IO^3: Interval-based Out-of-Order Event Processing in Pervasive Computing

Chunjie Zhou ⋆ , Xiaofeng Meng

School of Information, Renmin University of China, Beijing, China
{lucyzcj,xfmeng}@ruc.edu.cn

Abstract. In pervasive computing environments, complex event processing has become increasingly important in modern applications. A key aspect of complex event processing is to extract patterns from event streams to make informed decisions in real-time. However, network latencies and machine failures may cause events to arrive out-of-order. In addition, existing literatures assume that events do not have any duration, but events in many real world application have durations, and the relationships among these events are often complex. In this work, we first analyze the preliminaries of time semantics and propose a model of it. A hybrid solution including time-interval to solve out-of-order events is also introduced, which can switch from one level of output correctness to another based on real time. The experimental study demonstrates the effectiveness of our approach.

Key words: pervasive computing, complex event, out-of-order, time interval

1 Introduction

In pervasive computing environments, event processing has raised increased interest in the past few years [1, 4, 3]. A growing numbers of applications nowadays take the form of time ordered or out-of-ordered series, and every event has time duration. All these factors about time are very important in extracting patterns from atomic events in pervasive computing. However, the present literatures about time management in pervasive computing usually refer to only one aspect of the problem.

Existing research works [1] almost focus on instantaneous events. However, purely sequential queries on instantaneous events are not enough to express many real world patterns. For example, it has been observed that in many diabetic patients, the presence of hyperglycemia1 overlaps with the absence of glycosuria [10]. This insight has led to the development of effective diabetic testing kits. Clearly, there is a need for an efficient algorithm that can solve events with duration.

Meanwhile, the real-time processing in time order of event streams generated from distributed devices is a primary challenge in pervasive computing environments. However, most systems [2]assume a total ordering among event arrivals. It has been illustrated that the existing technology would fail in such circumstances, either missing

⋆ This research was partially supported by the grants from the Natural Science Foundation of China under Grant No. 60833005; National High-Tech Research and Development Plan of China under Grant No. 2007AA01Z155, 2009AA011904; the Ph.D. Programs Foundation of Ministry of Education of China No.200800020002

resulting matches or incorrectly producing incorrect matches. Let us consider a popular application for tracking books in a bookstore [3] where RFID tags are attached to each book and RFID readers are placed at strategic locations throughout the store, such as book shelves, checkout counters and the store exit. If a book shelf and a store exit sensed the same book but none of the checkout counters sensed it in between the occurrence of the first two events, then we can conclude that this book is being shoplifted. If events of sensing at the checkout counters arrive out-of-order, we cannot ever output any results if we want to assure correctness of results. Otherwise, if we focus on real-time alarm, we may output the wrong results. Clearly, it is imperative to deal with both in-order as well as out-of-order arrivals efficiently and in real-time.

The contributions and the rest of this work are organized as follows: Section 2 gives the related works. Section 3 provides some preliminaries, including the time semantics and the model of interval-based out-of-order events. Section 4 describes a hybrid solution. Section 5 gives the experiment results and we conclude in Section 6.

2 Related Works

About the out-of-order problem, from the aspect of different scenarios, the literatures can be divided into two types, one focus on real time,another pay more attention to correctness. Since the input event stream to the query engine is unordered, it is reasonable to produce unordered output events. So [6] permits unordered sequence output and proposes the aggressive strategy.

If ordered output is needed, additional semantic information such as K-Slack factor or punctuation is needed to ”unblock” the on-hold candidate sequences from being output. A native approach [2] is using K-Slack as a priori bound on the out-of-order of the input streams. The biggest drawback of K-slack is rigidity of the K that cannot adapt to the variance in the network latencies that exists in a heterogeneous RFID reader network. Another solution is applying punctuations, namely, assertions inserted directly in the data stream confirming that for instance a certain value or time stamp will no longer appear in the future input streams [6]. [6] use this technique and propose a solution called conservative method. However, such techniques, while interesting, require for some service to first be creating and appropriately inserting such assertions.

About the interval-based events, there has been a stream of research [7–10]. Kam et. al. [7] designs 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. [8] proposes the H-DFS algorithm to mine frequent arrangements of temporal intervals.This approach does not scale well when the temporal pattern length increases. Wu et. al. [9] devises an algorithm called TPrefix for mining non-ambiguous temporal pattern from interval-based events. TPrefixSpan has several inherent limitations: multiple scans of the database are needed and the algorithm does not employ any pruning strategy to reduce the search space. In order to overcome the above drawbacks, [10] gives a lossless representation to preserve the underlying temporal structure of the events, and proposes an algorithm to discover frequent temporal patterns from interval-based events. However, they only use this representation for classification but don’t consider out-of-order problems.

3 Preliminaries

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 respectively denote valid start and end time; O_s and O_e respectively denote occurrence start and end time; S_s corresponds to the system server clock time upon event arrival; K corresponds to an initial insert and all associated retractions, each of which reduce the S_e compared to the previous matching entry in the table.

Definition 1 (time interval) *Suppose $T_1 \subseteq \ldots \subseteq T_n$ is a linear hierarchy H of time units. For instance, $H = minute \subseteq hour \subseteq day \subseteq month \subseteq year$. A time interval T_i in H is an n-tuple $(t_{i_1}, \ldots, t_{i_n})$ such that for all $1 \leq i \leq n$, t_{i_i} is a time-interval in the time-value set of T_i [5].*

Definition 2 (out-of-order event) *Consider an event stream $S: e_1, e_2, ..., e_n$, where $e_1.ats < e_2.ats < ... < e_n.ats$. 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 flagged as an out-of-order event.*

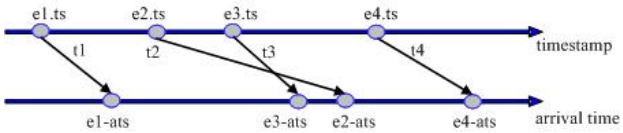

Fig. 1. out-of-order event

In the example shown in Figure 1, the timestamp of events $e_1 \sim e_4$ are listed in order. But we can see that the earlier event e_2 received later than event e_3, which is called out-of-order.

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 events that match the pattern must occur. Real-time Factor specifies the real-time requirement intensity 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::=overlap,before,after
   <time constraints>::= Time Window length W
```

In pattern queries among interval-based events, the state transition not only depends on the event type, but also the relationships and the predicates among events. So in this paper, we use tree-based query plans for query patterns.

4 Interval-based Out-of-Order Solution

Our method framework is shown in Figure 2, which includes three components. "Terminal Layer" is the sources of events. "Event Buffer" stores the received events from "Terminal Layer", and handles some query processing. "Database Management" conserves historical records, event relationships and some knowledge base rules, as we have introduced in [11].

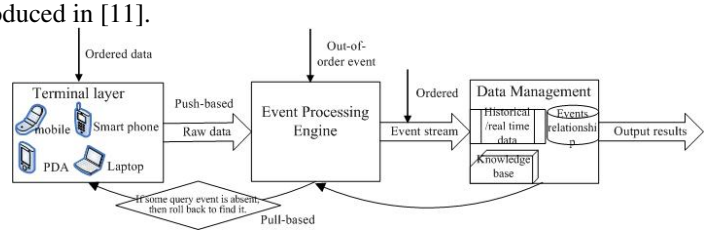

Fig. 2. Interval-based out-of-order solution framework

Besides the framework, specific query expression is also a challenge problem. The expression of interval-based queries may be ambiguous. For example, the same expression query (*A* (*overlap*) *B* (*overlap*) *C*) may have different meanings, as shown in Figure 3. In order to overcome this problem, the hierarchical representation with additional information is needed [10]. Then 5 variables is 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 composite event E to include the count variable is shown as follows:

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

Thus, the temporal patterns in Figure 3 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

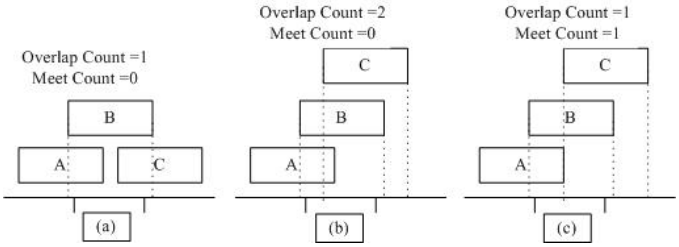

Fig. 3. Interpretation of pattern (*A* (*Overlap*) *B*) (*Overlap*) *C*

In 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. So we add an additional attribute ("Real-time Factor") into every query, as shown in section 3. If the users focus on real-time output, the "Real-time Factor" is set to "1"; Otherwise, it is valued as "0". Due to users' different requirements of consistency, there are two different methods, which are introduced as follows.

4.1 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. When users submit queries to "Events Buffer", which handles the query processing and outputs the corresponding results directly. 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 [6] because of the tree-plan expression, which can reduce the compensation time and frequency, as shown in section 3.

For example, the query is $(A(overlap)B(!D)(before)C)$ within 10 mins. A unique time series expression of this query $\{OS_a, OS_b, OE_a, OE_b, OS_c, OE_c\}$ can be gotten based on the above interval expression method. For the event stream in Figure 4, 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 constructed and output as $<+, \{OS_a(3), OS_b(6), OE_a(7), OE_b(9), OS_c(11), OE_c(12)\}>$. 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 and send out a compensation $<-, OS_a(13), OS_b(16), OE_a(17), OE_b(20), OS_c(22)>$.

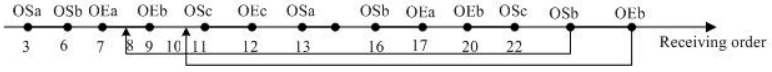

Fig. 4. Input events

4.2 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. When the user submit a query to "Events Buffer", we first extract the corresponding event type. Based on the event model introduced in section 3, we can get the event sequence by a backward and forward depth first search in the DAG. 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 sequence can be further filtered.

The buffer in the "Database Management" is proposed for event buffer and purging using the interval-based K-ISlack semantics. It means that both the start time and the end time of the out-of-order event arrivals is within a range of K time units. A CLOCK value which equals to the largest end time seen so far for the received events is maintained. The CLOCK value is updated constantly. According to the sliding window 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 event 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 Experiments

Our experiment involves two parts: one is the event generator; another is the event process engine. The event generator is used for generating different types of events

continuously. The event process engine includes two units: the receiver unit and the query unit.

The experiments are run on two machines, which CPU is 2.0GHz and RAM is 2.0G and 3.0G respectively. PC1 is used for running the Event Generator programs and PC2 for the Event Process Engine. In order to make the experimental results more credible, we run the program for 300 times, and take average value of all results. In the following, we will examine key performance metrics. $P_{io^3}$is varied from 0% to 45%, and in order to be simple, the effects of window buffer size will be considered in our future work.

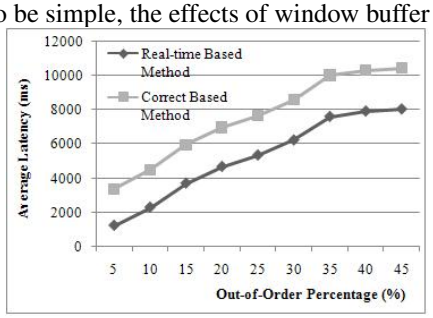

Fig. 5. Trend of Average Latency

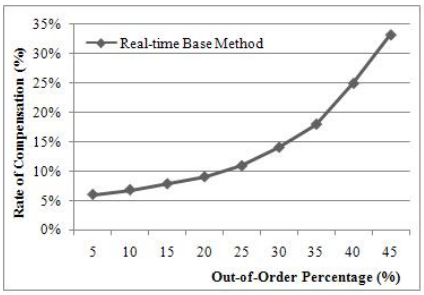

Fig. 6. Trend of Rate-of-Compensation

Figure 5 shows that the average event latency of both methods increases with the increase of out-of-order percentage, and the average latency of Correct Based Method increases faster than Realtime Based Method.

Figure 6 is only in connection with Realtime Based Method, which has compensation operations. The rate of compensation is determined by (NoC/NoR). From the figure, we can see the increasing out-of-order percentage leads to more compensation operations to be generated, which reduce the efficiency of algorithm.

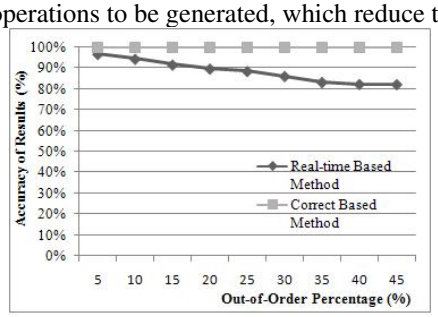

Fig. 7. Accuracy of Methods

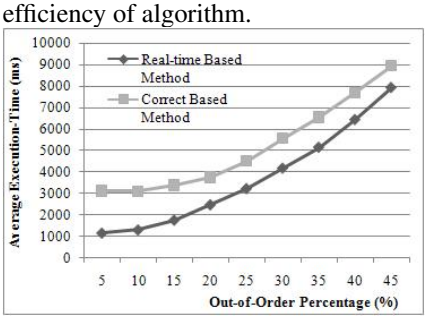

Fig. 8. Trend of Average Execution-Time

We also experiment accuracy of results. In Figure 7, we can see the result accuracy of Correct Based Method is independent of out-of-order percentage, while the result accuracy of Realtime Based Method drops with the increase of out-of-order percentage.

We examine the average excution time in Figure 8, which denotes the summation of operator execution times. From the figure, two observations can be found: 1) the average excution time increased as the out-of-order event percentage increases for more recomputing is needed; 2) the average execution time of Correct Based Method is larger than Realtime Based Method at beginning, while with the increase of out-of-order percentage, they will trend to the same.

6 Conclusion and Future Work

The goal of this work is to solve query processing of interval-based out-of-order events in pervasive computing. We analyze the preliminaries, propose a model of interval-based out-of-order events, and a hybrid solution including time-interval is also introduced. In the future work, we will consider other influence factors, including the size of buffer, the out-of-order percentage, the average step-length of out-of-order events, in order to find the balance point.

Acknowledgement

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

References

1. Pei, J., Han, J., Mortazavi, B., Pinto, H., Chen, Q.: Prefixspan: Mining Sequential Patterns Efficiently by Prefix-projected Pattern Growth. Proceedings of the 17th International Conference on Data Engineering (ICDE), pp. 215-226. (2001)
2. Babu, S., and et. al.: Exploiting K-constraints to Reduce Memory Overhead in Continuous Queries over Data Streams. ACM Transaction on Database Systems, Vol. 29, No. 3, pp. 545-580. (2004)
3. Wu, E., Diao, Y., Rizvi, S.: High Performance Complex Event Processing over Streams. Proceedings of the 32th SIGMOD International Conference on Management of Data (SIGMOD), pp. 407-418. (2006)
4. Mei, Y., Madden, S.: ZStream: a Cost-based Query Processor for Adaptively Detecting Composite Events. Proceedings of the 35th SIGMOD International Conference on Management of Data (SIGMOD), pp. 193-206. (2009)
5. Alex, D., Robert, R., Subrahmanian, V.S.: Probabilistic Temporal Databases. ACM Transaction on Database Systems, Vol.26, No.1, pp. 41-95. (2001)
6. Liu, M., Li, M., Golovnya, D., Rundenstriner, E.A., Claypool, K.: Sequence Pattern Query Processing over Out-of-Order Event Streams. Proceedings of the 25th International Conference on Data Engineering (ICDE), pp. 274-295. (2009)
7. Kam, P.S., Fu, A.W.: Discovering Temporal Patterns for Interval-based Events. Proceedings of the 2th International Conference on Data Warehousing and Knowledge Discovery (DaWak), pp. 317-326. (2000)
8. Papapetrou, P., Kollios, G., Sclaroff, S., Gunopulos, D.: Discovering Frequent Arrangements of Temporal Intervals. Proceedings of the 5th IEEE International Conference on Data Mining (ICDM). (2005)
9. Wu, S., Chen, Y.: Mining Nonambiguous Temporal Patterns for Interval-based Events. IEEE Transactions on Knowledge and Data Engineering, Vol. 19, Issue 6, pp. 742-758. (2007)
10. Patel, D., Hsu,W., Lee, M.L.: Mining Relationships among Interval-based Events for Classification. Proceedings of the 34th SIGMOD International Conference on Management of Data (SIGMOD), pp. 393-404. (2008)
11. Zhou, C.J., Meng, X.F.: A Framework of Complex Event Detection and Operation in Pervasive Computing. The PhD Workshop on Innovative Database Research (IDAR). (2009)