Expressing and Evaluating Complex Event Patterns over Streams

Yanlei Diao
University of Massachusetts, Amherst

Joint work with Jagrati Agrawal, Daniel Gyllstrom, and Neil Immerman
Complex Event Processing

- High-volume event streams
  - Sensing devices
  - Financial services
  - Network monitors
  - Application monitors

- Complex event processing (CEP)
  - Filtering
  - Correlation
  - Aggregation
  - Pattern matching
  - Transformation
Retail Management

**Shoplifting**: an item was **first read at a shelf and then at an exit but not at any checkout counter in between.**

**Misplacement**: an item was **first read at shelf but not at any checkout counter or back at shelf afterwards.**
Supply Chain Management

**Contaminated shipments**: all shipments that were co-located with products originating from a source of contamination or subsequently infected products.

**Expired/spoiled drugs**: drugs that went through the distribution network in more than 3 months or were exposed to temperature $> 100^\circ F$ for 24 hours.

**Warehouse management**: count pallets, detect missing pallets…

**Retail management**: shoplifting, misplaced inventory, fast shelf depletion…

Expired/spoiled drugs: drugs that went through the distribution network in more than 3 months or were exposed to temperature $> 100^\circ F$ for 24 hours.

Contaminated shipments: all shipments that were co-located with products originating from a source of contamination or subsequently infected products.
Challenges

- **Expressiveness**
  - Can existing stream systems express all those complex events? What is new in CEP?

- **Performance**
  - **Low-latency**: up-to-the-second information
  - **Scalability**: high-volume streams, many monitoring tasks, etc.

- **Robustness**
  - **Out-of-order events**
  - **Uncertain events**: generated from incomplete, noisy, and erroneous sensor data
Overview of the SASE+ Project

- **SASE+:** a compact, declarative event language
  - Sequencing, negation, Kleene closure, complex predicates, ...

- **Formal study of expressability and complexity**
  - Theoretical underpinnings of CEP
  - Relationships of SASE+ and other languages

- **Efficient implementation over streams**
  - Automata-based query plans
  - Compile-time and runtime optimizations

- **Robust event processing**
  - Event generation from uncertain data
  - Handling uncertain events, out-of-order events, …
Q1: Shoplifting

“Item seen at a shelf and then at an exit, but not at any checkout counter in between.”

<table>
<thead>
<tr>
<th>PATTERN</th>
<th>SEQ(Shelf_Reading x, ~ (Counter_Reading y), Exit_Reading z)</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHERE</td>
<td>x.TagId = z.TagId AND x.TagId = y.TagID /* Equivalent to [TagId] */</td>
</tr>
<tr>
<td>WITHIN</td>
<td>12 HOURS</td>
</tr>
<tr>
<td>RETURN</td>
<td>x.TagId, x.ProdName, x.AreaId, z.AreaId</td>
</tr>
</tbody>
</table>
Conceptual Model for Evaluation

Shoplifting Query

PATTERN SEQ(Shelf_Reading x, ~ (Counter_Reading y), Exit_Reading z)
WHERE [TagId]
WITHIN 12 hours
RETURN ...

Output Events
(s₃, e₇) (s₆, e₉)

Input Events

Timeline
Closure Property

Yanlei Diao, University of Massachusetts Amherst
Q2: Stock Trend Monitoring

“In an hour, the volume of a stock started high, but after a period of price increasing, the volume plummeted.”

PATTERN  
SEQ(Stock+ a[], Stock b)

WHERE  
[symbol] AND
a[1].volume > 1000 AND a[i].price > \text{avg}(a[...i-1].price) AND b.volume < 80\% \times a[a.\text{LEN}].volume

WITHIN  
1 hour

RETURN  
a[1].symbol, a[].(price, volume), b.(price, volume)
Event Selection Strategies

- **Strict Contiguity**
  - The two relevant events must be contiguous in the input stream.
  - Matches regular expressions against strings, DNA sequences...

- **Partition Contiguity**
  - The two relevant events must be contiguous in a logical partition.
  - Captures continuous trends in each partition.

- **Skip till next match**
  - Can skip any events that can’t be selected.
  - Ignores local fluctuations to detect broad trends.

- **Skip till any match**
  - Can both select and skip relevant events, non-deterministic actions
  - Computes transitive closure over a subset of events.
Q2 Using Partition Contiguity

To capture a continuous trend in each partition:

**PATTERN**
SEQ(Stock+ a[], Stock b)

**WHERE**
partition_contiguity(a[],b)
{
    [symbol] AND
    a[1].volume > 1000    AND
    a[i].price > a[...i-1].price    AND
    b.volume < 80% * a[a.LEN].volume
}

**WITHIN**
1 hour

**RETURN**
a[1].symbol, a[].(price,volume), b.(price,volume)
Q2 Using Skip Till Next Match

To capture a trend while ignoring local fluctuating values:

**PATTERN**  
SEQ(Stock+ a[], Stock b)

**WHERE**  
\[\text{skip\_till\_next\_match(a[],b)}\]

\[
\{ \text{[symbol]} \text{ AND}
\]

\[
a[1].\text{volume} > 1000 \text{ AND}
\]

\[
a[i].\text{price} > a[...i-1].\text{price} \text{ AND}
\]

\[
b.\text{volume} < 80\% \times a[a.\text{LEN}].\text{volume}
\]

\[
\}
\]

**WITHIN**  
1 hour

**RETURN**  
a[1].symbol, a[].(price, volume), b.(price, volume)
Q3 Using Skip Till Any Match

In a food supply chain, detect contaminated shipments.

**PATTERN**

SEQ(Alert a, Shipment+ b[])  

**WHERE**

skip_till_any_match(a, b[])  

{}  

a.type = 'contaminated' AND  

b[1].from = a.site AND  

b[i].from = b[i-1].to  

}  

**WITHIN**  

12 hours  

**RETURN**  

a.type, a.site, b[].to
A Formal Study

- **Expressive power of SASE+**
  - Analysis based on *Descriptive Complexity*
  - Natural sublanguages ↔ standard complexity classes
  - SASE+ vs. standard languages, recent event languages

- **Theoretical underpinnings of CEP**
  - Why can’t existing languages express certain patterns?
  - How do existing languages relate to each other?
    - Need a unified foundation.
  - Query complexity results:
    - What features make computation expensive?
    - What compiler support and runtime optimizations can we offer?
Expressive Power of SASE+

Complexity Hierarchy

- SASE+ w. any match
- SASE+ w. next match
- SASE+ w. contiguity
- SASE w.o. ‘+’
Relationships to Other Languages

Complexity Hierarchy

- Temp. Logic
- SQL-TS
- Cayuga
- CQL w. recursion
- CQL w.o. recursion

- "truly feasible"
- P
- NC
- NC^2
- log(CFL) SAC^1
- NSPACE[log n]
- DSPACE[log n]
- Regular
- NC^1
- TC^0
- Logarithmic-Time Hierarchy
- AC^0
- SASE+ w. any match
- SASE+ w. next match
- SASE+ w. contiguity
- SASE w.o. ‘+’
Automata-based Query Processing

- **New abstraction for sequence patterns**
  - Relational joins can be slow for sequence patterns and are inadequate for Kleene closure
  - Native sequence operator: NFA<sup>b</sup>

- **Automata-based query plans**
  - More complex patterns may involve negation, nested patterns, etc.
  - An NFA<sup>b</sup> operator will pipeline sequence matches to other NFA<sup>b</sup> or relational operators
  - Can leverage relational query processing
Sequence Scan & Construction

- **Finite Automata** are a natural formalism for detecting sequence patterns.
- To produce all sequence matches, we adopt two phases of processing:
  - **Sequence Scan** (SS→): scans input stream to detect matches.
  - **Sequence Construction** (SC←): searches backward (in a summary of the stream) to create event sequences.
Illustration of SSC: Structure Only

Simple sequence:
(A, B, D)

Nondeterministic Finite Automaton (NFA)

Cost of sequence construction depends on the window size!
Optimizing SSC

- “Match buffer” integrated with the NFA model

- SS → builds the match buffer in NFA execution
- SC ← searches the match buffer for event sequences
Compilation Techniques

PATTERN
SEQ(Stock+ a[1], Stock b)

WHERE
Event_Selection_Strategy(a[], b) {
  [symbol] AND
  a[1].volume > 1000 AND
  a[i].price > a[...i-1].price AND
  b.volume < 80% * a[a.LEN].volume
}

Compile time optimizations: push filtering and stopping conditions early in appropriate places in the automaton.
Runtime Techniques

Automaton

\[ \theta_{a[i]} \_begin \quad \theta_{a[i]} \_ignore \quad \theta_{a[i]} \_proceed \quad \theta_{b} \_ignore \quad \theta_{b} \_begin \]

Match Buffer

\[ \begin{align*}
e_2 & \quad \text{\_\_\_\_\_\_}
e_3 & \quad \text{\_\_\_\_\_\_}
e_5 & \quad \text{\_\_\_\_\_\_}
e_6 & \quad \text{\_\_\_\_\_\_}
\end{align*} \]
Runtime Optimizations

- **Challenge: simultaneous runs of the automaton**
  - Multiple events can initiate new runs.
  - Runs can branch due to non-determinism.

- **Shared storage for all runs**
  - Can’t blind merge buffers for individual runs!
  - Technique: *shared versioned match buffer*

- **Shared processing of equivalent runs**
  - Merge equivalent runs based on *computation state*
  - Techniques: *when to merge, how to merge...*
Performance Evaluation (1)

- Comparison to a stream system using joins

**SASE:**

- **PATTERN SEQ** \(E_1, E_2, \ldots, E_L\)
- **WHERE** \([attr_1 (, attr_2)]\)
- **WITHIN** \(W\)

**Parameters:**
- \(L\) – Sequence length
- \(W\) – Window size in # events
- \(V_1\) – domain size of \(attr_1\)
- \(V_2\) – domain size of \(attr_2\)

**Join-based Stream Processor:** \(L=3, W=10000, [attr_1]\)

- \(R\) As (Select * From ES e Where e.type = ‘\(E_1\)’)
- \(S\) As (Select * From ES e Where e.type = ‘\(E_2\)’)
- \(T\) As (Select * From ES e Where e.type = ‘\(E_3\)’)

\((\) Select * 
- From \(R\) r [range by 10000]
- \(S\) s [range by 10000]
- \(T\) t [range by 10000]

- \(r.attr_1 = s.attr_1\) and \(r.attr_1 = t.attr_1\) and
- \(r.time < s.time\) and \(s.time < t.time\)

- Offered hint on the most selective predicate to the stream optimizer
- Performance metric is throughput
Varying Sequence Length

\textbf{PATTERN SEQ}({\(E_1, E_2, \ldots, E_L\)})

\textbf{WHERE} [\textit{attr}_1]

\textbf{WITHIN} \textit{W}

Parameters:
\(L = 2-6\)
\(W = 10,000\)
\([\textit{Attr}_1] V_1 = 100\)

\textbf{SASE} scales better than \textbf{Stream-Join} for longer sequences.

- Stream Join: N-way joins, postponed temporal predicates
- SASE: NFA for sequences and predicate evaluation, both in SSC
Varying Selectivity of Predicates

\[
\text{PATTERN} \quad \text{SEQ}(E_1, E_2, \ldots, E_L)
\]
\[
\text{WHERE} \ [\text{attr}_1 \text{, attr}_2]?
\]
\[
\text{WITHIN} \quad W
\]

Parameters:
\[
L = 3
\]
\[
W = 10,000
\]
\[
[\text{Attr}_1] \ V_1 = 10 - 10,000
\]
\[
[\text{Attr}_2] \ V_2 = 20
\]

SASE produces fewer intermediate results than Stream-Join.

- Stream-Join: cascading joins, postponed temporal predicates
- SASE: both sequencing and predicates in SSC, before producing any intermediate results
Performance of Kleene Closure

**PATTERN SEQ** (S+ a[,], S b)

**WHERE S(a[,], b) {**

- `[attr1]` AND
- `a[1].attr2 % 500 = 0` AND
- `P(a[i])` AND
- `b.attr2 > a[a.LEN].attr2` }

**WITHIN** W

**Parameters:**

- **P** = (p1) true
  - (p2) `a[i].attr3 > a[i-1].attr3`
  - (p3) `a[i].attr3 > min(a[i-1].attr3)`

- **S** = (s2) partition contiguity
  - (s3) skip till next match

- **W** = 500

**Predicate selectivity:** strong effect on num of runs and result length, hence overall performance.

**Event selection strategy:** s3 can be more expensive than s2.
Comparing to a Backtrack Algorithm

SASE+ vs. Backtrack:

- SASE+ evaluates all runs of the automaton simultaneously; it processes each event only once.
- Backtrack handles one run at a time, backtracks upon failure or to find another match; it reprocesses events multiple times.
Benefit of Shared Processing

Benefits of merging runs of automata:
• Performance gains 40% to 110% across all queries.
• Throughput over 10,000 events/sec even for complex queries.
• Higher performance gains when partition window size increases.
Summary of the SASE Project

- **SASE+:** compact, rich event language
  - Sequence, negation, Kleene closure, complex predicates, ...
- **Formal study of expressive power**
  - Theoretical underpinnings of CEP
- **Query processing approach**
  - New automata-based query plans
  - Optimization techniques
- **Summary of results**
  - SASE+ is more expressive than most event languages
  - Automata-based query processing can be highly efficient
  - Relational stream systems not suited for complex event processing
  - SASE system can be integrated into stream systems.
Questions