ARTICLE IN PRESS

Pervasive and Mobile Computing ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

Out-of-order durable event processing in integrated wireless networks☆

Chunjie Zhou [a,*], Xiaofeng Meng [a], Yueguo Chen [b]

[a] School of Information, Renmin University of China, Beijing, China

[b] Key Laboratory of Data Engineering and Knowledge Engineering, Renmin University of China, Beijing, China

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
Durable events
Complex events
Wireless networks
Out-of-order

ABSTRACT

Many events in real world applications are long-lasting events which have certain durations. The temporal relationships among those durable events are often complex. Processing such complex events has become increasingly important in applications of wireless networks. An important issue of complex event processing is to extract patterns from event streams to support decision making in real-time. However, network latencies and machine failures in wireless networks may cause events to be out-of-order. In this work, we analyze the preliminaries of event temporal semantics. A tree-plan model of out-of-order durable events is proposed. A hybrid solution is correspondingly introduced. Extensive experimental studies demonstrate the efficiency of our approach.

1. Introduction

In recent years, the demand for all types of wireless services, as well as the increasing number of users, have led to wireless networks being widely used in various areas [1–4]. Meanwhile, a wide range of wireless network applications nowadays rely on the ability of query processing over event streams [2,5–7]. These event streams are sequences of durable (interval-based) events that are temporally ordered or out-of-order. The temporal relationships among atomic events are very important in identifying event patterns in integrated wireless networks. However, the existing literature on time management of events in integrated wireless networks focuses on either ordered durable events [8–11] or out-of-order instant events [1,12–14]. It is very important to support out-of-order durable event pattern detection in integrated wireless networks. To the best of our knowledge, no prior work on out-of-order events considers the time intervals of events. For pattern queries of durable events, the state transition depends on not only the type of events, but also the relationships and the predicates among events. However, due to NFA (Non-deterministic Finite Automata) explicitly ordering state transitions, the prior NFA-based methods are not straightforward to efficiently model negation and parallel events, which are common in durable events. In this paper, we propose to combine techniques on addressing the detection of durable events and out-of-order events.

Existing research works [5,6] almost focus on instant events, i.e., events with no duration. However, purely sequential queries on instant events are not enough to express many event patterns in the real world. For example, it has been observed that for many diabetic patients, the presence of hyperglycemia often overlaps with the absence of glycosuria [11,15]. This insight has led to the development of effective diabetic testing kits. There is a need for an efficient algorithm that can solve durable events. To model the relationships of events based on their time intervals, we propose two types of event query pattern, the sequential query pattern and the parallel query pattern.

Real-time processing of event streams generated from distributed devices is a primary challenge for today's monitoring and tracking applications. Most systems [11,16], either event-based or stream-based, assume events satisfy totally ordered

☆ This research was partially supported by the grants from the Natural Science Foundation of China (No. 60833005); the National High-Tech Research and Development Plan of China (No. 2009AA011904); and the Doctoral Fund of Ministry of Education of China (No. 200800020002).

* Corresponding author. Fax: +86 10 62512719.
E-mail address: lucyzcj@ruc.edu.cn (C.J. Zhou).

1574-1192/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.pmcj.2010.11.010

Please cite this article in press as: C.J. Zhou et al., Out-of-order durable event processing in integrated wireless networks, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.11.010

arrivals. However, in pervasive computing environments, event streams might be out-of-order at the processing engine due to machine or network failures. It has been illustrated that the existing technologies are likely to fail in such circumstances because of false missing or false positive matches of event patterns. Let us take an application for tracking books in a bookstore [7] as an example. In this application, RFID tags are attached to each book and RFID readers are placed at some strategic locations (e.g., book shelves, checkout counters and the exit) throughout the store. If two readers at a book shelf and an exit sensed the same book, but none of the checkout counters sensed the book in between the occurrence of the first two readings, then it is likely that the book is being shoplifted. If events of readings at the checkout counters arrive out-of-order, we may miss the desired results or generate false alarms for event monitoring. The correctness of event processing therefore cannot be guaranteed. Obviously, it is imperative to deal with both in-order as well as out-of-order event arrivals efficiently and in real-time.

In addition, most of the recently proposed complex event processing systems use NFA to detect event patterns [7,13,15]. However, the NFA-based approaches have three limitations [2]: (1) current NFA-based approaches impose a fixed evaluation order determined by their state transition diagram; (2) it is not straightforward to efficiently model negation in an NFA; (3) it is hard to support parallel events because NFA's explicitly order state transition. In this paper, we use tree-based query plans for both logical and physical representations of query patterns.

The main contributions of this work include:

1. We define the notations of temporal semantics and introduce two types of query patterns based on time intervals: sequential query pattern and parallel query pattern.
2. We propose a model of out-of-order durable events that includes the logical and physical expression, as well as the detection method of these events.
3. We use a tree-based query plan structure for complex event processing that is amenable to a variety of algebraic optimizations.
4. We develop a hybrid solution to solve out-of-order durable event processing that can switch from one level of output accuracy to another in real time.

The rest of this paper is organized as follows. Section 2 gives the related works. Section 3 provides the research background and some preliminaries. Section 4 introduces the model of out-of-order durable events and the detection method. Section 5 describes a hybrid solution and the optimization strategy. Section 6 gives the experiment results and we conclude in Section 7.

2. Related works

There has been some work on investigating the problems of out-of-order events and durable events respectively. However, to the best of our knowledge, there is no work considering the two aspects together, which is important in integrated wireless networks.

Studies on the problem of out-of-order events can be divided into two categories: one focuses on real time, whose output is unordered; the other pays more attention to the accuracy, whose output is ordered. Because the input event stream to the query processing engine is unordered, it is reasonable to produce unordered output events. In [13], the authors permit outputting unordered sequences and propose an aggressive strategy. The aggressive strategy produces maximal output under the assumption that out-of-order events are rare. In contrast, to tackle the unexpected occurrence of an out-of-order event, appropriate error compensation methods are designed for the aggressive strategy.

If ordered output is required, additional semantic information such as K-Slack factor [1,12] or punctuation [13] is required to "unblock" the on-hold candidate sequences from being output. The introduction of the two techniques is as follows. A naive approach [1,12] on handling out-of-order event streams is to use K-Slack as an a priori bound on the out-of-order input streams. It buffers incoming events in the input queue for K time units until the order can be guaranteed. The biggest drawback of K-Slack is the rigidity of K, which cannot adapt to the variance in the network latencies that exist in a heterogeneous RFID reader network. For example, one reasonable setting of K may be the maximum of the average latencies in the network. However, as the average latency changes, K may become either too large (thereby buffering unneeded data and introducing unnecessary inefficiencies and delays), or too small (thereby becoming inadequate for handling the out-of-order arriving events and resulting in inaccurate results). It also requires additional space and introduces more latency before evaluating events.

Another solution proposed to handle out-of-order data arrivals is to apply punctuations. This technique assumes assertions are inserted directly in the data stream in order to confirm that a certain value or time stamp will no longer appear in the future input streams [13,17]. The authors in [13] use this technique and propose a solution called the conservative method. It works under the assumption that out-of-order data may be common, and thus produces output only when the correctness can be guaranteed. A partial order guarantee (POG) model is proposed under which such correctness can be guaranteed. Such techniques are interesting, but they require some services first to be created, appropriately inserting such assertions.

On durable events, there have been a stream of research works [10,11,16,18]. Kam and Fu [18] designed an algorithm that uses the hierarchical representation to discover frequent temporal patterns. However, the hierarchical representation is ambiguous and many spurious patterns are found. Papapetrou et al. [16] proposed the H-DFS algorithm to mine frequent

Please cite this article in press as: C.J. Zhou et al., Out-of-order durable event processing in integrated wireless networks, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.11.010

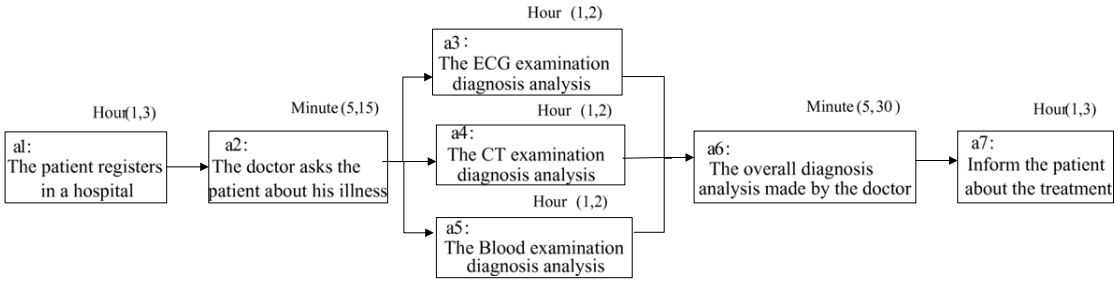


Fig. 1. An example of medical treatment flow.

arrangements of temporal intervals. Both these works transform an event sequence into a vertical representation using id-lists. The id-list of one event is merged with the id-list of other events to generate temporal patterns. This strategy does not scale well when the length of temporal patterns increases. Wu and Chen [10] devised an algorithm called TPrefix for mining non-ambiguous temporal patterns from durable events. TPrefix first discovers single frequent events from the projected database. Next, it generates all the possible candidates between the temporal prefix and discovered frequent events, and scans the projected database again for support counting. TPrefixSpan has several inherent limitations: multiple scans of the database are required and the algorithm does not employ any pruning strategy to reduce the search space. In order to overcome the above drawbacks, the authors of [11] give a lossless representation to preserve the underlying temporal structure of the events, and propose an IEMiner algorithm to discover frequent temporal patterns from durable events. However, they only use this representation for classification. The problem of out-of-order events is not considered.

3. Background and preliminaries

3.1. Research background

In medical industry, wireless network technology can effectively improve the efficiency of collaborative operations among doctors, nurses and relevant departments. Instead of cumbersome paper-based processes, most of the current hospitals track and share patient information such as medical reports, test results, and X-rays electronically. Wireless access to electronic medical records improves both the productivity of clinical staff and the quality of services. In Fig. 1, we take "the medical treatment flow" as an example to exhibit the motivation of our research.

As shown in Fig. 1, after a patient registers in a hospital, he will be dispatched to the corresponding department where he will consult a doctor about his illness. If the disease can be clearly judged, the patient will receive a diagnostic opinion directly. If further diagnosis analysis is required, the patient will be suggested to go to other departments for further examinations, such as ECG, CT and Blood tests. Doctors/nurses in these departments perform the suggested examinations and then return results back to the first doctor. The first doctor then synthesizes the diagnostic opinions from all sides, makes a decision and returns the diagnostic opinion back to the patient.

In this example, $Hour(1,\ 3)$ stands for the limits of duration of the event (the minimum is 1 h and the maximum is 3 h). Suppose many people take examinations in the hospital. Due to the network latency or some other reasons, an earlier examination may arrive at the doctor later. Although the ECG, CT and Blood samples can be sent to different departments in a certain order, the processing time might not be the same for each event, and therefore, the outgoing event stream might be out-of-order.

3.2. Temporal semantics

Each event has an ID and two timestamps. The application timestamp records the time that the event providers generate the events; the arrival timestamp is the time that events arrive at the consumer (responsible for processing the event). The application time can be further refined as a valid time and an occurrence time [3,19]. In the following we will introduce some special attributes of event timestamps in pervasive computing.

Definition 1 (*Time Granularity*)**.** The time granularity is the granularity used for describing the temporal constraints of events, such as second, minute, hour, day, week, month, year, and so on.

In the example of Fig. 1, the time granularity includes minute and hour. Before comparing them, a certain granularity should be chosen first. However, the proportion ratio between two cases of granularity may not be fixed. For example, the ratio of week and month, workday and hour are all not fixed.

Definition 2 (*Time Interval*)**.** Suppose $H = \{T_1, \ldots, T_n\}$ is a linear hierarchy of time units. In it, for all $1 \leq i < n,\ T_i \subseteq T_{i+1}$. For instance, $H = \{\text{minute, hour, day, month, year}\}$ and $minute \subseteq hour$. A time interval in H is an n-tuple $(t_1, \ldots, t_n)$ such that for all $1 \leq i \leq n$, t_i is a time-interval in the time unit of T_i.

Please cite this article in press as: C.J. Zhou et al., Out-of-order durable event processing in integrated wireless networks, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.11.010

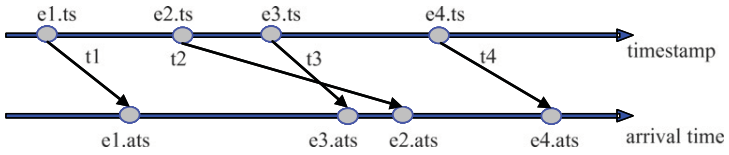


Fig. 2. Out-of-order event.

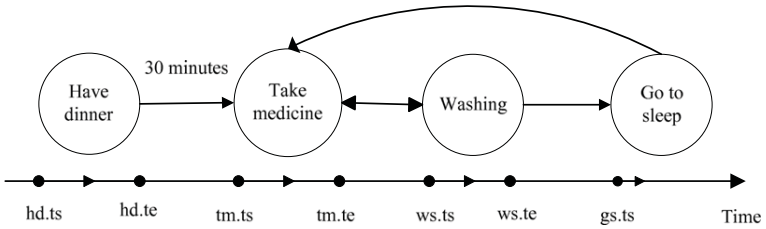


Fig. 3. Sequential query pattern.

Time intervals are ordered according to the lexicographic ordering $<_H$. Thus, time interval $T = (t_1, \ldots, t_n) <_H T' = (t'_1, \ldots, t'_n)$ iff there exists an $i (1 \leq i \leq n)$ such that $t_i <_{T_i} t'_i$ and $t_j = t'_j$ for all $j = i+1, \ldots, n$. Note that if $i = n$, then $t_j = t'_j$. When $T <_H T'$, we say that T occurs before T'. If $T = T'$, we say that T occurs simultaneously with T'.

For example, in Fig. 1, the total time interval of a1 and a2 is denoted as $T = Hour(1, 3) + Minute(5, 15)$, and the total time interval of a6 and a7 is $T' = Hour(1, 3) + Minute(5, 30)$. By using hierarchy $H = minute, H' = minute \subseteq hour, T <_{H'} T'$, because $15 <_H 30$.

Definition 3 (*Out-of-Order Event*)**.** Let $e.ats$ and $e.ts$ be the arrival timestamp and the occurrence timestamp of an event e. Consider an event stream S:$e_1, e_2, \ldots, e_n$, where $e_1.ats < e_2.ats < \cdots < e_n.ats$. For any two events e_i and e_j $(1 \leq i, j \leq n)$ from S, if $e_i.ts < e_j.ts$ and $e_i.ats < e_j.ats$, we say the stream is an ordered event stream. If however $e_j.ts < e_i.ts$ and $e_j.ats > e_i.ats$, then e_j is an out-of-order event.

In the example of Fig. 2, the timestamps of events $e1 \sim e4$ are listed in order. But we can see that event $e2$ arrives later than event $e3$, which is called out-of-order.

3.3. Query patterns

Query patterns specify how individual events are filtered and how multiple events are correlated via time-based and value-based constraints. Based on the time interval of events, query patterns can be classified into two categories: sequential query patterns and parallel query patterns.

3.3.1. Sequential query pattern

Sequential query pattern is a basic function supported by most event processing systems. The ability to synthesize events based on the ordering of previous events is useful for complex event detection. For efficiency in a stream setting, all operators that produce outputs involving more than one input event have a temporal constraint, denoted as w. For example, $seq(e_1, e_2, \ldots, e_n; w)$ outputs a complex event with $t_1 = e_1.starttime$, $t_2 = e_n.endtime$ if (i) $\forall i$ in $1, \ldots, n-1$ we have $e_i.endtime < e_{i+1}.starttime$ and (ii) $t_2 - t_1 \leq w$. Hence, *seq* constrains events of it to occur in order without overlapping.

For a concrete example of elder-care shown in Fig. 3, we should infer the elder's activities over time and remind them of something important (e.g., taking medicine), because they may be very forgetful. Suppose the man should take medicine within 30 min after having dinner. If we have already received the start and end time of having dinner, the start and end time of washing, but no start or end time of taking medicine, before the start time of going to sleep, an alarm should be issued to the man. For another example of Fig. 1, the tasks of cooperative medical treatment from "the patient registers in a hospital" to "the doctor asks the patient about his illness", to "the CT examination diagnosis analysis", to "inform the patient about the treatment", can also be regarded as a sequential query pattern.

3.3.2. Parallel query pattern

Parallel query pattern can be regarded as another feature for event processing systems, especially for durable events. In order to improve the performance, one may detect complex events concurrently. Parallel query pattern also has a temporal constraint, denoted as w. For example, $pal(e_1, e_2, \ldots, e_n; w)$ outputs a complex event with $t_1 = \min\{e_1.starttime, \ldots, e_n.starttime\}$, $t_2 = \max\{e_1.endtime, \ldots, e_n.endtime\}$, where $t_2 - t_1 \leq w$. Hence, *pal* permits overlapping among events and may cause events to be out-of-order. Parallel pattern includes conjunction and disjunction.

Please cite this article in press as: C.J. Zhou et al., Out-of-order durable event processing in integrated wireless networks, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.11.010

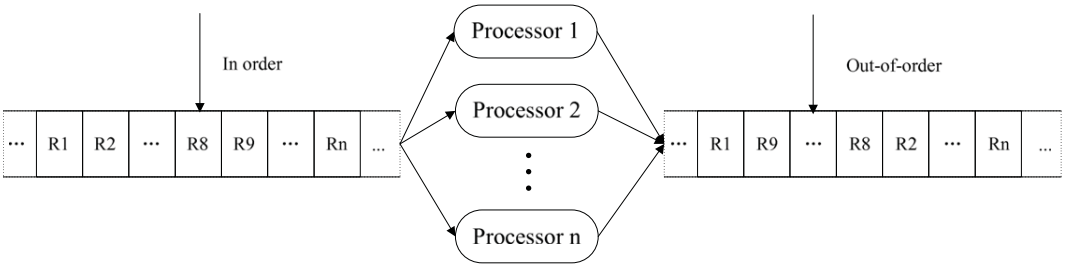


Fig. 4. Parallel query pattern.

Conjunction means both event *A* and event *B* occur within w, and their order does not matter. Disjunction means either event *A* or event *B* or both occurs within w.

For an example shown in Fig. 4, suppose there is a promotion in a bank in which the first *N* customers who satisfy certain conditions can get thank-you packs. Because there are a large number of customers in a bank, several processors are required to process the customers' requirements in parallel. Due to different service rates in different processors, the user who applied first may leave later. Also in the example of Fig. 1, the tasks of cooperative medical treatment "the ECG examination diagnosis analysis", "the CT examination diagnosis analysis" and "the Blood examination diagnosis analysis" can also be regarded as a parallel query pattern.

4. Out-of-order durable event detection

After introducing what out-of-order events are, and the problems caused by them, we start to present our method on how to detect out-of-order durable events in this section.

4.1. The model of durable events

As shown in Section 3, each event is denoted as $(ID, V_s, V_e, O_s, O_e, S_s, S_e, K)$. Here V_s and V_e denote the valid start and end time respectively; O_s and O_e denote the occurrence start and end time respectively; S_s corresponds to the system clock time upon event arrival; S_e means the system clock time when an event ends; K corresponds to an initial insert and all associated retractions, each of which reduces the S_e compared to the previous matching entry. Besides time stamps, the event may also have some other attributes, such as value, price, name, and so on.

In the following query format, Event Pattern connects events together via different event operators; the WHERE clause defines the context for event pattern by imposing predicates on events; the WITHIN clause describes the time range during which a matching event pattern must occur. Real-time Factor specifies the real-time requirement of different users.

```
<Query>::=EVENT <event pattern>
   [WHERE <value constraints>]
   [WITHIN <time constraints>]
   [Real-time Factor {0,1}]
   <event pattern>::=SEQ/PAL((Ei(relationship)Ej)
                    (!Ek)(relationship)El))(1≤i,j,k,l≤n)
   relationship::=contain,overlap,before,after,meet,finish,equal
   <time constraints>::= Time Window length W
```

The cooperative medical treatment flow in Fig. 1 can be expressed as Query Q1. *A* stands for the doctor asks the patient about his illness. *B*, *C* and *D* denote the ECG, CT and Blood examination diagnosis analysis respectively. *E* shows the overall diagnosis analysis made by the doctor. Events *B*, *C* and *D* can be executed in parallel, which are allowed to overlap with *E*.

```
  Query Q1. Cooperative Medical Treatment Pattern
EVENT PATTERN   SEQ(A(Before)PAL(B,C,D)(Overlap)E)
WHERE           A.Oe<B.Os
AND             A.Oe<C.Os
AND             A.Oe<D.Os
AND             B.price<C.price
AND             B.Oe>E.Os
AND             C.Oe>E.Os
AND             D.Oe>E.Os
WITHIN          6 workdays
Real-time Factor   1
```

Please cite this article in press as: C.J. Zhou et al., Out-of-order durable event processing in integrated wireless networks, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.11.010

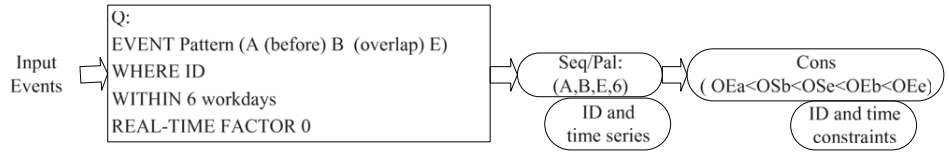
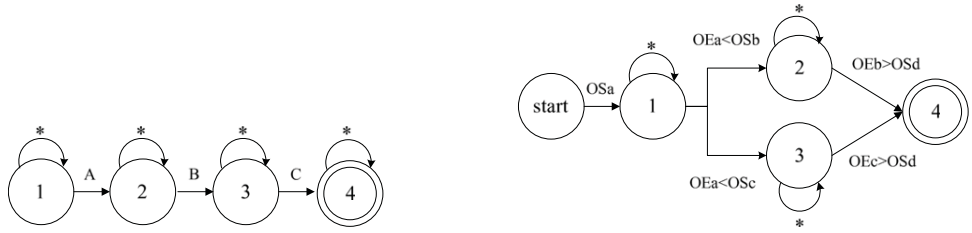


Fig. 5. Logical expression of query Q.



(a) NFA-based status transition. (b) The extended NFA expression.

Fig. 6. NFA-based expression.

4.1.1. Logical expression

A query expressed by the above language is translated into a query plan composed of the following operators: Sequential/Parallel Pattern (*Seq*/*Pal*), Negation Pattern (*Neg*), and Constraints (*Cons*) [7]. An event e_i is a positive (resp. negative) event if there is no '!' (resp. with '!') symbol used. The *Seq*/*Pal* operator denoted $Seq/Pal(E_1, E_2, \ldots, E_n, window)$ extracts all events matching to the positive event pattern specified in the query and constructs positive sequential/parallel events. *Seq*/*Pal* also checks whether all matched event sequences occur within the specified sliding window. The detailed definition and constraints of *Seq*/*Pal* can be found in Section 3.3. The *Neg* operator specified by $Neg(!E_1, (time\ constraint); \ldots; !E_m, (time\ constraint))$ checks whether there exist negative events within the indicated time constraint in a matched positive event pattern. In this paper, we don't introduce *Neg* in detail, because its operations are very similar to *Seq*/*Pal*, as in [13]. The *Cons* operator expressed as *Cons*(*P*), where *P* denotes a set of constraints on event attributes, further filters event patterns by applying the relationship specified in the query. Query Q1 can be simplified as *Q*, and Fig. 5 shows an example of the algebra plan for the pattern query *Q*.

4.1.2. Physical expression

Currently, non-deterministic finite automata (NFA) is the most commonly used method for evaluating complex event queries [2]. As the example shown in Fig. 6(a), the method starts at state 1, transits to state 2 when event *A* occurs, then to state 3 when *B* occurs, and finally to the output state when *C* occurs. A pattern is said to be matched when the NFA transitions into a final state. However, due to NFA's explicitly order state transitions, the prior NFA-based methods are not straightforward to efficiently model negation and parallel events. Here we extend the expression of NFA and process *Q* 1 as shown in Fig. 6(b). State 2 and 3 are internal states reached after a *B* or *C* event is received. State 4 is a final state indicating a match for the pattern. However, because there is a predicate involving events *B* and *C* ($B.price < C.price$), there is no simple way to evaluate the predicate when a *B* event arrives, because the corresponding event *C* may not arrive yet. Hence, it has difficulty to decide a transition for event *B*. As a result, this extended NFA also cannot be used for parallel query pattern processing with predicates, which is common in durable events.

For query patterns of durable events, the state transition depends on not only the type of events, but also the relationships and the predicates among events. In this paper, we propose to use tree-based query plans for both the logical and physical representation of query patterns. To process a query, we should first transform the query into an internal tree. Leaf nodes buffer atomic events as they arrive. Internal nodes buffer the intermediate results assembled from sub-tree buffers. Each internal node associates with one operator of the plan, along with a collection of predicates. Fig. 7(a) shows a tree plan for *Q* 1. This is a left-deep plan, because *A* and *Pal* are first combined, and their outputs are matched with *D*. A right-deep plan, where *Pal* and *D* are first combined, and then matched with *A*, is also possible.

Each node of the tree has a stack to temporarily store incoming events (for leaf nodes) or intermediate results (for internal nodes). Each stack contains a number of records, each of which has a vector of event pointers, including a start time and an end time of the event. For the start time of each instance in the stack, an extra field named *PreEve* records the nearest instance in terms of time sequence in the stack of the previous state (shown in Algorithm 1). While for the end time of each instance, *PreEve* is first set to its corresponding start time. When its start time becomes the *PreEve* of another instance, its *PreEve* changes to this instance. For example, in Fig. 7(b), the *PreEve* of $OE_a(7)$ is first set as $OS_a(3)$. The most recent instance in stack S_a of type A before $OS_b(6)$ is $OS_a(3)$. *PreEve* field of $OS_b(6)$ is set to $OS_a(3)$, as shown in the parenthesis preceding $OS_b(6)$, then the *PreEve* of $OE_a(7)$ changes to $OS_b(6)$. The start time and the end time of an internal node are the timestamps of the earliest and the latest atomic event comprising this complex event.

Please cite this article in press as: C.J. Zhou et al., Out-of-order durable event processing in integrated wireless networks, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.11.010

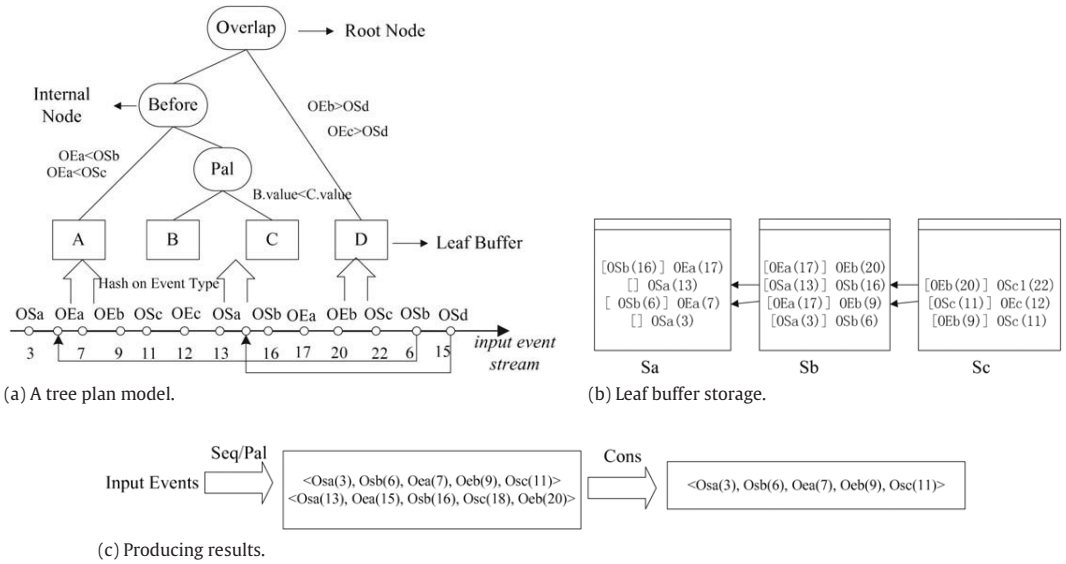


Fig. 7. Tree plan physical expression.

Algorithm 1 Storage Pattern of the Stack

Input:
The number of events in the stack, N;
The occurrence start time of the event, OS;
The occurrence end time of the event, OE;
One of the events that is already stored in the stack, k;

Output:
The correct TList of the stack;
string[ ] A;

```
for i = 0; i < N; i++ do
  for j = 1; j <= N; j++ do
    if A[i].OE has arrived and there is no k satisfying the prior pointer of A[k] is A[j].OS then
      the prior pointer of A[j].OS is set to A[i].OE;
      the prior pointer of A[j].OE is set to A[j].OS;
    else if A[i].OE has arrived and there is a k satisfying the prior pointer of A[k] is A[j].OS then
      the prior pointer of A[j].OS is set to A[i].OE;
      the prior pointer of A[j].OE is set to A[k];
    else if A[i].OE has not arrived and there is no k satisfying the prior pointer of A[k] is A[j].OS then
      the prior pointer of A[j].OS is set to A[i].OS;
      the prior pointer of A[j].OE is set to A[j].OS;
    else
      the prior pointer of A[j].OS is set to A[i].OS;
      the prior pointer of A[j].OE is set to A[k];
    end if
  end for
end for
```

Fig. 7(b) shows the stacks of each node in the tree model. In each stack, its instances are naturally sorted from top to bottom in the order of their arrival timestamps (shown in Algorithm 2). For in-order events, each received event is simply appended to the end of the corresponding stack and its *PreEve* field is set to the last event in the previous stack. However, this simple appended semantics is not applicable for the insertion of out-of-order events. An out-of-order event $e_i \in E_i$ will be allocated by the corresponding stack of type E_i in StateStack sorted by arrival timestamp. The *PreEve* field of event e_k's start time (i.e., $e_k.starttime$) in the adjacent stack with $e_k.starttime >_H e_i.starttime$ ($e_i.endtime$) will be set to $e_i.starttime$ ($e_i.endtime$), if the end time of all events in StateStack of type E_k has arrived; otherwise, it should wait until the absent end time. The *PreEve* field of event e_k's end time (i.e., $e_k.endtime$) is set to the corresponding start time $e_k.starttime$, if the start time has arrived. For example, in Fig. 7(c), suppose there are out-of-order events $OS_b(8)$ and $OE_b(10)$. The *PreEve* of $OS_b(8)$

Table 1
Annotated history table.

K	$Sync$	O_s	O_e	S_s	S_e
E_0	1	1	10	0	7
E_0	5	1	5	7	10

should not be set to $OS_a(3)$ until $OE_b(9)$ arrives, because before time point 8, there is only a start time of event type B, but no end time. In the same way, the *PreEve* of $OE_b(10)$ is set to $OS_b(8)$, because its corresponding start time has arrived.

Algorithm 2 Insert Operation of the Stack

Input:
The current number of events in the stack, N;
The occurrence start time of the event, OS;
The occurrence end time of the event, OE;
The new received event ID, k;
The maximal size of the stack, MS;
Output:
The correct TList of the stack;

```
1:  while k < MS do
2:    if all events in the stack are in order then
3:      the prior pointer of A[k].OS is set to A[N].OE;
4:      the prior pointer of A[k].OE is set to A[k].OS;
5:    else if there are out-of-order events in the stack, and the end time of all events have arrived then
6:      for i = N − 1; i >= 0; i − − do
7:        if A[k].OS > A[i].OS then
8:          the prior pointer of A[k].OS is set to A[i].OS;
9:        else
10:         the prior pointer of A[k].OS is set to A[i].OE;
11:         the prior pointer of A[k].OE is set to A[k].OS;
12:       end if
13:     end for
14:   else
15:     wait until there is no absent end time;
16:   end if
17: end while
```

4.2. The detection of out-of-order durable events

In order to detect out-of-order durable events, one more notation should be defined, named a synchronization point. We describe an annotated form of the history table which introduces an extra column, called *Sync*. A table with such a column is shown in Table 1. The extra column ($Sync$) is computed as follows: for insertions $Sync = Os$; for retractions $Sync = O_e$. The intuition behind the *Sync* column is that it induces a global notation of out-of-order events. For instance, if and only if the global ordering of events achieved by sorting events according to S_s is identical to the global ordering of events achieved by sorting events according to the compound key $\langle Sync, S_s \rangle$, then there are no out-of-order events in the stream.

5. Solution

5.1. Basic framework and expression

The framework of our method is shown in Fig. 8, which includes three components. The "Terminal Layer" component involves mobiles, PDAs, laptops, etc., which are the sources of events. The raw data generated from different data sources are ordered. "Event Processing Engine" stores the received events from "Terminal Layer", and handles some query processing. During the process of transmitting to "Event Processing Engine", network latencies and machine failures may cause events to be out-of-order. There are two data transition methods between "Terminal Layer" and "Event Processing Engine": push-based and pull-based, which will be discussed in Section 5.3. "Database Management" conserves historical records, event relationships and some knowledge base rules, as we have introduced in [20]. "Database Management" is really an integration of historical event records coupled with real-time event records. A knowledge base also should be stored in the "Database Management", which includes the extra information, such as the spatial location information, and the possible actions in a certain place. Relations identify the relationships among incoming atomic events in "Terminal Layer". In "Database

Please cite this article in press as: C.J. Zhou et al., Out-of-order durable event processing in integrated wireless networks, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.11.010

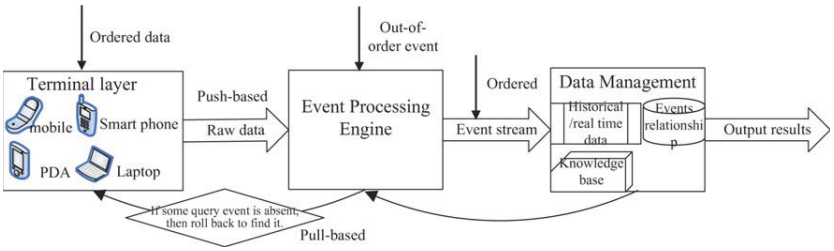


Fig. 8. Durable out-of-order solution framework.

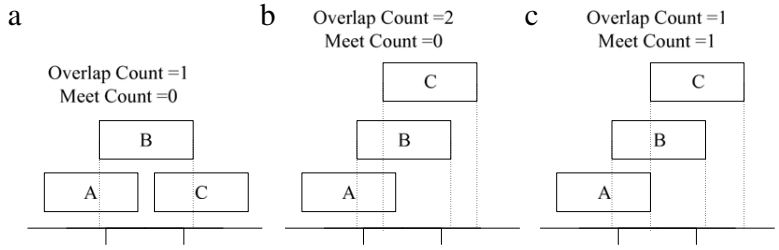


Fig. 9. Interpretation of pattern $(A\,(Overlap)\,B)\,(Overlap)\,C$.

Management" the instantaneous relation is used to denote a relation in the traditional bag-of-tuples sense, and a relation to denote a time-varying bag of tuples.

Besides the framework, specific query expression is also a challenge. Different from point-based queries, the expression of durable event queries may be ambiguous. Multiple interpretations may result in an incorrect inference of the exact relationship among events. For example, the same expression query $(A\,(overlap)\,B\,(overlap)\,C)$ may have different meanings, as shown in Fig. 9. In order to overcome this and distinguish the different interpretations of temporal patterns, the hierarchical representation with additional information is required [11]. Therefore, 5 variables are proposed, namely, contain count c, finish by count f, meet count m, overlap count o, and start count s to differentiate all the possible cases. The representation for a complex event E to include the count variable is shown as follows:

$$E = (\ldots (E_1\ R_1[c, f, m, o, s]\ E_2)\ R_2[c, f, m, o, s]\ E_3 \ldots\ R_{n-1}[c, f, m, o, s]\ E_n). \tag{1}$$

Thus, the temporal patterns in Fig. 9 are represented as:
(A Overlap [0,0,0,1,0] B) Overlap [0,0,0,1,0] C
(A Overlap [0,0,0,1,0] B) Overlap [0,0,0,2,0] C
(A Overlap [0,0,0,1,0] B) Overlap [0,0,1,1,0] C.

In the real world, different applications have different requirements for consistency. Some applications require a strict notion of correctness, while others are more concerned with real-time output. When exposed to users and handled by the system, users can specify consistency requirements on a per query basis and the system can adjust consistency at runtime. So we add an additional attribute ("Real-time Factor") to every query, as shown in Section 4.1. If the users focus on real-time output, the "Real-time Factor" is set to "1"; Otherwise, it is set to "0". Due to users' different requirements of consistency, there are two different methods, which are introduced as follows.

5.2. Real-time based method

If the "Real-time Factor" of a query is set to "1", the goal is to send out results with as small latency as possible based on the assumption that most data arrives in time and in order. Once out-of-order data arrival occurs, we provide a mechanism to correct the results that have already been erroneously output. This method is similar to, but better than the method in [13] because of the tree-plan expression, which can reduce the compensation time and frequency, as shown in Section 4.1.2.

At first, users submit queries to "Events Buffer", which handles the query processing and outputs the corresponding results directly. This method guarantees the real-time requirements and takes some urgent actions timely. However, in the case of out-of-order events, the output results may be wrong or the correct results may be lost. In order to compensate for this, two kinds of stream messages are used. Insertion $\langle +, E\rangle$ is induced by an out-of-order positive event [13], where "E" is a new event result. Deletion $\langle -, E\rangle$ is induced by an out-of-order negative event, such that "E" consists of the previously processed event. Deletion tuples cancel event results produced before which are invalidated by the appearance of an out-of-order negative event.

Please cite this article in press as: C.J. Zhou et al., Out-of-order durable event processing in integrated wireless networks, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.11.010

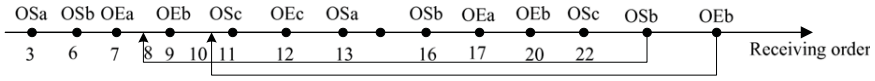


Fig. 10. Input events.

For example, the query is $(A(overlap)B(!D)(before)C)$ within 10 min. A unique time series expression of this query $\{OS_a, OS_b, OE_a, OE_b, OS_c, OE_c\}$ can be obtained based on the above interval expression method. For the event stream in Fig. 10, when an out-of-order *seq/pal* event $OS_b(6)$ is received, a new correct result $\{OS_a(3), OS_b(6), OE_a(7), OE_b(9), OS_c(11), OE_c(12)\}$ is output as $\langle +, \{OS_a(3), OS_b(6), OE_a(7), OE_b(9), OS_c(11), OE_c(12)\}\rangle$. when an out-of-order negative event $OS_d(15)$ is received, a wrong output result $\{OS_a(13), OS_b(16), OE_a(17), OE_b(20), OS_c(22)\}$ is found. So we send out a compensation $\langle -, \{OS_a(13), OS_b(16), OE_a(17), OE_b(20), OS_c(22)\}\rangle$.

When out-of-order events occur, these compensation operations should also be stored in "Database Management". Then, after a period of time, the query results generated from "Event Buffer" should first be transmitted into "Database Management", which checks the results again based on the historical records and knowledge base rules. If the results and the contents in "Database Management" are positively correlated, then output the final result; otherwise, the result should be held for a certain time, which is application-dependent. This optimization method consumes a little time, but it can eliminate many wrong results and compensation operations. From the global view, its performance is better than the existing methods.

5.3. Correct based method

If the "Real-time Factor" of a query is set to "0", the goal is to send out every correct result with less concern about the latency. Considering the time intervals, the existing methods can be improved as follows.

5.3.1. Query processing

Using our framework above, when the user submits a query to "Events Buffer", we first extract the corresponding sequential/parallel event patterns. Based on the event model introduced in Section 4.1, we can get the event sequence by a backward and forward depth first search in the DAG. The forward search is rooted at the start time of this instance e_i and contains all the virtual edges reachable from e_i. The backward search is rooted at the end time of event instances of the accepting state. It contains paths leading to and thus containing the event e_i. One final root-to-leaf path containing the new event e_i corresponds to one matched event sequence. If $e_i.endtime$ (resp. $e_i.starttime$) belongs to the accepting (resp. starting) state, the computation is done by a backward (resp. forward) search only.

Meanwhile, we can transform the query into a certain time series based on the above 5 variables, which make the representation of relationships among events unique. Compared with the time series of the query, the set of event sequences can be further filtered. For example, the precedence relationship among start time and end time of different events, the time window constraints, as well as negative events among the event sequence. After all these steps, the remaining event results are transmitted into a buffer in the "Database Management".

The buffer in the "Database Management" is proposed for event buffer and purging using the K-ISlack semantics. Different from the previous K-Slack method, we consider the time interval in this paper. It means that both the start time and the end time of the out-of-order event arrivals are within a range of K time units. That is, an event can be delayed for at most K time units. The buffer compares the distance between the checked event and the latest event received by the system. A CLOCK variable equal to the largest end time seen so far for the received events is maintained. The CLOCK is updated constantly. According to the sliding window of semantics, for any event instance e_i kept in the buffer, it can be purged from the stack if $(e_i.starttime + W) < CLOCK$. Thus, under the out-of-order assumption, the above condition will be $(e_i.starttime + W + K) < CLOCK$. This is because after waiting for K time units, no out-of-order events with start time less than $(e_i.starttime + W)$ can arrive. Thus e_i can no longer contribute to forming a new candidate sequence.

5.3.2. Optimization

In order to make some optimization, we divide the buffer into two parts: outdated event instances and up-to-date event instances, based on window constraints. A divider is set for the buffer: instances on or above it are outdated instances and instances below it are up-to-date ones. The part of outdated event instances stores the event sequence which falls out of the time window; while the up-to-date event instances keep the event sequence which is less than the allowed window range. For a stack without outdated events, the divider is set to *NULL*, while an in-order event triggers sequence construction. Only the events under the divider in each stack will be considered.

In addition, when out-of-order events occur, there may be some delay of attributes. For example, the end time of an event has been received, but no start time of it. As in Fig. 8, we take some active actions to the absent attributes, instead of waiting positively. If "Event Processing Engine" cannot find an attribute of the query, it will go back to data sources to see whether the absent one happened or not. If there is no such an attribute in data sources, then the corresponding query results are output or discarded directly. Otherwise, "Event Processing Engine" keeps waiting until the attribute arrives, then outputs through "Database Management". As introduced in Section 5.1, the "Database Management" involves both historical records and knowledge base data, which can filter some incorrect results before generating outputs.

Please cite this article in press as: C.J. Zhou et al., Out-of-order durable event processing in integrated wireless networks, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.11.010

Table 2
Parameters and performance metrics.

Terminology	Meaning
P_{io^3}	Out-of-order event percentage
Buf	Buffer size of tree pattern
QL	Event's query length
NoR	Number of results
NoC	Number of compensation results
NoCR	Number of correct results
K	Maximum delay of out-of-order events
AET	Average execution time
AL	Average latency
RoC	Rate of compensation, NoC/NoR
ACC	Accuracy of results, NoCR/NoR

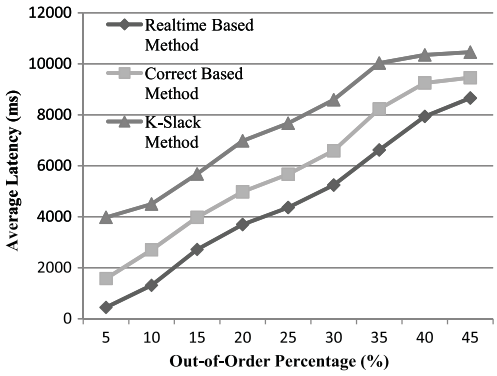


Fig. 11. Trend of average latency.

6. Experiments

In order to test and verify the above two algorithms, we designed an experimental environment to simulate the events generation and queries. A prototype using the C# language has been implemented.

6.1. Experimental environments

Our experiments involve two parts: one is the event generator; another is the event processing engine. The event generator is used for generating different types of events continuously. We adopt multi-thread to model different sensors to produce different events randomly. Then the generated events are sent to the event receiver, which is a part of event processing engine. The event processing engine includes two units: the receiver unit and the query unit. The former is just responsible for receiving the events from "sensors"; the latter takes charge of queries, and outputs the correct results. Meanwhile, it records the performance information which is shown in Table 2.

Our experiments run on two machines, with Intel Dual-Core 2.0 GHz and 2.5 GHz CPU, 2.0 G and 3.0 G RAM respectively. PC1 is used for running the Event Generator programs and PC2 for the Event Processing Engine. In PC1, we created about 1000 generators ("sensors"), each of which can produce more than 1000 different-type (A, B, C or D) events randomly. So at least 1,000,000 events will reach the receiver hosted in PC2 and wait to be queried. Based on such a large scale of event data, our experiments can test and verify the performance of the algorithms much better. Additionally, in order to make the experimental results more convincing, we run the program for 300 times, and take the average value of all results. In the following, we will focus on the key performance metrics shown in Table 2.

6.2. Experimental results

Figs. 11–15 mainly examine the impact of out-of-order percentage P_{io^3} on the performance metrics. P_{io^3} is varied from 0% to 45%. Fig. 11 shows the case when there no durable events arrive. From the figure, the average latency of three methods (Realtime Based Method, Correct Based Method and *K*-Slack Method) increases with the enlargement of out-of-order percentage, and Realtime Based Method increases faster than the other two methods, because we add the cost of compensation operations into the definition of average latency. However, if there are durable events, the naive *K*-Slack method will not work, while the trend of Realtime Based Method and Correct Based Method are almost the same, as shown in Fig. 11.

Please cite this article in press as: C.J. Zhou et al., Out-of-order durable event processing in integrated wireless networks, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.11.010

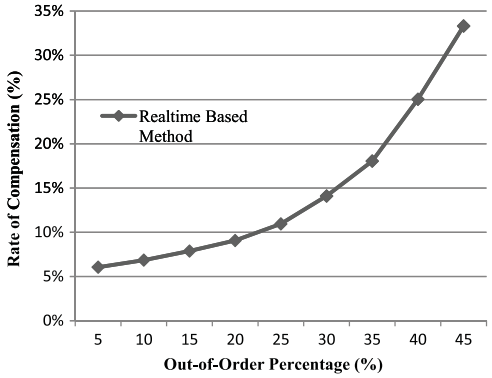


Fig. 12. Trend of rate-of-compensation.

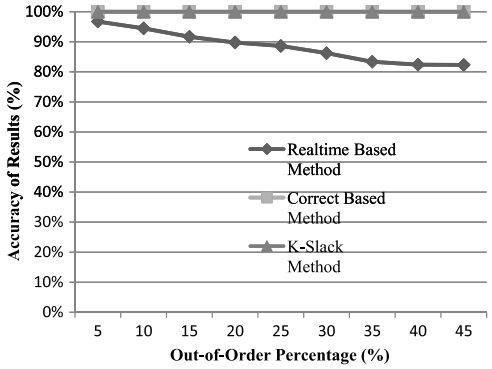


Fig. 13. Accuracy without durable events.

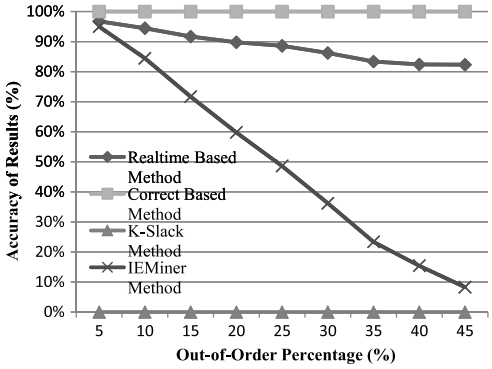


Fig. 14. Accuracy with durable events.

Fig. 12 just concerns Realtime Based Method, which has compensation operations. The rate of compensation is determined by (*NoC*/*NoR*). From the figure, we can see that with an increase of out-of-order percentage, more compensation operations are generated, and the speed of compensation rate is faster and faster.

The accuracy of results is also examined, defined as (*NoCR*/*NoR*). Fig. 13 shows the accuracy of three methods when there are no durable events. In this case, the accuracy of Correct Based Method and *K*-Slack Method are both independent of out-of-order percentage, while Realtime Based Method drops with the enlargement of out-of-order percentage. This is because with larger out-of-order percentage, more output results should be compensated.

Fig. 14 shows the accuracy of four methods (Realtime Based Method, Correct Based Method, *K*-Slack Method and IEMiner Method) when there are durable events. In this case, the accuracy of *K*-Slack Method is almost zero, because it cannot deal with out-of-order durable events. With the enlargement of out-of-order percentage, the accuracy of IEMiner Method drops fast, because it can only deal with durable events, but not out-of-order events. The accuracy of Realtime Based Method and Correct Based Method are similar to the case in Fig. 13.

Please cite this article in press as: C.J. Zhou et al., Out-of-order durable event processing in integrated wireless networks, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.11.010

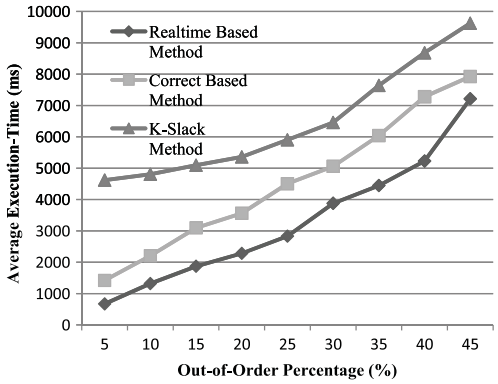


Fig. 15. Trend of average execution time.

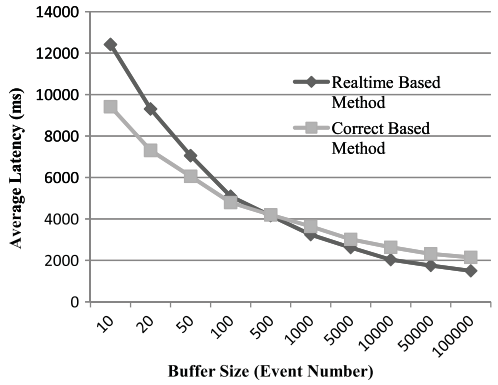


Fig. 16. Trend of average latency.

We examine the average execution time in Fig. 15, which denotes the summation of operator execution times. When there are no durable events, two observations can be found: (1) the average execution time increases as the percentage of out-of-order events increases because more recomputing operations are needed; (2) the average execution time of Correct Based Method is larger than Realtime Based Method at beginning, while with the enlargement of out-of-order percentage, they will tend to the same. But the execution time of K-Slack Method is always larger than the other two methods. If there are durable events, the execution time of K-Slack Method tends to be infinity, while the trend of Realtime Based Method and Correct Based Method are almost unchanged.

Figs. 16–18 show the impact of buffer size on performance metrics. Fig. 16 shows that the average latency of both methods decreases with the enlargement of buffer size. When the buffer size in tree-pattern is less than 500 *event number*, the average latency of Correct Based Method is less than Realtime Based Method; while the opposite situation happens when the buffer size is larger than 500 *event number*. I.e., the dropping ratio of Realtime Based Method is faster than Correct Based Method, or the buffer size has much more impact on Realtime Based Method. That is because when the buffer is too small, there must be a lot of incorrect results output, which cause too many compensation operations and extend the latency. While when the buffer size is large enough, the compensation results decrease quickly, so the average latency of Correct Based Method is larger than Realtime Based Method again.

Fig. 17 shows the accuracy trends of both methods with different buffer size. When the buffer size is near to zero, the accuracy of both methods is also about 0%, because there are almost no results generated now. However, when the buffer size is a little larger, the accuracy of both methods increases immediately. Because we can always get the prior events from the "Data Management Center", the accuracy of both methods almost stays the same with the enlargement of buffer size. That is to say, the parameter of buffer size has little effect on accuracy.

The trend of average execution time is shown in Fig. 18, which is similar to the trend of average latency. There is only a constant difference between them, from the first event's arrival time to the corresponding last event's.

Fig. 19 shows the impact of event query length on average execution time when there are no durable events. From the figure, we can see the trend can be divided into two parts. When the event query length is shorter, the average execution time of Correct Based Method and K-Slack Method is larger than Realtime Based Method. With the enlargement of event query length, they tend gradually to the same, and then Realtime Based Method becomes the largest. That is because when

Please cite this article in press as: C.J. Zhou et al., Out-of-order durable event processing in integrated wireless networks, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.11.010

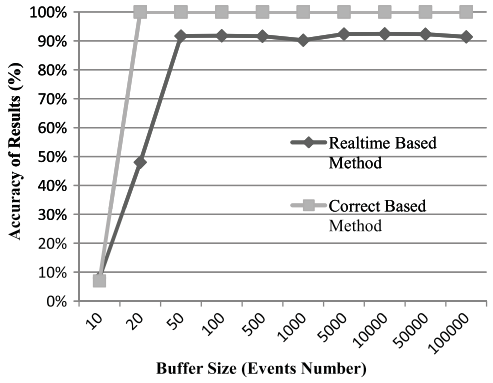


Fig. 17. Accuracy of methods.

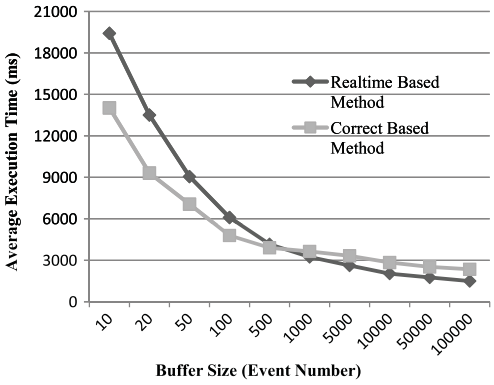


Fig. 18. Execution time on buffer size.

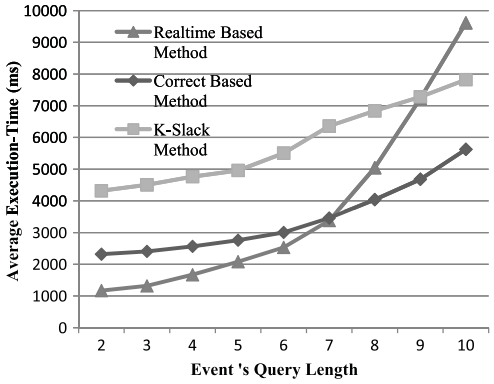


Fig. 19. Execution time on event length.

the event query length is too long, there must be many compensation operations of Realtime Based Method. The average execution time of K-Slack Method is always larger than Correct Based Method. Compared with Realtime Based Method, event query length has less impact on Correct Based Method and K-Slack Method. If there are durable events, the execution time of K-Slack Method tends to be infinite.

Fig. 20 shows the latency of the three methods increases with the increase of event query length when there are no durable events. From the figure, we can see that Realtime Based Method increases faster than the other two methods. The latency of K-Slack Method is always larger than Correct Based Method. If there are durable events, the latency of K-Slack Method tends to be infinite, because it cannot deal with durable events.

Please cite this article in press as: C.J. Zhou et al., Out-of-order durable event processing in integrated wireless networks, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.11.010

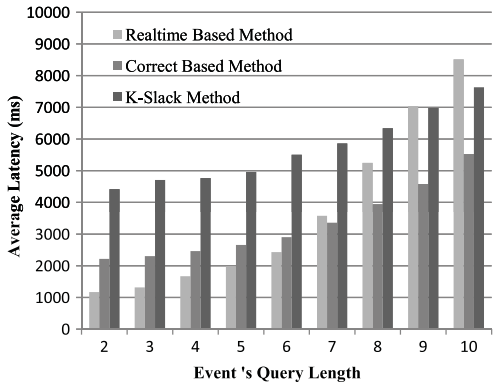


Fig. 20. Latency on event length.

7. Conclusion and future work

The goal of this work is to solve query processing of out-of-order durable events in integrated wireless networks. We proposed a tree-plan model of out-of-order durable events, which can give the logical and physical expressions. We also combined techniques on addressing the detection of durable events and out-of-order events. A hybrid solution to solve out-of-order events is studied, which can switch from one level of output accuracy to another in real time. The experimental study compares with the method with *K*-Slack and IEMiner methods, and demonstrates the effectiveness of our proposed approach.

In this paper, we primarily consider incoming events for occurrences of user-specified event patterns. In the future work, mining the automatic complex events patterns of out-of-order durable events will be studied. In addition, we will consider query optimization strategies and algorithms in the "Data Management Center", so that the query frequency can be regarded as another influencing factor on performance metrics.

Acknowledgement

We would like to thank Pengfei Dai of Beijing University of Post and Telecommunications for his helpful comments on the experiments.

References

[1] S. Babu, et al., Exploiting *K*-constraints to reduce memory overhead in continuous queries over data streams, ACM Transaction on Database Systems 29 (3) (2004) 545–580.
[2] Y. Mei, S. Madden, ZStream: a cost-based query processor for adaptively detecting composite events, in: Proceedings of the 35th SIGMOD International Conference on Management of Data, SIGMOD, 2009, pp. 193–206.
[3] M. Akdere, U. Cetintemel, N. Tatbul, Plan-based complex event detection across distributed sources, in: Proceedings of the 34th International Conference on Very Large Data Bases, VLDB, 1, 1, 2008, pp. 66–77.
[4] A.H.M. Amin, A.I. Khan, Collaborative-comparison learning for complex event detection using distributed hierarchical graph neuron (DHGN) approach in wireless sensor network, Australasian Conference on Artificial Intelligence (2009) 111–120.
[5] C. Antunes, A.L. Oliveira, Generalization of pattern growth methods for sequential pattern mining with gap constraints, Machine Learning and Data Mining in Pattern Recognition (2003).
[6] J. Pei, J. Han, B. Mortazavi, H. Pinto, Q. Chen, Prefixspan: mining sequential patterns efficiently by prefix-projected pattern growth, in: Proceedings of the 17th International Conference on Data Engineering, ICDE, 2001, pp. 215–226.
[7] E. Wu, Y. Diao, S. Rizvi, High performance complex event processing over streams, in: Proceedings of the 32th SIGMOD International Conference on Management of Data, SIGMOD, 2006, pp. 407–418.
[8] D. Alex, R. Robert, V.S. Subrahmanian, Probabilistic temporal databases, ACM Transaction on Database Systems 26 (1) (2001) 41–95.
[9] R.S. Barga, J. Goldstein, M. Ali, M.S. Hong, Consistent streaming through time: a vision for event stream processing, in: The 3rd Biennial Conference on Innovative Data Systems Research, IDAR, 2007.
[10] S. Wu, Y. Chen, Mining nonambiguous temporal patterns for interval-based events, IEEE Transactions on Knowledge and Data Engineering 19 (6) (2007) 742–758.
[11] D. Patel, W. Hsu, M.L. Lee, Mining relationships among interval-based events for classification, in: Proceedings of the 34th SIGMOD International Conference on Management of Data, SIGMOD, 2008, pp. 393–404.
[12] M.A. Hammad, et al. Scheduling for shared window joins over data streams, in: The 29th International Conference on Very Large Data Bases, Vol. 29, 2003, pp. 297–308.
[13] M. Liu, M. Li, D. Golovnya, E.A. Rundenstriner, K. Claypool, Sequence pattern query processing over out-of-order event streams, in: Proceedings of the 25th International Conference on Data Engineering, ICDE, 2009, pp. 274–295.
[14] S. Eyerman, L. Eeckhout, T. Karkhanis, J.E. Smith, A mechanistic performance model for superscalar out-of-order processors, ACM Transactions on Computer Systems (TOCS) 27 (2) (2009).
[15] F. Wang, S. Liu, P. Liu, Complex RFID event processing, The International Journal on Very Large Data Bases (VLDBJ) 18 (4) (2009) 913–931.
[16] P. Papapetrou, G. Kollios, S. Sclaroff, D. Gunopulos, Discovering frequent arrangements of temporal intervals, in: Proceedings of the 5th IEEE International Conference on Data Mining, ICDM, 2005.

Please cite this article in press as: C.J. Zhou et al., Out-of-order durable event processing in integrated wireless networks, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.11.010

[17] L. Ding, N. Mehta, E.A. Rundensteiner, G.T. Heineman, Joining punctuated streams, Advances in Database Technology (EDBT) (2004) 587–604.
[18] P.S. Kam, A.W. Fu, Discovering temporal patterns for interval-based events, in: Proceedings of the 2nd International Conference on Data Warehousing and Knowledge Discovery, DaWak, 2000, pp. 317–326.
[19] S. Roger, J.G. Barga, A. Mohamed, M. Hong, Consistent streaming through time: a vision for event stream processing, in: 3rd Biennial Conference on Innovative Data Systems Research, CIDR, 2007.
[20] C.J. Zhou, X.F. Meng, A framework of complex event detection and operation in pervasive computing, in: The Ph.D. Workshop on Innovative Database Research, IDAR, 2009.

Please cite this article in press as: C.J. Zhou et al., Out-of-order durable event processing in integrated wireless networks, Pervasive and Mobile Computing (2010), doi:10.1016/j.pmcj.2010.11.010