IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 26, 2019, accepted July 18, 2019, date of publication July 22, 2019, date of current version August 9, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2930313

A Universal Complex Event Processing Mechanism Based on Edge Computing for Internet of Things Real-Time Monitoring

LINA LAN[1,2], RUISHENG SHI[3], BAI WANG[2], LEI ZHANG[2], AND NING JIANG[3]

[1]School of Network Education, Beijing University of Posts and Telecommunications, Beijing 100088, China
[2]School of Computer, Beijing University of Posts and Telecommunications, Beijing 100876, China
[3]School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing 100876, China

Corresponding author: Ruisheng Shi (shiruisheng@bupt.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2016YFB0800302, in part by the National Natural Science Foundation of China under Grant 61802025, in part by the Fundamental Research Funds for the Central Universities under Grant 2018RC56, and in part by the Beijing Talents Foundation under Grant 2017000020124G062.

ABSTRACT The tremendous number of sensors and smart objects deployed in the Internet of Things (IoT) pose a huge potential for the IoT real-time monitoring applications to detect and react to the real world. The insufficient capacity of the IoT data real-time processing has hampered the growth of the IoT real-time monitoring applications. We focus on two issues of the IoT data real-time processing: 1) how to efficiently transform a large number of raw sensing data into meaningful complex event, and 2) how to adapt to the complexity and changeability of monitoring business logic. This paper proposes a universal complex event processing (CEP) mechanism for the IoT real-time monitoring. We propose a formalized hierarchical complex event model including raw event, simple event, and complex event, which reduces the complexity of event modeling. The model supports complex time and space semantics to define flexible complex events by a programming way. Based on this model, we propose a CEP system architecture, in which the system is deployed on the network edge between sensing devices in terminal and applications in the cloud. The complex event definition can be mapped to the CEP rule logic script to detect the potential abnormal event timely. The proposed CEP mechanism is universal and suitable to any heterogeneous sensing devices and CEP engine. We demonstrate the efficacy of the mechanism with two application case studies that highlight our proposed complex event model and evaluate the performance improvement with experiments.

INDEX TERMS Real-time monitoring, Internet of things, complex event processing, complex event definition, edge computing.

I. INTRODUCTION

With the rapid development of sensors, GPS position sensors, RFID tags and readers, smart objects and other IoT sensing technologies, the IoT real-time monitoring business is growing rapidly in many IoT innovative applications such as smart logistics, smart farm, environmental monitoring, intelligent transportation and smart power grid, etc [1]–[7]. A recent report predicted that 50 billion devices will be connected to Internet by year 2025 [8]. These IoT sensing devices are data generators which generate new perception data every moment, and provide huge potential for real-time monitoring and intelligent application in various industries.

The associate editor coordinating the review of this manuscript and approving it for publication was Chi-Yuan Chen.

For example, smart logistics application based on temperature and humidity sensors can monitor the logistics and storage status to grasp the storage situation of goods more comprehensively and accurately. Based on RFID electronic tag and reader, the whole process of transportation and storage can be timely tracked. Based on the camera's real-time image and video, the abnormal events can be detected timely. Therefore, the wide deployment of sensing devices makes all kinds of monitored objects becoming ''real-time visible''. Then, through data collection, processing and analysis, triggering specific response operations or business processes, the monitoring system can achieve various real-time monitoring goals.

The monitoring data volume of IoT is growing rapidly and has obvious big data characteristics [7], [9]. The IoT

monitoring system needs to process the data continuously in time when the data flow arrives to support real-time performance, and it should enhance its insight and decision ability to support complex business logic.

The data processing of traditional monitoring system is generally based on relational database. Before data processing, data needs to be stored and indexed, which causes great delay and lacks the ability to collect, process and analyze big data in real time. IoT monitoring data processing is facing new challenges. The architecture of the monitoring system needs to be redesigned for real-time data processing and responsiveness. The traditional B/S and C/S cloud-based architecture should change to be based on edge computing [4], [11]. Most of computing should be carried out on the edge device of intelligent monitoring equipment to improve the response speed [10]–[12]. The success of IoT monitoring applications largely depends on the ability to access, absorb and analyze heterogeneous data in a timely manner and to provide operators with advanced intelligent information. Real-time data processing technology based on data stream has become the key technology for monitoring IoT big data.

Complex Event Processing (CEP) is the main technology for extracting information [9]. CEP is a technique of data processing based on a set of predefined rules that dictate how data flows should be processed and what new event flows should be generated as output. Events can be a number of individual events of interest (e.g. higher than normal temperature) or complex events (e.g. continuous high temperature) that correspond to specific situations or patterns to the business.

CEP engine is a software system composed of a set of data processing algorithms and knowledge representation sets [13]. CEP engines for data flow processing typically have state management, fault tolerance, and high performance characteristics. IoT monitoring requires CEP engine to track the system status including event arrival and processing. CEP engine needs to process a large amount of continuous data with low latency, and needs to be able to recover quickly when system failure occurs.

At present, CEP is widely used in many fields such as industrial production monitoring [3], environmental monitoring [1], [2], [4], building safety monitoring [5], telemedicine monitoring [6], etc. Previous researches focus on data flow processing of CEP [9], [13], [14], architecture of CEP based on edge computing [6], [15], [16], high performance of parallel processing and system scalability [17], [18] of CEP. These studies have contributed to improve CEP performance in different ways.

However, focusing on real-time monitoring services, refactoring and computational performance are still unresolved issues for large-scale CEP system. How to efficiently transform a large number of raw sensing data into meaningful complex event is a challenge of high computational performance. On the other hand, how to adapt to the complexity and changeability of monitoring business logic of various fields is a challenge of the event refactoring capability. These challenges seriously restrict the development of IoT large-scale real-time monitoring applications. Existing research does not address these challenges well. Focusing on these two problems, this paper proposes a hierarchical event model and a general event processing system architecture to provide a solution for efficient real-time monitoring data processing. The event model supports flexible monitoring logic semantics, and be able to quickly reconstruct to adapt to monitoring requirements or changes in software and hardware environment. Meanwhile, the efficiency of event processing meets real-time requirements.

The main contributions of this paper are as follows:

1) It proposes a hierarchical model of IoT perceived event. The IoT event can be divided into original raw event, reusable simple event and customizable complex event according to the event granularity from small to large. The monitoring task is deconstructed, and complex event is defined by means of event modeling language.
2) It proposes a universal CEP system architecture based edge computing to support fast responsive application. The CEP system is deployed on the network edge between the sensing devices in terminal and the IoT applications in the cloud. The defined complex events can be mapped to business logic scripts. The complex event parsing and reasoning are implemented based on efficient CEP engine to support business event real-time monitoring efficiently.
3) Two application case studies of IoT monitoring for fruit transportation and city road manhole cover status are implemented based on the CEP system to highlight our proposed complex event model. And the performance evaluation is carried out to verify the efficiency of our CEP system.

The remainder of this paper is organized as follows: In Section II, we describe the related work. In Section III, we present the IoT event model, including raw event, simple event and complex event, and we describe two typical IoT monitoring instances for IoT complex event definition. In Section IV, we propose the universal complex event processing system architecture, and discuss the advantage characteristics of the system. In Section V, we introduce the performance evaluation of the system. Finally, we conclude the paper in Section VI.

II. RELATED WORK

In recent years, IoT monitoring system based on CEP has been applied in industrial production, environmental monitoring, water quality monitoring, public safety monitoring, telemedicine monitoring and many other fields.

Huang *et al.* [3] proposes a reactive model-based monitoring in RFID-enabled manufacturing, which applies CEP to RFID-assisted real-time production monitoring. Wong and Kerkez [1] presents a water quality monitoring application using web services based on real-time environment sensor data. Sun *et al.* [4] presents intelligent environmental monitoring system architecture based CEP engine to detect anomalies in real time, and demonstrates the system using a series of

real monitoring data from the geological carbon sequestration domain. Mongiello *et al*. [5] presents an IoT-based framework aiming at monitoring public building environmental parameters in order to support rescuers during emergencies such as a fire.

All the above studies basically use some kind of sensor equipment, such as RFID tags or sensors, to realize the monitoring in a certain field. The proposed scheme has limitations. So far, there is no general CEP system suitable for various IoT monitoring applications.

In recent years, previous researches focus on high performance technology of data flow processing of CEP, architecture of CEP based on edge computing, high performance of parallel processing and system scalability of CEP.

Flouris *et al*. [9] studies issues such as query optimization aspect to try to apply CEP techniques over big data, and expands on the synergies among predictive analytics and CEP. Akbar *et al*. [13] proposes a proactive architecture which exploits historical data using machine learning for prediction in conjunction with CEP, and evaluates it using a real-world use case of intelligent transportation system. da Silva *et al*. [14] introduces an approach for CEP query shipping to support distributed IoT environments to achieve the high efficiency for the emerging IoT paradigm.

Assuncao *et al*. [15] presents that more recently architecture has been proposed to use edge computing for data stream processing. Vrbaski *et al*. [16] proposes a micro-service based notification methodology that uses complex event recognition to handle the IoT system uncertainty.

Dhillon *et al*. [6] introduces an IoT-based CEP approach that uses a mobile device on the edge and a remote IoT Hospital Server (IHS) deployed on the cloud. In this architecture, complex event detection is performed on the edge to avoid queuing delays, and reduce the cost for data transfer between the mobile device and the hospital server. It is a seminal work by introducing CEP into edge computing for IoT. But there are some limitations of the CEP approach. 1) The CEP runs on mobile device (e.g. mobile phone) and it is small to be used for individuals. CEP shares mobile computing resources with other applications on the mobile phone, and the performance is affected. The storage space of mobile phone is small, and it is difficult to save more historical data. So it needs to cooperate with remote IHS to support the analysis of long-term historical data. The CEP is not suitable for large-scale telemedicine monitoring. 2) There are few types of monitoring sensors supported for the CEP. The method of how to support new sensors is not described in detail, so the scalability of the CEP is uncertain.

Mayer*et al*. [17] proposes a pattern-sensitive partitioning model for data streams to achieve a high degree of parallelism in detecting event patterns, which formerly could only consistently be detected in a sequential manner or at a low parallelization degree. Xiao *et al*. [18] propose a new parallelization model and parallel processing strategies for distributed complex event processing systems to enable the processing load for the overlap to be shared by the downstream machines to avoid wrong events decision.

These studies have contributed to improve CEP performance in different ways. But the event model of them is basically defined as an abstract way like e(s,t), without hierarchical or specific description mechanism, which cannot meet the complex event description of comprehensive monitoring business. Therefore, focusing on the event model, the event refactoring and computing performance remain the unsolved technical issues which are CEP system currently addressing.

III. IOT EVENT MODEL

IoT perceived event can be divided into raw event, simple event and complex event according to the granularity.

Raw event is a sensing data read event, which means that a sensing device detects a certain data at a certain time. Even a small IoT monitoring system will generate a large number of fragmentary, repeated and redundant raw events in a short time. For example, a RFID tag is repeatedly read in a certain period of time, but the application only cares about whether the RFID tag exists in a certain period of time. Therefore, raw events should be filtered and combined to instantly distil them into meaningful simple events that describe the important status of a single device or a group of devices.

Simple event only reflect simple fact, and the application system is more concerned with the space and time correlation between a group of simple events, namely complex event. Through the analysis of the relationship between simple events, such as membership, time relationship, causality, inclusion relationship, etc., and the data processing of filtering, aggregation, etc., the complex events are finally generated by simple events. CEP is the primary way to recognize whether simple events meet a business rule.

The process of IoT perceived event processing is shown in Figure 1.

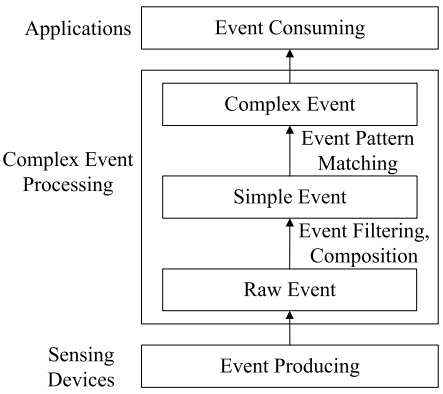


FIGURE 1. IoT perceived event processing process.

Figure 1 shows the entire process from event producing to event consuming. Raw events are generated by the sensor devices. Raw events are filtering and composing to generate simple events. Complex events are generated by event pattern matching from simple events. Complex events are sent to

applications for event consuming. The complex event processing is located between sensing devices and applications.

Both simple events and complex events are inferred from small-grained events. The difference between them is: the inference rules of simple events are universal, the granularity of simple events is moderate, and the reusability of simple events is high, while the inference rules of complex events are customized and determined by the actual business requirements.

A. RAW EVENT

Raw event is every piece of data detected and transferred by IoT sensor, also namely atomic (non-decomposable) event. Atomic event e is expressed as follows:

$$e = (\text{ID}, \text{DeviceID}, \text{DeviceType}, \text{Location}, \text{StartTime}, \text{EndTime}, \text{KeyValueList}) \quad (1)$$

In (1), ID is the unique identifier of an atomic event which is a sequence number.

DeviceID is the unique ID of the sensing device.

DeviceType stands for the type of sensing device.

Location means the location of the sensing device.

KeyValue List is a list of key-value pairs of <Key,Value>, representing multiple attributes and their values. It is extensible. For example, <Temperature,TemValue> represents the temperature value of temperature sensor, <Humidity,HumValue> represents the humidity value of humidity sensor, <Tag,TagValue> represents the tag value scanned by RFID reader.

StartTime and EndTime are the start time and end time of an event. The point time event satisfy StartTime = EndTime.

A raw event, for example, ''A temperature and humidity sensor of 'DS000580' locates in 'building 815' detects that temperature is 20°C, humidity is 35% at 9:30 in Oct. 14, 2018'', can be represented as:

e = (''0001'',''DS000580'',''building 815'',2018-10-14 9:30,2018-10-14 9:30, <Temperature, 20>, <Humidity, 35%>).

If StartTime ≠ EndTime, the event is not a point time event, but a duration time event. The duration time event may not be the original event, which can be obtained from the analysis of the original events, like the stay event of RFID tag. The duration time event can also be original event, such as camera video data event that record video for a duration time, thus Starttime is the start time of video and Endtime is the end time of video.

For complex event, Starttime is the minimum start time of all atomic events constituting the complex event, and Endtime is the maximum end time of all atomic events constituting the complex event.

The raw event of the various devices are shown in Table 1.

The different item in the raw event of various devices is the KeyValue List. The different perceived data can be uploaded through <Key,Value>.

TABLE 1. Raw event of devices.

Item	RFID	Sensor	Camera
ID	√	√	√
DeviceID	√	√	√
DeviceType	√	√	√
Location	√	√	√
StartTime	√	√	√
EndTime	√	√	√
KeyValue List	<Tag,TagValue>	<Temperature,TemValue> <Humidity,HumValue> <Angle, AngleValue> <Speed,SpeedValue> etc.	<Photo, PhotoValue> <Video, VideoValue>

For the convenience of description, the Raw Event can be simply denoted as RAE(d,l,t) to represent the instantaneous time and space state of a specific device, where d refers to the sensing device, l refers to the location space of the sensing device, and t refers to the time when the event occurs.

B. SIMPLE EVENT

A Simple Event is extracted from the Raw Event. The processing is listed as follows:

1) Filtering irrelevant events. For example, the raw sensing events not interested by applications can be ignored.

2) Filtering repeated event. Spatial repeated events generated by multiple sensing devices covering the same area can be filtered. For example, an event OE(d,l,t) describing the real-time spatial state of the sensing device d in the monitoring area l at time t. Most of the OE may be useless information for some moveable device such as RFID tag.

3) Filtering to get important events. For example, for an RFID tag, if it stays in a certain area for a long time, there will be a set of RAE(d,l,t), but the application usually only cares about the two moments when the tag enters to the area and leaves from the area, so the entry event AE(d,l,t) and exit event DE(d,l,t) can be filtered to be the important event from a set of RAE(d,l,t).

AE(d,l,t) describes that the sensing device d appears in the monitoring area l at time t. DE(d,l,t) describes that the device d disappears in the monitoring area l at time t.

SE(d,l,ts,te) describes device d stays in the monitoring area l from ts time to te time. SE(d,l,ts,te) is detected by matching the adjacent AE(d,l,t) and DE(d,l,t). SE(d,l,ts,te) is frequently used, and time reasoning with CEP wastes unnecessary computing cost, so SE(d,l,ts,te) is classified as a simple event.

As the same, the reappear event RPE(d,l,ts,te) describes device d reappears in regional l, ts and te refers to the moment of disappear time and reappear time. RPE(d,l,ts,te) is detected by matching the adjacent DE(d,l,t) and AE(d,l,t).

OE(d,l,t), AE(d,l,t) and DE(d,l,t) are Point Time events.

SE(d,l,ts,te) and RPE(d,l,ts,te) are Duration Time events.

4) Collection events are obtained by combination. Many applications focus on the number or collection of a certain

type of sensing data events in a specific region at a specific moment, while ignoring each individual data event. For example, the number of RFID tags that arrive at a certain area at a certain time, and the collection of RFID tags can be extracted by combination. The collection event $CE(E,l,t)_{(E.Devicetype=\text{“}RFIDTag\text{”})}$, where E refers to the device set, E.size refers to the number of devices in the set, l refers to the area, t refers to the event time, and subscript expression defines the attribute constraint, such as (E.Devicetype = ‘‘RFID Tag’’) defines the constraint condition: the device category is RFID Tag.

The above RFID events are used as an example to describe the possible events in the whole process of device movement. These simple events can represent a wider range of sensor events while the event properties are extended and the data properties are added in the KeyValue List as shown in Table 1. Therefore, the simple event definition is generic to most sensing devices.

The simple event processing algorithm is as follows:

Algorithm 1 Simple Event Processing

Input Raw event set: RAE(d,l,t)
Output Simple event, OE(d,l,t), AE(d,l,t), DE(d,l,t), SE(d,l,ts,te),RPE(d,l,ts,te), CE(E,l,t)

```
if d is unmovable device then //  sensor, camera, etc.
    OE(d,l,t) ←RAE(d,l,t);
    output OE(d,l,t);
    exit;
end if
// RFID tag, mobile device, etc.
if RAE(d,l,t) is a new AE(d,l,t) then // new AE
    output AE(d,l,t);
    match adjacent DE(d,l,t),AE(d,l,t);
     if math success then
        output RPE(d,l,ts,te);
     end if
     CE(E,l,t) ←d; // add the new device into CE
 end if
 if RAE(d,l,t) is a new DE(d,l,t) then // new DE
    output DE(d,l,t);
    match adjacent AE(d,l,t),DE(d,l,t);
    if math success then
        output SE(d,l,ts,te);
    end if
    CE(E,l,t)-d; //delete the disappeared device from CE
 end if
```

In Algorithm 1, the repeated RAE that is repeated AE or DE can be filtered not to be sent to CEP, which can lighten unnecessary burdens of CEP.

C. COMPLEX EVENT

Complex event is related to a group of events that reflect a particular rule. The events in the group are called sub-event. Sub-event can be simple event or complex event. Rules are defined by event operators.

E is the event set as follows:

$$E = \{e|e \text{ is an event defined as(1).}\} \tag{2}$$

So, the complex event C is:

$$C = f(E_1, E_2 \ldots, E_n), (E_n \in E, n > 0) \tag{3}$$

In (3), f is the event constructor function. The f function contains the various event operators. A complex event is usually defined by applying event constructors to constituent events, which are either simple events or other complex events.

The event operators are proposed as follows:

1) $E1 \wedge E2$, defines E1 and E2 occurred without time constraint. $(E1 \wedge E2)_T$ denotes that both E1 and E2 occurred in T time.

2) $E1 \vee E2$, defines E1 and E2 happen either.

3) $\neg$ E, means E is not happened. $\neg$ E is often used in conjunction with time window operator like $(\neg E)_T$.

4) $E_{C,}$ defines attribute constraints. C is composed of relational and logical operation expressions. The operators include: $\wedge$, $\vee$, $\neg$, etc. For example, $E_{(e.DeviceID\text{“}DS000580\text{”})\wedge(e.DeviceType=\text{“}TemperatureSensor\text{”})\wedge(e.Location=\text{”}L1\text{”})\wedge(T>30s)\wedge(e.TemValue>80°C)}$ means the complex event of that the temperature sensor ‘‘DS000580’’ in the area of ‘‘L1’’ detects temperature over 80 ° C, and the duration is more than 30s.

5) Aggregation, define the statistics information of a group of event instances or the attribute values of instances, including count(E), sum(E,key), max(E,key), min(E,key), avg(E,key),etc. It respectively count the number of instances, sum, maximum, minimum and average value of the key attribute of each instance.

6) $E_{T,}$ defines the effective time range of complex event E, where T represents the time window duration in units of s (seconds), m (minutes), h (hours), etc.

7) Time constraint operators, define the time relationship of the events, which are combined with the time length semantics based on the operators proposed by Allen *et al*. [19]–[21]. The time operators of events are shown in Table 2. (E.ts refers to E.StartTime, E.te refers to E.EndTime)

D. COMPLEX EVENT DEFINITION INSTANCE

We define the complex event using the event modeling language for IoT real-time monitoring applications.

Instance 1: IoT monitoring system for fruit transportation and storage. The monitoring application scenario is shown in Figure 2.

Assuming that the best storage condition of fruit is in temperature 5-10° C, humidity 85%-90%. Fruit supermarket are supplied by van. Two temperature and humidity sensor nodes DS000560 and DS000570 are respectively deployed in the fruit supermarket and the supplier van to detect temperature and humidity. Two RFID Reader A and B are respectively placed at the entrance of the supermarket and supplier’s van

TABLE 2. Time constraint operator.

Time operator	Grammar	Semantics description
before	(E1 before E2)$_{[T1,T2]}$	E1 E2 T1<= E2.ts-E1.te<=T2
after	(E1 after E2)$_{[T1,T2]}$	E2 E1 T1<=E1.ts-E2.te<=T2
meets	(E1 meets E2)$_{[T]}$	E1 E2 |E2.ts-E1.te|<=T
met-by	(E1 met-by E2)$_{[T]}$	E2 E1 |E1.ts-E2.te|<=T
includes	(E1 includes E2)$_{[T1,T2,T3,T4]}$	E2 E1 (T1<=E2.ts-E1.ts<=T2)&& (T3<=E1.te-E2.te<=T4)
during	(E1 during E2)$_{[T1,T2,T3,T4]}$	E1 E2 (T1<=E1.ts-E2.ts<=T2)&& (T3<=E2.te-E1.te<=T4)
equals	(E1 equals E2)$_{[T1,T2]}$	E1 E2 |E1.ts-E2.ts|<=T1)&& |E1.te-E2.te|<=T2)
overlaps	(E1 overlaps E2)$_{[T1,T2]}$	E1 E2 T1<=E1.te-E2.ts<=T2
overlapped-by	(E1 overlapped-by E2)$_{[T1,T2]}$	E2 E1 T1<=E2.te-E1.ts<=T2
starts	(E1 starts E2)$_{[T]}$	E1 E2 (|E1.ts-E2.ts|<=T)&&(E1.te<=E2.te)
started-by	(E1 started-by E2)$_{[T]}$	E1 E2 (|E2.ts-E1.ts|<=T)&&(E2.te<=E1.te)
finishes	(E1 finishes E2)$_{[T]}$	E1 E2 (|E1.te-E2.te|<=T)&&(E1.ts<=E2.ts)
finished-by	(E1 finished-by E2)$_{[T]}$	E1 E2 (|E2.te-E1.te|<=T)&&(E2.ts<=E1.ts)

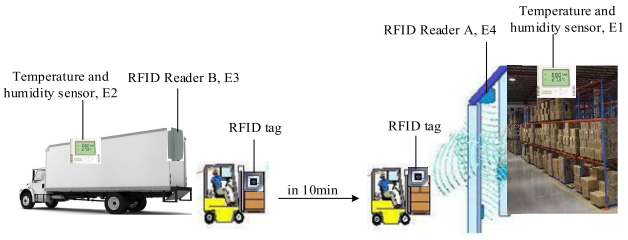


FIGURE 2. The monitoring application scenario for Instance 1.

door. Each box of fruit in the supplier's van is tagged with an RFID tag with a unique ID.

The requirements to transport the fruit successfully from the van to the supermarket are as follows: (1) Fruit supermarket is in temperature 5-10° C, and humidity 85%-90%. (2) Van is in temperature 5-10° C, and humidity 85%-90%. (3) It should take less than 10 minutes for the fruit to be transported from the van to the supermarket. If the above conditions are met, the fruit can be carried into the supermarket, otherwise alarm processing is performed.

The complex event definition for Instance 1 is in Figure 3.

(E3(e.DeviceID= "RFID Reader B")∧(e.DeviceType= "RFID Reader")∧(e.Location= "Van Door")
before
E4(e.DeviceID= "RFID Reader A")∧(e.DeviceType= "RFID Reader")∧(e.Location= "Supermarket Door")∧(e.TagValue=E3.TagValue))[0,10m]∧
(E1(e.DeviceID= "DS000560")∧(e.DeviceType= "TemperatureSensor")∧(e.Location= "Supermarket")∧(e.TemValue<=10℃)∧(e.TemValue>=5℃)∧(e.HumValue<=90%)∧(e.HumValue>=85%)
before
E3(e.DeviceID= "RFID Reader B")∧(e.DeviceType= "RFID Reader")∧(e.Location= "Van Door"))[0,10m]∧
(E2(e.DeviceID= "DS000570")∧(e.DeviceType= "TemperatureSensor")∧(e.Location= "Van")∧(e.TemValue<=10℃)∧(e.TemValue>=5℃)∧(e.HumValue<=90%)∧(e.HumValue>=85%)
before
E3(e.DeviceID= "RFID Reader B")∧e.DeviceType= "RFID Reader")∧(e.Location= "Van Door"))[0,10m]

FIGURE 3. Complex event definition for Instance 1.

Instance 2: IoT monitoring system for city road manhole cover status.

The inclination sensor and speed sensor are deployed at the bottom of the road cover to perceive the change in the position of the cover. If there are any abnormal events such as inclination, turnover or movement, an alarm will be issued. Such as when the tilt angle is more than 30°, the tilt event will occur. If the speed exceeds 0.5m/s, a cover movement event will occur.

The complex event E1∨E2 with E1 of manhole cover tilt event and E2 of manhole cover movement event are defined in Figure 4.

E1(e.DeviceType= "TiltAngleSensor")∧(e.Location= "ManholeCover001")∧(e.TiltAngleValue>=30°)
∨
E2(e.DeviceType= "SpeedSensor")∧(e.Location= "ManholeCover001")∧(e.SpeedValue>=0.5m/s)

FIGURE 4. Complex event definition for Instance 2.

IV. COMPLEX EVENT PROCESSING SYSTEM

A. SYSTEM OVERVIEW

The architecture of IoT complex event processing system is shown in Figure 5.

The CEP system of IoT deployed on edge is located between the sensing devices in terminal and the upper application system in the cloud, which forms the high-efficiency computing model of "terminal - edge computing - cloud computing".

The CEP system is deployed on the network edge nearly to the place where the raw data generated. The amounts of raw data are processed by CEP to generate complex event, and then transfer complex event to the monitoring applications in the cloud. Due to the complex event data quantity is far less than the original raw data, thus the amount of data transferred to the cloud can greatly reduce. Reducing the amount of data transmitted through the network can effectively improve the real-time processing performance of the whole system.

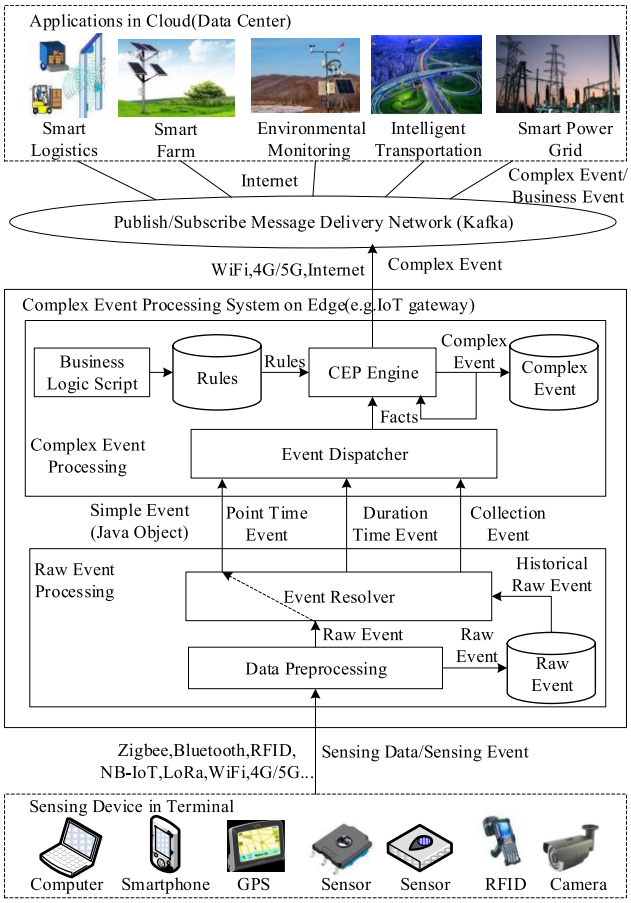


FIGURE 5. Architecture of IoT complex event processing system.

CEP on edge can process the raw data in time when the raw data generated and transferred to edge. The real-time performance of raw data processing on edge is better than transfer raw data to the cloud to process.

The simple event is introduced into the CEP model. The simple event processing such as filtering out repeated events, combining to get set events, etc. are carried out before the CEP engine can reduce the processing burden of the CEP engine and effectively improve the overall system processing performance.

The CEP system on edge can run on the IoT gateway or on a separate computer in the same LAN with the IoT gateway.

The sensing data/event of raw data can be transferred to the CEP system through various wireless communication network, such as Zigbee, Bluetooth, RFID, NB-IoT, Lora, WiFi, 4G/5G/LTE, etc. The CEP system transfers the complex events to the publish/subscribe message delivery network (e.g. Kafka) through WiFi, 4G/5G/LTE mobile communication network or Internet. The IoT monitoring applications run in the cloud (e.g. data center). The publish/subscribe system provides publish and subscribe of messages for CEP and applications to complete the delivery of complex events through Internet.

The CEP system consists of two modules: 1) Raw Event Processing module. Firstly, the sensing data uploaded by the sensing devices are preprocessed to generate the raw event and stored in the database. Then, the simple events are generated by refining from raw events. 2) Complex Event Processing module. It reasons complex events according to simple events.

The advantages of CEP system deployed on the network edge are as follows:

1) The storage and processing of original sensing data run on the CEP system of IoT, which is located at the edge of the network, can reduce the communication overhead of network transmission, and improve the monitoring service response speed and improve the performance of the whole IoT real-time monitoring system.

2) The original data carries a variety of sensitive information, such as time and location. After processing of CEP, the business data or complex event not containing sensitive information will be sent to the application in the form of complex events, which is conducive to the realization of privacy protection.

For example, for smart city real-time monitoring service, the local area data analysis can be carried out in the CEP system located on the edge, while the global data analysis task can be carried out in the cloud. The edge computing can realize rapid analysis and response of local data, and reduce unnecessary data transmission to the cloud. Also, it is conducive to the privacy protection of users.

B. RAW EVENT PROCESSING MODULE

Raw Event Processing module includes Data Preprocessing module and Event Resolver module. Data Preprocessing module collects data from Sensing Devices data source according to interval time, filters the repeated and irrelevant data, forms Raw Event and writes it to Raw Event database.

If the Raw Event is Point Time Event, Event Resolver will send it immediately to the Complex Event Processing module to ensure the real-time processing performance. For the Duration Time Event, Event Resolver will process the correlated Raw Events from the raw event database to generate the Duration Time Event to send to the Complex Event Processing module.

Event Resolver reads Raw Events from the database in real time, filters and combines them according Algorithm 1 to obtain Simple Events including: 1) Point time event, such as entry event AE, leaving event DE and existing event OE. 2) Duration time event. According to match AE and DE, stop event SE and reappear event RPE are obtained. 3) Collection event CE are obtained according to statistics of AE and DE.

The Event Resolver outputs all simple events as Java objects to the Complex Event Processing module. The simple event class definition of IoT is shown in Figure 6.

C. COMPLEX EVENT PROCESSING MODULE

1) CEP ENGINE

The Complex Event Processing module is implemented based on CEP rule engine. The original event stream facts

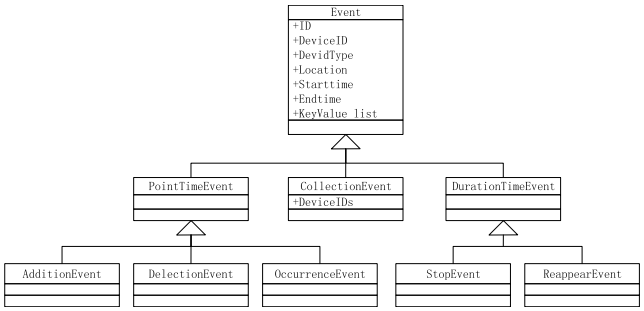


FIGURE 6. Class of IoT simple event.

performs pattern matching with rules in the rule library. If the matching is successful, complex events will be inferred and generated to send to the upper application. The generated complex event can be sent to CEP engine as a fact if needed.

The definition of a rule is the basis for complex event discovery. Different application systems define different rule libraries and get different high-level events.

The rule base is pre-defined according to the application requirements, and a large number of rules are stored in the rule base. Use rules to describe the judgment of complex events. Such as ''If a RFID tag entered a region, but the video have more than two faces recognition, the abnormal event should occur and the system should alarm'', ''If temperature C_2-C_1 $>10°$ C and C_2 occurred after C_1 in 10 seconds, the temperature rises too fast, the system should alarm''. It can be seen that the rule is composed of two parts. The first part is the condition, which represents the event pattern and describes the relationship among original events. The second part is the inference result, which is the specific processing of complex event occurs.

2) WHY SELECT DROOLS

The Complex Event Processing module is implemented on JBoss Drools [21] which is a popular CEP engine. Drools has some advantages compared with another popular CEP Esper as follows:

(1) Esper receives the event stream and regards each type of event as the event stream. Drools can receive event stream and store different types of events, and can record the source of the event which facilitates the causal tracing of the event.

(2) Esper does not provide application time model, but only provides engine time. Drools provides application time model, but it requires events to arrive at the engine in the correct order.

(3) Drools is better than Esper in supporting temporal relation. For example, Esper supports point time events, while Drools can define the duration of events to support duration time events. Drools support richer time operations to define complex event with complex time-dependent semantics.

For IoT monitoring scenario, it is necessary to record events including point time events and duration time events.

TABLE 3. Drools logic script (PART).

Complex event definition	Business logic script of Drools
$E1 \wedge E2$	E1($id:id,l==”l1”) and E2(id==$id,l==”l2”)
$E1 \vee E2$	E1($id:id,l==”l1”) or E2(id==$id,l==”l2”)
$\neg E$	not (E(l==”l1”))
E_C	E(type==”TemperatureSensor”,l==”l3”)
$(E2 \text{ after } E1)_{[T1,T2]}$	$e1:E1($id:id,l==”l1”) $e2:E2(id==$id,l==”l2”, this after [0,30s] $e1
$(E1 \text{ during } E2)_{[T1,T2,T3,T4]}$	$e1:E1($id:id,l==”l1”) $e2:E2(id==$id,l==”l2”, this during [1s,3s,4s,6s] $e1
E_T	E(l==”l1”) over window:time(2m)
count(E)	from accumulate(E(l==”l1”),count)
max(E,key)	from accumulate(E(l==”l1”),$k:key),max($k))

The time processing requirements of events are relatively high. Drools time operators can meet the requirements. Therefore Drools CEP is selected in this paper.

3) DROOLS RULE LOGIC SCRIPT

Drools supports for programmatic customization of complex event logic. Each complex event corresponds to a logic script. The logic script is parsed at runtime to process the receiving simple events.

In Figure 5, Complex Event Processing module includes Event Dispatcher module, CEP engine and Business Logic Script module.

The Event Dispatcher module gets the received simple Event Java objects as Fact objects, and put them into the event processing channel corresponding to CEP Engine. Each event processing channel represents a set of internally related and externally independent complex event logic, which is implemented by a logical script. CEP Engine conducts pattern matching between Facts and rules, calculates and deduces complex events, and then stores complex events into the complex event database, and sends the complex events to the upper application system through the message delivery system like Kafka.

The key to realize CEP is to realize the mapping between complex event semantics description and the rule logical script. The complex event modeling language proposed in this paper can be fully mapped to Drools logical scripts. Table 3 lists the logical script fragments of some event operators.

The Drools complex event logic script for Instance 1 is defined in Figure 7. The rule ''rule_fruit_transportation'' includes *when* section and *then* section, the *when* section is the complex event pattern that is the complex event logic script.

The Drools complex event logic script for Instance 2 is defined in Figure 8. The *when* section of the rule ''rule_road_manhole_cover_1'' is the complex event pattern that is the complex event logic script.

```
rule "rule_fruit_transportation_1"
    when
       //conditions
       $e3:E3(DeviceID=="RFID Reader B",DeviceType=="RFID Reader",
        Location=="Van Door") from entry-point "P1"
       $e4:E4(DeviceID=="RFID Reader A",DeviceType=="RFID Reader",
        Location=="Supermarket Door",TagValue==$e3.getTagValue(),
        this after[0s,10m] $e3) from entry-point "P1"
       $e1:E1(DeviceID=="DS000560",DeviceType=="TemperatureSensor",
        Location=="Supermarket",
        List["TemValue"]<=10,List["TemValue"]>=5,
        List["HumValue"]<=90,List["HumValue"]>=85,
        this before[0,10m] $e3) from entry-point "P1"
       $e2:E2(DeviceID=="DS000570",DeviceType=="TemperatureSensor",
        location=="Van",
        List["TemValue"]<=10,List["TemValue"]>=5,
        List["HumValue"]<=90,List["HumValue"]>=85,
        this before[0,10m] $e3) from entry-point "P1"
    then
       //actions
       count.addAndGet(1);
end
```

FIGURE 7. Drools logic script of complex event for Instance 1.

```
rule "rule_road_manhole_cover_1"
    when
        //conditions
        $e1:E1(DeviceType=="TiltAngleSensor",Location=="ManholeCover001",
         List["TiltAngleValue"]>=30) from entry-point "P2"
        or
        $e2:E2(DeviceType=="SpeedSensor",Location=="ManholeCover001",
         List["SpeedValue"]>=0.5) from entry-point "P2"
    then
        //actions
        count.addAndGet(1);
end
```

FIGURE 8. Drools logic script of complex event for Instance 2.

D. DISCUSS OF SYSTEM CHARACTERISTICS

The system has some advantage characteristics as follows:

1) The event model is formalized and readable. The formalized modeling language to define the complex event has better readability. It defines the complex event in the programmatic way supporting space and time semantics of IoT application, and can be completely mapped to the Drools rules logic script definition, so as to realize efficient definition and adapt to the complex and changeable business events in the IoT monitoring system.

2) The event model is universal which is independent of CEP engine. The complex event modeling language does not rely on specific implementation techniques, and can map to complex event definition scripts of other CEP engines to realize complex event processing.

3) The CEP system has high scalability. The division of event processing channel in the CEP module is distributed according to the characteristics of the monitoring system business. The monitor tasks can be assigned to different event processing channel to run the corresponding complex event. It promotes the scalability of the large-scale complex event processing through distributed software and hardware technology.

V. EXPERIMENTAL EVALUATION

The performance of CEP is the key to IoT real-time monitoring. The experimental setup of the CEP system is shown in Figure 9.

In Figure 9, the Sensing Data Generator is a simulator program to produce the sensing data/event according to the

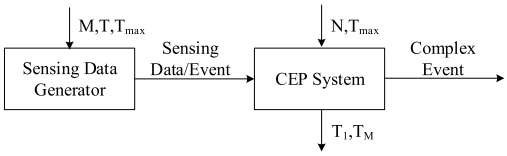


FIGURE 9. Experimental setup.

parameter M, T, and T_{max}. T is the time interval. T_{max} is the maximum time window. M is the number of sensing data/event generated in each T interval to send to CEP system.

The CEP system is the real system. It receives the sensing data and processes to generate complex event. There are some parameters of the CEP system. The input parameter N is the complex event definition number, and T_{max} is the maximum time window. The output parameter T_1 and T_M are used to compute the processing time T_c. T_1 is the system time when the first event received in a T interval. T_M is system time at the end of processing the last event M in a T interval.

Processing time T_c is the system running time to handle the M events in a T interval. The value of T_c is the average time of 5 consecutive runs under stable operation. The computing method of T_c is Formula (4) as follows. In (4), i is the number of CEP system runs.

$$T_c = \frac{1}{5}\sum_{i=1}^{5}(T_M - T_1) \tag{4}$$

The experiments are carried out on an Intel Core i5-6300HQ CPU 2.30GHz PC, with 8GB of RAM running on Windows 10.

Since complex event of time-dependent semantics need to deal with all historical events within the effective period, which has a great impact on performance, we divide the complex events into 2 kinds of events to evaluate the performance:

1) Complex event with time-independent semantics referring to complex event definition without time operators, e.g. the complex event "rule_road_manhole_cover_1" of Instance 2 in Figure 8.

2) Complex event with time-dependent semantics referring to the definition of complex event containing time operators including time constraints and time window, e.g. the complex event "rule_fruit_transportation_1" of Instance 1 in Figure 7.

A. PERFORMANCE COMPARISON OF TIME-INDEPENDENT SEMANTICS CEP and TIME-DEPENDENT SEMANTICS CEP

Experiment 1: performance comparison of time-independent semantics CEP and time-dependent semantics CEP.

1) THE PROCESSING TIME PERFORMANCE

The number of complex event definitions N is 1, 2, 5, 10, 20, the input simple event flow M is increased from 1×10^4 to 10×10^4, the processing time T_c are shown in Figure 10.

Only time-independent semantics events such as "rule_road_manhole_cover_1" are defined in Figure 10(a)

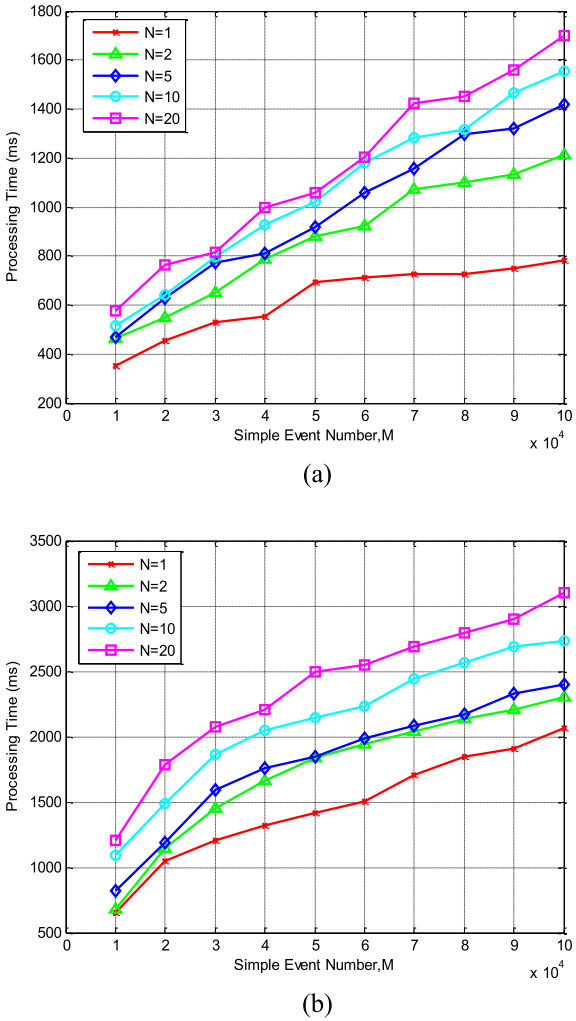


FIGURE 10. Processing time performance comparison. (a) Performance of time-independent semantics complex event processing. (b) Performance of time-dependent semantics complex event processing.

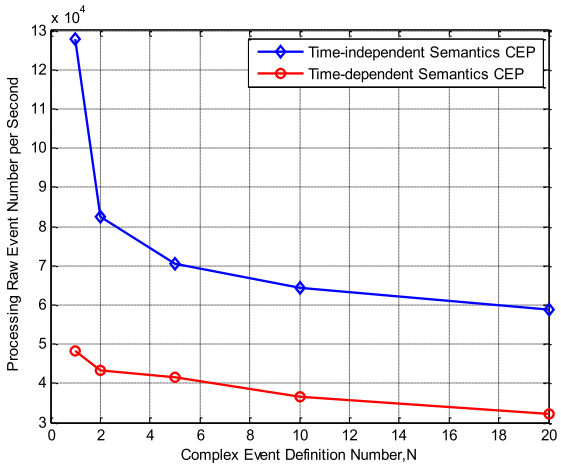


FIGURE 11. Throughput performance comparison.

experiment, while only time-dependent semantics events such as "rule_fruit_transportation_1" are defined in Figure 10(b) experiment.

The experimental results show that:

(1) The processing time T_c increases as the number of complex event definitions N increases for all of the two kinds of complex events.

(2) For time-independent semantics complex event processing, the number of complex event definitions N has little effect on the processing time T_c. For time-dependent semantics complex event processing, the number of definitions of complex events N has a greater impact on the processing time T_c than the time-independent semantics complex event processing.

In Figure 10(a), when $M = 10 \times 10^4$, the processing time T_c is about 1553ms when $N = 10$, and T_c is about 1700ms when $N = 20$. Though the number of complex event definitions N is doubled, the processing time T_c increases only 147ms, a 9.5% increase.

In Figure 10(b), when $M = 10 \times 10^4$, the processing time T_c is about 2731ms when $N = 10$, and T_c is about 3100ms when $N = 20$. The number of complex event definitions N is doubled, and the processing time T_c increases by 369ms, a 13.5% increase. T_c increases by 4% compared to 9.5% for time-independent semantics complex event processing.

(3) The processing time T_c of CEP without time operators is far greater than that with time operators, and the larger the N value is, the more obvious the performance is. The larger N is, the more complex event processing overhead is, and the larger T_c is.

In Figure 10(a), the processing time T_c of time-independent semantics CEP is about 1700ms when $N = 20$, $M = 10 \times 10^4$. In Figure 10(b), T_c of time-dependent semantics CEP is about 3100ms when $N = 20$, $M = 10 \times 10^4$, which is about 1.82 times more than 1700ms for time-independent semantics CEP.

2) THE THROUGHPUT PERFORMANCE

The number of complex event definitions N is 1, 2, 5, 10, 20, the input simple event flow M is 10×10^4, the throughput of processing raw event number per second of the time-independent semantics CEP and time-dependent semantics CEP are shown in Figure 11.

The experimental results show that:

(1) The throughput decreases as the number of complex event definitions N increases for all of the two kinds of complex events.

For time-independent semantics CEP, when $N = 1$, the throughput is 1.27×10^4 raw events per second, and when $N = 20$, the throughput is 5.88×10^4 raw events per second, a 54% decrease. For time-dependent semantics CEP, when $N = 1$, the throughput is 4.83×10^4 raw events per second, and when $N = 20$, the throughput is 3.22×10^4 raw events per second, a 33% decrease. The throughput degradation of time-independent semantic CEP is more significant than time-dependent semantic CEP.

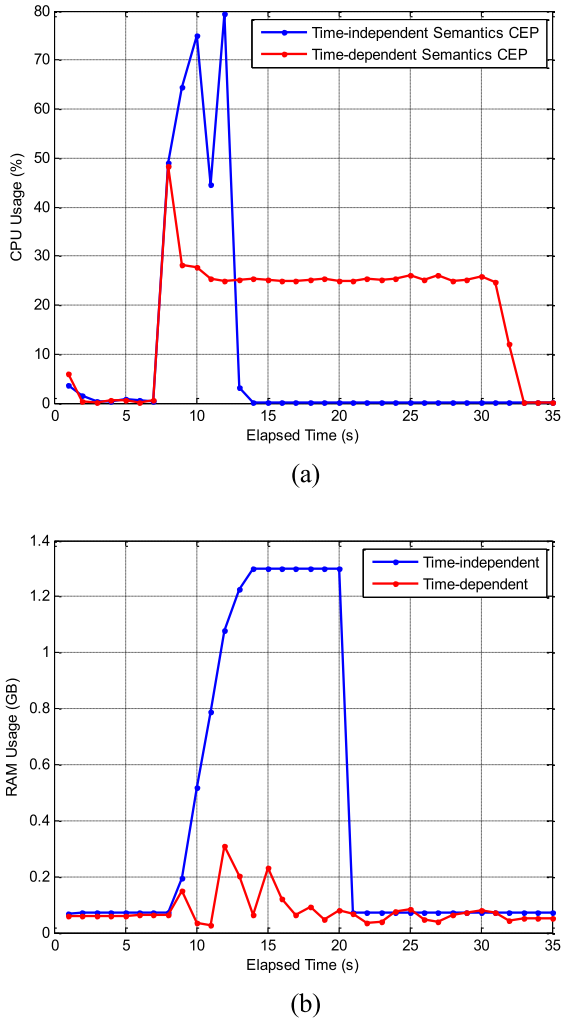


FIGURE 12. Computing resource usage. (a) CPU usage. (b) RAM usage.

(2) The throughput of time-independent semantics CEP is higher than time-dependent semantics CEP.

When N = 20, the throughput of time-independent semantics CEP is 5.88 × 10 ^{4} raw events per second, while the throughput of time-dependent semantics CEP is 3.22 × 10^4 raw events per second. The throughput of the former is 1.82 times than the latter which is consistent with the previous multiple of processing time T_c.

3) THE CPU AND RAM USAGE

We evaluate the computing resource usage of CEP. When N is 20, the input simple event flow M is 200 × 10 ^{4}, the CPU and RAM usage of the time-independent semantics CEP and time-dependent semantics CEP are shown in Figure 12.

The experimental results show that the computing resource of CPU and RAM usage of time-independent semantics CEP is higher than time-dependent semantics CEP. The former takes less processing time than the latter because the former has larger throughput than the latter.

For time-independent semantics CEP, the maximum CPU usge is 80%, and the average CPU usage is about 62%. For time-dependent semantics CEP, the maximum CPU usage is 48%, and the average CPU usage is about 26%.

For time-independent semantics CEP, the maximum RAM usage is 1.3GB, and the average RAM usage is about 1GB. For time-dependent semantics CEP, the maximum RAM usage is 0.3GB, and the average RAM usage is about 0.1GB.

4) THE PERFORMANCE METRICS SUMMARY

In summary, the performance metrics of the CEP system in the experimental environment are shown in Table 4.

TABLE 4. Performance metrics of the CEP system.

Performance Metric	Time-independent (Instance 2)	Time-dependent (Instance 1)
T_c (N=20,M=10×10^4)	1700ms	3100ms
Throughput (N=20, M=10×10^4)	5.88×10^4 events/second	3.22×10^4 events/second
CPU usage (N=20, M=200×10^4)	Maximum 80% Average 62%	Maximum 48% Average 26%
RAM usage (N=20, M=200×10^4)	Maximum 1.3GB Average 1GB	Maximum 0.3GB Average 0.1GB

If the required throughput of CEP is thousands level such as 1000-9000 events/second, or tens of thousands level but less than 3 × 10 ^{4} events/second, the hardware infrastructure of a PC of Intel Core i5 with 8GB RAM (our experiment environment hardware) is enough to support the CEP system running.

In practice, the choice of hardware of CEP system running depends on the requirements for throughput of CEP.

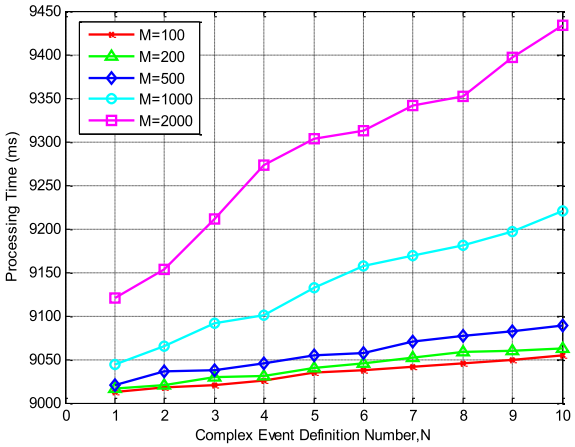
B. PERFORMANCE ANALYSIS OF TIME-DEPENDENT SEMANTICS CEP

Experiment 2: performance analysis of time-dependent semantics CEP.

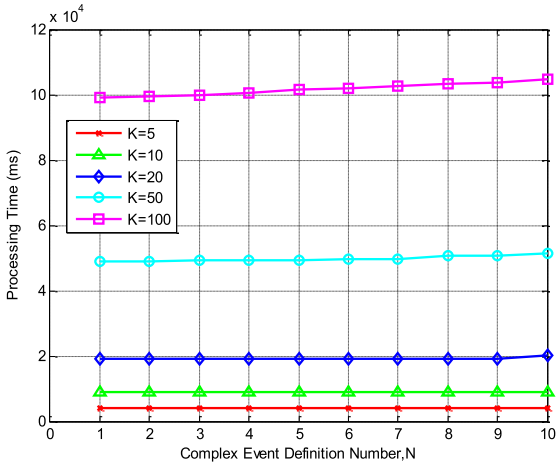
Time-dependent semantics need to consider historical events within the effective period, and the main parameter affecting performance is K = T_{max}/T, where T_{max} is the maximum time window, representing the maximum effective duration in the definition of complex events, which is determined by event failure time, time constraint duration and time window value. T is the sampling interval. K represents the number of event groups uploaded to complex event processing in the range of maximum effective time.

In Figure 13(a), K is fixed at 10, indicating the number of input event groups is 10. Let T_{max} = 10*s*, T = 1s, K = T_{max}/T = 10, the number of complex event definitions N increases from 1 to 10, and M takes 100, 200, 500, 1000, 2000 simple events in every 1 second, the processing time T_c are shown in Figure 10(a).

In Figure 13(b), the input simple event flow M is fixed. M is 500 in every 1 second, the number of complex event



(a)



(b)

FIGURE 13. Time-dependent semantics complex event processing performance. (a) Processing time when fixed K = 10. (b) Processing time when fixed M = 500 per 1s.

definitions N increases from 1 to 10, K is 5, 10, 20, 50, 100, the processing time T_c are shown in Figure 13(b).

The experimental results show that:

(1) The processing time T_c increases with the growth of M, N and K. T_c is proportional to M, N, and K.

Figure 13(a) shows that when K is fixed, the larger N is, the larger M is, and the faster T_c increases.

Figure 13(b) shows that when M is fixed, the performance time T_c is mainly affected by T_{max} and very little by N. The T_c value of each K value is basically unchanged, showing a horizontal trend with a small rise. For example, when K = 100, T = 1s, T_{max} = 100s, at this time, when M = 500, N = 1, T_c = 99.013s, and N = 10, T_c = 104.808s, with little change. It can be seen that T_c value is close to T_{max} value. With the increase of N, T_c increases very little, indicating that T_c is mainly affected by T_{max} and very little by N.

(2) To better define time-dependent semantics complex events, the parameters need to be carefully considered as follows: i) The time interval T should be set valid and reasonable. ii) The maximum time window T_{max} should be minimized as far as possible, so as to avoid affecting the event processing performance and reducing the efficiency of real-time monitoring.

For Instance 1, T_{max} should be set 10 minutes, because the fruit must arrive within 10 minutes from the van to the supermarket. The T collection interval can be set 1 minute. K = T_{max}/T = 10.

For Instance 2, complex events are time-independent semantics. There is no time operator in a complex event definition, and the processing time is independent of the T_{max} parameter.

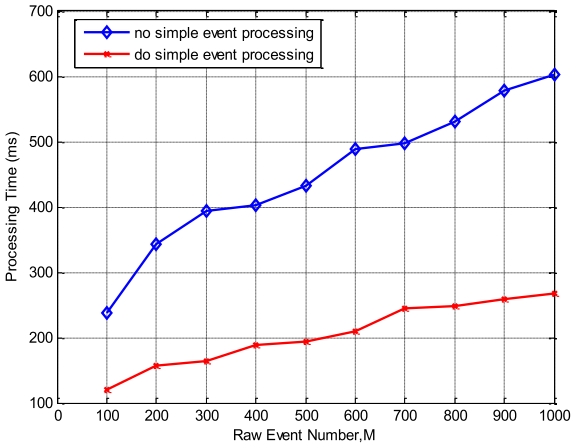


FIGURE 14. Performance improvement of do simple event processing before complex event processing of CEP engine.

C. PERFORMANCE COMPARISON OF HAVING SIMPLE EVENT PROCCESSING AND NO SIMPLE EVENT PROCESSING

Experiment 3: performance comparison of having simple event processing and no simple event processing.

The complex event processing with time-dependent semantics complex event costs much more time than time-independent semantics complex event. The simple event processing Algorithm 1 included in the system executing before complex event processing by CEP engine can reduce the computing cost and improve performance.

In Figure 14 experiment, the number of the time-dependent semantics complex event definitions N is 6, the input raw event flow M is from 100 to 1000. The number of repeated event of each raw event is 5. The processing time T_c of the two cases: 1) no simple processing, and 2) do simple event processing are shown in Figure 14.

The ''no simple event processing'' means that the function of Algorithm 1 simple event processing need to be implemented in CEP engine rules. So it need much more processing time. The ''do simple event processing'' means that the function of Algorithm 1 is implemented in simple event processing module running before the CEP engine which reduces the complexity and burden of CEP engine. So the processing time T_c of the system in case of ''do simple event

processing'' is smaller than T_c in case of ''no simple event processing''.

In Figure 14, when raw event number M is 100, the performance time T_c of ''no simple processing'' case is 238.2 ms, and T_c of ''do simple processing'' case is reduced to 120.4ms. When raw event number M is 1000, the performance time T_c of ''no simple processing'' case is 602.4 ms, and T_c of ''do simple processing'' case is reduced to 268.2ms. With M increases, the performance time T_c reduces even much more. The larger M is, the more significant performance improvement is.

The experimental result shows the system performance is obviously improved by having ''do simple event processing''. Therefore, the simple event processing is necessary to do before complex event processing for large amount of data real-time processing. It can effectively improve the overall performance of CEP system. The system architecture proposed in this paper is high efficiency.

VI. CONCLUSIONS

This paper proposes a universal complex event processing mechanism for Internet of things real-time monitoring. The CEP mechanism is suitable to any heterogeneous sensing devices and CEP engine. The proposed system architecture is based on the edge computing which deployed between the sensing devices in terminal and IoT applications in the cloud to process the data flow timely.

In this paper, a formalized complex event model is proposed to analyze IoT sensing events to establish hierarchical events as raw event, simple event and complex event, so as to reduce the complexity of event modeling. The model supports to define complex and changeable IoT monitoring business logic effectively. We demonstrate two typical IoT monitoring application case studies highlight the proposed event model.

Moreover, based on the complex event model, a complex event processing system architecture is proposed. The highly efficient CEP engine is adopted to construct the system. The complex event definitions are mapped to the CEP rule logic scripts, and the system extracts important status or abnormal information from a large amount of collected data in a timely manner.

Finally, the experimental evaluation demonstrates the performance of the CEP system. The performance metrics of the CEP system are evaluated, including processing time, throughput, CPU and RAM usage, etc. The configuration of CEP system to improve performance is suggested. The mechanism of IoT event model with do simple event processing before CEP engine improves the whole CEP system performance. The mechanism is efficiency to suffice the high requirements of IoT real-time data processing, and adapt to the complexity and changeability of monitoring business logic.

The future work will focus on the following: 1) Study the system performance and application scenarios of the CEP system running on edge smart devices, such as mobile phone and Raspberry Pi, etc. 2) Research on service collaboration of Internet of things application services based on CEP.

REFERENCES

[1] B. P. Wong and B. Kerkez, ''Real-time environmental sensor data: An application to water quality using Web services,'' *Environ. Model. Softw.*, vol. 84, pp. 505–517, Oct. 2016.

[2] C. Granell, D. Havlik, S. Schade, Z. Sabeur, C. Delaney, J. Pielorz, T. Usländer, P. Mazzetti, K. Schleidt, M. Kobernus, F. Havlik, N. R. Bodsberg, A. Berre, and J. L. Mon, ''Future Internet technologies for environmental applications,'' *Environ. Model. Softw.*, vol. 78, pp. 1–15, Apr. 2016.

[3] Y. Huang, B. C. Williams, and L. Zheng, ''Reactive, model-based monitoring in RFID-enabled manufacturing,'' *Comput. Ind.*, vol. 62, nos. 8–9, pp. 811–819, Oct./Dec. 2011.

[4] A. Y. Sun, Z. Zhong, H. Jeong, and Q. Yang, ''Building complex event processing capability for intelligent environmental monitoring,'' *Environ. Model. Softw.*, vol. 116, pp. 1–6, Jun. 2019.

[5] M. Mongiello, F. Nocera, A. Parchitelli, L. Riccardi, L. Avena, L. Patrono, I. Sergi, and P. Rametta, ''A microservices-based IoT monitoring system to improve the safety in public building,'' in *Proc. 3rd Int. Conf. Smart Sustain. Technol. (SpliTech)*, Jun. 2018, pp. 1–6.

[6] A. Dhillon, S. Majumdar, M. St-Hilaire, and A. El-Haraki, ''MCEP: A mobile device based complex event processing system for remote healthcare,'' in *Proc. IEEE Int. Conf. Internet Things (ICIOT)*, Jul./Aug. 2018, pp. 203–210.

[7] A. Gandomi and M. Haider, ''Beyond the hype: Big data concepts, methods, and analytics,'' *Int. J. Inf. Manage.*, vol. 35, no. 2, pp. 137–144, 2015.

[8] M. James, M. Chui, P. Bisson, J. Woetzel, R. Dobbs, J. Bughin, and D. Aharon, ''The Internet of Things: Mapping the value beyond the hype,'' McKinsey Global Institute, New York, NY, USA, Tech. Rep. 1, Jun. 2015.

[9] I. Flouris, N. Giatrakos, A. Deligiannakis, M. N. Garofalakis, M. Kamp, and M. Mock, ''Issues in complex event processing: Status and prospects in the Big Data era,'' *J. Syst. Softw.*, vol. 127, pp. 217–236, May 2017.

[10] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, ''Edge computing: Vision and challenges,'' *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[11] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, ''A survey on the edge computing for the Internet of Things,'' *IEEE Access*, vol. 6, pp. 6900–6919, 2018.

[12] L. Lan, R. Shi, B. Wang, and L. Zhang, ''An IoT unified access platform for heterogeneity sensing devices based on edge computing,'' *IEEE Access*, vol. 7, pp. 44199–44211, 2019.

[13] A. Akbar, A. Khan, F. Carrez, and K. Moessner, ''Predictive analytics for complex IoT data streams,'' *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1571–1582, Oct. 2017.

[14] A. C. F. da Silva, P. Hirmer, R. K. Peres, and B. Mitschang, ''An approach for CEP query shipping to support distributed IoT environments,'' in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PerCom Workshops)*, Mar. 2018, pp. 247–252.

[15] M. D. da Assunção, A. da S. Veith, and R. Buyya, ''Distributed data stream processing and edge computing: A survey on resource elasticity and future directions,'' *J. Netw. Comput. Appl.*, vol. 103, pp. 1–17, Feb. 2018.

[16] M. Vrbaski, M. Bolic, and S. Majumdar, ''Complex event recognition notification methodology for uncertain IoT systems based on micro-service architecture,'' in *Proc. IEEE 6th Int. Conf. Future Internet Things Cloud (FiCloud)*, Aug. 2018, pp. 184–191.

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

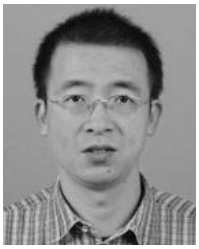
[18] F. Xiao, C. Zhan, H. Lai, L. Tao, and Z. Qu, ''New parallel processing strategies in complex event processing systems with data streams,'' *Int. J. Distrib. Sensor Netw.*, vol. 13, no. 8, Aug. 2017, Art. no. 1550147717728626.

[19] J. F. Allen, ''Maintaining knowledge about temporal intervals,'' *Commun. ACM*, vol. 26, no. 11, pp. 832–843, Nov. 1983.

[20] E. Yoneki and J. Bacon, ''Unified semantics for event correlation over time and space in hybrid network environments,'' in *Proc. OTM Confederated Int. Conf.*. Berlin, Germany: Springer, 2005, pp. 366–384.

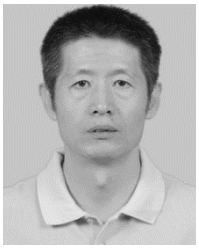
[21] *Drools Documentation*. Accessed: Apr. 3, 2019. [Online]. Available: https://docs.jboss.org/drools/release/7.20.0.Final/drools-docs/html_single/index.html#DroolsComplexEventProcessingChapter

LINA LAN is currently an Associate Professor with the School of Network Education, Beijing University of Posts and Telecommunications (BUPT), Beijing, China. Her current research interests include the Internet of Things, complex event processing, intelligent information processing, and software architecture.

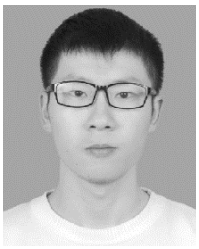


RUISHENG SHI received the Ph.D. degree in information and communication engineering from the Beijing University of Posts and Telecommunications (BUPT), China, in 2013, where he is currently an Associate Professor with the School of Cyberspace Security. His current research interests include the Internet of Things, service computing, publish/subscribe systems, and information security.

BAI WANG is currently a Professor with the School of Computer Science, BUPT, Beijing, China. Her current research interests include the Internet of Things, communication software, distributed computing technology, and intelligent information processing.



LEI ZHANG is currently a Professor with the School of Computer Science, BUPT, Beijing, China. His current research interests include the Internet of Things, communication software, network service, and intelligent information processing.



NING JIANG is currently pursuing the master's degree with the School of Cyberspace Security, BUPT, Beijing, China. His current research interests include the Internet of Things, complex event processing, and information security.