Journal of Intelligent Learning Systems and Applications, 2018, 10, 21-35
http://www.scirp.org/journal/jilsa
ISSN Online: 2150-8410
ISSN Print: 2150-8402

Interval-Based Out-of-Order Event Processing in Intelligent Manufacturing

Chunjie Zhou[1,2], Pengfei Dai[2], Zhenxing Zhang[1], Tong Liu[1]

[1]School of Information and Electrical Engineering, Ludong University, Yantai, China
[2]Yantai Cloudoer Software Co., Ltd., Yantai, China
Email: luckyzcj@163.com

How to cite this paper: Zhou, C.J., Dai, P.F., Zhang, Z.X. and Liu, T. (2018) Interval-Based Out-of-Order Event Processing in Intelligent Manufacturing. *Journal of Intelligent Learning Systems and Applications*, **10**, 21-35.
https://doi.org/10.4236/jilsa.2018.102002

Received: February 3, 2018
Accepted: May 5, 2018
Published: May 8, 2018

Abstract

Estimating the cycle time of each job over event streams in intelligent manufacturing is critical. These streams include many long-lasting events which have certain durations. The temporal relationships among those interval-based events are often complex. Meanwhile, network latencies and machine failures in intelligent manufacturing may cause events to be out-of-order. This topic has rarely been discussed because most existing methods do not consider both interval-based and out-of-order events. In this work, we analyze the preliminaries of event temporal semantics. A tree-plan model of interval-based out-of-order events is proposed. A hybrid solution is correspondingly introduced. Extensive experimental studies demonstrate the efficiency of our approach.

Keywords

Event Streams, Intelligent Manufacturing, Interval-Based Events, Out-of-Order Events

1. Introduction

In recent years, estimating the cycle time of each job over event streams in intelligent manufacturing applications has received a lot of attention from production control, soft computing, and operations management researchers because of its critical role in the competitiveness of intelligent manufacturing [1]. These event streams are sequences of interval-based events that are temporally ordered or out-of-order. The temporal relationships among events are very important in identifying event patterns in intelligent manufacturing. However, existing literatures on time management of events in intelligent manufacturing focus on either ordered interval-based events [2] [3] or out-of-order instant events [4] [5].

It is very important to support interval-based out-of-order event pattern detection in intelligent manufacturing. To the best of our knowledge, no prior work on out-of-order events considers the time intervals of events in intelligent manufacturing. For pattern queries of interval-based 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 interval-based events. In this paper, we propose to combine techniques on addressing the detection of interval-based events and out-of-order events.

Existing research works [6] [7] [8] 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 intelligent manufacturing. There is a need for an efficient algorithm that can solve durable events. Meanwhile, real-time processing of event streams generated from distributed devices is a primary challenge for intelligent manufacturing applications. Most systems [9], either event-based or stream-based, assume events satisfy totally ordered arrivals. However, in intelligent manufacturing 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. 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 event processing systems use NFA to detect event patterns [5]. However, the NFA-based approaches have three limitations: 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.

2) We propose a model of interval-based out-of-order events that includes the logical and physical expression of these events.

3) We use a tree-based query plan structure for event processing that is amenable to a variety of algebraic optimizations.

4) We develop a hybrid solution to solve interval-based out-of-order 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 some preliminaries. Section 4 introduces the model of interval-based out-of-order events. 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 interval-based events respectively. However, to the best of our knowledge, there is no work considering the two aspects together, which is important in intelligent manufacturing.

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 [5], 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 [4] or punctuation [5] is required to "unblock" the on-hold candidate sequences from being output. The introduction of the two techniques is as follows. A naive approach [4] 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 [5] [7]. The authors in [5] 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 [2] [3] [9] [10]. Kam and Fu [10] 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*. [9] proposed the H-DFS algorithm to mine frequent arrangements of temporal intervals. Both these works transform an event sequence into a vertical representation using IDLists. 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 [2] devised an algorithm called TPrefix for mining on-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. TPrefix Span 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 [3] 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. Preliminaries

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] [11]. In the following we will introduce some special attributes of event timestamps in intelligent manufacturing.

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

Time intervals are ordered according to the lexicographic ordering $< H$. Thus, time interval $T = (t_1, \cdots, t_n) < HT' = (t_1' \cdots, t_n')$ iff there exists an i $(1 \le i \le n)$ such that $t_i < T_i t_i'$ and $t_i = t_n'$ for all $j = i + 1, \cdots, n$. Note that if $i = n$, then $t_i = t_n'$. When $T < HT'$, we say that T occurs before T'. If $T = T'$, we say that T occurs simultaneously with T'.

Definition 2 (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: e1, e2, ···, en, where e1.ats < e2.ats < ··· < en.ats. For any two events ei and ej (1 ≤ i, j ≤ n) from S, if ei.ts < ej.ts and ei.ats < ej.ats, we say the stream is an ordered event stream. If however ej.ts < ei.ts and ej.ats > ei.ats, then ej is an out-of-order event.

In the example of **Figure 1**, the timestamps of events e1 - e4 are listed in order. But we can see that event e2 arrives later than event e3, which is called

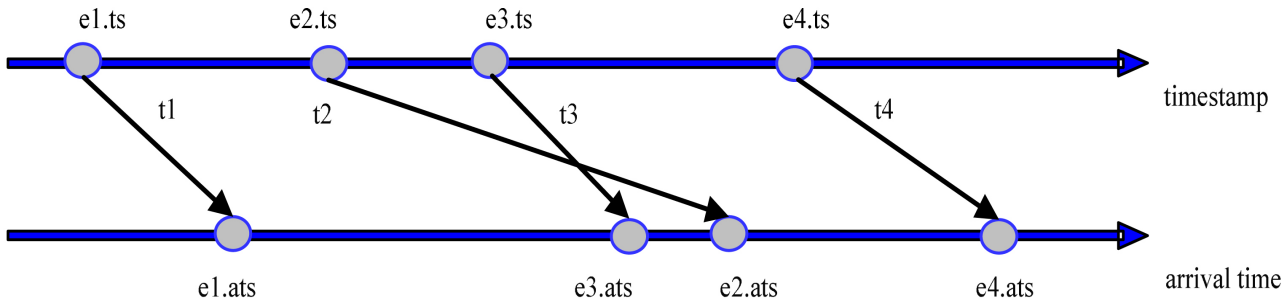

Figure 1. Out-of-order event.

out-of-order.

Different from point-based queries, the expression of interval-based 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 Figure 2. In order to overcome this and distinguish the different interpretations of temporal patterns, the hierarchical representation with additional information is required [3]. 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.

Thus, the temporal patterns in Figure 2 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.

4. The Model of Interval-Based Out-of-Order Events

As shown in Section 3, each event is denoted as (ID, Vs, Ve, Os, Oe, Ss, Se, K). Here Vs and Ve denote the valid start and end time respectively; Os and Oe denote the occurrence start and end time respectively; Ss corresponds to the system clock time upon event arrival; Se means the system clock time when an event ends; K corresponds to an initial insert and all associated retractions, each of which reduces the Se compared to the previous matching entry.

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($(E_i$(relationship)E_j)

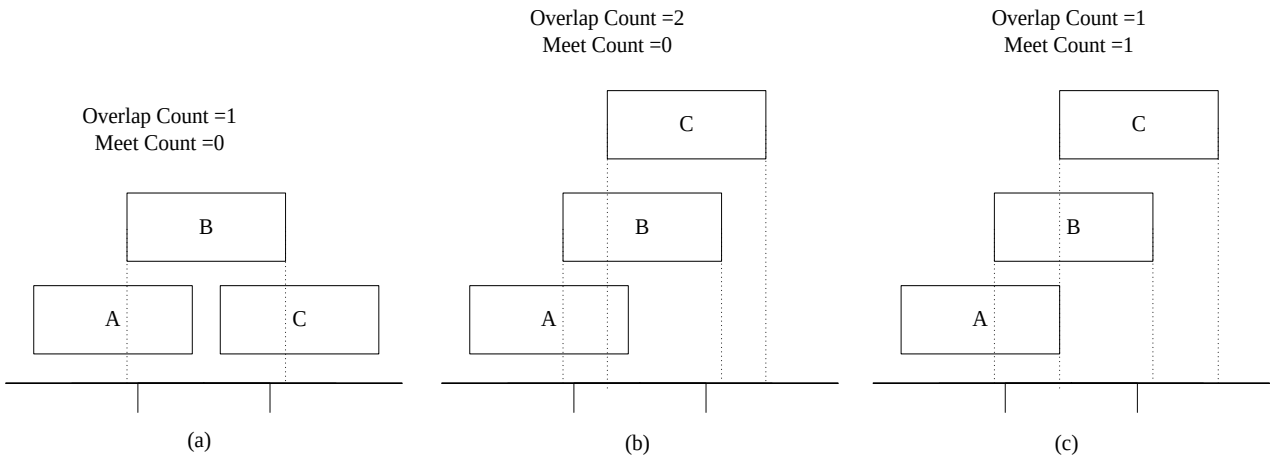

Figure 2. Interpretation of pattern (A (Overlap) B) (Overlap) C.

$(!E_k)(\text{relationship})E_l))(1 \le i, j, k, l \le n)$

relationship:: = {contain, overlap, before, after, meet}

<time constraints>:: = Time Window length W

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) [3]. An event ei is a positive (resp.negative) event if there is no "!" (resp. with "!") symbol used. The Seq/Pal operator denoted Seq/Pal (E1, E2, ···, En, 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 Neg operator specified by Neg (!E1, (time constraint); ···; !Em, (time constraint)) checks whether there exist negative events within the indicated time constraint in a matched positive event pattern. 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. **Figure 3** shows the logical expression of a pattern query Q.

5. Solution Methods

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. So we add an additional attribute ("Real-time Factor") to every query. 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.

5.1. 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 based on the assumption that most data arrives

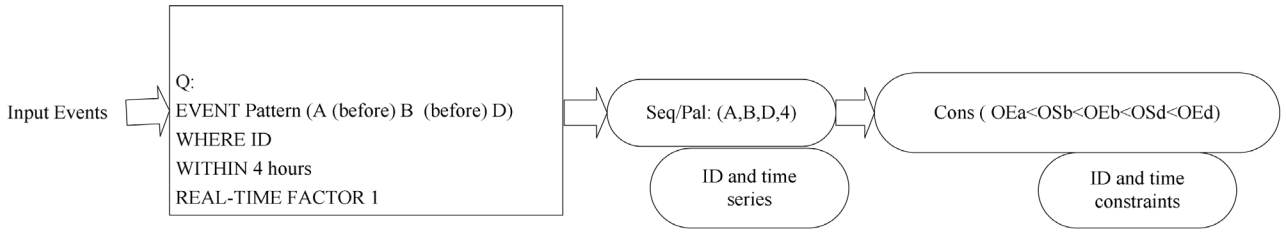

Figure 3. Logical expression of query Q.

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 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 <+, E> is induced by an out-of-order positive event [5], where "E" is a new event result. Deletion <–, E> 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.

For example, the query is (A(overlap)B(!D)(before)C) within 10 min. A unique time series expression of this query {OSa, OSb, OEa, OEb, OSc, OEc} can be obtained based on the above interval expression method. For the event stream in **Figure 4**, when an out-of-order seq/pal event OSb(6) is received, a new correct result {OSa(3), OSb(6), OEa(7), OEb(9), OSc(11), OEc(12)} is output as <+, {OSa(3), OSb(6), OEa(7), OEb(9), OSc(11), OEc(12)}>. When an out-of-order negative event OSd(15) is received, a wrong output result {OSa(13), OSb(16), OEa(17), OEb(20), OSc(22)} is found. So we send out a compensation <–, {OSa(13), OSb(16), OEa(17), OEb(20), OSc(22)}>.

5.2. 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. Considering the time intervals, the methods can be improved as follows.

Based on the event model introduced in Section 4, 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 ei and contains all the virtual edges reachable from ei. 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 ei. One final root-to-leaf path containing the new event ei corresponds to one matched event sequence. If e*i.e*nd time (resp. ei.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 there presentation of relationships among events unique. Compared with the time series of the query, the set of event sequences

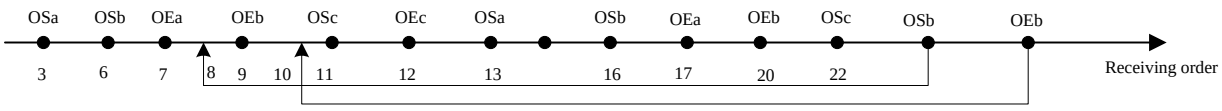

Figure 4. Input events.

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.

The buffer 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 ei kept in the buffer, it can be purged from the stack if (ei.starttime + W) < CLOCK. Thus, under the out-of-order assumption, the above condition will be (ei.starttime + W + K) < CLOCK. This is because after waiting for K time units, no out-of-order events with start timeless than (ei.starttime + W) can arrive. Thus ei can no longer contribute to forming a new candidate sequence.

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.

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 1.

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 about1000 generators, each of which can produce more than 1000 different-type (A, B, C or D) events randomly. Soat 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 1.

6.2. Experimental Results

Figures 5-9 mainly examine the impact of out-of-order percentage Pio3 on the performance metrics. Pio3 is varied from 0% to 45%. Figure 5 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 Figure 5.

Table 1. 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

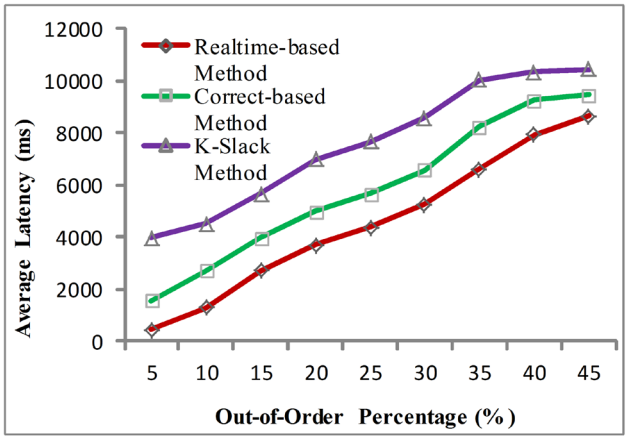

Figure 5. Trend of average latency.

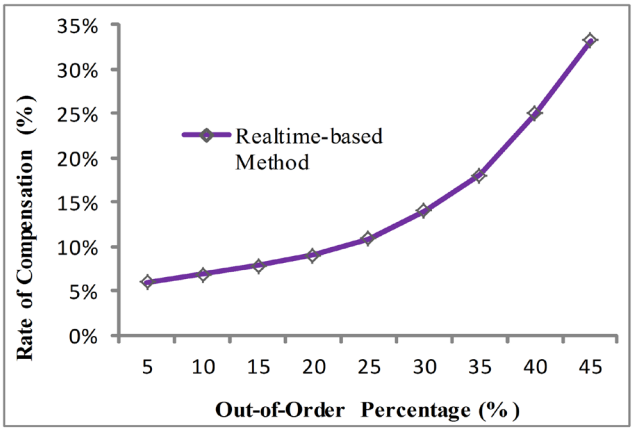

Figure 6. Trend of rate-of-compensation.

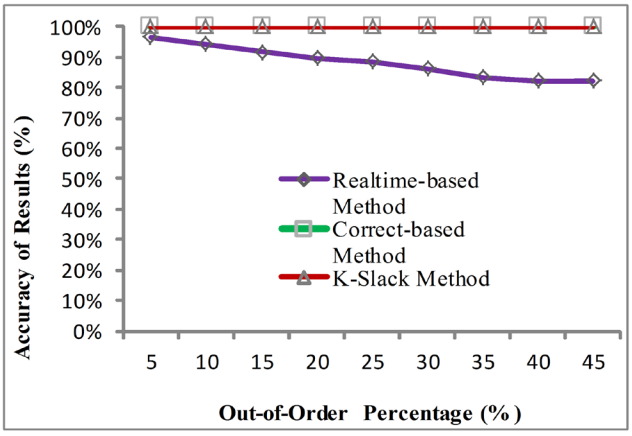

Figure 7. Accuracy without interval-based events.

Figure 6 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

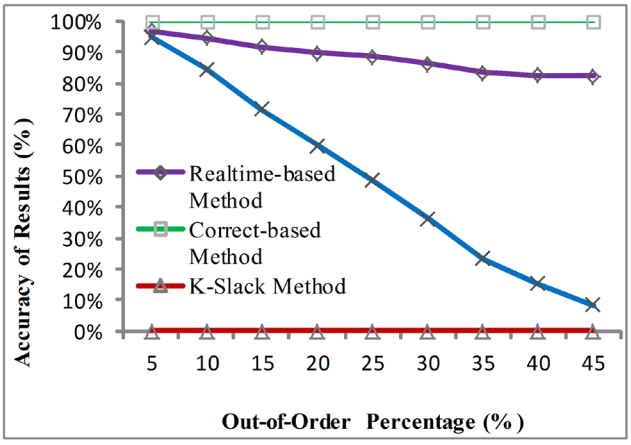

Figure 8. Accuracy with interval-based events.

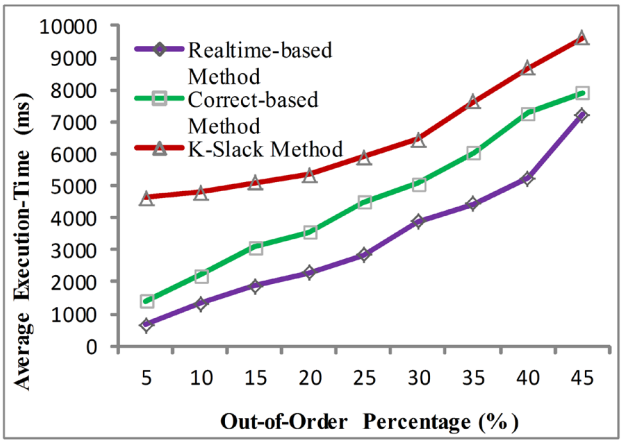

Figure 9. Trend of average execution time.

faster.

The accuracy of results is also examined, defined as (NoCR/NoR). Figure 7 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.

Figure 8 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 Figure 7.

We examine the average execution time in Figure 9, which denotes the summation of operator execution times. When there are no durable events, two ob-

servations 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.

Figures 10-12 show the impact of buffer size on performance metrics. Figure 10 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

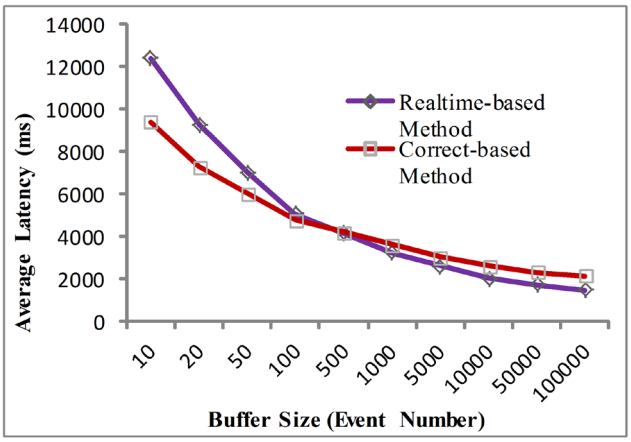

Figure 10. Trend of average latency.

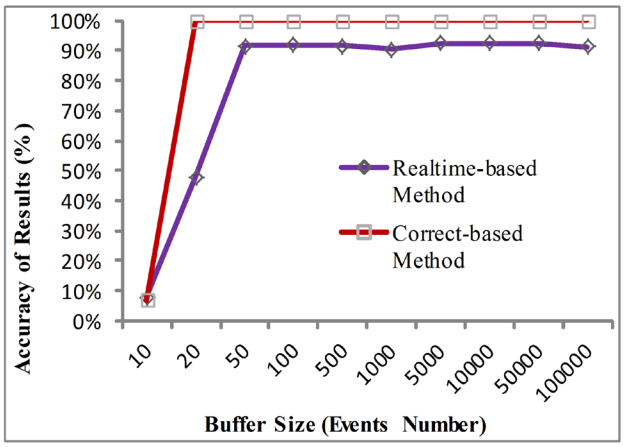

Figure 11. Accuracy of methods.

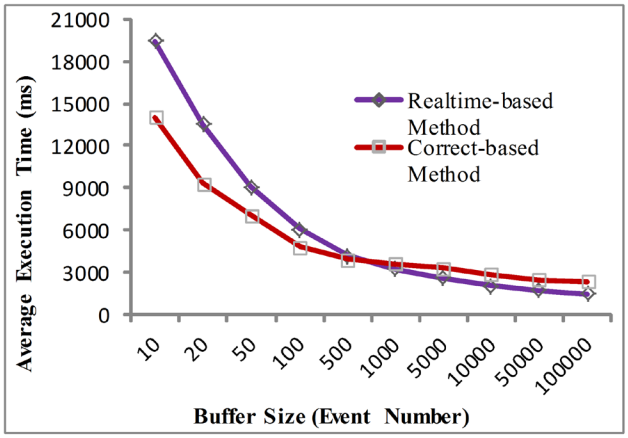

Figure 12. Execution time on buffer size.

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.

Figure 11 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. That is to say, the parameter of buffer size has little effect on accuracy.

The trend of average execution time is shown in Figure 12, 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.

Figure 13 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 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.

Figure 14 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.

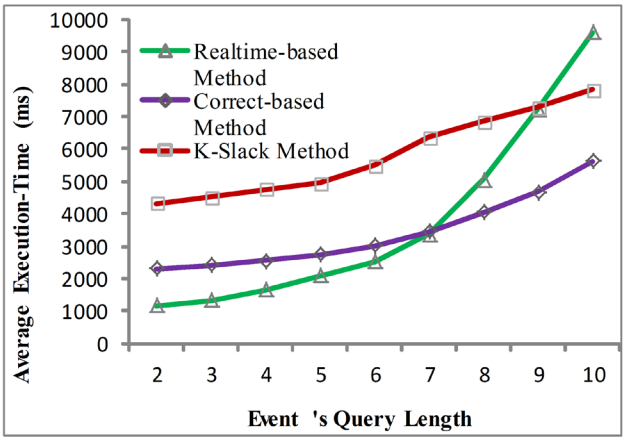

Figure 13. Execution time on event length.

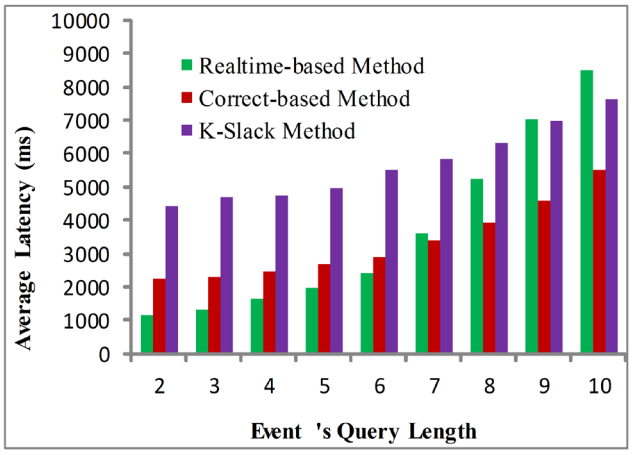

Figure 14. Latency on event length.

7. Conclusion

The goal of this work is to solve query processing of interval-based out-of-order events in intelligent manufacturing. We proposed a tree-plan model of interval-based out-of-order events, which can give the logical and physical expressions. 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.

Acknowledgements

This work was supported by the grants from the National Natural Science Foundation of China (No. 61202111, 61472141, 61273152); the Project of Shandong Province Higher Educational Science and Technology Program (No. J12LN05); the natural science foundation of Shandong province and special projects (Grant No. ZR2013FL009); the Doctoral Foundation of Ludong University (No. LY2012023).

References

[1] Chen, T. (2014) The Symmetric-Partitioning and Incremental Relearning Classification and Back-propagation-Network Tree Approach for Cycle Time Estimation in Wafer Fabrication. *Symmetry*, **6**, 409-426. https://doi.org/10.3390/sym6020409

[2] Wu, S. and Chen, Y. (2007) Mining Nonambiguous Temporal Patterns for Interval-Based Events. *IEEE Transactions on Knowledge and Data Engineering*, **19**, 742-758. https://doi.org/10.1109/TKDE.2007.190613

[3] Patel, D., Hsu, W. and Lee, M.L. (2008) Mining Relationships among Interval-Based Events for Classification. *Proceedings of the* 34*th SIGMOD International Conference on Management of Data* (*SIGMOD*), Vancouver, 10-12 June 2008, 393-404. https://doi.org/10.1145/1376616.1376658

[4] Babu, S., *et al*. (2004) Exploiting K-Constraints to Reduce Memory Overhead in Continuous Queries over Data Streams. *ACM Transaction on Database Systems*, **29**, 545-580. https://doi.org/10.1145/1016028.1016032

[5] Liu, M., Li, M., Golovnya, D., Rundenstriner, E.A. and Claypool, K. (2009) Sequence Pattern Query Processing over Out-of-Order Event Streams. *Proceedings of the* 25*th International Conference on Data Engineering* (*ICDE*), Shanghai, 29 March-2 April 2009, 274-295.

[6] Rani, K. and Mallikarjun, S. (2016) Holi: A Hybrid Model for Neurological Disordered Voice Classification Using Time and Frequency Domain Features. *Artificial Intellegent Research*, **5**, 87-94.

[7] Archimede, B., Letouzey, A., Memon, A. and Xu, J. (2014) Towardsa Distributed Multi-Agent Framework for Shared Resources Scheduling. *Journal of Intelligent Manufacturing*, **25**, 1077-1087. https://doi.org/10.1007/s10845-013-0748-8

[8] Miranville, A., Saoud, W. and Talhouk, R. (2017) Asymptotic Behavior of a Model for Order-Disorder and Phase Separation. *Asymptotic Analysis*, **103**, 57-76. https://doi.org/10.3233/ASY-171419

[9] Papapetrou, P., Kollios, G., Sclaroff, S. and Gunopulos, D. (2005) Discovering Frequent Arrangements of Temporal Intervals. *Proceedings of the* 5*th IEEE International Conference on Data Mining*, Houston, 27-30 November 2005, 354-361.

[10] Kam, P.S. and Fu, A.W. (2000) Discovering Temporal Patterns for Interval-Based Events. *Proceedings of the* 2*nd International Conference on Data Warehousing and Knowledge Discovery*, **1874**, 317-326. https://doi.org/10.1007/3-540-44466-1_32

[11] Aguado, J. Borzacchiello, D., Ghnatios, C., *et al*. (2017) A Simulation App Based on Reduced Order Modeling for Manufacturing Optimization of Composite Outlet Guide Vanes. Advanced Modeling and Simulation in Engineering Sciences, **4**, 11-26. https://doi.org/10.1186/s40323-017-0087-y