EdgeCEP: Fully-distributed Complex Event Processing on IoT Edges

Sunyanan Choochotkaew, Hirozumi Yamaguchi, Teruo Higashino
Graduation School of Information Science and Technology
Osaka University, Osaka, Japan
Email: {sunya-ch,h-yamagu,higashino}@ist.osaka-u.ac.jp

Megumi Shibuya, Teruyuki Hasegawa
KDDI Research, Inc.
Japan
Email: {shibuya, teru}@kddi-research.jp

***Abstract*—In this paper, we propose a general complex event processing (CEP) engine aiming for accomplishing at smart IoT edge devices in a fully distributed manner. We introduce a pseudo-source mechanism to cover a wide range of processing and obsolete prerequisite of source-specification at the same time, along with a brand-new event specification language defined to support relation-based processing. Against cloud-based approaches, our behind-edge approach can prevent data overflow and privacy issues, and fully distributed processing can draw the power of the edge devices. To achieve that in a resource-limited edge environment, we formulate an optimization problem of processing task assignment and stream delivery, and propose a fully-autonomous workload distribution mechanism. A large-scale simulation with a realistic smart-building scenario shows that our proposed method achieves about 6.6 times smaller flow volume and 2 times lower loss rate compared to centralization and is relatively superior to a hop-based distribution approach. Notably, a prototype engine is successfully deployed over an ad-hoc wireless sensor and actuator network through Intel Edison modules in the real environment.**

I. Introduction

Recently, the Internet of Things (IoT) is a promising technology for enabling the wide range of advanced services by interconnecting physical and virtual things. A general stream data processing engine, where user demands can be specified and processed effortlessly, is recognized as a vital factor to increase IoT utilization. According to the study in [1], early engines were realized by modifying the traditional databases for actively processing events and exploited the familiar query statement to define user demands. The concept of data streams has been introduced in data-stream management systems (DSMS) for supporting unbounded and high flowing rate data, also known as stream-based approaches [2][3][4][5][6]. In the meantime, some researchers apply a rule engine or an automaton for pattern matching to catch interest events. This is called event-based approach [7][8][9][10]. In the past decade, most research in this area gave attention to an event which is determined by multiple data sources called *complex event*. Generally, stream-based approaches focus on analyzing relations among a large amount of collected data to achieve hidden knowledge and a typical example application is intrusion detection [11]. At the same time, event-based approaches aim for detecting various patterns of event occurrences. So far, there are a number of complex event processing tools available on the Internet, for instance, Google

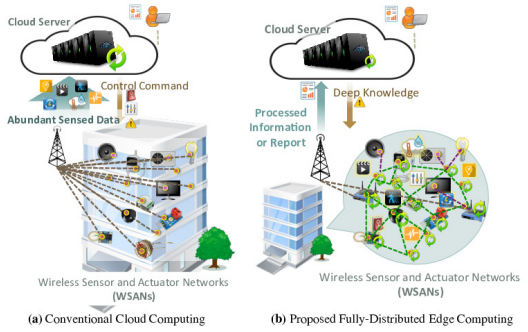


Fig. 1. Cloud Computing and Distributed Mesh Computing

Cloud Platform, where users can connect their streaming inputs and upload the customized applications to the servers. Correspondingly, the number of Internet-connecting devices grows exponentially and trends to reach 50.1 billion by 2020 [12]. Whereas, the wireless link resources development is relatively slow. In the next few years, pushing all data to the cloud may lead to intolerable latency and severe privacy concerns. To avoid such issues, *Edge computing* has been introduced as illustrated in Fig. 1. Potential nodes on Wireless Sensor and Actuator Networks (WSANs) called edge nodes can horizontally collaborate and process the data. It allows locally-distributed computing, instead of leaving all to the cloud [13].

Distributed systems on the edges are supposed to be self-organizing since connections are dynamic and unpredictable. The processing engines with the stream-based model are now available only for centralized or clustered environments with static connections and a distribution plan of such systems is mostly manually designated by users [2][3][4][5][6]. In contrast, the engines with the event-based data model allow fully-autonomous distribution. Nevertheless, a simple distribution policy can lead to a critical state at some edge nodes, especially in executing high-arrival-rate workloads such as video. For instance, the policy based on only the "nearest-to-source" heuristic, found in [7][8][9][14], intuitively causes the source-nearest processor nodes inevitably risky for overloading and might cause accumulating delays in overall. More complicated methods are commonly done by collecting information from

2325-2944/17 $31.00 © 2017 IEEE
DOI 10.1109/DCOSS.2017.14

all processor nodes and making a consistent distribution decision at one specific node [15][16]. Regardlessly, no one to the best of our knowledge provides a fully-distributed approach that concerns optimization from sources to the destination, which particularly exists on edge networks.

This paper proposes a general complex event processing engine distributedly deployed on self-organized ad-hoc wireless sensor and actuator networks, also known as edge networks towards IoTs. The proposed engine is composed of potential nodes called brokers. Particularly, a pseudo-source mechanism and a relation-supportive event specification language are originally introduced to retrieve advantages of both event-based and stream-based approaches. User-specified requests are consistently decomposed and distributedly executed. For efficient and adaptive execution under resource constraints such as workload on individual nodes and link-load limits, we formulate a task assignment and delivery problem to optimize communication cost along the path from source to destination. For the solution, we propose a heuristic algorithm applying tabu search with a flow-based greedy move. Notably, our algorithm is tailored to consider dynamics of flow volume for periodical, real-time decision update. We have evaluated our approach both in simulation and in a real environment. Simulations have been conducted with a large-scale nursing-care scenario, and the results show that our approach can reduce 6.6 times flow volume and mitigate 2 times packet loss compared with that of centralized and topology-based distributed flow processing approaches. We have also developed and deployed the prototype on Intel Edison and tested running multiple practical applications to show the applicability and novelty.

The paper is organized as follows. Firstly, related works and our contribution are mentioned in Section II. Then, Section III describes details about the proposed system and Section IV gives the problem definition and solution of task assignment and delivery on edge networks. Finally, Section V reports and concludes the experimental results.

II. Related Work

A. Information Flow Processing: History and Background

A long history of processing engines for continuous and timely information is early surveyed in [1]. The authors have scrutinized 34 works and classified them into three groups: (i) active database, (ii) data stream management, and (iii) complex event processing or CEP. Since the first group relies on persistent storages, it often has scalability issues caused by the growth of the number of rules and the frequency of event arrival rates. Over the past decade, it becomes obsolete. On the other hand, the second and third groups are still commonly investigated and a lot of research efforts have been dedicated so far. Owing to the progress of those research, principles mentioned in [1] are insufficient to determine the class of recent engines. The term *CEP* no longer implies the notification event data models, neither rule-based nor automaton-based approaches. Throughout this paper, we classify the CEP approaches by their data models, i.e., the stream-based model and the event-based model.

B. Stream-based model for CEP

A stream-based model has been developed from the relational database. Most of the query representations refer to SQL, e.g. CQL[2]. Commonly, the processing approach has three steps: (1) window streams (2) process relation tables, and (3) produce streams from result relations. It allows operations of high-relevant data such as video processing with comparatively low latency. To the best of our knowledge, the existing stream-based software such as *Stream*[2], *Apache Spark Streaming* [4], *Apache Storm*[5], *TIBCO StreamBase* [3], WSO_2 [6], are deployable only on a centralized, or clustered nodes with static topology. Unfortunately, most of them have no concern about environmentally-adaptive processing to cope with dynamicity of computational/network resource availability due to new query injection, mobility, and so on.

C. Event-based Stream Filtering

An event-based model generally adopts pattern matching and content filtering along with highly-expressive event specification. Instead of advanced setting as in the steam-based model, sources can be recognized by advertisements in runtime. Due to this nature, the event-based model allows fully-distributed deployment, meanwhile, operations of high-relevant data incur a great cost. The event-based CEP engines are usually driven by rules [8][7][17] or automata [9][10]. Formerly, a *PADRES*[7] was proposed as a rule-based distributed Pub/Sub system by mapping subscriptions to rules, and publications to facts. Composite subscriptions are straightforwardly decomposed if composited events are adverted from both sides of a binary tree. However, only conjunction and disjunction are available. *RACED*[8] widens expressiveness of the subscription by TESLA language, makes complex event detection available in a distributed manner, and proposes a master-slave subscription protocol to reduce the number of non-potential packets. *Adaptive Content-Based Routing in General Overlay Topologies* is proposed in [18] to handle cyclic and dynamic topology. The state-of-the-art rule-engine proposed in [17] aims for large-scale, real-time systems like smart building systems. It uses the minimal perfect hash function for filtering and dynamic adaption scheme for blocking non-potential rules.

D. Analysis of Existing CEP in Task Distribution Perspective

To overcome a scalability issue, many proposals attempt to distribute task executions over multiple nodes called brokers. For the event-based model, most of the processing tasks in distributed systems are assumed to be straightforwardly decomposable (e.g. conjunction and disjunction). A general framework of task distribution is stated in [9] concerning link usage and computational effort under-addressed distribution and detection policies. However, to the best of our knowledge, the existing policies rely on routing protocols and adopt only the nearest-to-source heuristic [7][8][14]. We refer to them as hop-based approaches. Meanwhile, since most of the stream-based engines leave decisions to the users, there is few research that deals with the mechanisms for autonomous

distribution. One of them is Ref. [19], which optimizes operation placement from the given queries. A few state-of-art research have focused on a task mapping and scheduling considering resource limitations in wireless sensor networks. A general algorithm is formulated in [15]. Some cost functions are introduced in terms of energy consumption [20][21]. Ref. [22] proposes a buyer-seller computational task-assignment framework for wireless sensor networks with an auction-based mechanism, which is similar to the hand-over algorithm in [23]. Most recently, Ref. [16] utilizes workload estimation from [24] together with operator-profiling for parallelizing. However, none of the above proposals considers delivery cost from processors to destinations, which is an additional concern for distribution on edge networks.

E. Our Contributions

Our contributions are three-fold listed as follows. Firstly, we design a general processing engine aiming for IoT edge devices over wireless networks. It can obsolete the assumption of source knowledge in stream processing and remain relational storage for efficiently processing high-relevant data *in a fully-distributed manner*. To achieve this, we especially define a new event specification language to express relational operation, and design and develop a broker-based middleware for its distributed execution. Secondly, we formulate a problem of task assignment and delivery planning to optimize the communication cost under node and link constraints and propose a fully distributed solution. Thirdly, we have evaluated our proposed algorithm with the centralized and hop-based approaches by simulating a large-scale nursing-care scenario, and most importantly, we have prototyped our proposed system on a wireless sensor network with realistic applications.

III. Design and Architecture

A. Architecture Overview

Our system is composed of multiple *brokers* collaborating over self-organized ad-hoc networks. Brokers might be just tiny computer modules like Intel Edison or even high-efficiency servers. There are three role players connecting to our system via the brokers: *subscriber*, *sensor* and *actor*. Subscribers commit subscriptions. A subscription describes the ways to detect, analyze and generate an output. Sensors sense surrounding environments and produce information flows. Actors activate an action in response to the processed outputs.

B. Supportive Language

Inspired by TESLA, we define a specification language for relational expressions with the structure represented below:

define	$Name(Att_1, ..., Att_n)$ [aggr] [every T]
$< pre >^*$	
[**case** n:]	
detect	$Pattern(content_1, ..., content_m)$
assign	$attr_1 = f_1, ..., attr_p = f_p;\ f = F(content)$
[**consuming**	$e_1, ..., e_h; e_i \in content$]
$< post >^*$	
where*	$Att_1 = g_1, ..., Att_n = g_n;\ g = G(attr)$
[***group by***	$(location \vert srcid)$]

*only for aggregation specification (aggr)

Basically, creating a specification has only three steps: (1) address the name and attributes of events at *define* key (2) state interest pattern of events in terms of a set of contents at *detect* key, and (3) declare how to produce an output from the detected events at *assign* key. Note that, we use the term "content" to denote a set of events with values that satisfy the specific conditions. For example, if the condition "Temperature.$val > 40°C$" is specified, the event "Temperature($val = 42°C$)" will be included in the content set of "Temperature.$val > 40°C$". The second and third steps can be done more than one time to define multiple interest patterns in some specifications. To illustrate, we assume *Tracking* sensors are always deployed at the border between two-sided locations and provide that *direction* equals to "0" when someone moves from right to left and "1" when moves from left to right as presented in Fig. 2. The *MoveIn* specification for an event when someone moves in one location can be defined as below.

define	$MoveIn(location, timestamp)$
case 0:	
detect	$Tracking(direction = 0)$
assign	$location = Tracking.left$,
	$timestamp = Tracking.timestamp$
consuming	$Tracking$
case 1:	
detect	$Tracking(direction = 1)$
assign	$location = Tracking.right$,
	$timestamp = Tracking.timestamp$
consuming	$Tracking$

In addition, not only name and attributes, the aggregation keyword (*aggr*) and time-window (**T**) to perform assignment and produce an event are able to be specified in the *define* key. When the aggregation keyword is stated, the specification must be specified into two parts: *pre* and *post*. In the same way as the map and reduce method, the former is supposed to be distributed detecting and assigning and the latter is for concluding the final results. *Pre* part is same as the basic specification mentioned above while *post* part has only one key, *where*, to declare the function of assigned values from *pre* to event attributes. Note that time-window must be defined if the specification is aggregation to designate the time to perform *post* assignments. The following example is the specification of people counts in each location in every 2 minutes. Note that, *group* keyword is used to represent group-parameters at runtime and *outer* stands for the outer join operation.

define	$PeopleCount2Mins(location, count, timestamp)$
	aggr every 2 mins
$< pre >$	
detect	outer *MoveIn* and outer *MoveOut*
assign	$location$=group.$location$,
	$count$=Count($MoveIn$)-Count($MoveOut$),
	$timestamp$=Max($MoveOut.timestamp$)
$< post >$	
where	$location$=group.$location$, $count$=Sum($count$),
	$timestamp$=Max($timestamp$)
group by	$location$

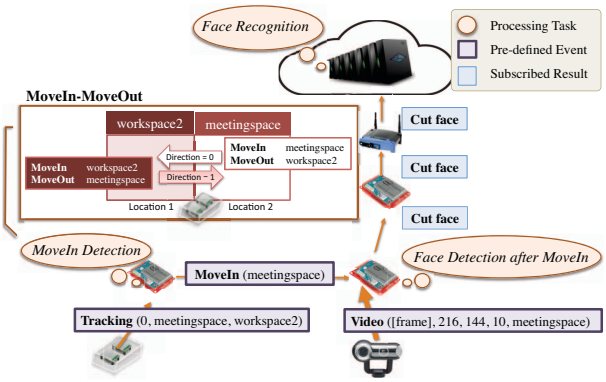


Fig. 2. Task Distribution: Face-Recognition Example

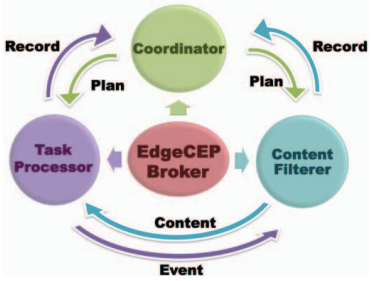


Fig. 3. Broker's Component Diagram

C. Broker Design

An individual broker is mainly composed of three packages: *Content Filterer*, *Task Processor*, and *Coordinator* (Fig. 3).

1) Content Filterer: This module works by the publish-subscribe mechanism. It provides pseudo-sources by mapping general form of an incoming event into *Interest Content*, before streaming to corresponding task agents in *Task Processor*. For instance, responding to the definition specification in *MoveIn*, an event, "Tracking(*direction* = 0)", will be matched to the content, "Tracking.*direction* = 0", and passed to the task agent *MoveIn* together with the unique id of the matching content. In other words, the filterer acts as a host with multiple sources. To achieve that, we apply minimal perfect hash to filter irrelevant events. Hash keys are combinations of source id, location, and event type (any of them can be omitted). Each hash item refers to a set of specified conjunction conditions. For instance, the specification, "$20°C <$ Temperature.*val* $< 25°C$", will generate conjunction conditions, "$val < 25°C \wedge val > 20°C$", to be mapped from a hash key *(-,-,Temperature)*.

2) Task Processor: In our system, there are two types of tasks called definition task and subscription task. The former is specified by the language stated in Section III-B beforehand and available as references while the latter is requested by users with a slightly different format (see Table I). In particular, the specification of subscription tasks has no attributes and requires subscriber, sequence, and actor. A task forks one agent for one detecting case. For example, two agents will be forced for the task *MoveIn*. An agent works as follows. Firstly, the interesting pattern, a sequence of subscribed contents specified in *detect*, is detected. For instance, the first subscription from Table I derives two states, *PeopleCount* and *AvgTempOut*, where *PeopleCount* comes before *AvgTempOut*. Each state owns a relational storage to memorize the relevant events. Note that, we use JDBC driver for SQLite connection. Secondly, when the producing condition is satisfied, it will produce an output as stated in *assign* from the events reaching the final state. A producing condition can be specified in *define* state with time-window T. If T is not specified, the output will be produced as soon as an automaton reaches the final state. Outputs of definition tasks are pushed back to *Content Filterer* while the outputs of subscription tasks are delivered directly to the destination broker.

3) Coordinator: In runtime, an individual broker will periodically make its own decision on whether or not to execute any specific tasks. Based on the decision, it will leave unexecuted tasks to other brokers. Through this paper, we use the terms *plan* to stand for the above-mentioned decision. A plan is determined by *Coordinator* to minimize resource utilization and keep the specified constraints satisfied using the knowledge of connections and historical records from *Content Filterer* and *Task Processor*. Our approach basically adopts the following two heuristics: (1) nearest to the source, and (2) shortest path to destinations. Fig. 2 shows an example of a simple distribution plan to cut face images from video stream when someone moved into the interest location. The task is to do *Face Detection* after *MoveIn* event occurs. Regardless of all constraints, the tasks, *MoveIn* and *Face detection*, are both assigned by the above heuristics. The particular problem definition and the solution are addressed in Section IV.

IV. Task Assignment and Delivery

A. Problem Definition

To decide the plan locally, every broker is supposed to maintain all definitions and subscriptions, referred to as tasks. Subscription tasks are always active while definition tasks will be active only when they are required for any subscription tasks. A plan is composed of (i) assigning plan, i.e., designating processing tasks to broker nodes, (ii) forwarding plan, i.e., determining paths of inputs from the originating broker to the designated broker, and (iii) delivering plan, i.e., determining paths of outputs from the designated broker to the destination. Presume information about available resources in the system, we formally give a problem definition to minimize resource utilization on edge networks in terms of the total flow volume from sources to destinations of all active tasks under resource and consistency constraints as below.

Problem Definition 1. Consider the edge network of k bi-directional, shared-channel connecting broker nodes $B = \{B_1, B_2, ..., B_k\}$. Presume global knowledge: (1) distance matrix $H = (h_{qps}) \in \mathbb{Z}^{k\times k\times k}$ as hop count from node q to node s via node p, (2) link-constraint matrix $L = (l_q) \in \mathbb{Z}^k$ as link capacity, and (3) node constraint $\Theta = (\theta_q) \in \mathbb{Z}^k$ as processing capability.

TABLE I
SUBSCRIPTION DEFINED IN REAL-ENVIRONMENT SYSTEM

Subscription	seq 1 from *User*1 to *AirController*
detect	last *PeopleCount2Mins*(*count* > 0) as *PeopleCount* and *AvgTemp5Mins*(*avg* < 20 or *avg* > 25) as *AvgTempOut* within 5 mins from *PeopleCount*
assign	*avg*=*AvgTempOut.avg*, *location*=group.*location*, *timestamp*=*ProcleCount.timestamp*
consuming	*AvgTempOut*
group by	*location*

Subscription	seq 2 from *User*1 to *AirController*
detect	last *PeopleCount2Mins*(*count* > 0) as *PeopleCount* and *AvgHumid5Mins*(*avg* < 30 or *avg* > 50) as *AvgHumidOut* within 5 mins from *PeopleCount*
assign	*avg*=*AvgHumidOut.avg*, *location*=group.*location*, *timestamp*=*ProcleCount.timestamp*
consuming	*AvgHumidOut*
group by	*location*

Subscription	seq 3 from *User*1 to *Gateway*
detect	*MoveIn* and *Video* within 5 mins from *MoveIn*
assign	*face*=FaceDetect(*Video*), *location*=group.*location*, *timestamp*=*MoveIn.timestamp*
consuming	*Video*
group by	*location*

Given a set τ of n processing tasks with the designated destination $Dest$, we let matrix $A = (a_{ij}) \in \{0,1\}^{n\times m}$ denote mapping of task i to content j from total m contents. Suppose statistical predictions for each node q to provide (1) $\Phi = (\phi_{qj}) \in \mathbb{Z}^{k\times m}$ as the input flow volume of content j, (2) $\Psi = (\psi_{iq}) \in \mathbb{Z}^{n\times k}$ as the output flow volume of task i and (3) $\Omega = (\omega_{iq}) \in \mathbb{Z}^{n\times k}$ as the execution cost of task i. Define a task-assignment plan $X : \tau \mapsto B$ and corresponding plans for forwarding $F : \tau \mapsto B$ and delivering $D : \tau \mapsto B$ as follows:

$$X : x_{iqt} = \begin{cases} 1;\ if\ task\ i\ of\ node\ q\ is\ entrusted\ to\ node\ t \\ 0;\ otherwise \end{cases}$$

$$F : f_{iqst} = \begin{cases} 1;\ if\ x_{iqt} = 1\ and\ node\ s\ \in Path(q,t) \\ 0;\ otherwise \end{cases}$$

$$D : d_{iqst} = \begin{cases} 1;\ if\ x_{iqt} = 1\ and\ node\ s\ \in Path(t, Dest_i) \\ 0;\ otherwise \end{cases}$$

A coefficient matrix of input flow can be computed as:

$$E^{k\times k\times k\times m} : e_{qstj} = \begin{cases} 1\ ; \exists_{i=1}^{n} f_{iqst} \cdot a_{ij} = 1 \\ 0\ ; otherwise \end{cases}$$

Then the problem is to find X, F and D with the following objective function and constraints.

$$Minimize\ \sum_{q\in B}\sum_{s\in B}\sum_{t\in B}\left\{\sum_{j=1}^{m} e_{qstj}\phi_{qj} + \sum_{i=1}^{n} d_{iqst}\psi_{iq}\right\}$$

subject to
(1) resource constraints:

$$\forall_{q\in B}\ \sum_{i=1}^{n} x_{iqq}\cdot\omega_{iq} \le \theta_q\ and\ \forall_{s\in B}\ \sum_{q\in B}\sum_{t\in B}\{\sum_{j=1}^{m} e_{qstj}\phi_{qj} + \sum_{i=1}^{n} d_{iqst}\psi_{iq}\} < l_s$$

(2) consistency constraints:

$$\forall_{i\in\tau}\forall_{q,s\in B;q\neq s}\ max(1 - x_{iqs}, x_{iss}) = 1$$

To solve problem 1 above, all brokers need to obtain the up-to-date global knowledge about topology and link/node capacities. Since such information is often unachievable in practical, we additionally define another problem for individual nodes to solve with limited local knowledge as below.

Problem Definition 2. Given that each node q maintains a set of local members $G = \{q\} \cup \{s; h_{qqs} \le r\}$, where r is a specified hop radius. Each member in G owns the following information: (1) one-hop neighbor $R_g \subset B$, (2) minimum hop to destination of all tasks $hoptodest_g^{\tau}$, (3) residual link capacity ll_g, (4) available processing capability $l\theta_g$, and (5) the last assigning plan $lastX_g : \tau \mapsto G_g$ and last forwarding plan $lastF_g : \tau \mapsto R_g$. With limited knowledge, the determined path to destination may be not available. Redefine the delivering plan (D) from Definition 1 as follows:

$$D : d_{iqst} = \begin{cases} 1;\ if\ x_{iqt} = 1\ and\ \exists\ Path(t, Dest_i) \\ hoptodest_s^i;\ if\ x_{iqt} = 1\ and\ \nexists\ Path(t, Dest_i) \\ 0;\ otherwise \end{cases}$$

Given tasks and statistical predictions in the same way as Definition 1, the problem is to find next individual plans $nextX \in \{0,1\}^n$ and $nextF : \tau \mapsto R$ that

$$Minimize\ \sum_{s\in G}\sum_{t\in G}\left\{\sum_{j=1}^{m} e_{qstj}\phi_j + \sum_{i=1}^{n} d_{iqst}\psi_i\right\}$$

subject to
(1) node constraints: $\forall_{s\in G}\ \sum_{i=1}^{n} x_{iqs}\cdot\omega_i \le l\theta_s$
(2) link constraints: $\forall_{s\in G}\ \sum_{t\in G}\left\{\sum_{j=1}^{m} e_{qstj}\phi_j + \sum_{i=1}^{n} d_{iqst}\psi_i\right\} \le ll_s$

In general, a solution from Definition 2 does not guarantee a global solution in Definition 1. However, if all of the vicinity groups cover all brokers and all plans are consistent, a set of local solutions is a global solution (see Theorem 1 in Appendix).

B. Solution

The problem addressed above can be induced to the minimization version of the Generalized Assignment (MINGAP) problem by fixing all tasks with a single input from only one source with multiple constraints. According to [26], MINGAP is equivalent to a Generalized Assignment Problem (GAP). The partition problem of a given set of positive integers into two equal-sized subsets with m = 2 can be reduced to GAP. Correspondingly, the above-defined problem is an NP-hard. So, we propose a heuristic algorithm running as follows.

To get the local solution respective to Definition 2 at any individual broker, we apply a well-known technique for MIN-GAP problem called *tabu search* with greedy move (Appendix Algorithm 2). For simplicity, we redefine assignment plan and delivery plan to $X = (x_i) \in \{1, ..., g\}$ and $F = (f_i) \in \{1, ..., r\}$ where $x_i = s$ means to suppose task i to be executed at node s and $f_i = s$ means to forward content flows of task i to neighbor s. Firstly, each node q pre-estimates and sorts flow volumes v_{is} for any assignment of task i to any vicinity group member s by the equation below:

$$v_{is} = \left\{ \sum_{j=1}^{m} a_{ij}(lh_{qs} \times \phi_j) \right\} + hoptodest_s^i \times \psi_i$$

For each move in tabu search, a candidate is orderly picked up from the sorted list from each task. The chosen local move is the candidate that produces a minimum value determined by the above-objective function in Definition 2 (see Appendix Algorithm 3). A valid move must not violate the node and link constraints defined in Definition 2. To avoid loop-forwarding, at node q, the task i cannot be forwarded to node s if it meets all of the following conditions in the same time: (1) node s currently forwards task i to itself, $lastF_s(i) = q$ (2) previous X used to forward task i to node s, $lastF(i) = s$. If there is no feasible solution in the iteration, it will choose X that does not violate loop-forwarding condition and has minimum total violation cost ($\sum_{s \in G} vv_s$), where

$$vv_s = \frac{Violate\Omega_s}{\theta_s} + \frac{Violate\Phi_s}{ll_s};$$

In term of computation complexity, let y and p denote the bounded number of iterations and possibilities to move for each iteration respectively. The complexity of pre-calculating program is $O(npk)$ and the complexity of greedy move depends on $estimateT$, $valid$, and tabu-searching which are $O(npk)$, $O(pk)$, and $O(y)$, respectively. So the complexity of our algorithm is $O(y(npk+pk))= O(ynpk)$ for an individual node and $O(ynpk^2)$ for k nodes, while the complexity of full searching is $O((pk)*k!)$.

For some processing tasks such as finding an average, the results could not be determined until knowing all relevant data collected from all brokers. In this paper, we use the term, *aggregation task*, to stand for such a kind of tasks. In common, the aggregation task needs to be done on a specific node. However, in distributed systems, the aggregation is usually divided into two steps. The first step is distributedly executed while the second step is carried out at one node. In the similar way, we allow distribution by *pre-post* specification, mentioned in Section III-B. To decide the header node to perform the second step, we exploit a blind bid protocol that leads to only one agreement by evaluating values of all candidates from topology and identification. Suppose the aggregation result to be periodically produced, the header node will make a request and wait for all replies with a specific timeout to execute post-processing. The output will be forwarded to all candidates if it is a definition task that any of referring tasks also refers to other non-aggregation tasks.

V. Implementation and Experimental Results

This section reports our findings from the logical experiment, performance of proposed distribution solution in the large-scale simulation, and utilization of the prototype in the real environment as follows.

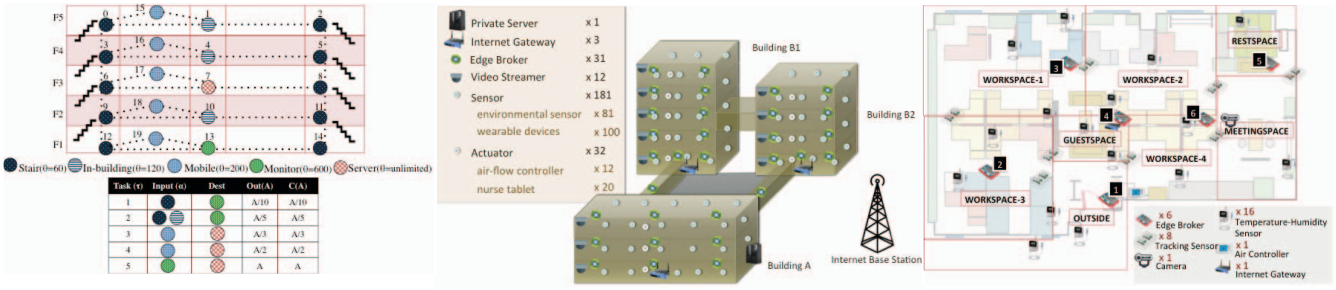
A. Logical Experiment

To study affecting factors of our method, we implement the proposed algorithm in JAVA and test with the controlled scenario as depicted in Fig. 4(a). This scenario assumes data processing and collection from different sensors installed in three buildings. To reduce the complexity of communication paths, link constraints are omitted in this experiment. Total flow volumes for different producing rates with different hop radius are summarized in Fig. 5(a). The results confirm that available knowledge, reflected by hop radius, has an influence on the distribution decisions and respectively affect the total volume. Particularly, we observe a few interesting results as follows. Firstly, insufficient knowledge, *producing rate = 100, radius = 1*, can cause extremely large volume and delays. Secondly, the optimal solution ($radius \geq 3$) needs to a larger total volume at early steps, as seen in Fig. 5(b), because it assigns the node farther from source for the better result in overall path. Thirdly, larger radius is not always better decision, e.g. *producing rate = 200, radius = 3*. According to Theorem 1 in Appendix, the all-covering radius can lead to the global solution. At the same time, the higher radius causes a larger amount of overheads. Thus, we have to trade off between this downside and the affected efficiency. In some scenario, the optimal solution can be guaranteed at a smaller radius than all-covering one. For example, in the test scenarios, we observe that the radius that covers the resource-abundant server can guarantee the global solution. With the same scenarios, we compare our proposed algorithm with the hop-based approach and the result, as presented in Fig. 5(c), shows that our proposed algorithm always achieves less or equal costs.

B. Large-scale Simulation

In this experiment, we use the same Java program as the above experiment to compute a plan responding to a smart nursing home scenario as depicted in Fig. 4(b). Concretely, input flows includes 1.5KB/min from an environmental sensor, 1.5KB/s from a wearable device, and 6.75MB/s from a video streamer and applications are (i) General surveillance sending to a local server, (ii) Face detection on Building A sending to Internet gateway, (iii) Average environmental information sending to an air flow controller for each floor, and (iv) Heart-rate pattern detection sending to the nearest nurse tablet.

The results are summarized in Table II(a), where *cover flood* denotes flooding to all nodes, *guarantee flood* denotes maximum hop to reach the server, and *optimal flood* denotes minimum hop that provides the same result as a cover flood. Our algorithm achieves about 6.6 times cost reduction from the centralized approach, and the cost is slightly lower than the hop-based approach. Table II(b) shows the overhead comparisons between assigning approaches. The total number



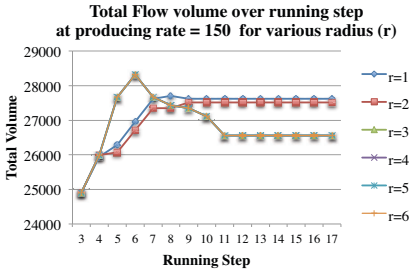
(a) Logical Test Scenario **(b)** Large-scale Smart Nursing Home Scenario **(c)** Real-environment Scenario

Fig. 4. Evaluation Scenarios

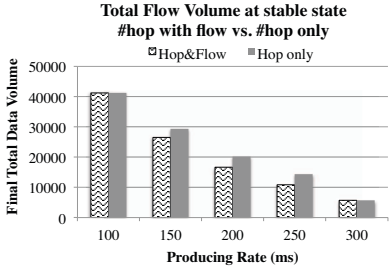
Total Flow Volume at stable state for various radius at each producing-rate scenarios

Producing rate \ Radius	1	2	3	4	5	6
100	73,440	41,280	41,280	41,280	41,280	41,280
150	27,623	27,515	26,555	26,555	26,555	26,555
200	16,670	16,670	16,950	16,670	16,670	16,670
250	10,824	10,824	10,824	10,824	10,824	10,824
300	5,650	5,650	5,650	5,650	5,650	5,650

(a)



(b)



(c)

Fig. 5. Flow-based Task Distribution Algorithm Results with Logical Test Scenario of Fig 4(a)

TABLE II
LARGE-SCALE SIMULATION RESULTS

(a) Flow statistics

Criteria	Flow based(n)	Hop based(n)	Centralized(n)
Data	11,833	11,838	84,501
Overheads	958	958	0
Total	12,791	12,796	84,501

(b) Overhead

Assigning Approach		Packets	Bytes
Distributed	Cover flood (Hop=11)	958	91,968
	Guarantee flood (Hop=9)	827	79,392
	Optimal flood (Hop=1)	31	3,976
Centralized	Server pushing	131	138,991

of packets in the distributed approach is higher than the centralized approach in general. However, the packet size of the centralized approach is much larger depending on the number of contents. In the simulated scenarios, the total size is even larger than the cover flooding in the distributed approach.

C. Real-environment Deployment

Our prototype is implemented in JAVA using JDBC, javaCV, and H2-group MHPF library, and installed in the Intel Edison with the yocto-standard kernel. Six broker nodes were connected to an ad-hoc network with the Better Approach To Mobile Adhoc Networking (B.A.T.M.A.N.) protocol [27]. The experiment was conducted in our laboratory as depicted in Fig. 4(c). Concretely, a video camera flooded 20 fps of the 216x144-size frame and 16 temperature-humidity sensors sensed every one minute. We define three subscriptions as presented in Table I.

The experiment runs for two hours with 10-minutes window. From the results concluded in Table III, the second and third subscriptions as well as required definition tasks are mainly executed at the destination-nearest brokers due to aggregation overhead while the first subscription is executed at the source-nearest brokers. In terms of performance, brokers 1 and 2 gain 25% higher communication cost affected by aggregation cost and information-exchange overhead. In the same time, communication costs on the rest brokers are reduced at least 60%, especially video stream from broker 6 to 4.

VI. CONCLUSION

This paper proposed a general complex event processing system driven by a fully-distributed collaboration of devices at edge networks called brokers. The proposed broker can achieve the event-based specification and detection, together with tuple-based processing by first-introduced pseudo-source mechanism employing publish-subscribe and content matching techniques. A new event specification language is defined to support relational operations. To distribute the processing tasks under resource constraints on edge networks in fully-distributed fashion, an optimization problem of task assignment and delivery is formally defined and a tabu search with flow-based greedy move algorithm is introduced as a solution. Evaluations show that the proposed algorithm always achieves the smallest total flow volume with sufficient local knowledge of brokers. According to the large-scale simulation results, our method approaches to 6.6 times total packets less than the centralization. Finally, a prototype engine is successfully deployed over an ad-hoc wireless sensor network composed of

TABLE III
AVERAGE FLOW VOLUME (BYTES PER WINDOW) FROM TWO-HOURS RUN OF REAL-ENVIRONMENT DEPLOYMENT (θ = 0.1 s, l = 100 KB/s)

ID	Tasks	Base Cost	Flow-based Cost				Reduction Ratio
			Output Flow	Aggregation Cost	Overhead	**Total**	
1	*PeopleCount, *AvgTemp, *AvgHumid Subscription1, Subscription2, Subscription3	9,201.4	659.3	4,255.3	1,988.9	11,312.9	0.8
2	PeopleCount, AvgTemp, AvgHumid, Subscription1	2,950.0	0	796.8	1,987.8	3,826.7	0.8
3	PeopleCount, AvgTemp, AvgHumid, Subscription1	9,010.4	0	756.6	1,989.0	3,869.5	2.3
4	PeopleCount, AvgTemp, AvgHumid, Subscription1	11,995.2	294	772.6	2,092.7	4,396.3	2.7
5	PeopleCount, AvgTemp, AvgHumid, Subscription1	5,996.6	0	756.2	1,988.7	3,847.8	1.6
6	PeopleCount, AvgTemp, AvgHumid, Subscription1	422.5M	294	796.7	1,988.7	4,646.6	90,923.4

* header of aggregation task

Intel Edison modules in the real environment and the running result presents decreasing of communication cost in general.

ACKNOWLEDGMENT

The research results have been achieved by "Research and development of Innovative Network Technologies to Create the Future", the Commissioned Research of National Institute of Information and Communications Technology (NICT), JAPAN. The work was also supported in part by JSPS KAKENHI JP26220001, JP15H02690 and JP16KT0106.

REFERENCES

[1] G. Cugola and A. Margara, "Processing flows of information: From data stream to complex event processing," *ACM Comput. Surv.*, vol. 44, no. 3, pp. 15:1–15:62, Jun. 2012.

[2] A. Arasu, S. Babu, and J. Widom, "The cql continuous query language: Semantic foundations and query execution," *The VLDB Journal*, vol. 15, no. 2, pp. 121–142, Jun. 2006.

[3] (2015) StreamBase. [Online]. Available: http://www.streambase.com

[4] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica, "Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters," in *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Ccomputing*, ser. HotCloud'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=2342763.2342773

[5] Storm: Distributed realtime computation system. [Online]. Available: http://storm.apache.org/

[6] WSO2. [Online]. Available: http://wso2.com/

[7] G. Li and H.-A. Jacobsen, "Composite subscriptions in content-based publish/subscribe systems," in *Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware*, ser. Middleware '05. NY, USA: Springer-Verlag New York, Inc., 2005, pp. 249–269.

[8] G. Cugola and A. Margara, "Raced: An adaptive middleware for complex event detection," in *Proceedings of the 8th International Workshop on Adaptive and Reflective MIddleware*, ser. ARM '09. NY, USA: ACM, 2009, pp. 5:1–5:6.

[9] P. R. Pietzuch, B. Shand, and J. Bacon, "A framework for event composition in distributed systems," in *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, ser. Middleware '03. New York, NY, USA: Springer-Verlag New York, Inc., 2003, pp. 62–82.

[10] G. Cugola and A. Margara, "Complex event processing with t-rex," *J. Syst. Softw.*, vol. 85, no. 8, pp. 1709–1728, Aug. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2012.03.056

[11] R. BHARGAVI and V. VAIDEHI, "Semantic intrusion detection with multisensor data fusion using complex event processing," *Sadhana*, vol. 38, no. 2, pp. 169–185, 2013. [Online]. Available: http://dx.doi.org/10.1007/s12046-013-0118-4

[12] Statista, "Internet of things (iot): number of connected devices worldwide from 2012 to 2020 (in billions)," January 2017. [Online]. Available: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

[13] 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," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, Sep. 2015. [Online]. Available: http://doi.acm.org/10.1145/2831347.2831354

[14] G. Cugola and A. Margara, "Deployment strategies for distributed complex event processing," *Computing*, vol. 95, no. 2, pp. 129–156, 2013.

[15] M. H. A. Awadalla, "Task mapping and scheduling in wireless sensor networks," *IAENG International Journal of Computer Science*, vol. 40, no. 4, pp. 257–265, 2013.

[16] R. Mayer, B. Koldehofe, and K. Rothermel, "Predictable low-latency event detection with parallel complex event processing," *IEEE Internet of Things Journal*, vol. 2, no. 4, pp. 274–286, Aug 2015.

[17] Y. Sun, T. Y. Wu, G. Zhao, and M. Guizani, "Efficient rule engine for smart building systems," *IEEE Transactions on Computers*, vol. 64, no. 6, pp. 1658–1669, June 2015.

[18] G. Li, V. Muthusamy, and H.-A. Jacobsen, *Middleware 2008: ACM/IFIP/USENIX 9th International Middleware Conference Leuven, Belgium, December 1-5, 2008 Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, ch. Adaptive Content-Based Routing in General Overlay Topologies, pp. 1–21.

[19] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer, "Network-aware operator placement for stream-processing systems," in *Proceedings of the 22Nd International Conference on Data Engineering*, ser. ICDE '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 49–. [Online]. Available: http://dx.doi.org/10.1109/ICDE.2006.105

[20] Y. Tian, E. Ekici, and F. Ozguner, "Energy-constrained task mapping and scheduling in wireless sensor networks," in *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference, 2005.*, Nov 2005, pp. 8 pp.–218.

[21] J. W. L. Heemin Park, "Task assignment in wireless sensor networks via task decomposition," *Information Technology And Control*, vol. 41, no. 4, pp. 340–348, 2012.

[22] A. T. Zimmerman, J. P. Lynch, and F. T. Ferrese, "Market-based computational task assignment within autonomous wireless sensor networks," in *2009 IEEE International Conference on Electro/Information Technology*, June 2009, pp. 23–28.

[23] B. Dieber, L. Esterle, and B. Rinner, "Distributed resource-aware task assignment for complex monitoring scenarios in visual sensor networks," in *2012 Sixth International Conference on Distributed Smart Cameras (ICDSC)*, Oct 2012, pp. 1–6.

[24] D. G. Feitelson, *Workload Modeling for Computer Systems Performance Evaluation*, 1st ed. New York, NY, USA: Cambridge University Press, 2015.

[25] G. Cugola and A. Margara, "Tesla: A formally defined event specification language," in *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '10. NY, USA: ACM, 2010, pp. 50–61.

[26] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations (Wiley Series in Discrete Mathematics and Optimization)*. Wiley, 1990.

[27] Freifunk, "B.a.t.m.a.n. advanced," January 2017. [Online]. Available: https://www.open-mesh.org/projects/batman-adv/

Appendix

Theorem 1. *If all of the vicinity groups cover all brokers and all plans are consistent, a set of local solutions is equivalent to the global solution.*

Proof: At any node q, once the vicinity group covers all brokers information (i.e. $G = B$ and $g = k$), every node has a path to all destinations as long as there is a feasible solution for global problem (i.e. $\forall_{t\in G}\forall_{1\le i\le \tau.size}\ Path(t, Dest_i)$ exists). As a result, the definition function of D in Definition 2 is equivalent to the definition function of D in Definition 1. The objective function in Definition 2 can be rewritten as:

$$Minimize \sum_{s\in B}\sum_{t\in B}\left\{\sum_{j=1}^{m} e_{qstj}\phi_j + \sum_{i=1}^{n} d_{iqst}\psi_i\right\}$$

Correspondingly,

$$\sum_{q\in B} \text{argmin} \sum_{s\in G}\sum_{t\in G}\left\{\sum_{j=1}^{m} e_{qstj}\phi_j + \sum_{i=1}^{n} d_{iqst}\psi_i\right\}$$
$$= \text{argmin} \sum_{q\in B}\sum_{s\in B}\sum_{t\in B}\left\{\sum_{j=1}^{m} e_{qstj}\phi_{qj} + \sum_{i=1}^{n} d_{iqst}\psi_{iq}\right\}$$

Construct a concatenated assigning plan, (xx) from all local assigning plans orderly.

$$[xx_i] = ([x_{i1}]|[x_{i2}]|...|[x_{ik}])$$

An assigning plan will be considered as consistent if the concatenated plan satisfy the following condition:

$$\forall_{i\in\tau}\forall_{q,s\in B;q\neq s}\ max(1 - xx_{iqs}, xx_{iss}) = 1$$

With the same routing function, *Path* will return the same matrix for the same assigning plan X, source, destination. The constraints in Definition 1 can be applied to the concatenated assigning plan and corresponding forwarding and delivering plan equivalently to those in Definition 2.

$\therefore$ a set of local solutions is equivalent to global solution ff all vicinity groups cover all brokers and all plans are consistent.

```
def initRefT(LH, G, τ, A):
    Data: node id−q
    Result: sortedT: τ.size-List of Candidate
    sortedT = new List[τ.size];
    for i = 1 to n do
        candidates = new List;
        fwdvol_i ← Φ_i · A_i; outvol_i ← Ψ_i · A_i;
        for s ∈ G do
            hop ← LH_qs; htd ← hoptodest_s^i;
            value ← fwdvol · hop + outvol · htd;
            add new Candidate(s,value) to candidates
        end
        sortedlist ←sort candidates by value
        sortedT[i] ← sortedlist
    end
```

Algorithm 1: Pseudo codes of sorted alternative table

For an arbitrary node:

```
Data: LH, G(R, hoptodest, ll, lθ, lastX, lastF),
      τ(Φ, Ψ, Ω, Dest), A, stop
Result: nextX, nextF
tabu ← φ;
sortedT=←initRefT(LH, G, τ, A);
if lastX_q=null then
    X ← sortedT.get(0); F ← getF(X);
else
    X=lastX_q; F=lastF_q;
end valid,value
←estimateV(X, F, LH, G, τ, A);
nextX ← X; nextF ← F;
minValid ← valid; minValue ← value;
add X to tabu;
while X ≠ null and !stopCondition() do
    (X, F, valid, value)
    ←next(tabu, sortedT, X, F, LH, G, τ, A);
    add X to tabu;
    if (¬minValid ∧ valid)∨
    ((¬minValid ∨ valid) ∧ (value < minValue)) then
        nextX ← X; nextF ← F;
        minValid ← valid; minValue ← value;
    end
end
```

Algorithm 2: Pseudo code of Local minimization

```
def next(tabu, sortedT, X, F, LH, G, τ, A):
    Result: nX, nF, nValid, nValue
    nX ←null; nF ←null; nValue ←infinity;
     nValid ← false;
    for i = 1 to n do
        for ci = 1 to sortedT[i].size do
            candidate ← sortedT[i].get(ci);
            tmpX ←replace(X,i,candidate.node) ;
            if tmpX not in tabu then
                X ← tmpX; F ← getF(X);
                (valid,value)
                ←estimateV(X, F, LH, G, τ, A);
                if (¬nValid ∧ valid)∨
                ((¬nValid ∨ valid) ∧ (value < nValue))
                 then
                    nX ← X; nF ← F;
                    nValid ← valid; nValue ← value;
                end
                break;
            end
        end
    end
```

Algorithm 3: Pseudo codes of Greedy Move