Complex Event Processing over Unreliable RFID Data Streams*

Yanming Nie, Zhanhuai Li, and Qun Chen

School of Computer, Northwestern Polytechnical University,
Xi'an 710072, China
`yanming@mail.nwpu.edu.cn`, {`lizhh,chenbenben`}`@nwpu.edu.cn`
`http://www.nwpu.edu.cn/jsj`

Abstract. Existing RFID complex event processing (CEP) techniques always assume that raw RIFD data has been first cleansed to filter out all unreliable readings upfront. But this may cause delayed triggering of matched complex events. Furthermore, since the cleansed event streams need to be temporarily buffered for CEP evaluation, it may generate a large number of intermediate results. To address these issues, we propose an approach to perform CEP directly over unreliable RFID event streams by incorporating cleansing requirements into complex event specifications, and then employ a non-deterministic finite automata (NFA) framework to evaluate the transformed complex events. Experimental results show that our approach is effective and efficient.

Keywords: Unreliable RFID data streams, CEP, NFA.

1 Introduction

CEP can correlate individual RFID readings and transform them into semantic-rich complex events, and therefore plays a key role in monitoring applications. For example, shoplifting in a retail store can be detected by processing a complex event in such scenario: an item has been picked up at a shelf and then taken out of the store without being first checked out in a specific time window.

Raw RFID data is inherently incomplete, noisy, and need to be cleansed. We classify the inaccuracy of RFID data into two categories: **unreliability** and **uncertainty**. Here, unreliability refers to the erroneous readings that can be corrected by deterministic cleansing rules. In a retail store, for instance, if an item has been picked up from a shelf and checked out later on, any other shelf readings for this item in-between should be false positives. Instead uncertainty refers to the inconsistent RFID readings which cannot be determinately eliminated by cleansing rules due to their ambiguities. An instance of uncertainty occurs when the two different shelf readers detect an item simultaneously. So either of the two readings can be false positive, but available RFID data cannot arbitrate. In

* A preliminary version of this work appeared as a 4-page paper at Application of Research of Computers, Vol 26 No 7, 2009 [15].

this paper, we focus only on RFID event stream containing unreliable readings, and leave the issues of CEP over uncertain RFID streams for future work.

The straightforward solution to CEP over unreliable RFID event stream is to filter out unreliable readings upfront, and then execute CEP. But this may cause CEP-enabled systems delaying to trigger corresponding response. Moreover, since cleansed events are needed to be temporarily recorded, it may generate large number of intermediate results. In this paper, we proposed an approach to incorporate cleansing requirements into CEP. Our approach is first to convert unreliability of RFID readings into its corresponding reliability constraints in complex event specifications, and then directly evaluate the transformed complex events over unreliable event streams. The main contributions are as following:

– We present a declarative **C**leansing **L**anguage for **U**nreliable RFID **E**vent **S**treams (**CLUES**), with which cleansing actions can be implicitly set without using a specific action clause.
– Based on CLUES, we propose an automated mechanism to enable evaluating complex event directly over unreliable RFID streams by implanting reliability check constraints into complex event specification.
– By extending the existing NFA implementation frameworks, we propose two approaches to evaluate complex events with reliability check constraints, i.e. a primitive approach and an advanced approach.

2 Related Work

Due to value-based constraints and sliding windows in RFID complex event specification, traditional evaluation frameworks [1][3] is no longer applicable. And their employed fixed data structures, such as tree [3], directed graph [1], finite automata [2], or Petri net [4], cannot adapt to necessary extensions required by RFID complex event specifications. To address these issues and to optimize CEP over huge-volume RFID data, Eugene et al [5] proposed a declarative specification language **SASE** and an NFA-based evaluation framework. Unfortunately, none of them can handle unreliable readings existed in RFID event streams.

RFID data is inherently incomplete, noisy, and need to be cleansed before being forwarded. **SMURF** [7] aims to capture the accurate time window of tag existence by viewing RFID stream as a statistical sample of the tags in physical world, and filters RFID data at the low level of edge device. In practice, however, RFID readings usually require to be analyzed and cleansed in a bigger context of business flow, so cleansing thus need to be executed probably within a large sliding window. A deferred approach was proposed in [8] for detecting RFID data anomalies by defining cleansing rules with **SQL-TS** and performing application-specific cleansing at query time. In [6], Wang et al provided with example rules for data filtering and cleansing. But these off-line and RDBMS-based solutions cannot be applied to RFID CEP. Other related work [11] [12] tried to perform interpretation and imputation over uncertain RFID data by fully exploiting the temporal and spatial relationships among the readings, but neither of them addressed the issues of RFID CEP.

There exist some work [14] [13] on event queries and evaluation over uncertain RFID data streams. A temporal model and some evaluation frameworks with corresponding optimizations were proposed in [13] to recognize patterns and perform CEP over the streams with imprecise timestamps. Event pattern detection and query evaluation techniques over correlated probabilistic streams was studied in [14]. As far as we know, there are no any related studies on CEP incorporate with data cleansing over unreliable RFID event streams yet.

3 Preliminaries

RFID data can be seen as a sequence of RFID tuples with the form of *(Oid, Rid, RTime)*, where *Oid* is the ID of an object, *Rid* is the ID of a reader, *Rtime* is the occurrence time of the reading. CEP is used to correlate individual RFID readings in event stream to form semantically meaningful complex events appropriate for end applications. Shoplifting monitoring in a retail store, for instance, requires to detect a sequence of occurrence or non-occurrence of events having same *Oid* within a specific time window. As an RFID complex event specification language, SASE [5] is declarative and has the overall structure as:

EVENT <event pattern>
[WHERE <qualification>]
[WITHIN <window>]

The *EVENT* clause describe a sequence pattern, and its components are occurrence or non-occurrence of component events in a temporal order. The *WHERE* clause specifies constraints on those events. The *WITHIN* clause specifies the sliding window for the whole sequence of events. For example, the complex event corresponding to shoplifting in a retail store can be specified as Q_1:

EVENT SEQ(SHELF x, !(COUNTER y), EXIT z)
WHERE x.Oid=y.Oid=z.Oid
WITHIN a hour

In Q_1, *SEQ* denotes sequence pattern. *SHELF*, *COUNTER* and *EXIT* are different event types. The sign ! denotes non-occurrence of an event (also called as a negation event).

The NFA approach of SASE [5] can flexibly implement attribute value comparisons between events. It creates an NFA for each query, uses instance stacks to record the events in different states, and partition the entries based on the attribute values to facilitate comparisons among events.

As for an RFID complex event, the value of *Oid* has a large domain, so the partitioning strategy is no longer applicable. Correlating the events with the same *Oid* thus requires to involve search operation on *Oid*. An example of NFA evaluation on a complex event with the same *Oid* is presented in Fig. 1. We denote an event type with a capital letter, a primitive event with a lowercase letter and a subscript (where the letter and subscript indicate event type and object ID, respectively). For instance, a_i is an event of type A and its *Oid* is i.

RFID readings usually contain false positives and false negatives. Directly applying the NFA approach to CEP over unreliable RFID event streams may produce incorrect results. We will detail our solutions in the next section.

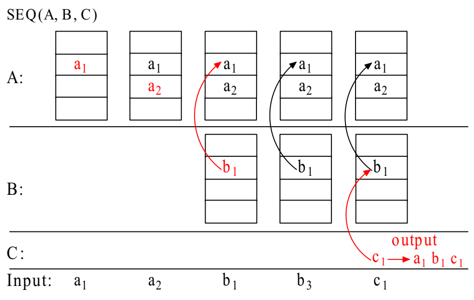


Fig. 1. An example of NFA evaluation

4 CLUES Cleansing Language

The unreliabilities of RFID event stream can be specified by cleansing rules. An effective specification language of cleansing rule is supposed to be declarative, sequence-based and easy-specifying. Unlike SQL-TS [8] which specifies cleansing rules in relational tables, CLUES is for RFID event streams. Its format is as:

IF <complex event specification>
THEN FORALL <Oid specification>
EXIST|NOT EXIST <event type>
WITHIN (BEFORE|AFTER) <window>

where the *IF* clause specifies the complex event. The *FORALL* clause specifies the objects may have unreliable readings. <*Oid specification*> can be a boolean combination of predicates with using comparison operators or containment operator (i.e. $\subset$). The keyword *EXIST* indicates that the reading may be false negative, while *NOT EXIST* false positive. The *WITHIN* clause specifies the temporal constraints for the specified complex event. The keywords *BEFORE* and *AFTER* are used to specify if the unreliable reading occurs before or after the complex event. The <*window*> specifies the size of valid time window between unreliable reading and the complex event. With the *BEFORE*, the time window refers to the occurrence time gap between unreliable reading and the first component event of the complex event. With the *AFTER*, the time window refers to the occurrence time gap between unreliable reading and the last component event. Also the <*window*> alone can be set as an absolute time interval.

The CLUES cleansing language is declarative, so we can construct event sequences and specify common RFID cleansing rules in a straightforward way. Note that there is no action clause defined in CLUES language, cleansing action has in fact been implicitly specified by the *EXIST* or *NOT EXIST* keywords. *EXIST* implies that the reading may be missing and thus need to be recovered if absented. In contrast, *NOT EXIST* implies that the reading may be a false positive, thus has to be dropped if presented. The *EXIST|NOT EXIST* clause specifies the inclusive or exclusive relationship between an unreliable reading and its triggering complex event. Various cleansing rules can thus be composed from the clauses of *EXIST*|*NOT EXIST*, *FORALL* and *WITHIN*, such as:

Example 1: Exclusive Rule. Consider an item is transported to location A (event type A). Suppose the transportation time is at least 1 minute and the items may be accidentally read by location B (event type B) nearby. We use the following cleansing rule R_e to remove all B readings before an A reading:

```
IF $a_1$
THEN FORALL (Oid = $a_1$.Oid)
NOT EXIST B
WITHIN BEFORE <1 minute>
```

According to R_e, any B event followed by an A event on the same object within one minute is supposed to be a false positive.

Example 2: Inclusive Rule. Suppose that an item detected at location A is always moved to location B within 2 minutes due to business flow. We use the following cleansing rule R_i to compensate some probably missed B readings:

```
IF $a_1$
THEN FORALL (Oid = $a_1$.Oid)
EXIST B
WITHIN AFTER <2 minutes>
```

According to R_i, any A event should be followed by a B event on the same object within two minutes.

Example 3: Cross Reading Rule. An item stays at location A, and may be accidentally read by location B. We use the following cleansing rule R_c to detect this type of cross readings:

```
IF ($a_1$.$a_2$
    WHERE $a_1$.Oid = $a_2$.Oid
    WITHIN 3 minutes)
THEN FORALL (Oid = $a_1$.Oid)
NOT EXIST B
WITHIN [$0 <$ $a_2$.Rtime - $a_1$.Rtime $\leq$ 3 minutes]
```

According to R_c, if two A events on the same object occur within 3 minutes, any B event on that object in-between is supposed to be a false positive.

Example 4: Packaging Rule. Consider a pallet and its contained cases is being moved together along a certain business path. Suppose that pallet tags are always more readable, but due to material interfering and tag orientation, case tags may fail to be detected. We use the following cleansing rule R_p to recover missed A readings for the cases:

```
IF ($a_1$ WHERE <$a_1$.Oid indicates a pallet>)
THEN FORALL (Oid $\subset$ $a_1$.Oid)
EXIST A
WITHIN [$a_1$.Rtime - 5 seconds, $a_1$.Rtime + 5 seconds]
```

Here $Oid \subset a_1.Oid$ denotes that the object (case) with unreliable reading is contained in a_1 (pallet). According to R_p, as a pallet reading occurs, the readings for the contained cases should be detected before or after it within 5 seconds.

5 Complex Event Evaluation

With CLUES cleansing rules, CEP over unreliable RFID event streams can be solved through imposing reliability constraints on component events and then evaluating the transformed complex event.

5.1 Complex Event Transformation

Suppose that the complex event specified in the *IF* clause is denoted by P_c. For false positive, we can use *[!*P_c*]* to represent its reliability constrain (! denotes non-occurrence of P_c), and the reliability of a reading depends on non-occurrence of P_c. Similarly, for false negative, we can use *[|*P_c*]* (| denotes an option), a missed reading should be recovered by verifying occurrence of P_c. Semantically, both predicates *[!*P_c*]* and *[|*P_c*]* impose additional constraints on events.

Suppose that the event type of false positive readings specified in a CLUES cleansing rule is A. Its reliability is formally specified as:

A[!(P_c *WHERE <Oid qualification> and <Rtime qualification>)]*

where P_c is the complex event specified in the *IF* clause of cleansing rule. *<Oid qualification>*, derived from *<Oid specification>* in the *FORALL* clause, specifies additional qualifications on the *Oid* of component events in P_c. *<Rtime qualification>*, derived from the *WITHIN* clause, specifies additional constraints on the *Rtime* of component events in P_c.

Generally, P_c consists of multiple component events, and may has its own attribute value comparisons and time window specifications. The reliability constraint shares the same event sequence pattern as P_c, but its *WHERE* clause has to integrate the Oid and Rtime qualifications specified in cleansing rule with those of P_c using $\wedge$. Consider the cleansing rule R_c in Section 4, a reading of type B should be imposed with a reliability constraint as:

B[!($A_1.A_2$ *WHERE (*A_1*.Oid =* A_2*.Oid) and*
  (B.Oid = A_1*.Oid) and*
  (A_1*.Rtime* $\leq$ *B.Rtime* $\leq$ A_2*.Rtime)*
  WITHIN 3 minutes)]

Note that the constraints *(B.Oid =* A_1*.Oid)* and *(*A_1*.Rtime* $\leq$ *B.Rtime* $\leq$ A_2*.Rtime)* are derived from the cleansing rule.

The unreliability constraint of false negative readings of type A is as:

A[|(P_c *WHERE <Oid qualification> and <Rtime qualification>)]*

Since a false negative reading is actually absented from event stream, *<Oid qualification>* and *<Rtime qualification>* do not impose additional qualifications on P_c. Instead, they delimit the probable *Oid* and *Rtime* values of the missed readings. Consider the cleansing rule R_i presented in Section 4, B events should be imposed with a reliability constraint as:

B[|(A WHERE (A.Oid = B.Oid) and
  (0 $\leq$ *B.Rtime - A.Rtime* $\leq$ *2 minutes))]*

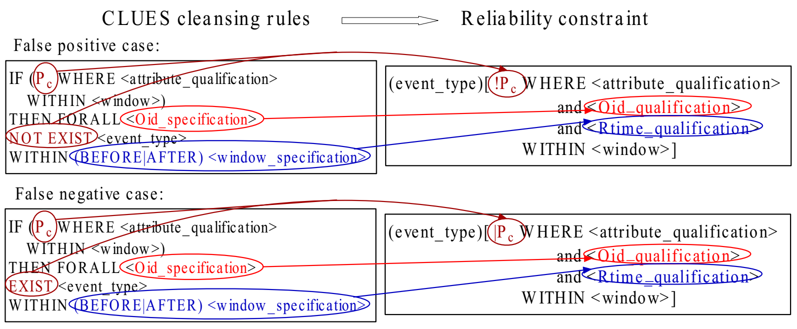

CLUES cleansing rules
Reliability constraint
False positive case:
IF (P_c WHERE <attribute_qualification> WITHIN <window>)
THEN FORALL <Oid_specification>
NOT EXIST <event_type>
WITHIN (BEFORE|AFTER) <window_specification>
(event_type)[!P_c WHERE <attribute_qualification> and <Oid_qualification> and <Rtime_qualification> WITHIN <window>]
False negative case:
IF (P_c WHERE <attribute_qualification> WITHIN <window>)
THEN FORALL <Oid_specification>
EXIST <event_type>
WITHIN (BEFORE|AFTER) <window_specification>
(event_type)[|P_c WHERE <attribute_qualification> and <Oid_qualification> and <Rtime_qualification> WITHIN <window>]

Fig. 2. Transforming CLUES rules into reliability constraints

Fig. 2 shows how to transform CLUES rules into reliability constraints. In false positive case, incorporating reliability constraint into complex event specification is straightforward as the event with constraints will be substantiated during evaluation. In false negative case, reliability constraints is to detect a missing event and specify the value of *Oid* and *Rtime*. The *Oid* of missing event may be $=$ or $\subset$ the *Oid* of P_c. But the occurrence time of a false negative reading depends on the occurrence time of its corresponding complex event specified in reliability constraint, as well as business flow. In this paper, we assume that a false negative may occur within the inferred time interval.

5.2 NFA-Based Evaluation Approaches

We use NFA-based evaluation framework for RFID complex event with reliability constraints. **The primitive approach** is to evaluate the complex event first, and then validate component events case-by-case using reliability constraints.

For false positive, the reliability constraint of a B reading is denoted as $B[!P_c]$. To check reliability constraint, the primitive approach has to record the instances of P_c. While validating a B event, it checks if it has a corresponding P_c instance by searching on *Oid* and *Rtime*. Note that if the *WITHIN* clause of cleansing rule has the keyword *BEFORE* or *AFTER*, the reliability constraint can be transformed into $B.!P_c$ or $!P_c.B$, respectively. For false negative, the reliability constraint is denoted as $B[|P_c]$. Since B event may be missing, the primitive approach has to detect all complex event instances with or without B event first. Then for each partially matched complex events, checks if there is a P_c instance which implies a corresponding B event occurred before.

The primitive approach produces all matching complex event instances without reliability constraints and record the instances of P_c specified in constraints. Next, we will present **an advanced approach** which interleaves reliability validation with CEP and evaluates reliability constraints eagerly, thus effectively reduce intermediate result size during evaluation.

For false positive $B[!P_c]$, the advanced approach detects the instances of P_c and uses them to filter out unreliable B events. It evaluates CEP and P_c

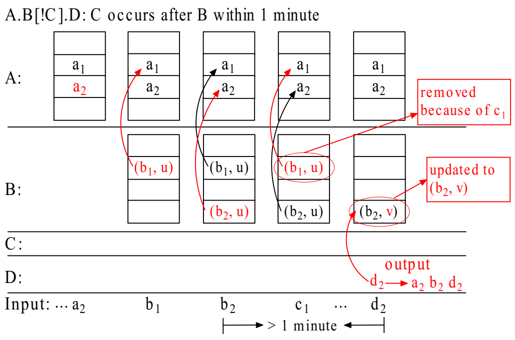


Fig. 3. A NFA running on complex event with false positive reliability check

simultaneously. Consider the case that B event is specified to occur before or within the time window of P_c in reliability constraint. The corresponding B events are considered as unreliable and removed from NFA as a P_c instance is found. A B event in NFA is considered as reliable only after it is out of the time window of $B[!P_c]$. So the P_c instances do not need to be recorded. For the case that B event is specified to occur after the time window of P_c, as a P_c instance is detected, its corresponding B event is unavailable, so the P_c instances have to be temporarily stored. The B event will be checked immediately against the existing P_c instances once available. A P_c instance can be dropped only after it is out of the sliding window of $B[!P_c]$. Each event in NFA is marked as *valid* or *uncertain*, and will be changed from *uncertain* to *valid* if it is out of the sliding window of $B[!P_c]$. It will immediately be removed once marked as *invalid*.

An evaluation example of $B[!P_c]$ is shown in Fig. 3. It supposes that the occurrence time gap between d_2 and b_2 is > 1 minute. (b_2, u) and (b_2, v) denotes b_2 to be *uncertain* and *valid*, respectively. Once d_2 is encountered, b_2 has fallen out of the sliding window of $B[!C]$, is thus considered as *valid*.

The advanced approach processes false negative $B[|P_c]$ in a similar way. Whenever a P_c instance is detected, its corresponding B event, whose probable occurrence time is specified as a time interval, should be inserted into NFA if not exists. In case that B event is specified to occur before or within the time window of P_c, it will be clear whether the corresponding B event is missing once the P_c is detected. For the case that the B event is to occur after the time window of P_c, even though the corresponding B event of P_c does not exist in NFA by the time P_c is detected, it may occur later. The supposed B event inferred from P_c should be replaced with this real one as it occurs and be inserted into NFA.

Suppose that $B[|P_c]$ is followed by D, and B event is specified to occur before or within the time window of P_c. When a D event occurs and its corresponding B event not existed in NFA, an imaginary B event will be generated and inserted. The imaginary B event is initially marked as *uncertain*, and updated to *valid* as its corresponding P_c is detected. An example of evaluating $B[|P_c]$ is shown in Fig. 4, where the corresponding b_2 event of c_2 is missed in the event stream.

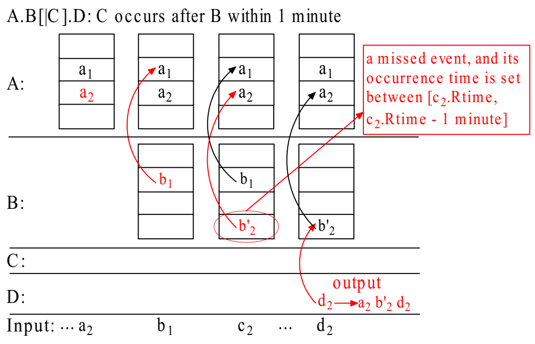

A.B[|C].D: C occurs after B within 1 minute
A:
B:
C:
D:
a1
a2
b1
b'2
a missed event, and its occurrence time is set between [c2.Rtime, c2.Rtime - 1 minute]
output
d2→a2 b'2 d2
Input: ··· a2 b1 c2 ··· d2

Fig. 4. A NFA running on complex event with false negative reliability check

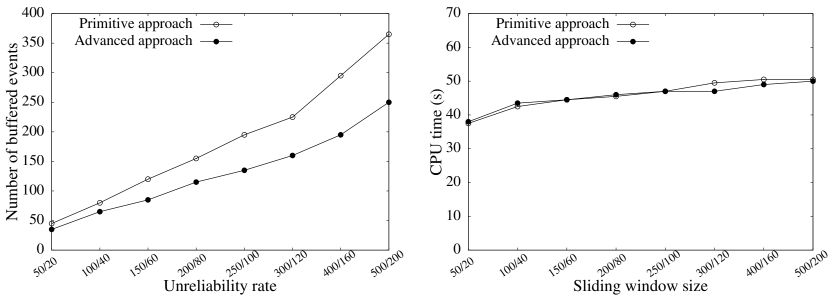
6 Experimental Study

We conduct a preliminary experimental study to validate our evaluation approaches on two metrics: memory usage and processing time. Used memeory is measured by the total number of buffered event instances. We first generated synthetic RFID data by emulating tags moving along a path of $A \rightarrow B \rightarrow D$. The complex event to be evaluated is $A.B.D$, and its components have same *Oid*. The B events are supposed to be unreliable, and can be determined by reliability constraints involving C events. We consider both false positive reliability constraint (i.e. *B[!C]*) and false negative one (i.e. *B[|C]*), where C event is specified to be after B event within a time window. All the tests is running on a Windows machine with Intel Core 2 Dual 2GHz CPU and 2GB RAM.

6.1 Advanced Approach vs. Primitive Approach

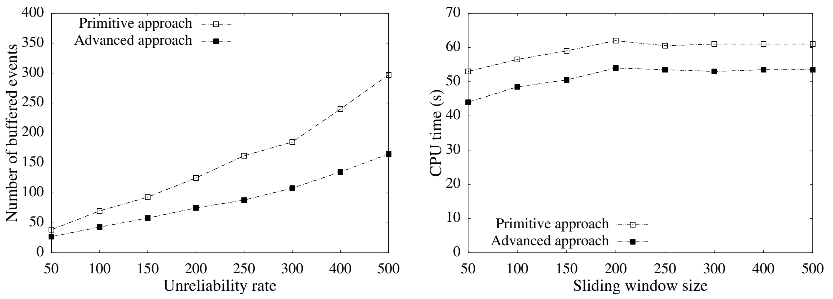
This test is to compare the performance between the advanced approach and the primitive one. Among the generated data, there are 50 thousand events can match $A.B.D$ with varying sliding window from 50 to 500, 10 thousand inaccurate $A.B.D$ whose B readings are followed by C readings with varying time window of *B[!C]* from 20 to 200 for false positive case, 10 thousand $A.D$ whose B events in-between are falsely missed but can be recovered if with the corresponding C events for false negative case, and some other noisy readings.

Fig. 5 shows the results for *A.B[!C].D*, where *50/20* on the X-axis means that the sliding window of $A.B.D$ is 50 while the time window of *B[!C]* is 20. The advanced approach is better on memory usage with the margins larger than 30% as sliding window > 300, as shown in Fig. 5(a). That's because the primitive approach has to record C readings and unreliable B readings, and the total number of these readings will grow accordingly as the sliding window increases. Even though the two approaches process C readings differently, they execute roughly the same number of event correlations and thus take nearly the same CPU time, as shown in Fig. 5(b).



(a) Memory usage (b) Processing time

Fig. 5. Advanced vs primitive on *A.B[!C].D*



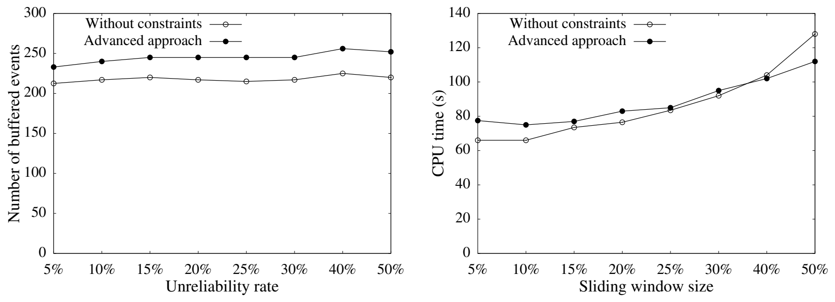
(a) Memory usage (b) Processing time

Fig. 6. Advanced vs primitive on *A.B[|C].D*

Fig. 6 shows the results for *A.B[|C].D*. The primitive approach records all C events and complex events of $A.D$, while the advanced one only buffers the C events with preceding A events and D events preceded by either B or C events. So the advanced approach uses less memory, as shown in Fig. 6(a). The primitive approach produces many inaccurate complex events of $A.D$ which need to be checked against C events. While the advanced one processes C events eagerly and does not need such checking. We can see that the advanced approach obviously outperforms the primitive one on CPU time, as shown in Fig. 6(b).

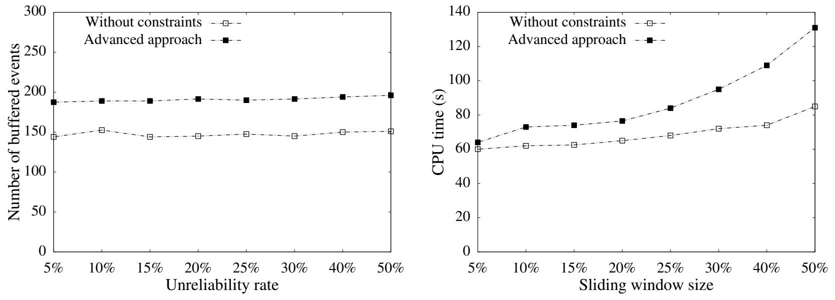
6.2 Advanced Approach vs. Traditional NFA Approach

In this test, we study the performance differences between the advanced approach and the traditional NFA approach without reliability constraints. The test RFID data includes 100 thousand accurate complex events of $A.B.D$ and other complex events involving false B events. Unreliability rate for an event stream is defined to be the ratio of the number of complex events with false B events to the total number of matching complex events. We varies unreliability rate from 5% to 50%. Instead, the traditional approach assums that each B event is reliable.



(a) Memory usage (b) Processing time

Fig. 7. Unreliable vs reliable in false positive case



(a) Memory usage (b) Processing time

Fig. 8. Unreliable vs reliable in false negative case

Fig. 7 shows the results for the false positive case. The advanced approach does not records C events, but need to cache the constructed instances of $A.B.D$ until the B events is out of the time window of *B[!C]*. The advanced approach performs worse by around 15%, as shown in Fig. 7(a). Note that the used memory of the advanced approach does not grow with the increasing unreliability rates. That is because even though the total number of false positive B events becomes larger as unreliability rates increase, they are distributed in a larger event stream. Therefore higher unreliability rates do not necessarily mean more memory usage given a fixed sliding window. The traditional approach does not process C events, but constructs more complex events of $A.B.D$. So the differences of CPU time between the two are negligible, as shown in Fig. 7(b).

Fig. 8 shows the results for the false negative case. As the advanced approach records the C events with corresponding A events, it performs worse on memory usage, as shown in Fig. 8(a). The extra is proportional to the number of complex events of $A.C$ within the sliding window of $A.B.D$. Because the advanced approach processes more C events, it performs worse on CPU time, as shown in Fig. 8(b). As the number of C events increase with the increasing unreliability rates, the degradation is roughly linear to the increase of unreliability rates.

7 Conclusion

In this paper, we study the issues of CEP over unreliable RFID event streams. First, we present CLUES, an RFID cleansing language, to define unreliability of RFID readings, and then transform reading unreliability into reliability constraint in complex event specification. Thus CEP can be directly performed over unreliable RFID event streams. Finally, we extend the current NFA evaluation framework to facilitate RFID CEP with reliability constraints. The preliminary experimental results show that our approach is effective and efficient.

There still exist such possible extensions for the CLUSE as: 1) The *IF* clause can be extended to accommodate boolean combination of multiple complex events, and 2) The *EXIST|NOT EXIST* clause can be enhanced to allow specifying one or even multiple complex events. Also we plan to investigate the issue of CEP over RFID event streams with both unreliable and uncertain readings.

Acknowledgments. This work is supported by the NSFC under Grant No. 60720106001 and No. 60803043, and the National National High Technology Development 863 Program of China under Grant No. 2009AA1Z134.

References

1. Hinze, A.: Efficient filtering of composite events. In: BNCOD, pp. 207–225 (2003)
2. Gehani, N.H., Jagadish, H.V., Shmueli, O.: Composite event specification in active databases: Model and implementation. In: VLDB, pp. 327–338 (1992)
3. Chakravarthy, S., Krishnaprasad, V., Anwar, E., Kim, S.: Composite events for active databases: Semantics, contexts and detection. In: VLDB, pp. 606–617 (1994)
4. Gatziu, S., Dittrich, K.R.: Events in an active object-oriented database system. In: RIDS, pp. 23–39 (1993)
5. Wu, E., Diao, Y., Rizvi, S.: High-performance complex event processing over streams. In: SIGMOD, pp. 407–418 (2006)
6. Wang, F., Liu, P.: Temporal management of RFID data. In: VLDB, pp. 1128–1139 (2005)
7. Jeffery, S.R., Garofalakis, M.N., Franklin, M.J.: Adaptive cleaning for RFID data streams. In: VLDB, pp. 163–174 (2006)
8. Rao, J., Doraiswamy, S., Thakkar, H., et al.: A deferred cleansing method for RFID data analytics. In: VLDB, pp. 175–186 (2006)
9. Chen, Q., Li, Z., Liu, H.: Optimizing Complex Event Processing over RFID Data Streams. In: ICDE, pp. 1442–1444 (2008)
10. Agrawal, J., Diao, Y., Gyllstrom, D., Immerman, N.: Efficient pattern matching over event streams. In: SIGMOD, pp. 147–160 (2008)
11. Cocci, R., Tran, T., et al.: Efficient Data Interpretation and Compression over RFID Streams. In: ICDE, pp. 1445–1447 (2008)
12. Gu, Y., Yu, G., et al.: Efficient RFID data imputation by analyzing the correlations of monitored objects. In: Zhou, X., Yokota, H., Deng, K., Liu, Q., et al. (eds.) DASFAA 2009. LNCS, vol. 5463, pp. 186–200. Springer, Heidelberg (2009)
13. Zhang, H., Diao, Y., Immerman, N.: Recognizing Patterns in Streams with Imprecise Timestamps. In: VLDB (2010)
14. Re, C., Letchner, J., et al.: Event queries on correlated probabilistic streams. In: SIGMOD, pp. 715–728 (2008)
15. Chen, Y., Li, Z., Chen, Q.: Complex event processing over unreliable RFID data streams. Application Research of Computers 26(7), 2538–2539, 2542 (2009)