An Efficient Complex Event Processing Algorithm based on INFA-HTS for Out-of-order RFID Event Streams

Jianhua Wang[1], Tao Wang[2], Lianglun Cheng[2], Shilei Lu[1]
[1] College of Electronic Engineering, South China Agricultural University
Guangzhou - P. R. China
[e-mail: wangjianhua655@163.com]
[2] College of Automation, Guangdong University of Technology
Guangzhou - P. R. China
[e-mail: wangtaosea@msn.com, llcheng@gdut.edu.cn, lvshilei@scau.edu.cn]
*Corresponding author: Jianhua Wang

Received November 17, 2015; revised July 19, 2016; accepted August 2, 2016; published September 30, 2016

Abstract

With the aim of solving the problems of long processing times, high memory consumption and low event throughput in the current processing approaches in out-of-order RFID event streams, an efficient complex event processing method based on INFA-HTS (Improved Nondeterministic Finite Automaton-Hash Table Structure) is presented in this paper. The contribution of this paper lies in the fact that we use INFA and HTS to successfully realize the detection of complex events for out-of-order RFID event streams. Specifically, in our scheme, to detect the disorder of out-of-order event streams, we expand the traditional NFA model into a new INFA model to capture the related RFID primitive events from the out-of-order event stream. To high-efficiently manage the large intermediate capturing results, we use the HTS to store and process them. As a result, these problems in the existing methods can be effectively solved by our scheme. The simulation results of our experiments show that our proposed method in this paper outperforms some of the current general processing approaches used to process out-of-order RFID event streams.

Keywords: Complex event processing, Out-of-order, RFID event stream, NFA, Hash table structure

This research was supported by the Joint Funds of the National Natural Science Foundation of China (No. U1201251). The Youth Science Foundation project of National Natural Science Foundation of China (No. 61502110) and (No.61602187). The science and technology projects in Guangdong Province (No.2016A020209007)and (No.2016A020210088), (No.2015A010103014)and(No.2015A020209161), The project of Natural Science Foundation of Guangdong Province (No. 2016A030310453).

http://dx.doi.org/10.3837/tiis.2016.09.016 **ISSN : 1976-7277**

1. Introduction

RFID radio frequency identification) technology is a non-contact automatic identification technology that uses radio frequency communications to achieve data acquisition. With its rapid development, RFID technology has been increasingly adopted in many fields, ranging from RFID tracking for supply chain management to industrial manufacturing monitoring. The wide distributions of RFID devices generate massive RFID event streams.

Because the massive RFID events generated by RFID devices are primitive events, and the semantic information inside the primitive events is limited, we can obtain only simple information from them. However, in real applications, we usually pay more attention to complex information. For example, we focus on mining customers' behaviours[1], predicting a vehicle's route[2], or measuring the similarity of PML documents[3] from a massive RFID event stream. Thus, how to quickly obtain valuable information from a massive event stream becomes a very important challenge when processing the event stream. Because Complex Event Processing (CEP) [4] technology can pick up valuable information from a massive RFID event stream by using the association between event attributes, matching rules and algebraic operations, CEP has recently obtained increasing attention in the field of event stream processing.

Recently, many complex event processing schemes have been studied to detect a complex event-over-event stream, such as a complex event detection method based on a diagram[5], a complex event detection method based on a tree[6], a complex event detection method based on finite automatons[7], a complex event detection method based on petri-nets[8], a complex event detection method based on a workflow[9], and some of their improved methods[10-12]. However, these methods all assume that the RFID event arrivals are totally ordered. They assume that the received order of events, for the events received by the detection system, is the same as the sent timestamp order. However, in practical application scenarios, network latencies, machine failures, node errors and other reasons can cause events that occur first to arrive later, which creates a large number of out-of-order phenomena among the event occurrences. The many occurrences of out-of-order RFID events can lead to output blocking, large system latencies, memory resource overflow and incorrect result generation, which influence the processing efficiency of the RFID event stream.

To address those problems, some complex event processing schemes based on out-of-order event streams have been presented recently, such as K-slack algorithms[13], the Conservative and Aggressive method[14-15], the punctuation-based method [16], the Speculation-based method[17], the Approximation-based method[18], and so on. The existing state-of-the-art methods listed above for addressing out-of-order events can be divided into two classes of approaches: buffering approaches and stream revision approaches. However, the existing two classes of methods have the problems of long processing time, high memory consumption and low detection throughput due to having too many event reorderings and recomputation operations, which influence the whole processing efficiency.

In this paper, with the aim of solving the problems mentioned above that exist in some current methods, an efficient complex event processing method based on INFA-HTS (Improved Nondeterministic Finite Automaton-Hash Table Structure) is presented. The achievements of this paper include the following:

(1) An INFA (Improved Nondeterministic Finite Automaton) model is first proposed in this paper to meet the processing requirement of disorder even in an out-of-order RFID event

stream on the basis of studying existing NFA models, which can solve the problems of long processing latency and high memory consumption by reducing many capturing operations when there are correlated out-of-order primitive events from an out-of-order RFID event stream.

(2) An HTS (Hash Table Structure) is proposed to store and process the large intermediate captured results in this paper; this approach can solve the problems of long processing latency and high memory consumption in out-of-order RFID event streams by applying the hash table mapping, hash table storing, hash table searching and hash table comparison technologies to significantly reduce the searching, inserting, sorting, and comparing operations compared with some of the general methods.

(3) A series of experiments are performed to verify the effectiveness of our proposed method in this paper. The experimental results show that our proposed scheme provides a significant improvement in terms of reducing the average execution time, lowering the average memory consumption and average result latency, and improving the average event throughput compared with some of the current processing methods.

The remainder of this paper is organized as follows. In section 2, the related knowledge on out-of-order RFID event streams is introduced. Our proposed scheme is presented in section 3. The experimental results and analysis using our proposed method are shown in section 4. In section 5, we provide some conclusions.

2. Related Knowledge

2.1 Related definitions

In this section, some definitions that are related to RFID primitive events, RFID complex events and out-of-order RFID events are given as follows:

RFID primitive event: A simple event that is generated by an RFID reader. This event consists of the identifier of a tag, the identifier of a reader and a timestamp when reader recognizes the tag. It can be expressed as e =<r, o, t>, where r represents the location at which the RFID primitive event is generated; o represents the unique identification of the RFID primitive event; and t represents the time at which the RFID primitive event occurs.

RFID complex event: A composite logical event that consists of primitive events based on some rules and some event algebraic operations for application services. There are many event algebraic operators such as AND, OR, NOT, SEQUENCE, INTERVAL, and so on.

Out-of-order RFID primitive event: A simple event that occurs as an out-of-order phenomenon. This event can be defined as follows: given any newly arrived RFID primitive event e_n, suppose that the events that the RFID device received before e_n are e_1, e_2, e_3,...,e_{n-1}; if there is any e_i that satisfies the relationship e_n.timestamp <e_i.timestamp ($1 \le i \le n-1$), then e_n is an out-of-order RFID primitive event.

Fig. 1 is an example of an out-of-order RFID primitive event in an RFID event stream. As seen from Figure 1, the event instance b should appear between event a and event d, and event c should appear between event d and event e; as a result, they do not appear in the correct position, perhaps because of network latencies, machine failures, or node errors. Thus, event b and event c are two out-of-order RFID primitive events in **Fig. 1.**

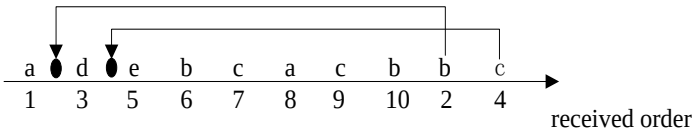


Fig. 1. Out-of-order RFID primitive events over an event stream

2.2 Processing difficulties for an out-of-order RFID event stream

As is known, there is an enormous difference in obtaining a complex event from an out-of-order RFID event stream compared with an ordered stream. The following are some of the processing difficulties when processing out-of-order an RFID event stream:

(1) It is difficult to establish corresponding NFA models to capture the related RFID primitive event from out-of-order RFID primitive events. As is known, it is easy to establish a corresponding NFA model according to pattern expression for capturing the related RFID primitive event due to the total time in an ordered RFID event stream, while it is difficult to build a unified NFA model for capturing related primitive events from an out-of-order RFID event stream due to the uncertainty in the arrival of the RFID events and the sequential order of the NFA model under the same detection condition. Take establishing the corresponding INFA models from out-of-order event pattern expression SEQ (A, B, C), for example, to illustrate the established difficulties for an out-of-order RFID stream. There are two established difficulties: First, in an out-of-order RFID stream, it is possible to have an RFID primitive event appear in any combination of event instances a, b, and c due to its uncertainty of arrival. Second, different combination forms of event instances a, b, and c require different detection models because of the sequential order of the NFA model. Thus, it is very difficult to establish a unified detection model for an out-of-order RFID event stream.

(2) It is difficult to determine the final state of a complex event. It is easy to determine the final state of a complex event in an ordered RFID event stream, which can be accomplished by judging the termination event due to its total time order. However, in an out-of-order RFID event stream, some primitive events composed into a complex event might not have fully arrived when the termination event arrives, due to the network latencies and other reasons. Under such conditions, it is very difficult to determine the final state of a complex event by only judging the termination event. We also consider the constituted timestamp of the out-of-order RFID event. The complexity of a double judgement adds to the processing difficulty for an out-of-order RFID event stream.

(3) It is difficult to judge the nonoccurrence or nonarrival for RFID primitive events composed into the complex event. It is easy to judge whether a correlated RFID primitive event is a nonoccurrence or nonarrival in an ordered RFID event stream due to the total time order. However, in an out-of-order RFID event stream, it is very difficult to judge whether a correlated out-of-order RFID primitive event does not occur or arrive due to the disorder of the RFID event stream, which also adds to the detection difficulty of a complex event over an out-of-order RFID event stream.

2.3. Related studies on out-of-order event streams

Currently, many complex event processing technologies have been proposed to detect a complex event from an out-of-order event stream. Babu et al.[13] and Mutschler et al[19] proposed a method, which is called K-slack, to address the out-of-order arrival of events. Li et al.[20] first proposed a correctness method based on the latency, output order and result correctness to address the out-of-order event stream. Then, they proposed two aggressive

methods and a conservative method to process out-of-order event streams for sequence pattern queries[14-15]. Tucker et al. [16] and Ding et al. [21] propose to use punctuation technology to handle out-of-order event streams. Mutschler et al. proposed to use reliable speculation and adaptive speculation technologies to address out-of-order event streams[17, 22]. Tirthapura et al. [18]and Cormode et al. [23] proposed to use an approximation-based method to address out-of-order event streams.

Barga et al. used a spectrum based on consistency levels and performance tradeoffs to address out-of-order deliveries[24]. In [25], an event processing method based on heartbeats is presented to address uncoordinated event streams. In [26], an accumulative FPGA accelerating towards a sliding window is presented for an out-of-order event stream. In [27], an event querying and matching mechanism with low memory access and a fast response time was presented. At the end of the paper, they also certified its effectiveness by using an experimental method. Paul et al. [28] proposed a complex event processing frame that is based on a speculative rule. Kim et al. proposed a data control method that is based on window processing technology and a data control method that is based on the time interval between tuples and their relationships for out-of-order event stream processing [29-30]. Paper [31] proposed a dataflow model for massive-scale out-of-order event streams. This dataflow model can balance the correctness, latency and cost for massive-scale out-of-order event streams. Papers [32-33] present a quality-driven continuous query execution scheme and a quality-driven processing of sliding window aggregates approach for out-of-order event stream processing, respectively. In their paper[34], Xiao et al. proposed to use a latency distance and purging time to perform real-time processing of out-of-order event streams.

All of the existing state-of-the-art methods above for addressing out-of-order events can be divided into two classes of approaches: the buffering approach and the stream revision approach. However, the buffering approach requires a long latency for the event ordering, while the stream revision approach requires large system overloads for the event ordering, which cause a high memory consumption for an out-of-order event stream. In contrast to the processing methods above, in this paper, efficient complex event processing based on INFA-HTS (Improved Nondeterministic Finite Automaton-Hash Table Structure) is presented for out-of-order RFID event streams based on the analysis and study of the current processing algorithms described above.

3. Proposed scheme

In this section, we proposed an efficient complex event processing method that is based on INFA-HTS (Improved Nondeterministic Finite Automaton-Hash Table Structure) for out-of-order RFID event streams.

3.1. Motivation resource

In our scheme, to solve the detection difficulties exhibited by current detection models based on automatons that cannot adequately detect out-of-order RFID event streams because of the uncertain arrival of RFID primitive events and the sequential order restriction of the NFA model, we proposed an INFA model on the basis of analysis and study the traditional NFA model. To improve the storing and processing efficiency for the large intermediate matching results that exist in detection processing, we proposed to use an HTS (Hash Table Structure) to store and process the large intermediate result. To provide a fast output detection result, we use hash table search technology to output the desired detection results instead of depth-first search technology.

3.2. Working principle of the INFA-HTS algorithm

The basic working principle of our proposed INFA-HTS algorithm is that we first use INFA (Improved Nondeterministic Finite Automaton) to capture the related primitive events from an out-of-order RFID event stream, and then, we utilize HTS (Hash Table Structure) to store and process the large intermediate detection result; last, we use the hash table searching technology to output the complex event sequences. As a result, the problems of having a long detection time, high memory consumption and low event throughput, which exist in the current processing methods for processing out-of-order RFID event streams, can be effectively solved by our scheme.

3.3. Composition structure of the INFA-HTS algorithm

Fig. 2 is the composition structure of our method. **Fig. 2** shows that our proposed INFA-HTS method contains four important composition parts: Read RFID primitive event, INFA match, HTS store and process, and output result, which compose our proposed INFA-HTS method. The Read RFID primitive event mainly executes the reading operation for the RFID primitive events from an out-of-order RFID event stream. INFA match mainly executes the capturing operation for the related RFID primitive event from the reading RFID primitive event above. HTS store and process mainly executes the function of storage and sorting. Output result mainly realizes the output function of the complex event sequences by using hash table search technology instead of depth-first search technology.

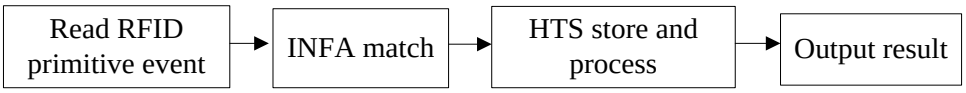


Fig. 2. Composition structure of the INFA-HTS algorithm

In our suggested method, HTS store and process includes the following functions: RFID event mapping, RFID event search, RFID event insertion, RFID event storage and RFID event delete. Where RFID event mapping mainly maps related RFID primitive events into a hash table structure by using a predefined hash mapping function, RFID event search mainly determines the right insertion location by the comparing timestamp among the RFID primitive event. RFID event insertion mainly executes the inserting operation for related RFID primitive events. RFID event storage mainly uses a hash table structure to store the large intermediate detection result; HTS delete mainly deletes outdated RFID primitive events.

3.4. Detection process of the INFA-HTS algorithm

Fig. 3 is the detailed detection process of our INFA-HTS algorithm. In **Fig. 3**, the capital letters (e.g., *E*) represent event types. The lower-case letters (e.g., *a*, *b*, *c*) stand for event instances. For example, b_f represents an event instance of event type E_b with attributes f; b_f, a_f, c_f represent RFID primitive event instances of event type E_b E_a E_c with the same associated attributes. The out-of-order RFID event mainly offers the RFID event resource. It consists of a series of various event instances. The timestamp refers to the received timestamp of an RFID primitive event.

The INFA model is an improved nondeterministic finite automaton model and is mainly used to capture the related out-of-order RFID events. It is constructed by using extending technology on the basis of an NFA model. Its construction process mainly includes the following parts: initialization, construct NFA model, traverse NFA model, merge NFA model and produce INFA model. Initialization mainly makes some necessary preparations before the

program starts to run. Construct NFA model primarily establishes all of the possible NFA models through given event pattern expressions and realizes the construction function of the NFA model for out-of-order RFID events. Traverse NFA mainly traverses each constructed NFA model above by going depth-first from its starting state. Merge NFA mainly executes the merging function by adding the runtime state of the NFA model into the INFA model for all of the NFA models. The ①,②,③ ……stand for automata states.

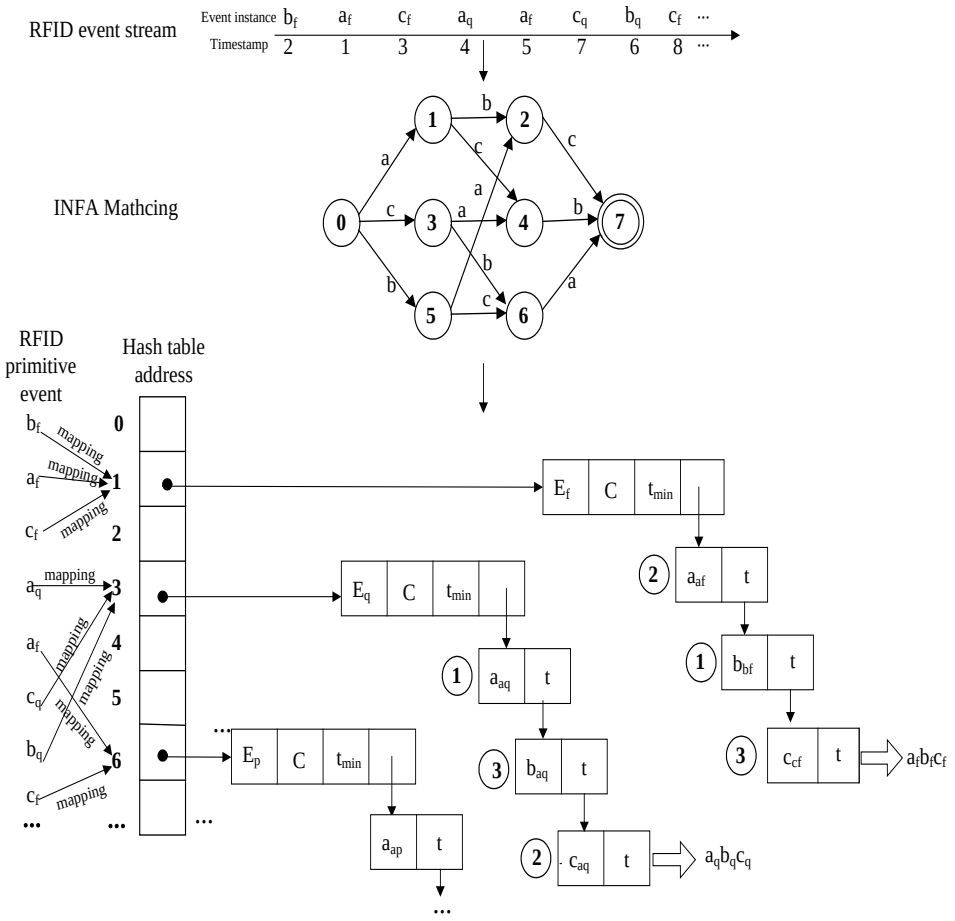


Fig. 3. Detection process of the INFA-HTS algorithm

In our proposed scheme, the hash table mapping function mainly realizes the mapping function for mapping the same associated attributes of the RFID primitive event into the hash table address through some predefined rules. Because the related RFID primitive events in an event pattern expression have the same associated attributes, for example, a_f, b_f, and c_f, we can map a_f, b_f, and c_f event instances into the same hash table address through some predefined rules.

C represents the counter, which is used to calculate the number of related RFID primitive events in the main node. In our proposed scheme, we must execute the output operation only when the counter value of the main node in the main chain is equal to the length value of the INFA. The t_{min} represents the minimum time of the RFID primitive event in the child chain. In our proposed scheme, we can output complex events by hash table search technology when the

timestamp of the current RFID primitive event - the minimum timestamp t_{min} of the main node in the main chain < Time Window.

The main node designed consists of event type E, counter C, minimum timestamp t_{min} and pointers. The child node includes the event instance, timestamp of the event instance and pointers.

In our scheme, after using INFA (Improved Nondeterministic Finite Automaton**)** to capture the related primitive events from an out-of-order RFID event stream, we utilize HTS (Hash Table Structure) to store and process the large intermediate detection result, and last, we use hash table searching technology to output the complex event sequences. RFID event mapping, RFID event search, RFID event insertion, RFID event storage, RFID event delete and RFID event output technologies are used to reduce the many searching, inserting, sorting, comparing operations, which improves the processing performance. We realize the out-of-order RFID event sorting operations by comparing the timestamp of the RFID event on the basis of taking good advantage of the hash table search and hash table insertion technology in this paper. The detailed detection process is shown in **Fig. 3.**

3.5. Realizing the steps of the INFA-HTS algorithm

Realizing the steps for our proposed INFA-HTS algorithm can be summed up in the following steps:

Step 1, build the corresponding INFA according to complex event expression and calculate its length, and in addition, create and initialize a new hash table.

Step 2, read the RFID primitive event from the out-of-order RFID event stream.

Step 3, judge whether the RFID primitive event is accepted by INFA. If yes, jump to step 4 to operate; otherwise, skip to step 2 to operate.

Sep 4, map the event type of the RFID primitive event into the corresponding hash array by using a predefined hash mapping function and judge whether it already exists in the hash array by searching for the event type. If not, then first add a main node in a main chain of this hash array. The main node includes the event type, primitive event counter and minimum timestamp, where the event type refers to the event type of this RFID primitive event. The primitive event counter mainly counts the number of primitive events in the child node, and the minimum timestamp mainly records the minimum time of the RFID primitive event in the child chain. Second, add a child node in a child chain according to the timestamp order, which includes only the event type and the timestamp of the RFID primitive event. If yes, then only add a child node in the right location of the child chain with the event type and the timestamp of the primitive event by comparing the timestamp of this RFID primitive event with other primitive events in succession.

Step 5, judge whether the counter value of the main node in the main chain is equal to the length value of the INFA. If yes, then jump to step 7 to operate; otherwise, skip to step 2 to execute next.

Step 6, judge the timestamp of the RFID primitive event - the minimum timestamp of the main event node in the main chain < Time Window. If yes, then output the complex event by hash table search technology. Otherwise, skip to step 3 to execute next.

Algorithm 1 is the partial pseudo code of the INFA-HTS algorithm. It mainly includes Read RFID primitive event, INFA match, HTS store and process, and Output result, which are four functions, and is realized by using Build_INFA(), read(), hash_array(), compare(), search_hash_table(), and so on. It is the use of the function above that allows us to achieve our suggested algorithm in this paper.

Partial pseudo code of the INFA-HTS algorithm

Algorithm1: complex event processing algorithm based on INFA_HTS
Input: out_of_order RFID stream S, matching expression Q , hash function H
Output: event sequence M

```
(1) initialization
(2) Build_INFA ◄— Q ;
(3) Calculate_INFA_length();
(4) Create_initialize_hashtable();
(5) read  e_i ◄— S
(6) if( e_i is accepted by INFA) then
(7)     hash array◄—H(e_i)
(8)     search E(e_i)
(9)     if(E(e_i) is not in hast array) then
(10)        Mainnode ◄—{ Ee_i,C,T_mix}
(11)        childNode◄—{ Ee_i,t_ei};
(12)        T_mix ◄— t_ei;
(13)        Counter+1;
(14)    end if
(15)  else then
(16)       Compare(e_i,e_i-1)
(17)       add childNode ◄—{ E_ei,t_ei};
(18)       if (t_ei<T_mix) then
(19)           T_mix ◄— t_ei;
(20)           end if
(21)       Counter +1;
(22)   if( value of C == value of ENFA) then
(23)       if(t_ei - T_min<Time Window) then
(24)           M ◄— Search_hash_table() ;
(25)           return M;
(26)       else  then
(27)           go to (5)
(28)   else  then
(29)       go to (5)
(29) else
(30)    go to (5)
```

3.6. Instance study of the INFA-HTS algorithm

Taking the detection of the complex event expression SEQ (A, B, C) from an out-of-order RFID event stream, for example, to illustrate the detailed realization process for our proposed method.

In step 1, build the corresponding INFA according to the complex event expression SEQ (A, B, C), which is shown in **Fig. 3**, and calculate its length value for 3, and in addition, create and initialize a new hash table.

Detect RFID primitive event b_f:

In step 2, read the RFID primitive event b_f from the out-of-order RFID event stream.

In step 3, because the RFID primitive event b_f can be accepted by INFA, the detection program jumps to step 4 to execute next.

In step 4, map the event type E_f of the RFID primitive event b_f into the corresponding hash array by using a predefined hash mapping function. Because there is no event type E_f in this hash array by searching the event type E_f, the detection program first adds an event node in the main chain of the corresponding hash array, which includes the event type E_f of the primitive event b_f, the primitive event counter C and the minimum time stamp t_{min}. Then, add a child node in a child chain with an event type b_{af} of the primitive event b_f and a timestamp t_b of the primitive event b_f. Last, the detection program increases by 1 for the value of the primitive counter C and updates the minimum timestamp t_{min} with t_b.

In step 5, because the value of counter C in the main node (1) is not equal to the length of the value of INFA (3), the detection program skips to step 2 to execute next.

Detect RFID primitive event a_f:

In step 2, read the primitive event a_f from the out-of-order RFID event stream.

In step 3, because the primitive event a_f can be accepted by INFA, the detection program jumps to step 4.

In step 4, map event type E_f of primitive event a_f into the corresponding hash array by using a predefined hash mapping function. Because there is already an event type E_f in the main chain of the corresponding hash array, searching the event type E_f, detection program, first adds a child node with event type a_{bf} of primitive event a_f and time stamp t_a in the precursor of the b_f event by judging $t_a < t_b$ and, second, adds the value of counter C for 2.

In step 5, because the value of counter C in main node (2) is not equal to the length value of INFA(3), the detection program still skips step 2, Additionally, it updates the minimum timestamp t_{mix} with t_a.

Detect RFID primitive event c_f;

In step 2, read the primitive event c_f from the out-of-order RFID event stream.

In step 3, because the primitive event c_f can be accepted by INFA, the detection program jumps to step 4.

In step 4, map the event type E_f of the RFID primitive event c_f into the corresponding hash array by using the predefined hash mapping function. Because there is already an event type E_f in the main chain of the corresponding hash array by searching the event type E_f, the detection program first only adds a child node with event type c_{cf} of RFID primitive event c_f and timestamp t_c of RFID primitive event c_f subsequent to event b_f by comparing $t_a < t_b < t_c$. Second, the detection program adds the value of counter C to 3.

In step 5, because the value of counter C in the main node (3) is equal to the length value of INFA(3), the detection program jumps to step 6 to execute next.

In step 6, because the timestamp t_c of RFID, the primitive event c_f - the minimum timestamp in main node< Time Window, and detection program outputs a complex event a_f b_f c_f by using hash table search technology.

The other RFID primitive events over the out-of-order RFID event stream, such as a_f, c_q, b_q, and so on, can be detected by a detection method similar to the above.

4. Experimental Results and Analysis

In this section, to verify the effectiveness of our proposed method above, we perform some experiments. Our designed experiments mainly include four parts: build experimental environment, test average execution time, average memory consumption, average application latency, average accuracy rate and average event throughput in different out-of-order event stream percentages.

4.1. Build experimental environment

Our simulation environment was conducted on two Windows PCs with Microsoft Windows 7 operating systems, AMD A6-3420M 4 core CPU Processors, 2 G of memory and a 500 G hard disk. One PC is used to generate the event streams and send out the event streams to the other PC, which is used as an event query engine. In our experiment, we use the Visual C++ 6.0 tool to develop an event generator. This event generator can be configured with parameters as listed in **Table 1**. The input event stream includes events of 10 different event types. Each

event source can send out event instances with the same type. The event distributes among event types A to J.

Table 1. Main experimental parameters in our experiment

Parameter name	parameter value
Bit number of hash table	1~10
Sliding window size	40
Number of event types	10
Sequence queries expression	SEQ(A, B, !C, D, E, F, G)
Length of sequence queries	7
Out-of-order event percentage of event type	0~50(%)
Maximum arrival delay of event type	10 time unit
Size of event stream	10^5
Number of attributes of each event	2
POGs percentage.	20%

In our experiment, we select SEQ(A, B, !C, D, E, F, G) as the testing sequence queries for all of the experiments, and the length of the testing sequence queries is set to 7. To keep the memory consumption and the overhead of *POGs* relative small, se set the percentage of *POGs* to 20%. The comparison indicators in our experiment are selected for processing time, accuracy rate, average application latency, memory consumption and event throughput. The processing time refers to the total executed time for addressing the whole out-of-order event stream. The accuracy rate denotes the accuracy for outputting the desired results from the out-of-order event streams. The average application latency refers to the average time difference between the sequence output time and the maximum arrival time of the event instances, which compose the sequence result. The memory consumption refers to the total used memory for addressing the out-of-order event streams. The event throughput refers to the output desired result from the out-of-order event streams.

The Conservative method, Aggressive method and K-slack method are selected as our comparison methods in our experiment, whereas the aggressive method is mainly used to produce a maximal output when the out-of-order event arrival is rare. To tackle any premature erroneous result generation when an out-of-order event arrives, appropriate error compensation methods are designed for this method. The conservative method is mainly used to produce a guaranteed output when out-of-order events are common. To guarantee the correctness result, a POGs model is proposed to guarantee the result correctness. K-slack algorithms are mainly used to buffer the arriving data for K time units because it generally assumes that the event might arrive out-of-order at most by some constant K time units. The out-of-order event percentage in each event stream is defined as follows: Out-of-order event percentage=the total number of out-of-order events/the total number of instances received by our processing system.

The accuracy rate can is defined as follows: Accuracy rate=(the number of pattern-matching results on the ordered event stream $\cap$ the number of pattern-matching results over the out-of-order event stream)/ the number of pattern-matching results over the ordered event stream. The average application latency can be defined as follows: Average application latency= Σ (the time of sequence output from our system − the maximum event arrival time)/Number of event types. The detailed experimental results are shown next.

4.2 Test average execution time

In this subsection, we evaluate the average execution time of our proposed scheme with the other three general methods in different out-of-order event stream percentages. **Fig. 4** shows the experimental results.

From **Fig. 4**, we can see clearly that our proposed algorithm shows the least execution time of the four methods. The Aggressive algorithm follows. The K-slack method costs the largest average execution time. The reasons for these findings are that, in our scheme, we first use INFA to reduce the capturing operations for the related out-of-order events, which can save much constructing and capturing time. Second, we use a hash table structure to store and address the large intermediate capturing results by using hash mapping, searching and inserting technologies, which can save a large number of execution operations for disordered events, therefore saving a large amount of average execution time. The K-slack strategy consumes the largest average execution time due to its many waiting operations for the maximum delay before processing. The aggressive algorithm spends less average execution time at the lower out-of-order event percentage compared with the conservative method, but when there is an increase in the out-of-order events percentage and when it reaches a certain threshold (e.g., 40% in this experiment), it starts to require more average execution time than the conservative strategy because of its extra recomputation operations for compensation tuples. The conservative method shows the opposite performance compared with the aggressive method.

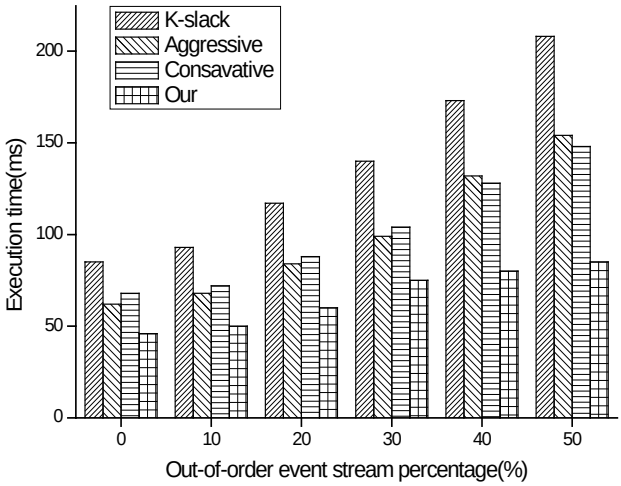


Fig. 4. Execution time in different out-of-order event data percentages

4.3. Testing the average memory consumption

In this subsection, we test the average memory consumption for our suggested scheme. The testing results are shown in **Fig. 5**.

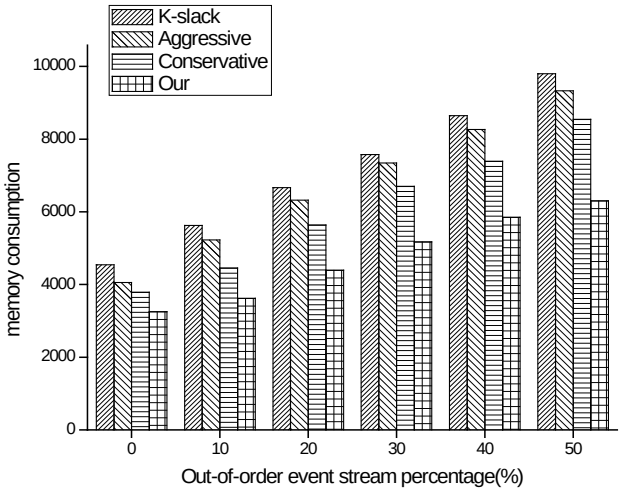


Fig. 5. Memory consumption in different out-of-order event data percentages

From **Fig. 5,** we can observe that our proposed method presents a good average memory consumption savings rate compared to the other three methods. The Conservative strategy follows. The aggressive solution and K-slack solution show the highest average memory consumption among the four approaches. The reasons for these findings are the following: In our scheme, on the one hand, an INFA model is established to reduce the capturing operation for the related out-of-order events, which can save much capturing memory for out-of-order events. On the other hand, the hash table structure is used to process the large intermediate capturing results, which can reduce the many inserting, storing and comparing operations of the related primitive events by applying hash mapping technology, thus also saving a large amount of processing memory. The Conservative method presents a low average memory consumption because the POGs in the conservative method can help the system to purge the related event instances over time, which can result in larger savings in memory. Because the aggressive solution must generate more recomputations for compensation tuples with more arrivals of out-of-order events, more memory is consumed. The K-slack strategy has a high average memory consumption because of its waiting operations for the maximum delay before processing.

4.4. Testing the average application latency

In this subsection, we mainly evaluate the performance of the average application latency for our proposed scheme. **Fig. 6** shows the testing results of the average application latency in different out-of-order event data percentages.

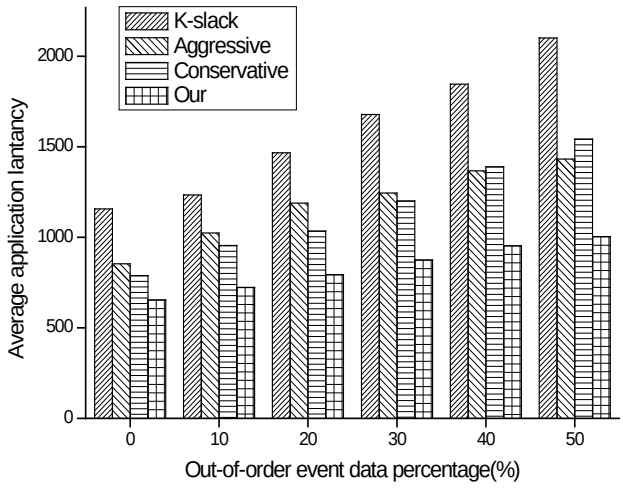


Fig. 6. Average application latency in different out-of-order event data percentages

From **Fig. 6,** we can observe that our proposed method presents the least average application latency compared with the other three methods, with the same percentage of out-of-order percentage. The aggressive and conservative algorithms follow. The K-slack strategy shows the largest average application latency of the four approaches. The reasons for these findings are the following: In our scheme, first, an INFA model is established to reduce the capturing operation for the related out-of-order events, which can save a large amount of capturing latency for out-of-order events; Second, a hash table structure is used to process the large intermediate capturing results, which can reduce many of the inserting, storing and comparing operations of related primitive events by applying hash mapping technology, thus also saving a large amount of processing latency. Because the K-slack strategy must wait for the maximum delay before processing, this arrangement leads to a large average application latency. The aggressive solution shows a small average application latency compared with the conservative strategy when the out-of-order percentage of event is relatively low, but with the increase in the out-of-order events percentage, when it reaching a certain threshold (e.g., 40% in this experiment), it starts to generate the largest latency compared with the conservative strategy. The reason is that the aggressive solution must generate more recomputations for compensation tuples with the arrival of out-of-order events, which delays the generation of the sequence results. In contrast, the average application latency of the conservative method presents a decrease with an increment in the out-of-order percentage of events. This finding occurs because the application latency is mainly determined by POGs in the conservative method, which can help the system to purge the related event instances over time.

4.5. Testing the average accuracy rate

In this subsection, we evaluate the average accuracy rate of our proposed scheme. The testing results of the average accuracy rate for the four methods in different out-of-order event stream percentages are shown in **Fig. 7**.

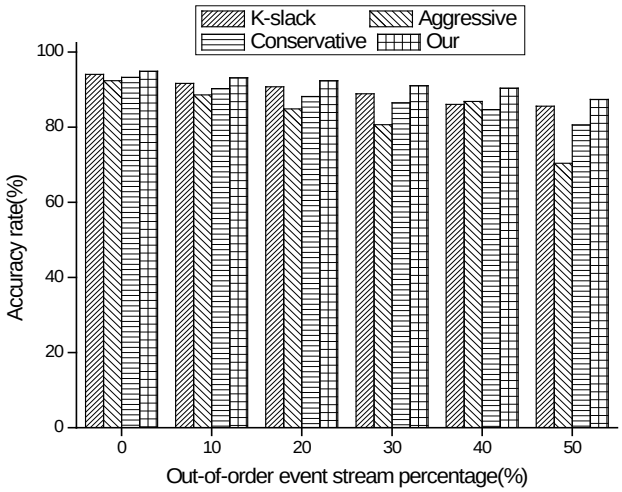


Fig. 7. Accuracy rate with different out-of-order event data percentages

Fig. 7 reveals that our algorithm has a similar average accuracy rate as the conservative method and K-slack. The aggressive algorithm shows the worst accuracy rate of the four methods. The reasons can be explained as follows: In our proposed scheme, a hash table structure is used to process the large out-of-order capturing results by taking advantage of the searching, inserting, storing and comparing functions of hash table technology after using the INFA model to capture the related out-of-order events, which can reduce many wrong operations, thus keeping up the high average accuracy rate. The K-slack method can achieve an average good accuracy rate due to its setting a high value for K, which can reduce many erroneous results, but at the same time, it sacrifices an enormous processing time, memory consumption and application latency. The conservative method can realize a high accuracy rate because it can take good advantage of POGs to guarantee the correctness of the output by reducing many of the of operator states and generated results, thereby having a high accuracy rate of the detection results with the increase in the out-of-order events percentage. The aggressive algorithm shows a low accuracy rate in the out-of-order events percentage because of the use the aggressive strategy in its methods. It needs to use an error compensation mechanism to tackle any premature erroneous result generation when each out-of-order event appears, thereby influencing its accuracy rate.

4.6. Testing the average event throughput

In this subsection, we evaluate the average event throughput of our proposed scheme. **Fig. 8** shows the experimental results.

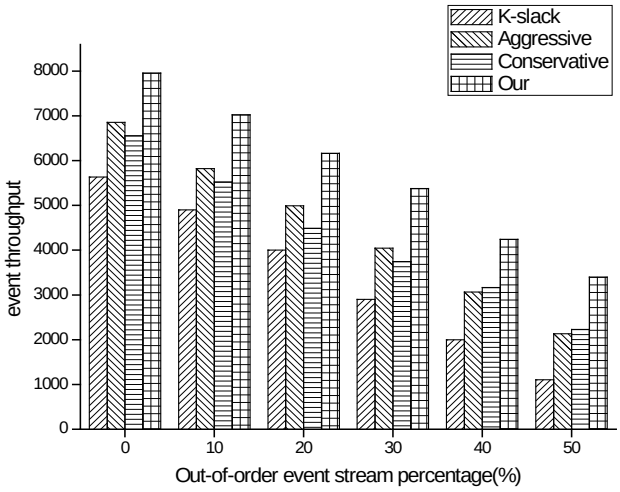


Fig. 8. Event throughput in different out-of-order event data percentages

From **Fig. 8,** we can see clearly that our proposed algorithm shows a good average event throughput for the four methods. The aggressive algorithm and conservative method follow. The K-slack method presents the worst processing performance. The reasons for these findings are that in our scheme, we use INFA and hash table structure to reduce the capturing, storing and processing operations for the related out-of-order events, therefore improving its whole processing speed. The aggressive algorithm outputs a high average event throughput due to the aggressive strategy in its method, while the conservative method has a good event throughput due to its conservative strategy. The K-slack strategy shows the worst processing performance because of its long waiting operations for the maximum delay before processing. **Fig. 8** also shows that the processing times in the four methods all present an increase with growth in the out-of-order event stream percentages, but our suggested method increases in a relatively flat way compared with the other algorithms.

In addition, from **Fig. 4** through **8** above, we can obtain the conclusion that the four methods above have their own different applicability scopes. The aggressive solution is more suitable for the scene with low out-of-order events. The conservative aggressive is better fit to the scene with a high out-of-order event. The K-slack method is better to use in an environment that has a constant latency. However, our proposed scheme in this paper can apply to all of the scenes and obtains better processing results.

5. Conclusions

In this paper, an efficient complex event processing scheme is presented for an out-of-order RFID event stream. In our scheme, first, we use INFA to capture the related primitive events from the out-of-order RFID event stream. Second, we use HTS to store and process the large intermediate detection result. As a result, some of the problems that exist in some of the current methods can be effectively solved by our scheme. The simulation experiments demonstrate the effectiveness of our proposed approach in terms of the execution time, memory consumption, application latency, accuracy rate, and event throughput compared with some of the current general processing methods.

References

[1] Wang Z, Ye N and Malekian R, et al. "Measuring the similarity of PML documents with RFID–based sensors," *International Journal of Ad Hoc and Ubiquitous Computing*, vol.17, no.2-3, pp. 174-185, 2014. Article (CrossRef Link).

[2] Wang Z, Ye N, Malekian R, et al. "TMicroscope: Behavior Perception Based on the Slightest RFID Tag Motion," *Elektronika ir Elektrotechnika*, vol.22, no.2, pp.114-122, 2016. Article (CrossRef Link).

[3] Ye N, Wang Z and Malekian R, et al. "A method for driving route predictions based on hidden Markov model," *Mathematical Problems in Engineering*, 2015, 2015. Article (CrossRef Link).

[4] Del G, Eugene W, Hee J, et.al. "SASE: Complex Event Proeessing over Streams," in *Proc. of 3rd Biennial Conference on Innovative Data Systems Researeh*, pp.407-411, Dec 20-22, 2007. Article (CrossRef Link).

[5] L. Bai, S. Lao, A. F. Smeaton, N. E. O'Connor, D. A. Sadlier, and D. Sinclair, "Semantic analysis of field sports video using apetrinet of audio-visual concepts," *Computer*, vol.52, no.7, pp.808-823, 2009. Article (CrossRef Link).

[6] Xiangwei Sun, Rong Chen, Zhenjun Du. "Composite Event Detection Based on Automata," in *Proc. of 2009 IEEE International Conference on Intelligent Human-Machine Systems and Cybernetics*, pp. 160-163, Aug 26-27, 2009. Article (CrossRef Link).

[7] Y Mei, S Madden, "ZStream: A cost-based query processor for adaptively detecting composite events," in *Proc. of the SIGMOD2009*. Providence, USA, pp.193-206, June 28-29, 2009. Article (CrossRef Link).

[8] F Wang, S Liu, P. Liu, "Bridging physical and virtual worlds: complex event processing for RFID data streams," in *Proc. of the 10th International Conference on EDBT*, pp. 588- 607. March 26-31, 2006. Article (CrossRef Link).

[9] C Zang, Y Fan, "Complex event processing in enterprise information systems based on RFID," *Enterprise Information Systems*, vol.1, no.1, pp.3-23, 2007. Article (CrossRef Link).

[10] Liu H Y, Li J H, "The study and application of tree-based RFID complex event detection algorithm," in *Proc. of the Second International Symposium on Web Information Systems and Application*, Nanchang , China, pp.520-524, May 22-24, 2009. Article (CrossRef Link)

[11] Ke J, Zhan Y Z, Chen X J, et al. "Detection of complexity video event based on hypergraph model," *Application Research of Computers*, Vol.29, no.12, pp.4770-4774, 2012. Article (CrossRef Link).

[12] Bai L, Lao S, Smeaton A F, et al. "Semantic analysis of field sports video using apetrinet of audio-visual concepts," *Computer Journal*, vol.52, no.7, pp. 808-823, 2009. Article (CrossRef Link).

[13] Babu S, Srivastava U. and Widom J., "Exploiting k-Constraints to Reduce Memory Over-Head in Continuous Queries over Data Streams," *ACM Transitionson Database Systems*, Vol. 29, No.3, pp. 545_580, 2004. Article (CrossRef Link).

[14] Liu M, Li M, Golovnya D, Rundensteiner E A and Claypool K. "Sequence Pattern Query Processing over Out-of-Order Event Streams," in *Proc. of 2009 ICDE*, Shanghai, China, pp. 784-795, March 29-April 2, 2009. Article (CrossRef Link).

[15] Wei M. and Liu M, "Supporting a Spectrum of Out-of-Order Event Processing Technologies: From Aggressive to Conservative Methodologies," in *Proc. of 2009 SIGMOD*, Providence, U S A, pp. 1031-1033, June 29-July1, 2009. Article (CrossRef Link).

[16] Tucker P A, Maier D, Sheard T and Fegaras, L.,"Exploiting Punctuation Semantics in ContinuousData Streams," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 15, No. 3, pp. 555-568 ,2003. Article (CrossRef Link).

[17] C Mutschler, "Adaptive Speculative Processing of Out-of-Order Event Streams," *ACM Transactions on Internet Technology*, vol.14, no.1, pp.4-8, 2014. Article (CrossRef Link).

[18] S. Tirthapura and D. P. Woodruff, "A general method for estimating correlated aggregates over a data stream," *Algorithmica*, Vol.73, no.2, pp.235-260, October 2015. Article (CrossRef Link).

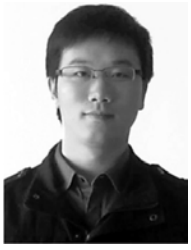
[19] C Mutschler, M Philippsen, "Distributed Low-Latency Out-of-Order Event Processing for High

Data Rate Sensor Streams," in *Proc. of IEEE 27th International Symposium on Parallel and Distributed Processing*, Boston, pp.1133-1144, May 20-24, 2013. Article (CrossRef Link).
[20] M Li, M Liu. L P Ding, et al. "Event stream processing with out-of-order data arrival," in *Proc. of 27th International Conference on Distributed Computing Systems Workshops*, pp. 67-67, Toronto, June 22-29, 2007. Article (CrossRef Link).
[21] L Ding, N Mehta E A. Runden steiner, and G. T. Heineman, "Joining punctuated streams," *EDBT*, pp. 587-604, March 14-18, 2004. Article (CrossRef Link).
[22] C Mutschler, M Philippsen, "Reliable speculative processing of out-of-order event streams in generic publish/subscribe middlewares," in *Proc. of the 7th ACM International Conference on Distributed Event-Based Systems*, Arlington, United states, pp.147-158, June 27-29, 2013. Article (CrossRef Link).
[23] Cormode G, Korn F, Tirthapura S, "Time-decaying aggregates in out-of-order streams," in *Proc. of* PODS, Vancouver, Canada, pp.89–98, June 9-12, 2008. Article (CrossRef Link).
[24] R S Barga et. al. "Consistent streaming through time: A vision for event stream processing," in *Proc. of 3rd Biennial Conference on Innovative Data Systems Research*, pp. 363–374, January 7-10, 2007. Article (CrossRef Link).
[25] J. Kr amer and B. Seeger, "Semantics and implementation of continuous sliding window queries over data streams," *ACM Trans. Database Syst*, vol.34, no.1, pp.4:1-4:49, 2009. Article (CrossRef Link).
[26] Y Oge, M Yoshimi T Miyoshi, et al. "Wire-Speed Implementation of Sliding-Window Aggregate Operator over Out-of-Order Data Streams," in *Proc. of IEEE 7th International Symposium on Embedded Multicore Socs*, pp.55-60, Sept 26-28, 2013. Article (CrossRef Link).
[27] K Wang, Y Yu. "A query–matching mechanism over out–of–order event stream in IOT," *International Journal of Ad Hoc and Ubiquitous Computing*, vol.13, no.3, pp.197-208, 2013. Article (CrossRef Link).
[28] R Fodor, D Anicic, S Rudolph, "Results on Out-of-Order Event Processing," in *Proc. of 13th International Symposium.* New York, pp.220-234, January24-25, 2011. Article (CrossRef Link).
[29] H G Kim, W L Kang, M H Kim, "Efficient Window Processing over Disordered Data Streams," *IEICE Transactions on Information and Systems*, vol.E93D, no.3, pp.635-638, 2010. Article (CrossRef Link).
[30] H G Kim, C Kim, M H Kim, "Adaptive disorder control in data stream processing," *Computing and Informatics*, vol. 31, no. 2, pp.393-410, 2012. Article (CrossRef Link).
[31] Akidau T, Bradshaw R, Chambers C, et al. "The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," in *Proc. of the VLDB Endowment*, vol.8, no.12, pp.1792-1803, 2015. Article (CrossRef Link).
[32] Ji Y, Zhou H, Jerzak Z, et al. "Quality-Driven Continuous Query Execution over Out-of-Order Data Streams," in *Proc. of the 2015 ACM SIGMOD International Conference on Management of Data*, pp. 889-894, May 25-27, 2015. Article (CrossRef Link).
[33] Ji Y, Zhou H, Jerzak Z, et al. "Quality-driven processing of sliding window aggregates over out-of-order data streams," in *Proc. of the 9th ACM International Conference on Distributed Event-Based Systems*, ACM, pp.68-79, 2015. Article (CrossRef Link).
[34] Xiao Y, Jiang T, Shen Y, et al. "Efficient Strategy for Out-of-Order Event Stream Processing," *Journal of Applied Science and Engineering*, vol.17, no.1, pp.73-80, 2014. Article (CrossRef Link).

JianHua Wang was born on February 6, 1982 in Guangdong, China. He received his B.S degree in Electronic Information Science and Technology from Shaoguan University, Guangdong, China, in 2006. He received his Ph.D degree in Control Science and Engineering at Guangdong University of Technology, Guangdong, China, in 2015. Currently he is a teacher of college of electronic engineering, south china agricultural university, Guangzhou, China. His research interests include wireless video transmission, cyber-physical systems and IoT.

Tao Wang was born on October 21, 1983. He received the PHD Degree in network security from Sun Yat-Sen University, Guangzhou, China, in 2010. Currently he is a teacher of College of Automation, Guangdong University of Technology, Guangzhou, China. His current research interest includes context-aware computing, service composition, protocol optimization in wireless sensor network and cyber-physical system.

LiangLun Cheng was born on August 22, 1964 in Hubei. He received his M.S and Ph.D degrees from Huazhong University of Science and Technology, Hubei, China in 1992 and Chinese academy of Sciences Jilin, china in 1999 respectively. He is a Prof and doctoral supervisor of Guangdong University of Technology. His research interests include RFID and WSN, IoT and CPS, production equipment and automation of the production process, embedded system, the complex system modeling and its optimization control, software of automation and information, etc



Shilei Lu was born on January 17, 1984. He received his B.S. degree in electrical automation engineering from Northeastern University, Qinhuangdao, China, in 2006 and received his PHD Degree from Sun Yat-Sen University in 2013. Currently he is a teacher of college of electronic engineering, south china agricultural university, Guangzhou, China. His research interests include Agricultural Internet of things and intelligent optimization, RFID network security based on IOT.