Optimization of Event Processing in RFID-Enabled Healthcare

Shanglian Peng and Jia He

1 Introduction and Motivation Example

With increasing smart devices introduced in healthcare monitoring applications, it becomes possible to manage people and objects in real time in the Internet of Things (IoT) era. Radio-frequency identification (RFID) is one of the most popular techniques used in healthcare monitoring scenarios which can be used to identify and monitor elderly people and patients, track hospital assets and medical instruments, validate patients' drug compliance, check status of operations, etc. [1] In RFID-enabled healthcare applications, streams of data are collected in real time and need to be processed within second response time in order to reduce risk of decisions.

To ensure error-free decision-making in life-critical RFID-enabled monitoring applications, a careful and fast responsive computing model is needed. Complex event processing (CEP) [2], as a stream-based computing paradigm, has been widely used in time-critical stream processing systems. Over RFID streams, queries of interest are considered as complex patterns (or complex events) which can be defined using SQL-style declarative [3] or rule-based languages [4]. To evaluate these patterns (queries), a non-deterministic or tree-based model is needed which could be running over the real-time RFID event streams [5]. However, existing event detection engines are limited in optimization algorithms over RFID-enabled healthcare applications [6, 7]. This paper is motivated by the need to efficiently

S. Peng (✉) · J. He
College of Computer Science and Technology, Chengdu University of Information Technology, Chengdu, China
e-mail: psl@cuit.edu.cn; hejia@cuit.edu.cn

© Springer International Publishing AG, part of Springer Nature 2019

M. Jiang et al. (eds.), *The Proceedings of the International Conference on Sensing and Imaging*, Lecture Notes in Electrical Engineering 506,
https://doi.org/10.1007/978-3-319-91659-0_29

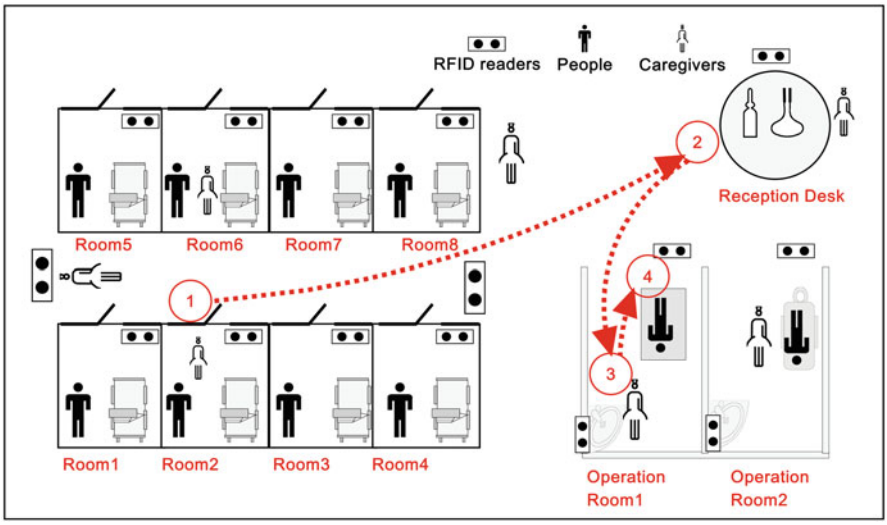

Fig. 1 Motivation example of RFID-enabled healthcare monitoring

run event detection queries in a healthcare RFID application with specialized optimization algorithms supported.

Motivation Example As shown in Fig. 1, in a RFID-enabled healthcare application, suppose we want to monitor people and caregiver's activity and movement in order to make error-free decisions. Monitored people are attached with RFID tags, the objects such as medicine, dosages, and instruments are also tracked by RFID tags. Due to cost consideration, we assume to use passive tags. RFID tags are read at fixed points and mobile readers. The readings of RFID tags, which we call simple events (records), will be pushed to a local server. Event patterns (queries) can be defined over these event streams. In a healthcare application, many activities need to be monitored with specific workflow. For example, in Fig. 1, to track whether a person in Room2 gets the right caring process, we can define an event detection pattern query over the RFID streams with a declarative pattern definition language proposed in [8, 9]:

Query 1:
PATTERN SEQ(Room r, ReceptionDesk rd,
! SEQ(SPD spd, Decontamination d, Packing p, Sterilization s, StoragePatch sp, spd.id = d.id = p.id = s.id = sp.id),
Washing w, Operating o, rd.id=w.id=o.id)

Here, the control flow of RFID-based sterile processing [10] is shown in Fig. 2 which can be formulated as a subpattern in Query1 SEQ(SPD spd, Decontamination d, Packing p, Sterilization s, StoragePatch sp, spd.id = d.id = p.id = s.id = sp.id).

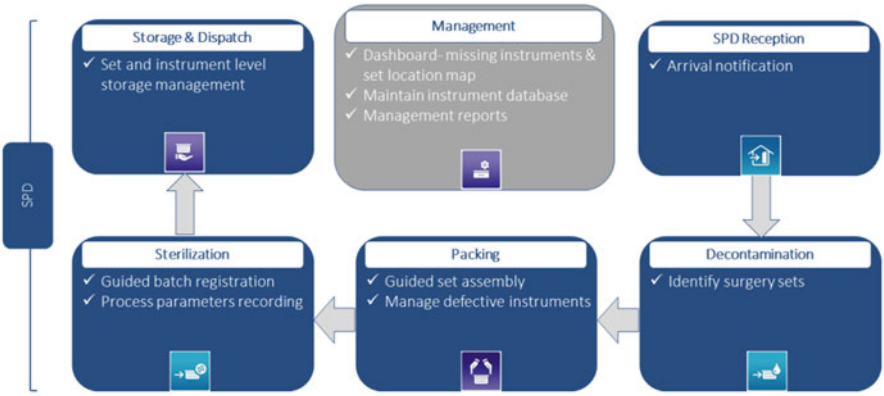

Fig. 2 The life cycle and workflow of surgical items in healthcare and hospital [10]

Query1 is a long sequence pattern query with a negated nested SEQ pattern imbedded. Semantic meaning of Query1 is that a person moving from the room area to the operation room should be followed by a reception desk reading and a washing reading, and if the tools used for the operation are not (!) disinfected by the flow in Fig. 2, then a complex event is triggered. The state-of-art complex event processing (CEP) works [1, 2, 6] do not support such nested pattern evaluation. In the Cayuga system [11], authors define composable queries with negation operator which is only applied to a single atomic event type of SEQ pattern. NEEL [8, 9] aims to solve nested pattern queries with query rewrite and some data structures, and their works are the closest to the work in this paper. In this paper, we address the problem of evaluate negation nested sequence queries in a healthcare context. The rest of the paper is organized as follows: Section 2 introduces the event model. Section 3 presents the NFA-based evaluation model. In Sections 4, we present optimization method of the evaluation model. Section 5 discusses experimental analysis of the evaluation model, while in Section 8, we draw a conclusion of the paper.

2 Event Model

Event type is a specification or class label of objects that have the same semantic meaning. Primitive event/atom event is an event which cannot be divided into smaller events. Each primitive event has an event type. A complex event is an event which is a combination of primitive events and/or complex events connected by event operators. Event operators used in our event model include SEQ, Negation (!), AND, and OR. An event instance denotes occurrence of a primitive or composite event. Primitive event instance is denoted by lowercase letters, for instance, in Query1 “s”. Event instance has temporal information denoted with start time and end time. For a primitive event instance, start time equals to end time.

Event type is denoted as E_i which includes attributes of the event instances of this type. Primitive event types are predefined in the application domain. For example, event type of decontamination event in Query 1 is denoted as "D," while $d_i \in D_i$ denotes d_i is an event instance of event type D_i. Each event type has attributes which is denoted as e_i. $attr_j$ meaning the *jth* attribute of event instance e_i [3, 8].

3 Evaluation Model

To evaluate a pattern query as described in Query, there are some models such as tree-based model [12], petri-net-based model [13], rule-based model [14], and NFA-based model [3]. We use the NFA-based model to evaluate pattern queries because NFA is suitable for fast stream processing and is easy to be implemented.

For an event query defined with SASE language, the query is first transformed into a query plan, and the query plan is then transformed into a NFA model. The transformation of Query 1 is shown in Fig. 3.

In Fig. 3, the event pattern is first compiled into a query plan tree, event operators are round nodes in the tree (SEQ in in Fig. 3.), and event types are transformed into rectangular square nodes. Attribute constraints of different operators are attached to corresponding nodes.

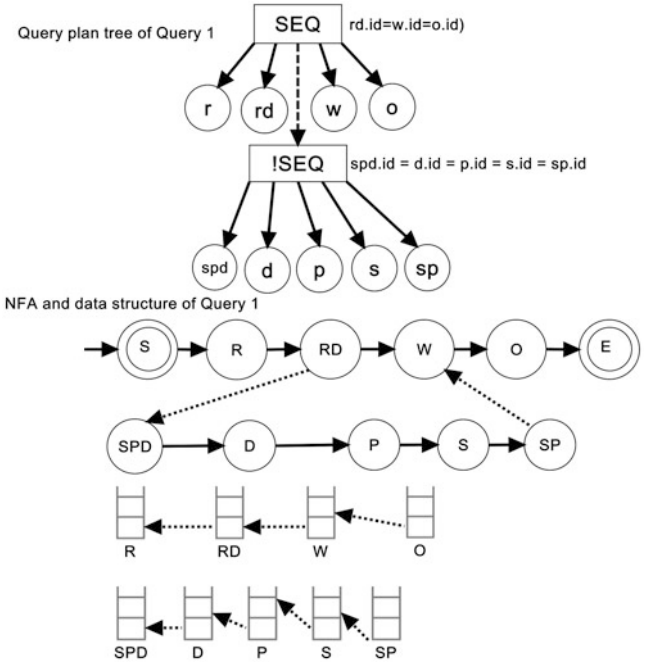

Fig. 3 Query plan and NFA of Query1

Table 1 Running example of Query1 over a stream snapshot of a healthcare application

Timestamp	Location	TagID	Timestamp	Location	TagID
8:00:24 13-04	**R1**	**0001**	9:05:28 13-04	RD	0007
8:05:33 13-04	**W**	**0006**	9:05:33 13-04	R1	0003
8:08:55 13-04	R2	0002	9:11:55 13-04	R2	0004
8:12:21 13-04	**RD**	**0001**	**9:22:21 13-04**	**W**	**0001**
8:24:35 13-04	RD	0003	**9:30:35 13-04**	**O**	**0001**
9:00:01 13-04	RD	0002	9:32:01 13-04	O	0002

The query plan tree is then transformed into a NFA model. A start and end node is constructed for a query as shown in Fig. 3. As there exists a Negation SEQ pattern in Query, we construct a sub-NFA embedded into the overall NFA. A stack-based data structure is used to store event instances of different event types. Events of the same tag ID in different event types are connected with forwarded arrows.

Running Example For a given event stream snapshot as shown in Table 1, the evaluation works as follows: For event types of R (R1-R8 in Fig. 1), each primitive event should be stored in stack of type R in Fig. 3. In Table 1, caregivers with TagID 0001, 0002, 0003, and 0004 are stored into a stack first, and then according to the NFA model in Fig. 3, we should check whether there exist subsequent events that fulfill the attribute constraint described in Query 1. This operation is checked by traversing the corresponding stacks of RD, W, and O of the NFA and checks the TagID attribute constraint over each stack. So we need to keep all the events of types R, RD, W, and O in the evaluation process which would result in great memory consumption and search operations. As shown in Table 1, the event with TagID 0001 in red color would satisfy partial matches of Query 1; we need to check whether the caregiver with TagID 0001 took the disinfected tools to people in the operation room with sequence pattern SEQ(SPD spd, Decontamination d, Packing p, Sterilization s, StoragePatch sp, spd.id = d.id = p.id = s.id = sp.id). This pattern should be detected over workflow streams in Fig. 2; in this running example, we omit the stream processing of this pattern.

4 Optimization Algorithm

In the evaluation process of the NAF of an event query, we need to keep all the partial matches of the subpattern until we slide forward to the next processing window. With this mechanism, we need to keep partial matches in the stack and delete related events which would not contribute to future matches. As stack operations are hard to apply to batch deletions over the event streams, we propose to use an ordered B-tree-based data structure to optimize the deletion operation of partial matches. The ordered B-tree structure is shown in Fig. 4.

In Fig. 4, we only store a pointer in the stack data structure of NFA, and the event data is stored in leaf nodes of a time-ordered B-tree data structure. Each TagID is stored only once. Batch deletion of events from the memory works as the following steps:

1. At the end of the sliding window, compute the time range which indicates the start and end time of the window in the ordered B-tree.
2. For each TagID in the ordered B-tree, search for related stack, and search for the subsequent events from the pointers between different types of stacks; output the patterns that satisfy the query.
3. Delete the partial matches that do not generate matches in the time window, and re-initialize the stack, and readjust the ordered window.

The event detection processing will move to the next sliding window after the deletion process.

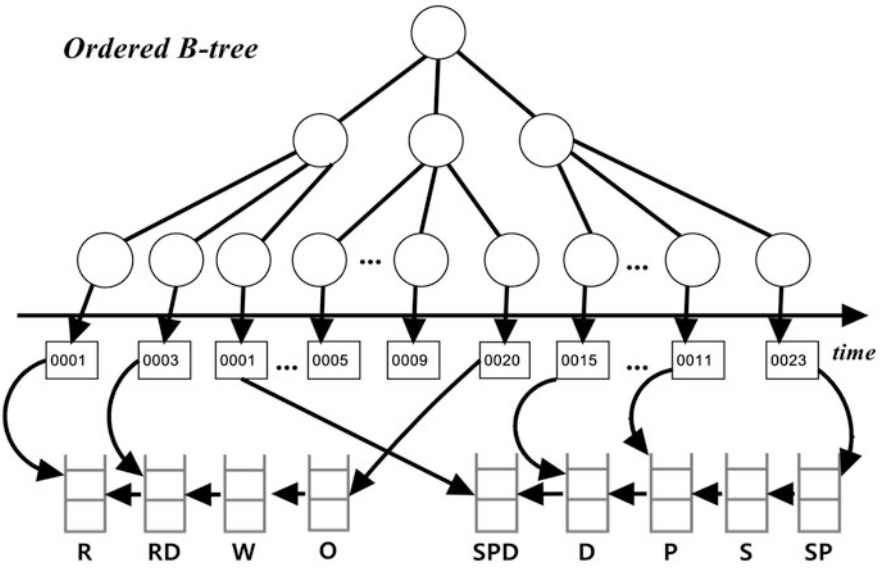

Fig. 4 Illustration of the ordered B-tree of Query1

5 Experimental Analysis

In this section, we present experimental study of the event detection model with a simulated RFID-enabled healthcare application. In the application, RFID read interval is set to seconds that means we generate RFID data stream from different RFID readers in seconds. We generate events in Fig. 1 with multithread styles to simulate a practical scenario. The streams vary in hours long with different volumes. Events of different event types fulfills a predefined normal probability distribution.

The simulation system is implemented in Visual Studio with C ++; data generator is implemented with C#. The computer used in the experiment is with i5 core processor and 4G memory. To compare the model and algorithm, we use NEEL [9] as the benchmark.

We have tested response time and memory consumption of the algorithms over different volume of streams with sliding window set 30 min. The results are shown in Figs. 5 and 6.

From Fig. 5, we can see NEEL outperforms the NFA implementation of the straightforward algorithm in this paper due to their optimization over a nested pattern match, while with optimization method, our algorithm outperforms NEEL as we delete partial matches in a batch manner.

Figure 6 is the memory comparison of different methods. As we can see, the straightforward implementation algorithm of this paper uses less memory than other methods. NFA + B-tree utilizes the most memory because it needs a B-tree to store all events and the stack data structure to store relationship between different event types. But considering time is critical in monitoring applications, we think it is acceptable to use more memory to get a quick response.

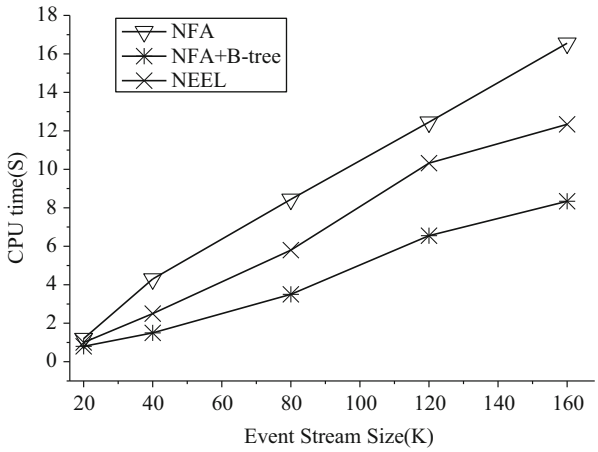

Fig. 5 CPU time consumption of three algorithms of Query1 (Window 30 mins, SEQ query length 4, imbedded SEQ length 4)

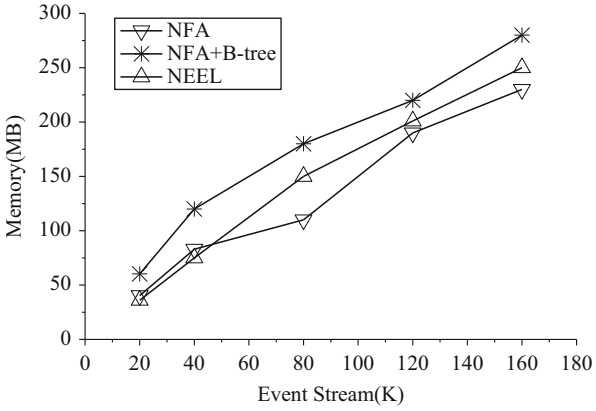

Fig. 6 Memory consumption of three algorithms of Query1 (Window 30 mins, SEQ query length 4, imbedded SEQ length 4)

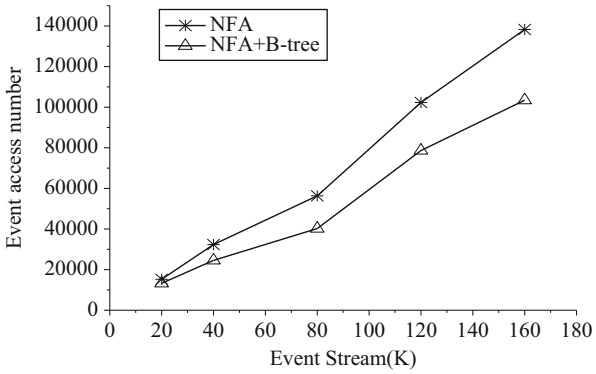

Fig. 7 Event access number of two algorithms of Query1 (Window 30 mins, SEQ query length 4, imbedded SEQ length 4)

We test the event access number of the NFA-based methods to see the efficiency of searching different data structures. The result is shown in Fig. 7.

From Fig. 7, we can see that B-tree-based method utilizes few searches than the straight method because the batch operation reduces some event access in the memory which would also reduce partial match management cost in a way.

6 Conclusion

In this paper, we present event detection method over RFID-enabled healthcare data streams with NFA-based model. To enhance fast partial matches deletion and reduce event access numbers, we utilize ordered B-tree to store the events. Experimental

results show that while our optimization method consumes more memory, it can provide fast response and utilizes few searches over streams. In the future, we would like to optimize the query execution plan over multiple queries.

Acknowledgments This work has been supported by funds from Chengdu University of Information Technology, China (J201410), the Applied Basic Research Key Project of Sichuan Province (2017JY0011), and the China Scholarship Council.

References

1. Yao W, Chu C-H, Li Z (2011) Leveraging complex event processing for smart hospitals using RFID. J Netw Comput Appl 34(3):799–810
2. Luckham D (2002) The power of events, vol 204. Addison-Wesley, Reading
3. Wu E, Diao Y, Rizvi S (2006) High-performance complex event processing over streams. In: Proceedings of the 2006 ACM SIGMOD international conference on management of data. ACM
4. Cugola G, Margara A (2010) TESLA: a formally defined event specification language. In: Proceedings of the fourth ACM international conference on distributed event-based systems. ACM
5. Gyllstrom D et al (2008) On supporting kleene closure over event streams. In: Data engineering, 2008. ICDE 2008. IEEE 24th international conference on. IEEE
6. Wamba SF (2012) RFID-enabled healthcare applications, issues and benefits: an archival analysis (1997–2011). J Med Syst 36(6):3393–3398
7. Tu Y-J, Zhou W, Piramuthu S (2009) Identifying RFID-embedded objects in pervasive healthcare applications. Decis Support Syst 46(2):586–593
8. Liu M et al (2011) High-performance nested CEP query processing over event streams. In: Data engineering (ICDE), 2011 IEEE 27th international conference on. IEEE
9. Liu M et al (2010) NEEL: the nested complex event language for real-time event analytics. In: International workshop on business intelligence for the real-time Enterprise. Springer, Berlin/Heidelberg
10. The ORLocate Solution. http://www.haldor-tech.com/products/the-orlocate-solution/
11. Demers AJ et al (2007) Cayuga: a general purpose event monitoring system. CIDR 7:412–422
12. Mei Y, Madden S (2009) Zstream: a cost-based query processor for adaptively detecting composite events. In: Proceedings of the 2009 ACM SIGMOD international conference on management of data. ACM
13. Gatziu S, Dittrich KR (1994) Detecting composite events in active database systems using petri nets. In: Research issues in data engineering, 1994. Active database systems. Proceedings fourth international workshop on. IEEE
14. Teymourian K, Paschke A (2009) Semantic rule-based complex event processing. In: International workshop on rules and rule markup languages for the semantic web. Springer, Berlin/Heidelberg