Computers and Mathematics with Applications 66 (2013) 1808–1821

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

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

Complex event processing over distributed probabilistic event streams

Y.H. Wang[*], K. Cao, X.M. Zhang

College of Information Science and Engineering, Hunan University, Changsha 410082, China

ARTICLE INFO

Keywords:
Internet of things
Complex event processing
Distributed probabilistic event streams

ABSTRACT

With the rapid development of Internet of Things (IoT), enormous events are produced every day. Complex Event Processing (CEP), which can be used to extract high level patterns from raw data, becomes the key part of the IoT middleware. In large-scale IoT applications, the current CEP technology encounters the challenge of massive distributed data which cannot be handled by most of the current methods efficiently. Another challenge is the uncertainty of the data caused by noise, sensor error or wireless communication techniques. In order to solve these challenges, in this paper a high-performance complex event processing method over distributed probabilistic event streams is proposed. With the ability to report confidence for processed complex events over uncertain data, this method uses probabilistic nondeterministic finite automaton and active instance stacks to process a complex event in both single and distributed probabilistic event streams. A parallel algorithm is designed to improve the performance. A query plan-based method is used to process the hierarchical complex event from distributed event streams. Query plan optimization is proposed based on the query optimization technology of probabilistic databases. The experimental study shows that this method is efficient in processing complex events over distributed probabilistic event streams.

1. Introduction

Internet of Things (IoT) is an integrated part of future internet and could be defined as a dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual "things" have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network. With the help of novel information and communication technologies in recent years, bandwidth, processing power and storage are no longer restricting factors in IoT applications. The key issue is how to process the huge events produced by IoT applications. Consider, for example, the possibility of incomplete data streams and streams including inaccurate data.

In a large-scale IoT application, event processing applications must process events that arrive from various kinds of sources such as sensors which constitute the wireless sensor network, RFID readers, GPS, social media, etc. We call the events that are generated by all these resources as "event cloud". The events that are generated by RFID readers or sensors directly are primitive events. The semantic information inside a primitive event is quite limited and we can only get some simple information from a primitive event. In a real application, we pay more attention to complex information such as business logic and rule. For example, each reading operation of the RFID reader at a garage generates a primitive event but a

[*] Corresponding author. Tel.: +86 13975813465.
E-mail addresses: yh.wang.cn@gmail.com (Y.H. Wang), caokening@sina.com (K. Cao), zhangxm19712003@yahoo.com.cn (X.M. Zhang).

0898-1221/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.camwa.2013.06.032

complex event like ''the car leaves the garage'' is the kind of event that a user is really concerned with. To get such a complex event, we need to combine many primitive events based on some rule. An IoT application system converts business logic into complex events and then detects business logic based on detecting complex events. Complex Event Processing (CEP) [1] is used to process huge primitive events and get valuable information from them.

Complex event processing has been studied widely in active database. Most of the methods are based on fixed data structure such as tree, graph, finite automaton or Petri net [2–4]. But those methods cannot support optimizing event query language flexibly and extending event query language according to the change of requirement. In order to resolve those problems some query plan-based method was proposed, such as SASE [5]. SASE is optimized for large sliding windows and large intermediate result size to get high performance and scalability. But the original SASE method cannot process the hierarchical complex event and it cannot get acceptable performance if the sliding window is very large. The traditional methods are also not able to process distributed and uncertain event streams.

Many state-of-the-art systems assume that data is precise and certain, or that it has been cleansed before processing. As a result, processing this data is deterministic in nature. However, in many real-time IOT applications, cleansing the data is not feasible due to time constraints. Imprecise primitive events can also be caused by other factors such as the limitation of measuring accuracy, signal disturbance or privacy protection. The uncertainty is usually treated with probabilities. Therefore it is important to develop an event processing engine that be able to deal with probability. Much work with probabilistic databases has been done to process probability in databases [6,7]. Recently some papers about probabilistic complex event processing have been proposed [8–10]. Most of these methods extend the CEP engine based on NFA to process probabilistic events. Some methods for optimization of the query plan are also proposed. Mainly there are two challenges in processing the probabilistic event stream: (1) how to detect complex events satisfying query requirements from a large number of possible worlds in high-speed and real-time updating stream; (2) how to calculate the probability of a complex event aggregated from correlated uncertain events.

Big data is recognized as one of the three technology trends at the leading edge a CEO cannot afford to overlook in 2012 [11]. Big data is characterized by volume, velocity, variety and veracity (data in doubt). Internet of things is one of the important areas that need to process big data. In this paper we propose a high-performance Distributed Probabilistic Complex Event Processing method (DPCEP). DPCEP extends the original method of SASE based on Nondeterministic Finite Automaton (NFA) and Active Instance Stacks (AIS). A parallel algorithm is designed to improve the performance for both single and distributed streams. A query plan-based method is used to process the hierarchical complex event from distributed event streams. Query plan optimization is proposed based on the query optimization technology of probabilistic databases.

The remainder of this paper is organized as follows. In Section 2, we give a general review of the related work of complex event processing technology and probabilistic event processing methods. In Section 3, we describe the event model and system architecture. The DPCEP method is described in detail in Section 4. In Section 5, we introduce our experiments for DPCEP. Then a discussion and conclusion is presented in Section 6.

2. Related work

2.1. Complex event processing

Complex event processing recognizes complex events based on a set/sequence of occurrences of single events by continuously monitoring the message flow, and then reacts to those detected situations. In their book, Etzion et al. defined the basic concept and architecture of complex event processing [12]. Event Processing Agent (EPA) is a component that, given a set of input events, applies some logic to generate a set of complex events as output. Event Processing Network (EPN) is a network of a collection of EPAs, event producers, and event consumers linked by channels. The network is used to describe the event processing flow execution. Luckman introduced the event processing network first in the field of modeling [1]. The conceptual model of EPN based on this idea was further elaborated by Sharon and Etzion [13]. A context is a named specification of conditions that groups event instances so that they can be processed in a related way [12].

The key ideas of complex event detection have four steps. (1) Primitive events are extracted from large volume data. (2) Event correlation or event aggregation is detected to create a business event with event operators according to specific rules. (3) Primitive or composite events are processed to extract their time, causal, hierarchical and other semantic relationships. (4) Response is sent to the actionable business information because of guaranteed delivery of events to the subscribers.

Most of the CEP methods in active databases and RFID applications use fixed data structure. Yuanping et al. used a tree-based CEP method and optimized the algorithm by event grouping [2]. Fusheng et al. used a directed graph to process the complex event in RFID stream [3]. Xingyi et al. used TPN (Timed Petri-Net) to detect the complex event from RFID stream [4]. Recently, there has been still much work on Petri-Net-based CEP since an uncertain event is important in many applications. For the CEP methods with fixed data structures, it is difficult to optimize the implementation of the event query flexibly. It is also hard to support the extension of event query language according to the change of application.

SASE [5] is a high-performance complex event detection method which uses NFA and AIS. SASE is optimized for large sliding windows and large intermediate result size to get high performance and scalability. But SASE does not support the hierarchical complex event and it cannot handle distributed event streams. Recently some researchers extended SASE to make it more powerful. Jagrati et al. proposed an improved NFA model to support more powerful query ability [14]. In Haopeng's work, the SASE model is improved to support imprecise timestamps when processing complex events in stream [15].

The traditional centralized CEP architecture requires greater bandwidth and computational capability as the number of clients increases, and the system is hardly robust and scalable because of single point failure or network break. On the other hand, some IOT applications are naturally distributed and we need to detect complex events from the distributed system. Therefore recently some work on distributed CEP has been proposed. Plan-based CEP across distributed sources has been studied by Mert et al. [16]. Tao et al. proposed a method for distributed complex event processing for RFID application [17]. Compared with our work, these methods do not support uncertain events.

2.2. Probabilistic event processing

Probabilistic database systems propose new data models, extending the traditional relational model, able to capture incomplete or imprecise data usually appearing in the physical world. In the work of [6,7], the main idea is that the state of the database, i.e. the instance I, is unknown. Instead the database can be in any one of a finite number of possible instances $I_1, I_2, \ldots,$ called possible worlds, each with some probability and the probabilities of all the possible worlds must sum up to 1. A systematic effort is made to enable traditional relational operators to handle values with probabilities. Trio project [18] defines a probabilistic model called ULDB. They exploit lineage in ULDB to avoid the "safe" problem in [7] and make confidence computation efficient.

ProbLog is a famous probabilistic logic programming framework to support event recognition under uncertainty [19]. Probabilistic facts in a ProbLog program represent random variables and ProbLog makes an independence assumption on these variables. ProbLog uses Event Calculus (EC) which is a logic programming language for representing and reasoning about events and their effects [20]. ProbLog also uses Binary Decision Diagrams (BDDs) [21] to compactly represent the proofs of a query. A BDD is a binary decision tree in which redundant nodes are removed and isomorphic sub-trees are merged. But ProbLog is not designed for detecting high level event patterns from low level event streams as CEP does.

A CEP engine needs to process streams of events with timestamp and, therefore, numerous event pattern recognition methods are based on sequential variants of probabilistic graphical models, such as Hidden Markov Models [22], Dynamic Bayesian Networks [23] and Conditional Random Fields [24]. Markov Logic Networks (MLNs) [25] have also been used to handle the uncertainty in event pattern recognition. MLNs combine first-order logic and probabilistic graphical models. With the help of first-order logic, event patterns including complex (temporal) constraints are supported. An important feature of MLNs is that they are supported by a variety of machine learning algorithms. Biswas et al. used MLNs for event pattern detection from uncertain event stream [26]. In the work of Sadilek et al. [27], hybrid-MLNs are used to recognize successful and failed interactions between multiple humans using noisy location data. Just like pure MLN methods, the knowledge base of hybrid-MLNs consists of composite event patterns. However, hybrid formulas are also included to remove the noise from the location data. A method that uses interval-based temporal relations is proposed in [28]. The aim of the method is to determine the most consistent sequence of composite events based on the observations of event producers. The method uses MLNs to express composite events using common sense rules. Compared with our work, the current methods based on probabilistic graphical models are not efficient enough when processing massive distributed event streams in large-scale IoT applications.

Recently some work on detecting complex events in probabilistic event stream based on NFA has been proposed. In the work of [9], a data structure called Chain Instance Queues (CIQ) is used to detect complex events satisfying query requirements with single scanning probabilistic stream. Conditional Probability Indexing-Tree (CPI-Tree) is defined to store the conditional probabilities of a Bayesian network to improve the performance. In another paper [10], an optimized method is proposed to not only calculate the probability of outputs of compound events but also obtain the value of confidence of the complex pattern given by a user against uncertain raw input data stream generated by distrustful network devices. This method is based on an existing stream processing engine SASE+, and extends its evaluation model NFA^b automaton to a new type of automaton in order to manage the runtime against probabilistic stream. In the work of [8], a query language is designed which allows users to express Kleene closure patterns when detecting probabilistic events. This method uses a new data structure AIG to detect sequence patterns over probabilistic data streams. With the benefit of lineage, the probability of an output event can be directly calculated without considering the query plan. These NFA-based methods are the closest ones to our work but just like SASE, these methods do not support hierarchical complex event and distributed event streams. Furthermore, we use parallel methods to improve the performance and query plan-based optimization methods to handle the event type query.

3. Event model and system architecture

In order to illustrate the event model and function of the system, we provide an example of the transporting system based on Internet of Things. Using RFID, Radar, GPS and camera, the system can get information of vehicles such as ID, location, speed. Much other information such as temperature and brightness can be obtained through the wireless sensor network.

Using the CEP engine in this paper a user can query the system to get useful event patterns which can be used for decision support.

Mainly there are two types of uncertainty in complex event processing: instance uncertainly and attribute uncertainly. In this paper we only deal with instance uncertainly.

Definition 3.1 (*Probabilistic Primitive Event*)**.** A primitive event in a stream means an atomic occurrence of interest in time. A probabilistic primitive event is represented as $\langle A, T, \mathrm{Pr}\rangle$ where A is the set of attributes and T is the timestamp that the event occurs. Pr is the concrete probability value used to present the occurrence probability of the event. The probability value represents the possibility that one event is converted accurately from truthful data of nature to digital data used for computing in electronic devices.

In the transporting system example, each read operation of the devices generates a primitive event. Sometimes the primitive event is not certain. For example, two RFID readers may find the same object at the same time. It may also be uncertain when a car accident is detected through a camera. A probability value is used to represent such an uncertainty.

Definition 3.2 (*Probabilistic Complex Event*)**.** A complex event is a combination of primitive events or complex events by some rule. A probabilistic complex event is represented as $\langle E, R, \mathrm{Ts}, \mathrm{Pr}\rangle$ where E represents the elements that compose the complex event, R represents the rule of the combination, Ts represents the time span of the complex event and Pr is the probability value.

Definition 3.3 (*Event Type*)**.** An event type is a specification for a set of event objects that have the same semantic intent and the same structure. Every event object is considered to be an instance of an event type. An event type can represent either primitive events deriving from a producer or complex events produced by an event processing agent.

In this paper, the event type is represented by a capital letter such as A. A primitive event is represented by a lowercase letter and number, e.g., a_1 where 1 is the timestamp. Like in SASE, we assume that the timestamps of events give rise to a natural total order.

In the following we define some operators that can be used to query a complex event. A_i represents an event type and $A_i(t) = \text{True}$ means an instance of A_i occurred at time t.

Definition 3.4 (*ANY Operator*)**.** $\mathrm{ANY}(A_1, A_2, \ldots, A_n)(t) \equiv \exists\, 1 \le i \le n\, A_i(t)$. The ANY operator selects any event that occurs at time t from the input event sequence.

Definition 3.5 (*SEQ Operator*)**.** $\mathrm{SEQ}(A_1, A_2, \ldots, A_n)(t) \equiv \exists\, t_1 < t_2 < \cdots < t_n = t\, A_1(t_1 \wedge A_2(t_2) \wedge \cdots \wedge A_n(t_n))$. The SEQ operator selects a given sequence of events from the input event sequence.

Definition 3.6 (*AND Operator*)**.** $\mathrm{AND}(A_1, A_2)(t) \equiv A_1(t) \wedge A_2(t)$. If two input events occur at the same timestamp t, the AND operator outputs the two events without order.

Definition 3.7 (*¬ operator*)**.** $\neg(A_1) \equiv A_1(t) = \text{False}$. The '¬' operator means that the event does not exist at a given timestamp t.

Definition 3.8 (*FIRST Operator*)**.** $\mathrm{FIRST}(A) \equiv \{a_i(t_i) \in A \mid \forall t_j \ne t_i, t_j > t_i\}$. The FIRST operator selects the first event from an event sequence.

Definition 3.9 (*LAST Operator*)**.** $\mathrm{LAST}(A) \equiv \{a_i(t_i) \in A \mid \forall t_j \ne t_i, t_j < t_i\}$. The LAST operator selects the last event from an event sequence.

Now we provide the definition of query language in DPCEP.

Definition 3.10 (*Complex Event Query Language*)**.** The query language structure of DPCEP is defined as

```
EVENT <event pattern>
[WHERE <qualification>]
[WITHIN <time_window>]
[GROUP BY <attribute_set>]
[HAVING <confidence_qualification>].
```

The basic query is the event instance query which returns the event instances that satisfy the qualification. An example is shown by Q1. In the event instance query, there is an event instance name after the event type, like '*A a*'. The having clause can be used to specify a threshold value for the probability of the result. In this example, assume event type A, B and D mean the RFID readers at location L_A, L_B and L_D find vehicles. Event type O means overspeed at location L_A. Then the meaning of

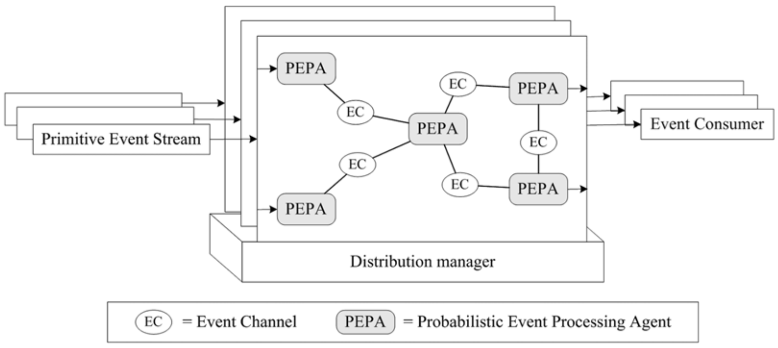


Fig. 1. The system architecture of DPCEP.

Q1 is: query all the vehicles which passed L_A, L_B and L_D in sequence and ran overspeed at L_A within 12 h with confidence larger than 0.8.

```
Q1:
EVENT AND (O o,(SEQ(A a, B b, D d)))
WHERE a.id = b.id ∧ b.id = d.id ∧ o.id = d.id
WITHIN 12 hours
HAVING CONF(∗) > 0.8.
```

Besides the event instance query, the query language also supports the event type query. In event type query, like Q2, there is no event instance name after the event type. Query Q2 returns the joint distribution of event type A and B. In the transporting system example, assume that *O* means overspeed and *X* means accident. Then the meaning of Q2 is: query the joint distribution of vehicles that ran overspeed and encountered accident within 12 h.

```
Q2:
EVENT AND(O, X)
WHERE O.location = X.location
WITHIN 12 hours.
```

In some applications we need to support the group query of complex events. As an example consider query Q3. The probability for group is called rank. In the transporting system example, assume that *O* and *X* have the same meaning as Q2. The semantic of Q3 is: query the joint distribution of vehicles that ran overspeed and encountered accident within 12 h and group the result according to the type of vehicles.

```
Q3:
EVENT AND(O, X)
WHERE O.location = X.location
WITHIN 12 hours;
GROUP BY O.type.
```

The architecture of the system is shown in Fig. 1. As described in [12], there are different types of EPA, such as filtering, matching, and derivation. In this paper we only consider event matching EPA which is the most important one. PEPA is the extension of EPA that supports probabilistic event processing and we also call it probabilistic CEP engine. Multiple PEPAs are connected by an event channel to create an EPN. An event channel is a processing element that receives events from one or more source processing elements, makes routing decisions, and sends the input events unchanged to one or more target processing elements in accordance with these routing decisions. In this paper routing is not considered and we assume that a channel is manifested as an edge in the graph. We assume that the primitive event streams are distributed and processed at different nodes. The distribution manager is designed to connect PEPAs at different nodes to support distributed complex event processing.

4. Distributed probabilistic CEP

4.1. Parallel probabilistic CEP over single event stream

SEQ event detection is the main part of the CEP engine. We first consider the SEQ event processing for a single event stream. The basic SEQ event detection method uses NFA and AIS [5]. The event detection is started at the "start" state of the

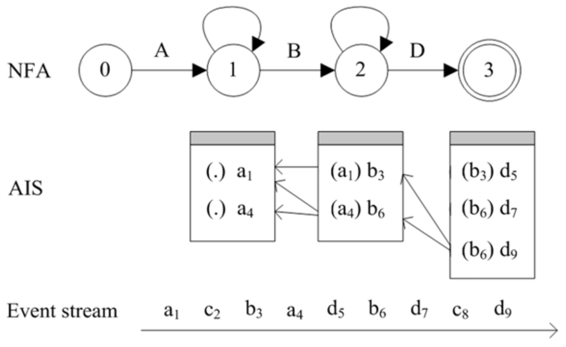


Fig. 2. Example of the basic method for SEQ event processing.

NFA and it enters the next state when an event is scanned. When the "end" state of the NFA is entered, the detection enters the accepting state which means that the SEQ event is detected. In order to improve the performance, the events that trigger the change of states are logged into AIS and an event link between neighboring states is created. We can search the event link in the AIS to detect the SEQ event.

Example 4.1 (*Basic Method for SEQ Event Processing*)**.** The NFA and AIS that are created for SEQ(A, B, D) are shown in Fig. 2. We can easily find that the SEQ events that end with d_9 are (a_1, b_3, d_9), (a_1, b_6, d_9), (a_4, b_6, d_9).

Using the method of basic SEQ event processing, if the event stream has large amount of data and the sliding window is very large, the performance is not acceptable for application that needs fast response. We propose a parallel event detection method to improve the performance. The parallel method is based on data partition. The data is partitioned into N (N is a number that can be set which is usually related to the number of processors or calculating nodes) partitions and we run a CEP process for each partition. The AIS stacks of different partitions can be connected to produce global result. The idea of the algorithm is as follows.

(1) The controlling process partitions the main task into N subtasks and starts N task processes for subtasks.
(2) Task process P_i runs the basic SEQ event detecting method. The input is the data partition P_i and the result of the subtask is outputted directly.
(3) The controlling process connects the AIS that created at each subtask and output the remainder results based on the global AIS.

In step 2, the result of the subtask is outputted directly because it must be part of the final result. The main problem is how to connect the AIS of the subtasks.

Lemma 4.1 (*AIS Connection in Parallel SEQ Event Detection*)**.** *Let $P_1, P_2, \ldots, P_n$ represent n task processes and NFA has k states $0, 1, \ldots, k-1$, then the events which are related to state j $(1 < j \leqq k-1)$ in the AIS of partition P_i $(1 < i \leqq n)$ can be linked to the events which are related to state $j-1$ in the AIS of partition p_{i-1}.*

Rationale. The timestamp of the events in P_i is larger than the timestamp of the events in p_{i-1} because we assume that the input events are ordered and the partition is based on timestamp. Since all the events in the AIS of p_{i-1} which are related to state $j-1$ of NFA occur before the events in the AIS of p_i which are related to state j of NFA, corresponding links between the events can be created based on the rule of AIS creation.

After linking the AIS in subtasks and creating total AIS, we can search the total AIS and get the global result of the event detection. Now we analyze the performance of the parallel method. Since the parallel method and nonparallel method create the same number of links, we can ignore the time for creating links in the AIS. Let Tl_i represent the searching time of local result in process i and Tg represent the searching time for global result. The running time of the nonparallel method can be calculated by $\sum_{i=1}^{N} Tl_i + Tg$. For the parallel CEP in a single stream, assume that we use a server with multiple processors and shared memory. The communication cost can be ignored and the total running time of the parallel method can be calculated by $\arg\max_{i\in(1,\ldots,N)} Tl_i + Tg$. We can see that the parallel method will be much more efficient than the normal method if the data is properly partitioned.

There are different kinds of dependency relationships among uncertain events in a complex event processing system. In this paper, we assume that the event instances from different event streams are independent. For an event type query, like Q2, we assume that the dependency relationship is unknown which means that the random variables may or may not be independent. We use a possible world model to process such a case which is described in detail in 4.3. When calculating SEQ events in a single stream, some of the primitive events have Markov property which means that the probability of the next event in the sequence is only related to the current event in the sequence but has nothing to do with previous events. For example, in supply chain management the location probability of an object at time $i+1$ depends on the location at

Conditional Event	Probability
$b_3 \mid a_1$	0.6
$b_6 \mid a_1$	0.7
$d_7 \mid b_3$	0.9
$b_9 \mid a_8$	0.6
$d_{14} \mid b_9$	0.8
$b_{13} \mid a_1$	0.7
$d_{14} \mid b_6$	0.9
...	...

Fig. 3. Part of the condition probability table.

time i and the information detected by some reader at time $i + 1$. Like the work of [9], we use Condition Probability Table (CPT) to save and sort the condition probability. Note that the probabilistic distribution and the CPT are obtained through machine learning from historical data which is beyond the scope of this paper. The probability of the sequence consists of these Markovian events can be calculated by Bayesian formula. Besides these Markovian events, we assume that there might be some primitive events that are independent of others. Therefore, we can calculate the probability of an SEQ event based on Definition 4.1.

Definition 4.1 (*Probability Calculation of an SEQ Event in a Single Stream*)**.** In an SEQ event $E = \text{SEQ}(e_1, e_2, \dots, e_n)$; the primitive events are partitioned into two sets S and T. Set T contains the independent primitive events while set S contains one or more Markov chain. The probability of SEQ event can be computed as follows:

$$\Pr(E) = \prod_{e_j \in T} \Pr(e_j) \cdot \prod_{s_i \in S} \left(\Pr(e_{i1}) \prod_{m=1}^{|s_i|-1} \Pr(e_{m+1} \mid e_m) \right) \tag{4.1}$$

where s_i means the Markov chain in set S and e_{i1} means the first event in Markov chain s_i. The conditional probability $\Pr(e_{m+1} \mid e_m)$ can be found in the condition probability table.

The extended AIS is called Probabilistic AIS (PAIS). An example is shown in Example 4.2.

Example 4.2 (*Parallel Probabilistic SEQ Event Processing*)**.** Assume that the input event stream with probability is $a_1(0.6), c_2(0.7), b_3(0.5), a_4(0.9), d_5(0.8), b_6(0.6), d_7(0.8), a_8(0.7), b_9(0.5), c_{10}(0.9), a_{11}(0.6), d_{12}(0.8), b_{13}(0.6), d_{14}(0.7)$. Part of the condition probability table is shown in Fig. 3. The NFA and AIS that are created for SEQ(A, B, D) are shown in Fig. 4. The event stream is partitioned into two sub-streams so that they can be processed in parallel. Based on Definition 4.1 we can get $\Pr(a_1, b_3, d_7) = 0.6 \times 0.6 \times 0.9 = 0.324$ and $\Pr(a_1, b_{13}, d_{14}) = 0.6 \times 0.7 \times 0.9 = 0.378$. By searching the local AIS and global AIS we can find all the instances of SEQ(A, B, D) with probability.

In the real application, usually there is a threshold for the probability of the resulting complex event. If there is a threshold, the following heuristic methods are used to improve the performance. (1) If the probability of a primitive event is less than the threshold, it will not be added into the PAIS. (2) When searching the AIS event link, the probability of the current path is calculated. If this value is less than the threshold, then the searching process will be terminated.

4.2. Probabilistic CEP over distributed event streams

In DPCEP, there are two methods to support a distributed SEQ event: moving event data and moving temporal result. In the moving event data method, we first partition the data window according to the starting time points and ending time points of event streams. For example, in Fig. 5 there are four event streams at different nodes. Each event stream is ordered. According to the starting time points and ending time points, the entire data window is partitioned into 7 fragments. In order to process the data in parallel using the method in 4.1, the data moving plan should make sure that there is no data overlap after moving. If one time fragment contains data of more than one stream, select the node that has the maximum data size as the main node and move all data in other nodes on that time fragment into the main node. Assume that there are r fragments after partition; the data size that needs to be moved is $d_1, d_2, \dots, d_r$; then the total data moving size is $D = \sum_{i=1}^{r} d_i$. Obviously this value is related to the overlapping rate of the event streams. After data moving, we can use the method in 4.1 to process the SEQ event in parallel. Unlike the parallel method in a single stream, in distributed processing we need to select a node with largest PAIS as the main node and send the PAIS of other nodes to the main node.

The moving event data method can be inefficient if the overlapping rate is large. The moving temporal result method does not need to move the primitive event data. In this method, the PAIS is created at each node according to the local event stream and local result is outputted. Then select the node with the largest PAIS as the main node and ask other nodes to

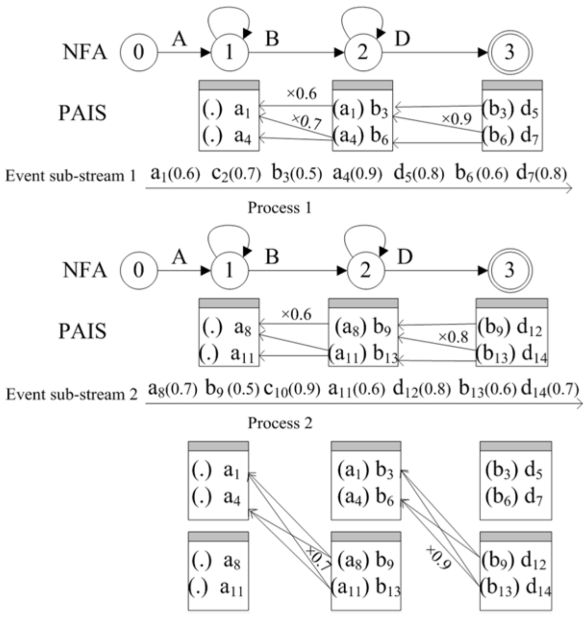

NFA
A
B
D
PAIS
×0.6
×0.7
×0.9
Event sub-stream 1 a1(0.6) c2(0.7) b3(0.5) a4(0.9) d5(0.8) b6(0.6) d7(0.8)
Process 1
×0.8
Event sub-stream 2 a8(0.7) b9(0.5) c10(0.9) a11(0.6) d12(0.8) b13(0.6) d14(0.7)
Process 2

Fig. 4. Example of parallel probabilistic CEP.

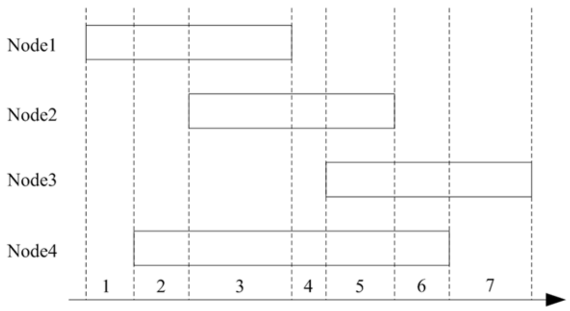


Fig. 5. Data distribution on different nodes.

send their PAIS to the main node. The main node merges the PAIS from other nodes into its own PAIS and creates new links using Algorithm 4.1.

Algorithm 4.1: create new links for distributed PAIS
Input: merged PAIS
Output: PAIS with new links
Method:

```
(1) For i = 2 to stack_number
(2)     For j = 1 to Count($S_i$)
(3)         For m = 1 to Count($S_{i-1}$)
(4)             If(IsLinked($S_{i-1,m}$, $S_{i,j}$) == false and Time($S_{i-1,m}$) < $S_{i,j}$)
(5)                 CreateLink($S_{i-1,m}$, $S_{i,j}$)
(6)             Else break
```

In Algorithm 4.1, for each instance in the PAIS, search the previous stack and create links (if there are none) with the instances that have a smaller timestamp. An example is shown in Fig. 6 (the probability is omitted).

Assume that there are k SEQ events in the result $CE_1, CE_2, \ldots, CE_k$. The complex event CE_i contains primitive event type $E_{i1}, E_{i2}, \ldots, E_{im}$. The total size of data moving can be estimated by Eq. (4.2) where count(E_{ij}) calculates the event number of

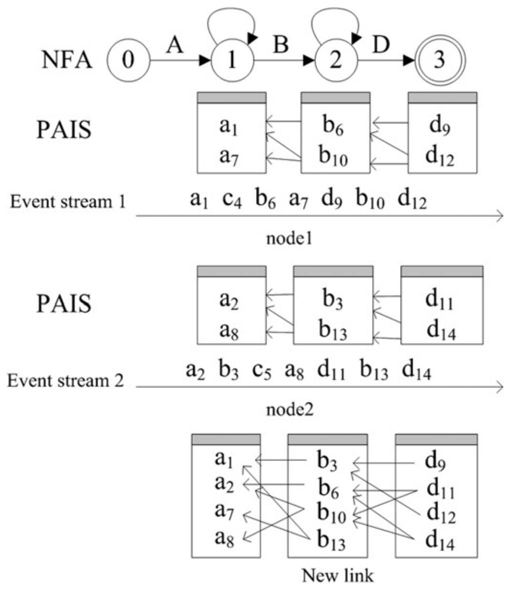
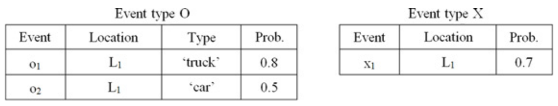


Fig. 6. Creating new links for distributed PAIS.


Event type O

Event	Location	Type	Prob.
o_1	L_1	'truck'	0.8
o_2	L_1	'car'	0.5

Event type X

Event	Location	Prob.
x_1	L_1	0.7

Fig. 7. Probabilistic events in different streams.

the primitive events of type E_{ij} and λ_{ij} means the rate that events of type E_{ij} are instances in PAIS:

$$D = \sum_{i=1}^{k} \sum_{j=1}^{im} \text{count}(E_{ij})\lambda_{ij}(N-1)/N. \tag{4.2}$$

As we can see, the moving temporal result method has less data moving than the moving event data method but the algorithm complexity of the former is higher. In real application, we prefer the moving temporal result method since data moving is the main cost to be considered. But sometimes using the moving event data method can get higher performance if the total data moving is not large and there are many queries on the same data window.

4.3. Hierarchical probabilistic CEP over distributed event streams

The traditional method with NFA does not support hierarchical complex event detection. In order to support hierarchical complex event detection across distributed event streams, we use the EPN as shown in Fig. 1. One EPA can process some type of complex event and send the result to another EPA to process the higher level complex event.

According to our system architecture, the leaf nodes in the EPN can come from different event streams. For the event instance query like Q1, we assume that the events in leaf nodes are independent and the joint probability is easy to calculate. But for the event type query like Q2, in order to calculate the probability of complex event, we need to use some query plan and evaluation technology of the probabilistic database. In the probabilistic database, an important problem is how to calculate the probability when joining two tables. We can regard event type in our event model as table in the probabilistic database. Then the probability calculation problem of "SEQ" and "AND" operator is converted into the calculation of the probability of join operator in the probabilistic database. We now define the possible world event database which is used to calculate the probability.

Definition 4.2 (*Possible World Event Database*)**.** A possible world event of type A is a probability distribution on the set of all deterministic events of type A. A possible world event database of complex event CE is a probability distribution on all possible event instances that satisfy pattern CE.

The example of Q2 is shown in Fig. 7. Event type O and X are from different event streams. Each event type has some instances with probability. In order to calculate the probability of AND(O, X), we create the possible world event database as shown in Fig. 8. The probability of D_1 is calculated like the following: $\Pr(D_1) = \Pr(o_1)*\Pr(o_2)*\Pr(x_1) = 0.8*0.5*0.7 = 0.28$.

World	Prob.
$D_1=\{o_1, o_2, x_1\}$	0.28
$D_2=\{o_1, x_1\}$	0.28
$D_3=\{o_2, x_1\}$	0.07
$D_4=\{x_1\}$	0.07
$D_5=\{o_1, o_2\}$	0.12
$D_6=\{o_1\}$	0.12
$D_7=\{o_2\}$	0.03
$D_8=\phi$	0.03

Fig. 8. Possible world event database.

Event type O

Event	Location	Type	Prob.
A_1	L_1	‘truck’	0.8
A_2	L_1	‘car’	0.5

Event type X

Event	Location	Prob.
B_1	L_1	0.7
B_2	L_1	0.8
B_3	L_2	0.9

Fig. 9. Probabilistic events' example for Q3.

The probability of D_3 is calculated by $\Pr(D_3) = \Pr(\neg o_1) * \Pr(o_2) * \Pr(x_1) = (1 - 0.8) * 0.5 * 0.7 = 0.07$. In the possible world event database both A and B are included in D_1, D_2 and D_3. So $\Pr(\text{AND}(O, X)) = \Pr(D_1) + \Pr(D_2) + \Pr(D_3) = 0.63$.

As we can see, if the number of instances is large, the possible world event database can be very large. In order to improve the performance, we use Definition 4.3 to calculate the probability of $\text{AND}(A, B)$ or $\text{SEQ}(A, B)$.

Definition 4.3 (*Joint Probability Calculation*)**.** Assume that an event type A has n instances $a_1, a_2, \ldots, a_n$; the joint possibility can be calculated by

$$\Pr\left(\prod(A)\right) = 1 - \prod_{i=1}^{n} \Pr(\neg a_i) \tag{4.3}$$

$$\Pr(\text{AND}(A, B)) = \Pr\left(\prod(A)\right) \times \Pr\left(\prod(B)\right) \tag{4.4}$$

where $\prod(A)$ and $\prod(B)$ mean the projection on event type. For the example in Figs. 7 and 8, $\Pr(\prod(O)) = 1 - \Pr(\neg o_1) * \Pr(\neg o_2) = 1 - 0.2 * 0.5 = 0.9$ and $\Pr(\text{AND}(O, X)) = \Pr(\prod(O)) * \Pr(\prod(X)) = 0.9 * 0.7 = 0.63$.

4.4. Rank calculation

For the rank query like Q3, another probabilistic events' example is shown in Fig. 9. Using the possible world event database, we can get the rank table but this method is inefficient.

In order to improve the performance when the possible world database is large, we present a query plan based on Definition 4.4.

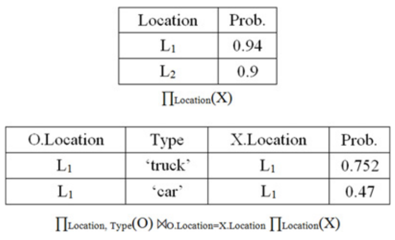
Definition 4.4 (*Optimized Plan for Rank Calculation*)**.** Assume that θ is the qualification that connects event type A and B. $A_\theta \in A$ and $B_\theta \in B$ are the attribute sets that are included in θ. $A_g \in A$ is the attribute that is used to group the result and $A_g \cap A_\theta = \Phi$. The rank of complex event $\text{AND}(A, B)$ can be calculated by the following plan:

$$\prod_{A_g}\left(\prod_{A_g \cup A_\theta}(A) \bowtie_\theta \prod_{B_\theta}(B)\right). \tag{4.5}$$

For the example of Q3 and Fig. 10, the corresponding query plan is $\prod_{\text{Type}}(\prod_{\text{Location,Type}}(O) \bowtie O.\text{Location} = X.\text{Location} \prod_{\text{Location}}(X))$ which is illustrated in Fig. 10. The final result is shown in Fig. 11. The probability of $\prod_{\text{Location}}(X)$ when Location $= L_1$ is calculated by $1 - (1 - 0.7) * (1 - 0.8) = 0.94$. Using this query plan, when the number of events is large, we can avoid the exponential explosion problem of the possible world event model and get much better performance.

5. Experimental evaluations

In this section, we report our experimental study on DPCEP. We developed a traffic simulation system based on SUMO [29] in which realistic mobility trace of cars is supported using an OpenStreetMap [30] road map of a subpart of Beijing.



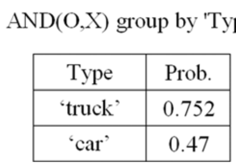
Location	Prob.
L_1	0.94
L_2	0.9

$\prod_{Location}(X)$

O.Location	Type	X.Location	Prob.
L_1	'truck'	L_1	0.752
L_1	'car'	L_1	0.47

$\prod_{Location,\ Type}(O) \bowtie_{O.Location=X.Location} \prod_{Location}(X)$

Fig. 10. Query plan for the example in Fig. 9.



AND(O,X) group by 'Type'

Type	Prob.
'truck'	0.752
'car'	0.47

Fig. 11. Rank of event.

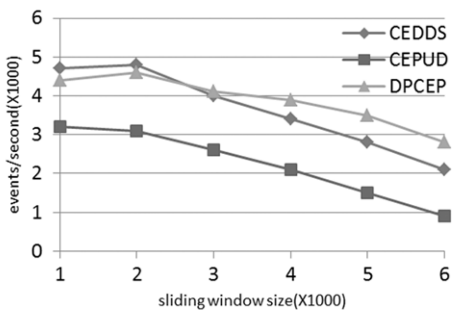


Fig. 12. Performance of single stream processing with different window size. The length of SEQ event pattern is set to 7.

Many "induction loops" are placed on the roads which can detect cars that pass corresponding areas. Virtual RFID or GPS readers are simulated by external applications which use the TraCI interface of SUMO to get the induction loop variables. Each induction loop covers a region. The closer a car is to the center of the region, the higher the probability of the event detected. Based on the data generated from this simulation system we compared the performance of DPCEP with two other related works. We also evaluated the relationship between the performance of DPCEP and some important factors such as CPU number and node number in the cluster. We used Lenovo ThinkServer RD series servers with 4 GB memory and the operating system is Ubuntu Server 12.

Currently we have not found other method which has the same function as ours. Therefore we compared the performance of our method with a plan-based complex event detection method across distributed sources (CEDDS) [16] and a complex event processing method over uncertain data streams (CEPUD). When evaluating CEDDS we ignored the probability of events since it supports distributed sources but not uncertainty. CEPUD supports uncertainty but not distributed event streams. In order to evaluate CEPUD in a distributed environment, we combined it with Storm bolt. Although distributed streams are supported, the CEP part in CEPUD is still centralized.

In the first experiment we evaluated the performance of the three methods for a single stream with different sliding window size (the length of SEQ event is set to 7). A server with a Xeon E5 processor (6 cores and 12 threads) was used. The result is shown in Fig. 12. As we can see, CEDDS and DPCEP have better performance than CEPUD because the latter is not parallel. When the sliding window size becomes larger, the performance of DPCEP outperforms that of CEDDS because the parallel execution of NFA-based CEPs can support large sliding window better.

In the next experiment we evaluated the performance of the three methods for four event streams with different number of event types. The size of sliding window was fixed at 3000 and the average value of λ is 0.05. Four servers were used in the cluster and each server has one Xeon E5 processor (6 cores and 12 threads). The result is shown in Fig. 13. As we can see, the performance of CEDDS and DPCEP decreases when the number of event types becomes larger but DPCEP still outperforms CEDDS. The reason is that the CEP method becomes complex and more data need to be moved but DPCEP benefits from the parallel method and joint probability optimization. The performance result for different λ values is shown in Fig. 14. The λ value does not affect the performance of CEPUD obviously because the CEP method in CEPUD is centralized. But when λ becomes larger the performance of CEDDS and DPCEP decreases quickly. The reason is that more data needs to be moved when λ becomes larger.

Then we evaluated the performance of the three methods over four distributed event streams for different sliding window size. Four servers are used like the previous experiment and the result is shown in Fig. 15. In contrast with Fig. 12, the

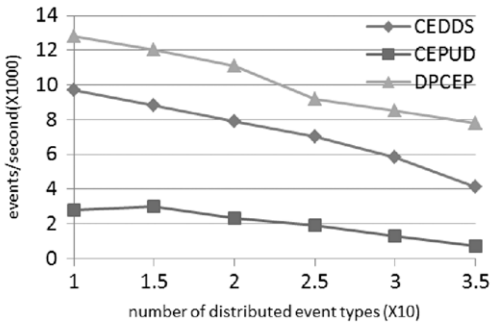


Fig. 13. Performance of 4 distributed streams with different number of event types. The size of sliding window is fixed at 3000 and the average value of λ is 0.05.

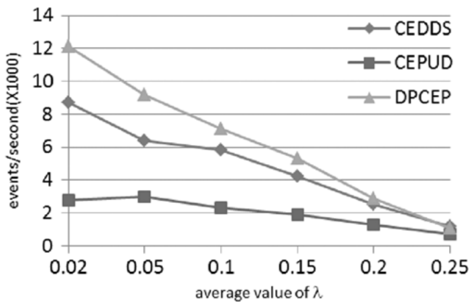


Fig. 14. Performance of 4 distributed streams with different λ values. The size of sliding window is fixed at 3000 and the number of event types is fixed at 25.

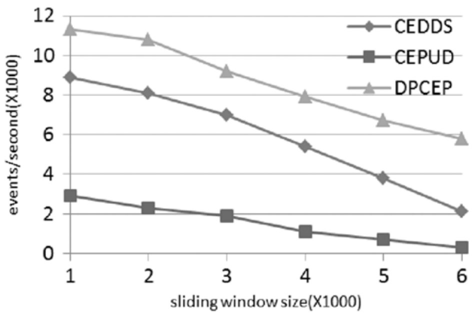


Fig. 15. Performance of 4 stream processing with different window size. The length of SEQ event pattern is set to 7. The number of distributed events is fixed at 25 and the average value of λ is 0.05.

performance of DPCEP outperforms that of CEDDS by much more. The reason is that DPCEP has a parallel method for both cluster and server.

In the last experiment we studied the performance of DPCEP for different sliding window size when the CPUs and node clusters are changed. The performance for three clusters with different CPUs is evaluated and the result is shown in Fig. 16. As we can see the performance becomes higher when the server has more CPUs or CPU has more cores or threads. The reason is that DPCEP executes in parallel inside a server and it can use more calculating resources to improve performance. Then the performance for three clusters with different servers is evaluated and the result is shown in Fig. 17. If the number of event streams is now very small, the performance becomes higher when the cluster has more servers because DPCEP supports parallel execution among nodes in the cluster.

From all the experiments we can see that DPCEP can detect hierarchical complex probabilistic events from distributed event streams. Using the heuristic and parallel method together with optimized query plan, it gets better performance and scalability than other related methods.

6. Discussion and conclusion

In this paper we propose a high-performance hierarchical probabilistic complex event processing method over distributed event streams. This method extends the basic NFA model to support probabilistic event processing in both single

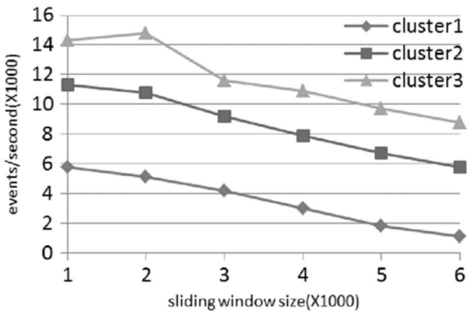


Fig. 16. Performance of 4 stream processing by DPCEP with different window size and different CPUs. The length of SEQ event pattern is set to 7. The number of distributed events is fixed at 25 and the average value of λ is 0.05. Cluster1: 4 nodes, each has a Xeon E5 CPU with 4 cores 4 threads; Cluster2: 4 nodes, each has a Xeon E5 CPU with 6 cores 12 threads; Cluster3: 4 nodes, each has 2 Xeon E5 CPUs with 6 cores 12 threads.

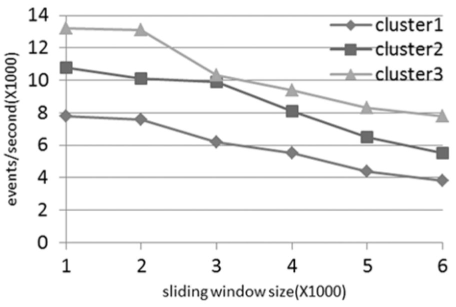


Fig. 17. Performance of 6 stream processing by DPCEP with different window size and different node numbers. The length of SEQ event pattern is set to 7. The number of distributed events is fixed at 25 and the average value of λ is 0.05. Cluster1: 2 nodes; Cluster2: 4 nodes; Cluster3: 6 nodes (each node has a Xeon E5 CPU with 6 cores 12 threads).

and distributed event streams. In order to support hierarchical complex event processing over distributed event streams, it uses the EPN-based model. Heuristic method, parallel operation and optimized query plan are also used to improve the performance. The experimental evaluations show that this method is effective when processing hierarchical probabilistic complex events over distributed event streams with large sliding window.

There are some limitations in our method. First, the distributed CEP algorithm needs data moving which is costly. Second, the optimization of the query plan in DPCEP is not flexible enough. In the future we need to improve our method to resolve these problems.

Acknowledgments

This project was sponsored by the "Context-aware and proactive complex event processing for large scale internet of things (13JJ3046)" project of Hunan Provincial Natural Science Foundation and the "complex event processing in large scale internet of things (K120326-11)" project of Changsha technological plan.

References

[1] D.C. Luckham, The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems, Addison Wesley, Boston, 2002.
[2] Y. Li, J. Wang, L. Feng, Accelerating sequence event detection through condensed composition, in: Proceedings of the 5th International Conference on Ubiquitous Information Technologies & Applications, Sanya, China, December, 2010.
[3] F. Wang, S. Liu, P. Liu, Bridging physical and virtual worlds: complex event processing for RFID data streams, in: Proceedings of the 10th International Conference on Extending Database Technology, EDBT'2006, pp. 588–607.
[4] X. Jin, X. Lee, N. Kong, Efficient complex event processing over RFID data stream, in: Proceedings of the Seventh IEEE/ACIS International Conference on, 2008, pp. 75–81.
[5] E. Wu, Y. Diao, S. Rizvi, High-performance complex event processing over streams, in: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, 27–29 June, Chicago, IL, USA, 2006.
[6] N. Dalvi, D. Suciu, Management of probabilisitic data foundations and challenges, in: PODS, 2007, pp. 1–12.
[7] N. Dalvi, D. Suciu, Efficient query evaluation on probabilistic databases, The VLDB Journal 16 (4) (2007) 523–544.
[8] Z. Shen, H. Kawashima, H. Kitagawa, Probabilistic event stream processing with lineage, in: Proceedings of the Data Engineering Workshop, 2008.
[9] C. Xu, S. Lin, W. Lei, Complex event detection in probabilistic stream, in: Proceedings of the 12th International Asia-Pacific Web Conference, APWeb 2010, pp. 361–363.
[10] H. Kawashima, H. Kitagawa, X. Li, Complex event processing over uncertain data streams, in: Proceedings of the Fifth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 2010, pp. 521–526.
[11] A. Artikis, O. Etzion, Z. Feldman, Event processing under uncertainty, in: Proceedings of the Sixth ACM International Conference on Distributed Event-Based Systems, DEBS 2012, Berlin, Germany, 16–20 July, 2012, pp. 32–43.

[12] O. Etzion, P. Niblett, Event Processing in Action, Manning Publications, 2010.
[13] G. Sharon, O. Etzion, Event-processing network model and implementation, IBM Systems Journal 47 (2) (2008) 321–344.
[14] J. Agrawal, Y. Diao, D. Gyllstrom, Efficient pattern matching over event streams, in: Proceedings of the SIGMOD Conference, 2008, pp. 147–160.
[15] H. Zhang, Y. Diao, N. Immerman, Recognizing patterns in streams with imprecise timestamps, Proceedings of the VLDB Endowment 3 (1) (2010) 244–255.
[16] M. Akdere, U. Çetintemel, N. Tatbul, Plan-based complex event detection across distributed sources, Proceedings of the VLDB Endowment 1 (2008).
[17] T. Ku, Y. Zhu, K. Hu, A novel distributed complex event processing for RFID application, in: Proceedings of the 2008 Third International Conference on Convergence and Hybrid Information Technology, Vol. 1, 2008, pp. 1113–1117.
[18] P. Agrawal, O. Benjelloun, A system for data, uncertainty, and lineage, in: Proceedings of the 32nd International Conference on Very Large Data Bases, 2006, pp. 1151–1154.
[19] A. Kimmig, B. Demoen, L.D. Raedt, On the implementation of the probabilistic logic programming language ProbLog, Theory and Practice of Logic Programming 11 (2011) 235–262.
[20] J. Filippou, A. Artikis, A. Skarlatidis, A probabilistic logic programming event calculus, Technical Report, Cornell University Library, 2012, http://arxiv.org/abs/1204.1851v1 (last accessed 02.2013).
[21] R. Bryant, Graph-based algorithms for Boolean function manipulation, IEEE Transactions on Computers 35 (8) (1986) 677–691.
[22] L. Rabiner, B. Juang, An introduction to hidden Markov models, IEEE ASSP Magazine 3 (1) (1986) 4–16.
[23] K. Murphy, Dynamic Bayesian networks: representation, inference and learning, Ph.D. Thesis, University of California, 2002.
[24] J.D. Lafferty, A. McCallum, F.C.N. Pereira, Conditional random fields: probabilistic models for segmenting and labeling sequence data, in: ICML, Morgan Kaufmann, 2001, pp. 282–289.
[25] M. Richardson, P. Domingos, Markov logic networks, Machine Learning 62 (1–2) (2006) 107–136.
[26] R. Biswas, S. Thrun, K. Fujimura, Recognizing activities with multiple cues, in: Proceedings of the Workshop on Human Motion, in: Lecture Notes in Computer Science, vol. 4814, Springer, 2007, pp. 255–270.
[27] A. Sadilek, H. Kautz, Location-based reasoning about complex multi-agent behavior, Journal of Artificial Intelligence Research 43 (2012) 87–133.
[28] V.I. Morariu, L.S. Davis, Multi-agent event recognition in structured scenarios, in: Computer Vision and Pattern Recognition (CVPR), 2011, pp. 3289–3296.
[29] M. Behrisch, L. Bieker, J. Erdmann, D. Krajzewicz, Sumo—simulation of urban mobility: an overview, in: Proceedings of the Third International Conference on Advances in System Simulation, Barcelona, Spain, October 2011, pp. 63–68.
[30] M. Haklay, P. Weber, Openstreetmap: user-generated street maps, IEEE Pervasive Computing 7 (4) (2008) 12–18.