2018 Third ACM/IEEE Symposium on Edge Computing

In-situ Resource Provisioning with Adaptive Scale-out for Regional IoT Services

Yugo Nakamura[1,2], Teruhiro Mizumoto[1], Hirohiko Suwa[1], Yutaka Arakawa[1], Hirozumi Yamaguchi[3], Keiichi Yasumoto[1]
[1]Nara Institute of Science and Technology, Ikoma, Nara 630-0192, Japan
[2]Research Fellow of the Japan Society for the Promotion of Science
[3]Osaka University, Suita, Osaka 565-0871, Japan
Email: {nakamura.yugo.ns0, yasumoto}@is.naist.jp

***Abstract*—In an era where billions of IoT devices have been deployed, edge/fog computing paradigms are attracting attention for their ability to reduce processing delays and communication overhead. In order to improve Quality of Experience (QoE) of regional IoT services that create timely geo-spatial information in response to users' queries, it is important to efficiently allocate sufficient resources based on the computational demand of each service. However since edge/fog devices are assumed to be heterogeneous (in terms of their computational power, network performance to other devices, deployment density, etc.), provisioning computational resources according to computational demand becomes a challenging constrained optimization problem. In this paper, we formulate a delay constrained regional IoT service provisioning (dcRISP) problem. dcRISP assigns computational resources of devices based on the demand of the regional IoT services in order to maximize users' QoE. We also present *dcRISP+*, an extension of dcRISP, that enables resource selection to extend beyond the initial area in order to satisfy increasing computational demands. We propose a provisioning algorithm, in-situ resource area selection with adaptive scale out and in-situ task scheduling based on a tabu search, to solve the dcRISP+ problem. We conducted a simulation study of a tourist area in Kyoto where 4,000 IoT devices and 3 types of IoT services were deployed. Results show that our proposed algorithms can obtain higher user QoE compared to conventional resource provisioning algorithms.**

***Keywords*—Internet of Things, Edge computing, Resource provisioning, Quality of Experience.**

I. Introduction

Recently, Internet of Things (IoT) has been attracting considerable attention due to its high potential for drastically changing our society. The number of IoT devices grows exponentially and is expected to reach more than 50 billion by 2020 [1]. It is expected that a variety of IoT services aimed at improving the quality of life of people and the values of communities are being developed. An example is the Smart and Connected Communities (SCC) framework[1] promoted by the US government which aims to deploy low-cost, sustainable and resilient IoT devices in cities. We need a platform to realize and maintain delay-sensitive, low-cost regional IoT services in order to make smart communities a reality. Regional map services with various timely information (e.g., crowdedness at shops/sightseeing spots, buses/vehicles location) and community-wide mutual elderly monitoring services are typical examples of delay-sensitive regional IoT services. These services must also be resilient to infrastructure network failures (e.g., cellular network) due to natural disasters like earthquake/floods and to privacy data leakage by attacks.

[1]https://www.nitrd.gov/sccc/

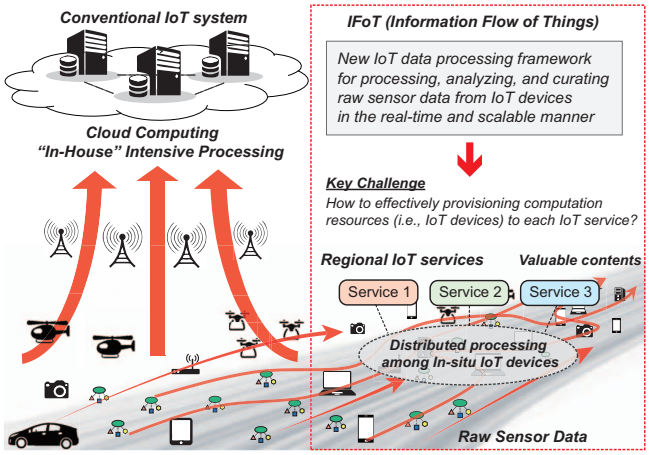


Fig. 1. Key challenge for realizing IFoT (Information Flow of Things)

Currently, cloud computing is considered as a preferred platform for IoT services [2]. However cloud-based approach may not be suitable for regional IoT services. Since the scale of regional IoT service is smaller than the global IoT service, service deployment to the cloud servers may be an over specification and may result in high operating costs. Also, cloud-based approach will not work when the connection with the servers are lost due to network failure. Some regional IoT services like vehicle location service and elderly monitoring have delay constraints, but cloud-based approaches may violate this constraint since executing tasks for IoT data processing at a server far from the data source includes more communication overhead. To address the limitations of the cloud-based approach, a new and promising computing paradigm called Fog or Edge Computing has recently been attracting attention [3] [4]. These paradigms adopted an approach that assigns delay-sensitive tasks to computation resources closer to the service users.

Inspired by these trends, we proposed a new IoT data processing framework called the *Information Flow of Things (IFoT)* [5], for processing, analyzing, and curating IoT data in a real-time and scalable manner based on distributed processing among in-situ IoT devices as shown in Fig 1. The IFoT framework flexibly utilizes computation resources of edge IoT devices existing at the data source, aiming to realize and maintain low-cost, delay-sensitive regional IoT services efficiently. In delay-sensitive regional IoT services that we are targeting in this paper, raw sensor data must be quickly

978-1-5386-9445-9/18/$31.00 ©2018 IEEE
DOI 10.1109/SEC.2018.00022

(within delay constraints) converted to valuable information or an IoT content, like *Smart xx map* (where xx can be anything), by executing a set of tasks programmed in a task graph according to users' requests. A key challenge then is how to effectively assign computational resources (i.e., IoT devices) to each IoT service while considering QoE. In addition, such a distributed system of heterogeneous IoT devices have varying computing power, network performance between devices, devices deployment density, etc. Thus, efficiently provisioning computation resources of edge IoT devices to satisfy the computation demand for each service becomes a challenging constrained optimization problem. We refer to this problem as the delay constrained Regional IoT Service Provisioning (*dcRISP*) problem.

In this paper, we formulate the dcRISP problem and its extended version called *dcRISP+* problem. Given computational resources graph in the target area and the set of services, queries and task graphs for all queries, the dcRISP problem is finding how to assign processing tasks queried by users to IoT devices, such that the user's QoE is maximized. To solve these problems, we propose *in-situ resource area selection with adaptive scale-out* and *in-situ task scheduling based on a tabu search* algorithms.

Our contributions in this paper are summarized as follows:

1) We formulated a delay constrained regional IoT service provisioning (dcRISP) problem and the extended version *dcRISP+* which aim to create regional IoT service contents efficiently while maximizing users' QoE.
2) We concretely design algorithms composed of in-situ resource area selection with adaptive scale-out and a tabu search based task scheduling to solve the dcRISP+ problem in reasonable time.
3) We conducted a simulation to evaluate the effectiveness of the proposed methods in the target regional area given a famous tourist area in Kyoto where 4,000 IoT devices and 3 types of IoT services were deployed. We confirmed that tabu search and adaptive scale-out could improve users' QoE for delay-sensitive IoT services compared to conventional methods.

II. Related Work

Fog Computing [3] and Edge Computing [4] are novel paradigms that mitigate server load by processing data on the fog/edge server located near IoT devices. Moreover, these approaches delegate the processing tasks of cloud servers to other distributed systems to realize service components such as proximity, intelligence, trust and control outside the cloud. In general, in Edge and Fog computing platforms, the edge nodes have roles such as IoT device management, network management and data processing and transferring [6], [7]. OpenFlow switches using Software-defined network (SDN) technology as novel edge platforms, which perform network management such as QoS control, mobility management, user management, and AAA (Authentication, Authorization, and Accounting), have been proposed recently [8], [9]. Moreover, IoT edge computing platforms for Home Energy Management (HEM) [10], Intelligent Transport System (ITS) [11], and disaster information gathering [12] have also become popular. Other systems such as Tasklets [13], [14], developed for the easy integration of heterogeneous computing systems, and distributed computation offloading algorithms [15], achieving efficient computation offloading between edge and cloud in a distributed manner have been proposed.

Optimizing assignment of tasks to processors in distributed computing systems has been of interest for many years. Shem et al. [16] proposed a task assignment model for distributed computing systems, based on a graph matching approach. They expressed task assignment as a graph matching problem which is an optimization problem, with the goal of finding a homomorphism from the task graph to the processor graph with minimum task turnaround time. However, they only solved the task assignment for single task graph by applying the state-space search algorithm. Salman et al.[17] proposed a task assignment algorithm based on particle swarm optimization to solve the multiple task assignment problem for homogeneous distributed computing system. However, their approach can not be applied to heterogeneous IoT-based fog computing environments. In this paper, we attempt to solve multiple task graph assignment on heterogeneous environments.

Cloud computing system is a type of distributed computing system and the task assignment in cloud computing systems is of interest similar to the distributed computing systems. The main purpose of the research [18][19][20] for task assignment in cloud computing systems is to adaptively provide the processing resources while meeting the deadline for all jobs in cloud applications while keeping running costs (money) to a minimum. They did not consider the limitation of processing resources because they assumed a nearly infinite number of virtual machines as resources. In addition, they have not considered the data transfer delay even though they have considered the processing delay because they are focused on minimizing the running costs of virtual machines. Therefore these approaches are not suitable for the IoT-based edge computing environment due to its limitations on available resources and transfer delay between tasks.

Provisioning resources in Edge and Fog computing systems has been gaining attention recently. Xu et al. [21] proposed a platform that performs location-based and time-sensitive applications such as location-based augmented reality games in edge computing systems. The platform provisions the processing resources from micro data centers on the edge or large-scale data centers on the cloud. However, the platform needs high capital and maintenance costs because it requires the deployment of the many edge servers to cover a large service area and to reduce delay. Ardagna et al. [22][23] proposed a resource provisioning platform combining cloud computing and fog computing. This platform adopts an approach that provides the processing resources of a fog colony (the set of fog nodes) and provides the cloud resources only if additional resources are needed. However, even with increased demand, this approach does not scale-out beyond the fog colony, resulting in delays.

Our approach in this paper is different from these existing studies. To the best of our knowledge, this paper presents the first formal study of the regional IoT services only with in-situ edge IoT devices. Our proposed approach provides

a novel and efficient mechanism for resource management and task assignment/execution while taking into account delay constraints of the IoT service and users' QoE.

III. Regional IoT Service Provisioning Problem

This section describes assumptions on the target regional IoT area and services, and designs the *delay constrained Regional IoT Service Provisioning (dcRISP)* problem that assigns service tasks to IoT devices for execution in an in-situ manner.

A. Assumptions

1) Target regional area: Fig. 2 shows our *target regional area*. In Fig. 2, letters are arranged from top to bottom and left to right. In target regional area, where many *IoT devices* (Fig.2.j) are already deployed and each IoT device connect to a *regional network* (Fig.2.b) via nearest base stations (Fig.2.c). We assume that each IoT device has some computation resource that can be utilized to execute assigned tasks (assuming Raspberry Pi computation power). Let R denote the *resource graph* that represents computation resources (i.e., IoT devices) and the overlay links connecting them. Specifically, we define $R = (P, L)$ as a graph where P is the set of vertices (IoT devices) and L is the set of undirected links (overlay links between two vertices). We call each vertex in R, *processor*.

2) Regional IoT services: In the target area, let $S = \{s_1, s_2, ..., s_n\}$ denote the set of *regional IoT services* (Fig.2.a) available. We assume the presence of regional IoT services that generate up-to-date geospatial contents (Fig.2.e). A typical example is *smart real-time map service* that allows a user to view a live street map of any place in the target area with various timely information (based on sensor data) such as shops/spots/streets crowdedness, temperature, UV, pollution (e.g., the density of PM2.5), scenic information (e.g., flowers) at every location, and/or vehicles positions for efficient transportation and driving safety.

Each IoT service s has some tuple parameter $\langle area(s), pd_s, md_s, SR, TR, G_s \rangle$. Here, $area(s)$ is called the *service area* (Fig.2.g) of s where the data generated by IoT devices deployed in $area(s)$ are utilized to provide the service s. pd_s and md_s are *preferable* (i.e., satisfactory) delay and *marginal* (i.e., tolerable) delay thresholds, respectively, Here, $0 < pd_s \leq md_s$. SR and TR are the maximum spatial resolution (e.g., number of points per m^2) and temporal

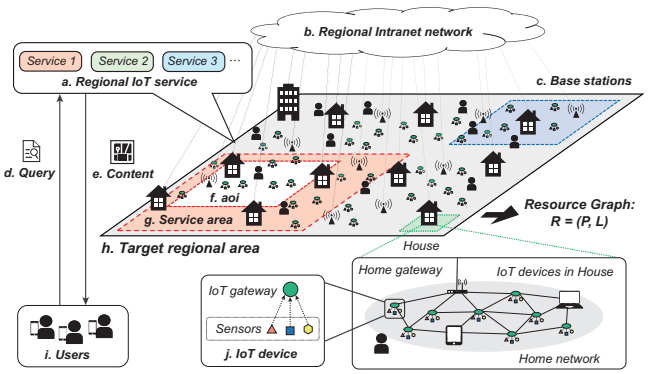


Fig. 2. Target regional area

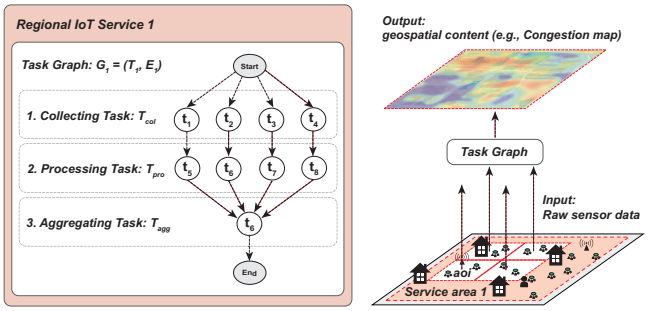


Fig. 3. An example of a task graph for a service

resolution (e.g., number of updates per second) that the service can provide, respectively. G_s is a task graph that describes the detailed procedure for providing the service s. A task graph G_s is described as a directed acyclic graph (DAG) with a depth of five as shown in Fig. 3, where each path starting from *Start* to *End* includes a *collecting task*, a *processing task* and an *aggregating task*.

We assume that task graph G_s has only one aggregating task but can have more than one collecting and processing tasks. The number of parallel tasks in G_s is determined when a query is received so that the requested resolutions and delay constraints are satisfied by executing tasks on the selected IoT devices. Fig. 3 is an example of a task graph where four collecting tasks (t_1, t_2, t_3, t_4) and four processing tasks (t_5, t_6, t_7, t_8) are executed in parallel.

3) User and Query: Each user in target area is allowed to use a service $s \in S$ by sending a query (Fig.2.d). Let Q denote the set of queries sent from all users in target area. A query $q \in Q$ is given as a tuple $\langle s, aoi, sr, tr \rangle$, where s is the target service, $aoi \subseteq area(s)$ is the *area of interest* (Fig.2.g) in the service area $area(s)$ and sr and tr are spatial and temporal resolutions that must be no more than SR and TR of s.

For each query $q \in Q$, a task graph $G_q = (T_q, E_q)$ is appropriately generated from $G_{q.s}$. Here, T_q is the set of tasks and E_q is the set of edges respectively. To simplify the notation, we define the unified task graph G for all queries of Q as follows.

$$G = (T, E) \text{ where } T = \bigcup_{q \in Q} T_q, E = \bigcup_{q \in Q} E_q \tag{1}$$

B. Formulation of dcRISP Problem

Given the resource graph $R = (P, L)$, the set of queries Q in target area and the unified task graph $G = (T, E)$ for all queries of Q, the dcRISP problem is defined as finding the optimal assignments of all tasks in T over R which would satisfy the given constraints and maximize the QoE of users (query senders) when receiving services.

1) Definitions of task assignment and delay: For every pair of task $t \in T$ and vertex $p \in P$, we define a variable $x_{t,p}$ which becomes 1 if t is assigned to p and 0 otherwise. We assume that every task is assigned to one processor. Then the following condition must hold.

$$\forall t \in T, \sum_{p \in P} x_{t,p} = 1 \tag{2}$$

We assume that each task t has a required computation amount denoted by $comp(t)$ and each processor p has a predefined computation capacity denoted by $cap(p)$. p accepts as many tasks as possible within its capacity. Therefore, the following condition must hold.

$$\forall p \in P, \sum_{t \in \{t | x_{t,p}=1\}} comp(t) \leq cap(p) \tag{3}$$

For each query $q \in Q$, we assume that task graph G_q is generated from $G_{q.s}$ by appropriately selecting the number of parallel collecting and processing tasks based on the resolution parameters $q.sr$ and $q.tr$. Here, $q.s, q.sr, q.tr$ are parameters of q. The total processing time of the task graph T_q is the maximum processing and communication delay among all execution paths from Start to End where each path contains one collecting task, processing task and aggregating task in this order (see Fig. 3).

Let n_q denote the number of paths in task graph $G_q = (T_q, E_q)$. Let $path_{q,i} = \langle Start, tc_{q,i}, tp_{q,i}, ta_{q,i}, End \rangle$ denote the i-th execution path of G_q, where $0 \leq i \leq n_q$ and $tc_{q,i}$, $tp_{q,i}$, $ta_{q,i}$ represent collecting, processing and aggregating tasks, respectively. Here, $T_q = \bigcup_{0 \leq i \leq n_q} \{tc_{q,i}, tp_{q,i}, ta_{q,i}\}$.

Let $p(t)$ denote the processor where task t is assigned and $dsize(t)$ the size of data that t outputs. We assume that a function $pt(t, p)$ for estimating the processing time when task t is executed on processor p is available. We also assume that a function $dt(l, dsize)$ for estimating the communication delay when data of size $dsize$ is transfered through link $l \in L$ is available.

For each execution path, the sum of computation and communication delay denoted by $delay(path(q, i)) = delay(\langle Start, tc_{q,i}, tp_{q,i}, ta_{q,i}), End \rangle$ can be calculated by the following equation.

$$\begin{aligned} &delay(path(q,i)) = \\ &\sum_{p \in P} [pt(tc_{q,i}, p) \cdot x_{tc_{q,i},p} + pt(tp_{q,i}, p) \cdot x_{tp_{q,i},p} \\ &\qquad + pt(ta_{q,i}, p) \cdot x_{ta_{q,i},p}] \\ &\quad + dt(link(Start, tc_{q,i}), dsize(tc_{q,i})) \\ &\quad + dt(link(tc_{q,i}, tp_{q,i}), dsize(tc_{q,i})) \\ &\quad + dt(link(tp_{q,i}, ta_{q,i}), dsize(tp_{q,i})) \\ &\quad + dt(link(ta_{q,i}, End), dsize(ta_{q,i})) \end{aligned} \tag{4}$$

where $link(t, t') = (p(t), p(t'))$ for every $t, t' \in T_q$

Then, the total processing and communication delay for executing G_q denoted by $D(q)$ can be represented by the following formula.

$$D(q) = \max_{0 \leq i \leq n_q} delay(path(q, i)) \tag{5}$$

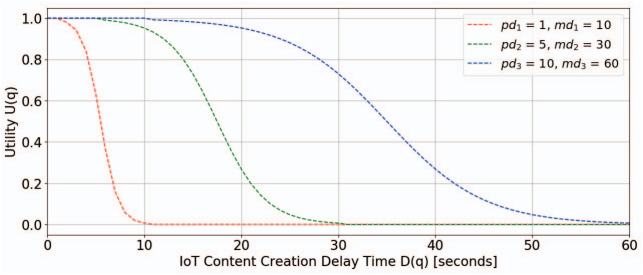


Fig. 4. An example of utility function form

2) Utility function: Let pd_q and md_q denote the preferable delay and the marginal delay for query q, respectively. In our target environment, the users desire to receive the requested service content within the preferable delay time pd_q. In other words, the user's QoE is maximized when contents are provided within the preferable delay time pd_q. On the other hand, if the contents are provided beyond the marginal delay time md_q set by the service, the user's QoE is equal to zero. Therefore, we assume that the user's QoE will decrease as the content creation delay time approaches the marginal delay time md_q. In summary, the user's QoE is maximum when the delay is within the pd_q, it then gradually decreases as the delay approaches md_q, and finally reaches 0 when the delay exceeds md_q. Based on existing work [24] on QoE, we design the QoE-based utility function $U(q)$. $U(q)$ is determined by the time delay in excess of the preferable delay time pd and the marginal delay time md by the following equation (Note: the actual values for pd and md are defined for each service). Fig. 4 illustrates examples of the utility function with different values of pd_q and md_q.

$$U(q) \stackrel{\text{def}}{=} \begin{cases} 1 & (D(q) < pd) \\ 1 - \dfrac{1}{1 + e^{5(ad - D(q))/(ad - pd)}} & (pd \leq D(q) \leq ad) \\ \dfrac{1}{1 + e^{5(D(q) - ad_q)/(md - ad)}} & (ad < D(q) \leq md) \\ 0 & (md < D(q)) \end{cases} \tag{6}$$

where

$$ad_q = \frac{pd_q + md_q}{2} \tag{7}$$

3) Objective function: The purpose of dcRISP is to find the optimal assignments of tasks T to processors P, in order to satisfy the given constraints and maximize the QoE of users. Thus, the objective function is defined as follows.

$$\textbf{Maximize} : \sum_{q \in Q} U(q) \tag{8}$$

4) Definition of dcRISP problem: Given the resource graph $R = (P, L)$ of the target area, the set of queries Q and the set of respective task graphs $\cup_{q \in Q} G_q (= (T_q, E_q))$, the dcRISP problem is to find assignments $x_{t,p}$ for every pair of $t \in \cup_{q \in Q} T_q$ and $p \in P$ which maximizes $\sum_{q \in Q} U(q)$ subject to (2) and (3).

This problem is NP-hard since it implies, as a special case, the Resource-Constrained Project Scheduling Problem (RCPSP) which is known to be NP-hard [25].

C. dcRISP problem with adaptive in-situ resource selection

In the dcRISP problem, computational resources required for executing task graphs of queries are allocated from the whole target area. However, as the target area becomes larger and more IoT devices are deployed over the area, possible allocation patterns (i.e., the search space) for seeking the optimal solution (task allocation pattern) will increase. Aiming to reduce the search space and shorten communication delay, we employ a strategy called *in-situ resource area selection* to limit the resource graph of each service to within its service area. However, this strategy may cause a shortage of resources in some services in spite of sufficient resources in the whole target area. Therefore, we employ an extended strategy called *adaptive in-situ resource area selection* that limits the resource graph of each service to within its service area but can adaptively extend the area for searching resources (called *resource area*) to increase available resources and improve QoE of the users of the service unless the extension deteriorates QoE of other services. Assuming that the service area is rectangular, the resource area is expanded by increasing the distance between the center and the sides equally (see Fig. 7 (b)).

Below, we will define the extended version of dcRISP called *dcRISP+*.

1) Definition of extendable resource graph: For each service $s \in S$, let R_s denote the sub-graph of $R = (P, L)$ within the service area of s. R_s can be defined as follows. Here, $pos(p)$ denotes the position of processor p and $area(s)$ denotes the service area of s.

$$R_s = (P_s, L_s) \tag{9}$$

where

$$P_s = \{p | p \in P_A \wedge pos(p) \in area(s)\}$$
$$L_s = \{l | pos(l.st) \in area(s) \wedge pos(l.ed) \in area(s)\}$$

Then, we define extendable resource graph R_s^{i+} as follows. Let $R_s^{i+} = (P_s^{i+}, L_s^{i+})$ denote the resource graph of s extended from $R_s^{(i-1)+} = (P_s^{(i-1)+}, L_s^{(i-1)+})$ where $R_s^{0+} = R_s$ and $|P_s^{i+}| = |P_s^{(i-1)+}| + 1$. That means one processor is added to R_s^{i+} by extending the resource area. For each service, its own computational demand is monitored regularly. If increased computational demand over the computational resource is observed, adaptive scale-out (extending the resource area) is executed. Onece adaptive scale-out is invoked, the resource area is extended until the shortage of computational resources is resolved. Here, note that the resource area of each service can be extended up to the whole target area and can overlap with those of other services.

2) Objective function: The objective function of dcRISP+ is defined as follows.

$$\textbf{Maximize}: \sum_{s \in S} \sum_{q \in Q_s} U(q) \tag{10}$$

3) Definition of dcRISP+ problem: Given the resource graph $R = (P, L)$ of the target area, the set of services S, the set of queries Q_s for each $s \in S$ and the set of respective task graphs $\cup_{q \in Q_s} G_q (= (T_q, E_q))$, the dcRISP+ problem is that, for every $s \in S$ and every $q \in Q_s$, we need to find assignments $x_{t,p}$ for every pair of $t \in T_q$ and $p \in P_s^{i_s+}$ and resource area extension level i_s, which maximizes $\sum_{s \in S} \sum_{q \in Q_s} U(q)$ subject to (2) and (3).

IV. Adaptive In-situ Task Scheduling Algorithm

The *dcRISP* problem is a special case of the *dcRISP+* problem. Thus, in this section, we focus only on the *dcRISP+* problem. As this problem is NP-hard, we give a heuristic algorithm for solving it.

A. Basic Policies

For heuristics, we employ the following two basic policies to solve the *dcRISP+* problem: *(1) In-situ task allocation*: task graph G_q generated by a query q is preferentially allocated and processed by IoT devices in the service area $area(q.s)$ and its extended resource area according to computational demand of s, and *(2) FCFS task scheduling/execution*: for the set of queries Q, task graph G_q of each query $q \in Q$ are scheduled in a first-come-first-serve (FCFS) manner. All tasks of task graph G_q are assigned to the available IoT devices (processors) in the order of collecting, processing, and aggregating tasks. Then, each processor executes tasks in a first-come-first-serve (FCFS) manner. Basically, since we assume that any resource can be accessed from any processor, our proposed method initially assigns processors (to derive the initial solution) without any constraint (not considering the dependency between the preceding and succeeding tasks). After that, the proposed method adjusts the initial task assignment by using Tabu search toward the more appropriate task assignment considering inter-task dependency.

B. System model

We show the system model of the proposed algorithm in Fig. 5. The model consists of two types of brokers: *Resource broker (or R-broker)* shown in Fig.5 (A) and *Service brokers*

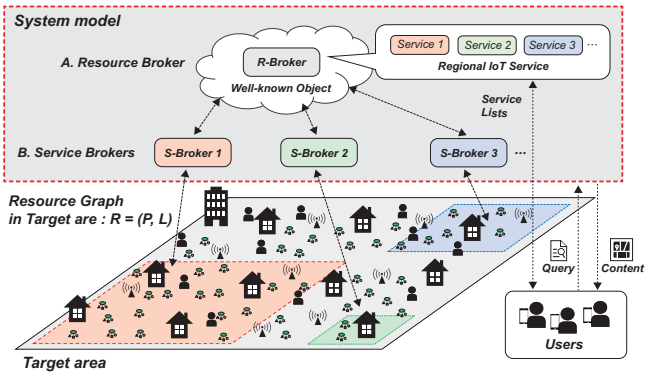


Fig. 5. System model of proposed algorithms

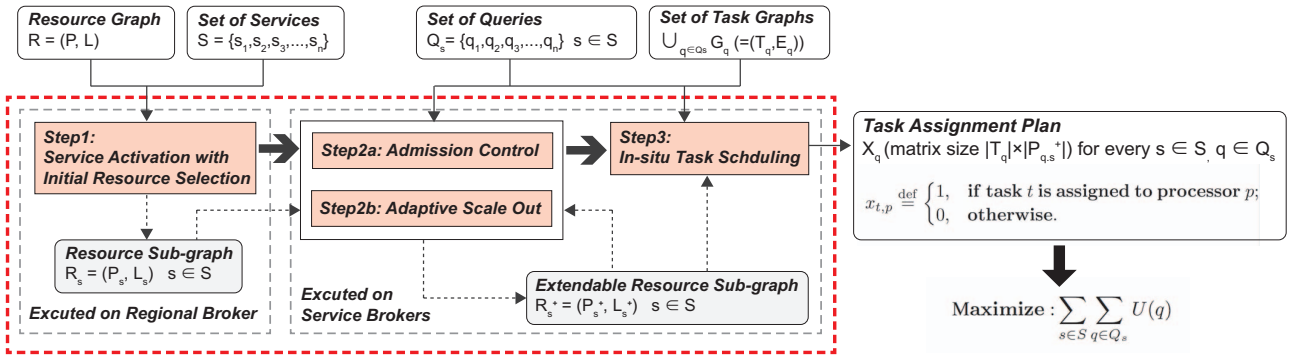


Fig. 6. Steps for the adaptive in-situ resource provisioning

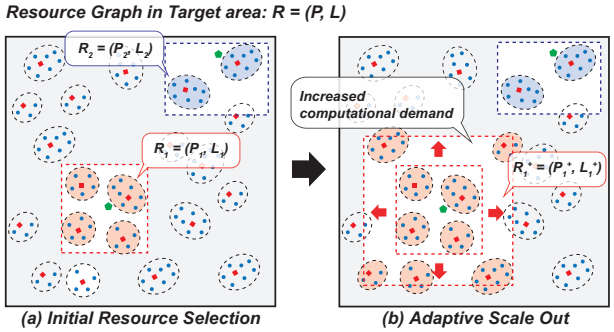


Fig. 7. In-situ initial resource selection and adaptive scale-out

(or S-broker) shown in Fig.5 (B). The R-broker is a well-known server that all devices know beforehand[2]. The R-broker is responsible for management of all processors (computation resources) existing in the target area and regional IoT services deployed in that area, while the S-broker manages the resource provisioning (task assignment) of a specific service in charge. Each S-broker assigned for service $s \in S$ carries out admission control of each query for s and scheduling of its task graph.

Given this system model, we design the adaptive in-situ resource provisioning algorithm that solves the *dcRISP+* problem.

C. Steps for providing regional IoT services

In order to provide regional IoT services, (Step 1) activation of each service, (Step 2) admission control of each query and adaptive scale-out of resource selection area, and (Step 3) task scheduling for the admitted query are needed. We show these steps in Fig. 6.

Step 1: Service activation with initial resource selection: Step 1 is executed on the R-broker. In this step, the R-broker adds s to the set of services S and generates resource sub-graph $R_s = (P_s, L_s)$ and selects a most powerful or most stable processor (IoT device) as an S-broker from P_s as shown in Fig.7 (a). The selected S-broker is activated to execute Step 2 and Step 3.

[2]A cloud server or a fog server is typically selected as an R-broker, but a powerful IoT device/gateway can also be selected.

Step 2: Admission control and adaptive scale-out: This step consists of two sub-steps: Step 2a: admission control and Step 2b: adaptive scale-out. In Step 2a, when the S-broker receives a query q, task graph G_q is generated. Specifically, the task graph G_q is generated according to the following procedure.

(1) First, the input data size for task graph $G_{q.s}$ is calculated by the parameters aoi, sr and tr specified in q;

(2) Next, the required computation amount of each task t (collecting, processing, aggregating) in the default task graph $G_{q.s}$ is calculated by the function $comp(t)$ (see Sect. III.B-1)) and the input data.

(3) Finally, each task t in $G_{q.s}$ is divided into multiple subtasks $s_1, ..., s_{k_t}$ so that $comp(s_i) \leq cap(p)$ $(1 \leq i \leq k_t)$ holds. Then, the task graph of the query G_q is generated by replacing each task t by the parallel execution of subtasks $s_1, ..., s_{k_t}$ in $G_{q.s}$

After generating G_q, S-broker determines in a heuristic manner whether the task graph G_q can be processed within the marginal delay $md_{q.s}$ using the current resource subgraph R_s (or R_s^{i+}). In Step 2b, the S-broker periodically checks resource usage and performs *adaptive scale-out* which generates resource graph R_s^{i+} by expanding resource selection area if the computation resources of service s is insufficient for queries being processed.

Step3: In-situ task scheduling: Finally in Step 3, the S-broker schedules the task graph $G_{q.s}$ for the accepted query q and generates the semi-optimal task assignments X_q with matrix size $|T_q| \times |P_q.s^{i_s+}|$ using tabu search algorithm.

Detailed algorithms for Step 2 and Step 3 are described below.

D. Admission control and adaptive scale-out

When S-broker receives a query q, it determines if q is acceptable or not in terms of the specified delay constraint (i.e., if the delay is no more than $md_{q.s}$). Algorithm 1 shows procedure $isAcceptable$ for admission control for a query q. The input parameters are task graph G_q for query q, resource sub-graph $R_{q.s}$ (or extended resource graph $R_{q.s}^{i+}$) and, $md_{q.s}$. The output is a boolean value that indicates if a query q is acceptable or not. If acceptable, initial resource assignment $X_{q.init}$ is generated in a greedy manner (each task $t \in T_{q.s}$

Algorithm 1 Admission control algorithm

Require: $G_q = (T_q, E_q)$, $R_{q.s} = (P_{q.s}, L_{q.s})$, $md_{q.s}$
Ensure: Bool Value that indicates if a query is acceptable, X_0 (if acceptable), CPD (if not acceptable)

```
1  procedure ISACCEPTABLE(G_q, R_{q.s}, md_{q.s})
2      P ← P_{q.s}
3      X ← O
4      CPD ← 0
5      falsecount ← 0
6      determine values of md_c, md_p, md_a so that md_c + md_p + md_a = md_{q.s}
7      for all t ∈ T_q do
8          md ← md_c/md_p/md_a depending on type of t
9          find p ∈ P s.t. proctime(t, p) ≤ md
10         if such p is found then
11             x_{t,p} ← 1
12             P ← P\{p}
13         else
14             CPD ← CPD + comp(t)
15             falsecount ← falsecount + 1
16     end for
17     if falsecount = 0 then
18         X_{q.init} ← X
19         return true
20     else
21         return false
22 end procedure
```

from task graph G_q is assigned to the processor $p \in P_{q.s}$ from resource graph $R_{q.s}$, with the processing time less than each task's marginal delay).

In Algorithm 1, variables $P, X, CPD, falsecount$ are initialized (lines 2–5) where P denotes the set of unassigned processors, X denotes the current task assignment matrix whose element is $x_{t,p}$, CPD denotes the sum of insufficient computation resource when processors that can run the tasks are not found and $falsecount$ denotes the number of mis-assignments. In line 6, marginal delay times denoted by md_c, md_p and md_a for collecting, processing and aggregating tasks respectively are determined by dividing $md_{q.s}$, where the ratio of division is empirically determined (given in advance). In lines 7–16, task assignments between tasks T_q and processors P are determined. Specifically, if an assignment of t to p that satisfies the delay constraint is found, $x_{t,p}$ is set to 1 and p is removed from P (lines 9–12). Otherwise, the computation resource required for executing t is summed up in CPD and $falsecount$ is incremented (lines 13–15). Finally, in lines 17–21, true (also the final task assignments are recorded in X_0) or false (CPD as well) is returned depending on the value of $falsecount$.

X_0 is a byproduct of $isAcceptable$ and used for Algorithm 2. CPD is used for adaptive scale-out (Step 2b in Fig. 6). If $CPD > 0$, the S-broker increases i for the extended resource graph R_s^{i+} to compensate a shortage of computational resources represented by CPD, as shown in Fig.7 (b).

E. In-situ task scheduling algorithm

Once query q is accepted by $isAcceptable$, the task graph G_q is scheduled. In this step, S-broker tries to find better task assignments than those derived by $isAceptable$ so as to increase the sum of utility function $\sum_{q \in Q} U(q)$ as much as possible. Thus, we propose an algorithm that improves the initial assignments $X_{q.init}$ by using meta-heuristic search. Specifically, we use tabu search algorithm [26] for adjusting assignments. Tabu search is a meta-heuristic based on a local heuristic search procedure designed to explore the solution space beyond local optimality where it keeps track of the regions of the solution space which have already been searched so as not to repeat a search near these areas. Tabu search has been successfully applied to a number of combinatorial optimization problems. In our proposed method, tabu search works to adjust/improve the initial solution obtained with admission control described above.

Algorithm 2 Task Scheduling algorithm based on Tabu search

Require: $G_q = (T_q, E_q)$, $R_{q.s}^{i+} = (P_{q.s}^{i+}, E_{q.s}^{i+})$, $X_{q.init}$
Ensure: X_q: Task Assignments

```
1  /*Initialization*/
2  X, X* ← X_0
3  Tabu ← O
4  cnt ← 0
5  while cnt ≤ MAXCNT do
6      reduce_best ← ∞
7      /* obtain the best neighbor plan*/
8      X', X̄ ← X
9      t' ← randselect(T_q), p' ← p(t')
10     for all t ∈ T_q do
11         for all p ∈ P_{q.s}^{i+} | p ≠ p(t) do
12             if Tabu_{t,p} = 0 & comp(t) ≤ remcap(p) then
13                 x̄_{t,p(t)} ← 0, x̄_{t,p} ← 1
14                 gain ← Utility(X̄) − Utility(X)
15                 if gain > gain_best then
16                     gain_best ← gain
17                     t' ← t, p' ← p
18         end for
19     end for
20     x'_{t',p(t')} ← 0, x'_{t',p'} ← 1
21     /* update the best plan*/
22     if gain_best ≥ 0 then
23         if Utility(X') > Utility(X*) then
24             X* ← X'
25     else
26         Tabu_{t',p(t')} ← 1
27     X* ← X'
28     cnt ← cnt + 1
29 X_q ← X*
30 return X_q
```

Algorithm 2 shows the proposed task scheduling algorithm based on Tabu search. The input parameters are task graph G_q for query q, resource sub-graph $R_{q.s}$ and, initial task assignments $X_{q.init}$ obtained from $isAcceptable$.

In Algorithm 2, variables $X, X^*, Tabu, cnt$ are initialized (lines 1–4), where X, X^* are the current task assignments and the best assignments, respectively. $Tabu$ is the tabu matrix and cnt is the counter to control the number of iterations. $p(t)$ denotes the processor where task t is assigned. The main process is in lines 5–28 where $MAXCNT$ in line 5 is

the constant value given to specify the number of iterations. In line 8, the best neighbor solution X' and the neighbor solution $\bar{X}$ are initialized by X. In line 9, variables t' and p' are initialized, where t' and p' are a task randomly selected from T_q and its assigned processor in X, respectively. The loop in lines 10–19 finds the best neighbor solution (task assignments) X' selected among all neighbor solutions $\bar{X}$ generated by changing one task assignment in the current solution X. Specifically, for every task assignment $x_{t,p(t)} = 1$ is changed to a new assignment $\bar{x}_{t,p} = 1$ if p has a remaining computation capacity ($remcap()$) for executing t (lines 12–13). If the overall utility to execute the task graph by the new assignment is increased, then X' is updated (lines 14–17 and 20). In line 14, $Utility(X)$ is the function to estimate the QoE when executing the task graph based on the current task assignments X (first the delay for processing the task graph is calculated and then utility is calculated based on the delay). In lines 22–27, if X' is better than the current best solution X^*, X^* is updated to the best neighbor solution X'. After iterations stop, the current best solution X^* is substituted to X_q as the final solution (lines 29–30).

V. Performance Evaluation

In this section, we evaluate the resource provisioning performance of the proposed algorithm and conventional algorithms through computer simulations. In these simulations, we use delay time, number of accepted queries, and utility function value as metrics to evaluate the performance of the algorithms.

TABLE I. Resource parameters for simulation

Parameter	Value(s)
Number of workers (nodes)	4000
Number of M-nodes	1000
Number of S-nodes	3000
Number of workers per section	0 to 24
Processing power of M-nodes	1000 to 3000
Processing power of S-nodes	500 to 1000
Network speed of M-node to R-broker	5 to 50 (Mbps)
Network speed of S-node to M-node	50 to 500(Mbps)

TABLE II. Services parameters for simulation

Parameter	Value(s)
Input sensor data size range of Home service	1 to 20 (MB)
Collecting task cost of Home service	100
Processing task cost of Home service	1000
Aggregating task cost of Home service	100
Input sensor data size range of Office service	5 to 50 (MB)
Collecting task cost of Office service	500
Processing task cost of Office service	2000
Aggregating task cost of Office service	300
Input sensor data size range of Tourism service	10 to 200 (MB)
Collecting task cost of Tourism service	1000
Processing task cost of Tourism service	5000
Aggregating task cost of Tourism service	500

TABLE III. Simulation case I (Daytime on weekday)

Parameter	Value(s)
Simulation duration	1 (h)
Simulation runs	10 Runs
Query arrival rate per section of Home service λ_h	1 (q/min)
Query arrival rate per section of Office service λ_o	6 (q/min)
Query arrival rate per section of Tourism service λ_t	2 (q/min)

TABLE IV. Simulation case II (Daytime on holiday)

Parameter	Value(s)
Simulation duration	1 (h)
Simulation runs	10 Runs
Query arrival rate per section of Home service λ_h	2 (q/min)
Query arrival rate per section of Office service λ_o	1 (q/min)
Query arrival rate per section of Tourism service λ_t	6 (q/min)

TABLE V. Simulation case III (Nighttime)

Parameter	Value(s)
Simulation duration	1 (h)
Simulation runs	10 Runs
Query arrival rate per section of Home service λ_h	6 (q/min)
Query arrival rate per section of Office service λ_o	1 (q/min)
Query arrival rate per section of Tourism service λ_t	2 (q/min)

A. Simulation design and parameter setting

In order to clarify the effectiveness of the proposed algorithm, we created a virtual IoT environment as shown in Fig 8 (explained in detail in the next subsection), and evaluate the performance of the algorithm using this simulation environment. The simulator was implemented on Python using frameworks SimPy and NetworkX. The simulation consists of the R-broker, IoT devices (consisting of *M-nodes* and *S-nodes*), regional IoT services, and queries. M-nodes and S-nodes refer to high power and low power IoT devices respectively. In the simulation, we assume that three services are running in different sub-areas as explained in-depth in the next subsection. We assume that the resource graph of the target area as well as the resource sub-graph and the task graph for each service are given in advance.

In the simulation, first, S-broker is selected for each service area and it generates a resource graph for the service. Then, the S-broker of each service area receives the query and generates a task graph for the query and evaluate it by admission control (Algorithm 1), and if accepted, the task graph is scheduled by using each algorithm described below by assigning each task in the task graph to a processor[3].

The general parameters of the simulation are shown in Table I. The parameters related to processing power and network speed are determined based on the actual measurement in the previous studies [27], [28].

Two conventional algorithms: random task scheduling and greedy task scheduling were used as baselines against the two proposed algorithms (tabu search and tabu search with adaptive scale-out).

B. Three cases for simulations

As a target regional area for simulations, we picked up a 2Km $\times$ 2Km tourist area in the Higashiyama district of Kyoto, Japan, as shown in the left of Fig. 8. The actual target area comprises three different areas: residential (home) area, office area (including shopping streets), and tourist area (temples and shrines), that we represent by grey, blue and green boxes, respectively in the simulation area shown in the right of Fig. 8. We suppose that three different regional IoT services: (1) Home service, (2) Office service and (3) Tourism service are provided in these areas, respectively. Since populations in these

[3]We assume that when multiple tasks are allocated to a processor, its processing power is decreased according to the load of the tasks.

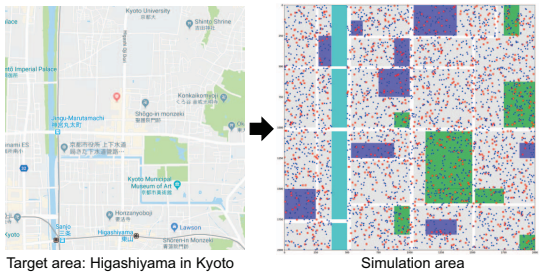


Fig. 8. Target regional area for simulation: Higashiyama area in Kyoto

areas are different depending on the day and time, we set three query arrival rates of each service for three cases: weekday daytime (WD), holiday daytime (HD) and night time (NT) as shown in Table III. The Query arrival rate is defined for each block (125m $\times$ 125 m) where the whole target area is divided into 8 $\times$ 8 blocks. We run simulations with 1hour of simulation time for 10 times.

Queries are issued to each service based on the Poisson arrival process with parameters shown in Table II. The table also specifies other parameters: data size and computation cost for each of collecting, processing and aggregating tasks. The input data size of each task graph for each query is randomly selected in the range specified in the parameters by considering the number of sensors and the generated raw data size in each service area and spatio-temporal resolutions specified in each query. The computation costs for each service is determined by supposing surveillance for suspicious person using motion sensors, crowdedness estimation at streets using laser range scanners, and tourist content curation using photo/video data in Home, Office and Tourist areas respectively.

C. Task scheduling algorithms for comparison

We compare four task scheduling algorithms, two conventional (baseline) algorithms, and two proposed algorithms, defined below;

- Baseline method 1 - Random task scheduling: It assigns each task $t \in T_{q.s}$, in arbitrary order, to the randomly selected available processor $p \in P_s$ from resource graph R_s, regardless of the task's expected execution time on that processor [29].
- Baseline method 2 - Greedy task scheduling: It assigns each task $t \in T_{q.s}$, in arbitrary order, to the processor $p \in P_s$ from resource graph R_s, with the minimum completion time for that task [29].
- Proposed method 1 - Tabu search based task scheduling: It assigns each task $t \in T_{q.s}$, to the processor $p \in P_s$ from resource graph R_s based on Tabu search algorithm described in the section IV-E.
- Proposed method 2 - Tabu search based task scheduling with adaptive scale-out: It performs adaptive scale-out in addition to the above proposed method 1. It assigns each task $t \in T_{q.s}$, to the processor $p \in P_s^{i+}$ from extended resource graph R_s^{i+} based on Tabu search algorithm.

The metrics to evaluate the performance of each algorithm are utility function value, delay time and number of accepted queries. Utility function value is calculated based on the utility function curve of pd = 1.0s and md = 10.0s.

D. Results and Discussion

1) Utility function value: Fig. 9 shows the sum of the utility function values for all queries of the four methods in each case. The results suggest that the proposed methods 1 and 2 have better QoE than the baseline methods in all cases. In addition, the proposed method 2 that performs adaptive scale-out achieved QoE than the proposed method 1 in all cases.

2) Delay time and acceptance rate: Fig. 10 shows cumulative distribution function (CDF) curves of the delay time of all queries for each method in each case.

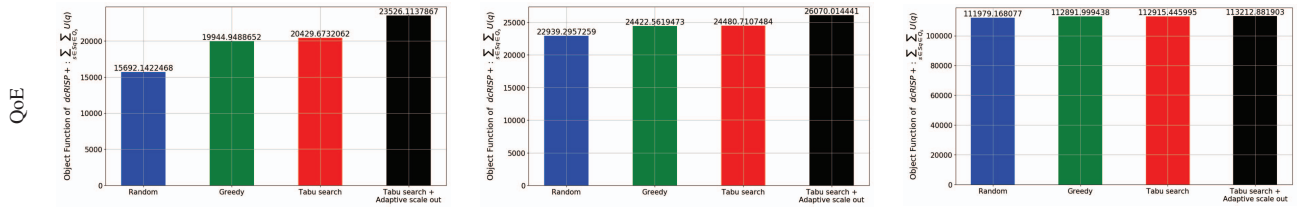
The highest point of each curve in each graph shows the acceptance rate of the queries. In all figures, the acceptance rate (the value at the highest point) of the proposed method 2 (tabu search with adaptive scale-out) is higher than other methods. We see that the proposed method 2 most effectively worked for the case 1 (weekday daytime) for Office service.

For the Tourist service, the proposed method 2 worked much better than others. However, the acceptance rates of all methods are low (between 0 and 30%) because the processing devices of the devices in the target area was not sufficient. When we consider the cloud resource for inclusion in our adaptive scale-out, we can increase the acceptance rate, but this is part of our future work. It means that the adaptive scale-out of the proposed method 2 is effective for such a case requiring high computation demand.

In most figures, we see that the delays of queries in the CDF curve are lower in the proposed methods 1 and 2 (tabu search and tabu search with adaptive scale-out). In case 2 (home and office service), the performance of proposed method 2 is lower than greedy and tabu search (without adaptive scale-out). These results are influenced by adaptive scale-out that the computational resources belonging to the home and office service were also used for the tourism service where the computational demand was high. However, this is not a concern because the proposed method 2 is superior to the overall QoE as shown in the Fig. 9.

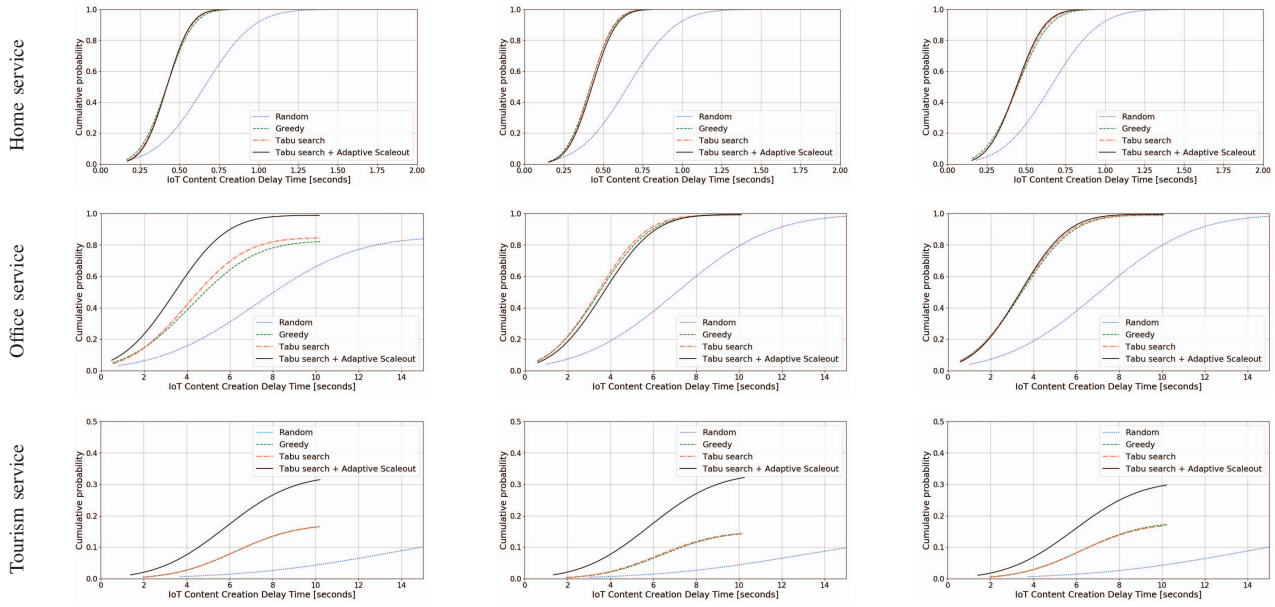
Fig. 11 shows the average delay time for each method of each service that includes 3 simulation cases results. Error bars show the 95% confidence interval (CI). This result shows that the proposed methods 1 and 2 have shorter delay times than random and greedy methods. In addition, we see the delay time of proposed method 2 is shorter than the proposed method 1.

Snapshots of adaptive scale-out during the simulation in case2 (daytime on holiday) are shown in Fig 12. Red zones are the resource selection area of each service. Depending on the requested queries, some services execute the adaptive scale-out for their service areas, as shown in Fig. 12 (the resource selection area protrudes the service area). In case 2, we assumed that the tourism service performs adaptive scale-out more frequently because the tourist service receives more queries and the computational demand is higher than the other two services (home, office). In the initial state, adaptive scale-out is not yet applied to any service area as shown in the left of Fig. 12. In the intermediate state in Fig. 12, tourism service scaled out their service areas and the resource selection



Case 1 (Weekday Daytime) Case 2 (Holiday Daytime) Case 3 (Night time)

Fig. 9. Sum of Utility function $U(q)$ value for all queries in Q in each simulation set



Case 1 (Weekday Daytime) Case 2 (Holiday Daytime) Case 3 (Night time)

Fig. 10. Cumulative distribution function (CDF) curves of the delay time of all queries

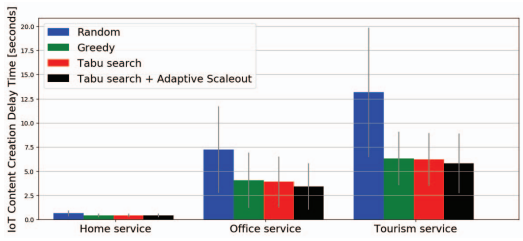


Fig. 11. Average delay time for each service with 95% confidence interval

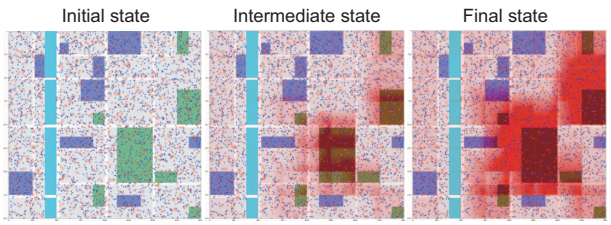


Fig. 12. Snapshot of adaptive scale-out in case 2

overlapped the home service area. It means that tourism service uses the resources which belong to home service. In the final state in Fig. 12, the resource selection area of tourism service reached the office service area. Fig 12 shows that services without enough resources for handling the queries are adaptively expanding the resource selection area using the adaptive scale-out. In case 2, since the overlapping of resource selection areas occurs (intermediate and final states in Fig. 12), the performances of the tabu search with adaptive scale-out in the home and office service are slightly inferior to greedy and tabu search as shown in Figure 10.

3) Effect of Tabu search and Adaptive scale-out: We summarize the evaluation results as follows.

- We confirmed that the proposed methods 1 and 2 using tabu search shorten the average delay time for processing queries (Fig. 10). Reducing the delay time improves the quality of service. From this, we believe that tabu search contributes to improving the quality of service for query processing in regional IoT services.
- We confirmed that the proposed method 2 using the adaptive scale-out increases the number of accepted queries (Fig. 10). An increase in the number of accepted queries means an improvement of service availability. From this, adaptive scale-out contributes to the improvement of service availability.
- In the proposed method 2, tabu search improves the quality of service for each query, and adaptive scale-out improves the service availability. As a result, the

method is able to provide the higher QoE to users.

VI. Conclusion

In this paper, we formulated the delay constrained regional IoT service provisioning (*dcRISP*) problem of assigning tasks for processing queries for geo-spatial information to IoT devices in the target regional area and its extended version, *dcRISP+* problem that allows the extension of the resource selection area. To solve these problems, we proposed the adaptive in-situ task scheduling algorithm composed of in-situ resource area selection with adaptive scale out and in-situ task scheduling based on tabu search technique. We evaluated the proposed methods through computer simulations supposing a regional area with 4,000 IoT devices. The results showed that tabu search and adaptive scale out shortens the delay for processing queries and improve Quality of Experience (QoE) of service users. This shows that the proposed flexible and in-situ resource provisioning scheme based on the demand for services is effective in providing regional IoT services. In future work, we update/extend the problems and models that consider more various constraints such as specific data, task (collecting, processing, aggregating task) and device type (e.g. sensors and actuators). We also evaluate the performance through more realistic simulation and a real-world testbed.

Acknowledgement

This work was supported in part by JSPS KAKENHI Grant Numbers 17J10021,16H01721 and 26220001.

References

[1] D. Evans, "The internet of things: How the next evolution of the internet is changing everything," http://blogs.cisco.com/news/the-internet-of-things-infographic/, accessed: 2017-06.

[2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: ACM, 2012, pp. 13–16.

[4] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[5] K. Yasumoto, H. Yamaguchi, and H. Shigeno, "Survey of real-time processing technologies of iot data streams," *Journal of Information Processing*, vol. 24, no. 2, pp. 195–202, 2016.

[6] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati, "Replisom: Disciplined tiny memory replication for massive iot devices in lte edge cloud," *IEEE Internet of Things Journal*, vol. 3, no. 3, pp. 327–338, 2016.

[7] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 27–32, 2014.

[8] O. Salman, I. Elhajj, A. Kayssi, and A. Chehab, "Edge computing enabling the internet of things," in *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*. IEEE, 2015, pp. 603–608.

[9] X. Sun and N. Ansari, "Edgeiot: Mobile edge computing for the internet of things," *IEEE Communications Magazine*, vol. 54, no. 12, pp. 22–29, 2016.

[10] M. A. Al Faruque and K. Vatanparvar, "Energy management-as-a-service over fog computing platform," *IEEE internet of things journal*, vol. 3, no. 2, pp. 161–169, 2016.

[11] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the internet of things realize its potential," *Computer*, vol. 49, no. 8, pp. 112–116, 2016.

[12] T. Higashino, H. Yamaguchi, A. Hiromori, A. Uchiyama, and K. Yasumoto, "Edge computing and iot based research for building safe smart cities resistant to disasters," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 1729–1737.

[13] D. Schafer, J. Edinger, J. M. Paluska, S. VanSyckel, and C. Becker, "Tasklets:" better than best-effort" computing," in *Computer Communication and Networks (ICCCN), 2016 25th International Conference on*. IEEE, 2016, pp. 1–11.

[14] J. Edinger, D. Schäfer, C. Krupitzer, V. Raychoudhury, and C. Becker, "Fault-avoidance strategies for context-aware schedulers in pervasive computing systems," in *Pervasive Computing and Communications (PerCom), 2017 IEEE International Conference on*. IEEE, 2017, pp. 79–88.

[15] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[16] C.-C. Shen and W.-H. Tsai, "A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion," *IEEE Transactions on Computers*, vol. 100, no. 3, pp. 197–203, 1985.

[17] A. Salman, I. Ahmad, and S. Al-Madani, "Particle swarm optimization for task assignment problem," *Microprocessors and Microsystems*, vol. 26, no. 8, pp. 363–371, 2002.

[18] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. ACM, 2010, pp. 304–307.

[19] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2011, p. 49.

[20] D. Ardagna, M. Ciavotta, and M. Passacantando, "Generalized nash equilibria for the service provisioning problem in multi-cloud systems," *IEEE Transactions on Services Computing*, vol. 10, no. 3, pp. 381–395, 2017.

[21] J. Xu, B. Palanisamy, H. Ludwig, and Q. Wang, "Zenith: Utility-aware resource allocation for edge computing," in *Edge Computing (EDGE), 2017 IEEE International Conference on*. IEEE, 2017, pp. 47–54.

[22] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner, "Resource provisioning for iot services in the fog," in *Service-Oriented Computing and Applications (SOCA), 2016 IEEE 9th International Conference on*. IEEE, 2016, pp. 32–39.

[23] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, "Towards qos-aware fog service placement," in *Fog and Edge Computing (ICFEC), 2017 IEEE 1st International Conference on*. IEEE, 2017, pp. 89–96.

[24] I. Abdeljaouad and A. Karmouch, "Monitoring iptv quality of experience in overlay networks using utility functions," *Journal of Network and Computer Applications*, vol. 54, pp. 1–10, 2015.

[25] P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch, "Resource-constrained project scheduling: Notation, classification, models, and methods," *European journal of operational research*, vol. 112, no. 1, pp. 3–41, 1999.

[26] S. C. Porto and C. C. Ribeiro, "A tabu search approach to task scheduling on heterogeneous processors under precedence constraints," *International Journal of high speed computing*, vol. 7, no. 01, pp. 45–71, 1995.

[27] Y. Nakamura, H. Suwa, Y. Arakawa, H. Yamaguchi, and K. Y. Yasumoto, "Design and implementation of middleware for iot devices toward real-time flow processing," in *Distributed Computing Systems Workshops (ICDCSW), 2016 IEEE 36th International Conference on*. IEEE, 2016, pp. 162–167.

[28] ——, "Middleware for proximity distributed real-time processing of iot data flows," in *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*. IEEE, 2016, pp. 771–772.

[29] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions," in *Heterogeneous Computing Workshop, 1998.(HCW 98) Proceedings. 1998 Seventh*. IEEE, 1998, pp. 79–87.