitored status considering mobile devices. In this case, communication costs can be reduced by preparing redundant components beforehand and dynamically re-connecting the components according to the movement of a user.

Reference [39] proposed a network function placement method to minimize the latency of all clients in MEC. It finds an optimal combination of all clients and base stations for the minimization. If an application requires data aggregation at POPs, a similar combination must also be considered in our proposal. However, our proposal currently leaves the base station selection to underlying 5G services and focuses on the controls of the application layer.

Reference [40] tackled a joint communication and resource allocation for minimizing the delay of all devices in MEC environments. Collaborating the edge and the cloud is similar to our proposal, however, targets are different. Our targets are application deployment of individual services considering delays and budgets, but their targets are optimization of entire delays. As described in Ref. [40], since they calculate the summed up delay of all devices, it is possible not to satisfy each service latency re
requirement. In our case, it is critical, and their proposal cannot be applied to our case. On the other hand, in the view of the dataflow platform provider, minimizing the whole delay is required, and it is possible to be applied.

Anthos [41] is a platform utilizing Kubernetes [11], Istio [42], Knative [43] and so on, to realize a service mesh. A service mesh is one of the technology to process the dataflow, and Anthos realizes that by using a Kubernetes’s component management function, an Istio’s service discovery function to connect components and a Knative’s multi-cloud environments construction function. Since Anthos is provided by Google, the cloud function is locked in, but the other cluster such as on-prem can be combined by the function of Knative. Thus, the large-scale network can be constructed. While Kubernetes does not directly provide the component deployment function considering the network hierarchies, it provides a label function. The label function has a possibility to identify the network layer as a cluster id, but it requires an extension similar to our proposal. The purpose of Anthos is like our proposal, and there are many interesting features, however, the deployment layer selection method is not proposed.

6. Summary and Future Work

In this paper, a data platform considering a hierarchical network was proposed. Based on the proposal in Ref. [8], a dataflow orchestrator is newly introduced, and the component management approach and the component deployment layer selection method are proposed. In a component placement method, in order to evaluate the placement, the cost metrics considering data transfer, resource consumption and communication/processing delay of components were defined. We proposed a method for finding cost-minimizing component placement by extracting and classifying the parameters required to estimate the costs. Finally, we implemented a simple dataflow application and verified the validity of our proposed method using two use cases.

In this paper, we assumed that even if the targeted dataflow graph has a complex topology including branch/join, a critical dataflow path that requires the most dominant delay and operation budgets can be extracted uniquely. And based on the assumption, we have proposed the simple algorithm that assigns resources in order from the extracted critical path. On the other hand, if the critical path is not uniquely determined, the total delay and budgets may become worse by the path assigned later. Currently, in our use case, since the paths including the frame rate converter and the passenger movement extractor become the most dominant and the other divided paths are partially reusing the dominant path, the later assigned paths will not produce the worse results. However, the complicated dataflow graph may include such a situation. It is necessary to consider a method for extracting the candidate critical paths and allocating resources for the extracted paths.

The proposed placement method assumes that the required resources are estimated by the function predictRequiredResources(). However, depending on the component implementation, it becomes difficult to estimate the required resources only from the input data size. Further research is required for better estimation.

We assume there is only one network in each hierarchy, and it is not clear how to select the optimum network when there are multiple upstream hierarchies. For example, when the data are sent from a moving bus, and if there are several POPs that can be used for processing, it is impossible to select a POP in the closest distance without flexible messaging functions as discussed in Section 2.5. Furthermore, since the proposed method for estimating computing resource consumption was too simple for reflecting practical usages, the continuous investigation is required.

Acknowledgments

This work was partially supported by JSPS KAKENHI Grant Numbers 20H04183 and 20H04186.

References

[1] Ishihara, S., Tanita, S. and Akiyama, T.: A Dataflow Application Deployment Strategy for Hierarchical Networks, IEEE 43rd Annual Computer Software and Applications Conference (2019).
[2] Hu, Y.C., Patel, M., Sabella, D., Sprecher, N. and Young, V.: Mobile Edge Computing - A key technology towards 5G, ETSI White Paper, Vol.11 (2015).
[3] Bonomi, F., Milito, R., Zhu, J. and Addepalli, S.: Fog Computing and its Role in the Internet of Things, Proc. 1st Edition MCC Workshop Mobile Cloud Comput., p.1316 (2012).
[4] Flinn, J. and Satyanarayanan, M.: Energy-Aware Adaptation for Mobile Applications, Proc. 17th ACM Symp. Operating Systems Principles, pp.48–63 (1999).
[5] Amazon Web Services, Inc.: Amazon Kinesis Data Streams (online), available from (https://aws.amazon.com/kinesis/data-streams/) (accessed 2019-10-09).
[6] Microsoft, Inc.: Azure Stream Analytics (online), available from (https://azure.microsoft.com/en-us/services/stream-analytics/) (accessed 2019-10-11).
[7] Google, Inc.: CLOUD DATAFLOW (online), available from (https://cloud.google.com/dataflow/) (accessed 2019-02-13).
[8] Teranishi, Y., Kimata, T., Yamanaoka, H., Kawai, E. and Harai, H.: Dynamic Data Flow Processing in Edge Computing Environments, IEEE 41st Annual Computer Software and Applications Conference (2017).
[9] Microsoft, Inc.: Azure IoT Edge (online), available from (https://azure.microsoft.com/en-us/services/iot-edge/) (accessed 2019-10-13).
[10] Google, Inc.: Google Cloud IoT Core - Cloud IoT Edge (online), available from (https://cloud.google.com/iot-core/) (accessed 2019-02-13).
[11] The Linux Foundation.: Kubernets: Production-Grade Container Orchestration (online), available from (https://kubernetes.io/) (accessed 2019-01-15).
[12] Schulz, P., Matthe, M., Kilessig, H., Simsek, M., Fettweis, G., Ansari, J., Ashraf Ali, S., Almeroth, B., Voigt, J., Riedel, I., Puschmann, A., Mitschele-Thiel, A., Muller, M., Elste, T. and Windisch, M.: Latency critical IoT applications in 5G: Perspective on the design of radio interface and network architecture, Proc. IEEE Communications Magazine, Vol.55, No.2, pp.70–78 (2017).
[13] Teranishi, Y., Banno, R. and Akiyama, T.: Scalable and Locality-Aware Distributed Topic-Based PalySub Messaging for IoT, Proc. IEEE Globecom 2015, pp.1–7 (2015).
[14] Nakashima, H., Arai, I. and Fujikawa, K.: Passenger Counter Based on Random Forest Regressor Using Drive Recorder and Sensors in Buses, IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops) (2018).
[15] SoftBank Corp.: Mappings of NTT-west PoPs (in Japanese) (online), available from (http://www.eaccess.net/cgi-bin/servicearea_ext.cgi?address=28) (accessed 2019-02-07).
[16] TTT DOCOMO, INC.: Reliable system for disaster preparedness in Japan (Japanese) (online), available from (https://www.nttdocomo.co.jp/corporate/csi/disaster/reliable/system/index.html) (accessed 2019-02-07).
[17] Wang, S., Zhang, X., Zhang, Y., Wang, L., Yang, J. and Wang, W.: A Survey on Mobile Edge Networks: Convergence of Computing, Caching and Communications, IEEE Access, Vol.5, p.67576779 (2017).
[18] Satyanarayanan, M., Bahl, V., Caceres, R. and Davies, N.: The Case for VM-based Cloudlets in Mobile Computing, IEEE Pervasive Comput., Vol.8, No.4, pp.14–23 (2009).
[19] Ha, K. and Satyanarayanan, M.: OpenStack++ for Cloudlet Deployment, School of Computer Science Carnegie Mellon University Pittsburgh (2015).
[20] Bernstein, D.: Containers and Cloud: From LXC to Docker...
Toyokazu Akiyama received his B.E., M.E. and Ph.D. degrees in Information Systems Engineering from Osaka University, Japan in 1997, 1999 and 2003, respectively. Since 2000, he worked as a Research Associate (Assistant Professor) at the Cybermedia Center, Osaka University. Currently, he is a Professor of the Faculty of Information Science and Engineering, Kyoto Sangyo University. His research interests include distributed systems and IoT applications. He is a member of the IEEE Computer Society, IEICE. He is a vice-chair of JSPS 163rd Committee on Internet Technology and an assistant of a Secretary-General of Cyber Kansai Project.

Shintaro Ishihara received his B.E degrees Faculty of Computer Science and Engineering and M.E. degrees in Graduate school of Frontier Informatics from Kyoto Sangyo University, in Japan in 2016 and 2018, respectively. Currently, he is a Ph.D. student of the same university. His research interests include Edge Computing and IoT.