Efficient Pattern Matching over Event Streams

Jagrati Agrawal
Yanlei Diao
Daniel Gyllstrom
Neil Immerman

University of Massachusetts, Amherst
A New Stream Processing Paradigm

- Pattern matching over event streams

Matching Results

```
... R_3 | R_2 | R_1
```

Input Events

```
... e_5 | e_4 | e_3 | e_2 | e_1
```
Applications

- Existing and emerging applications
  - Financial services
  - RFID-based supply chain management
  - Click stream analysis
  - Electronic health systems
  - Network monitoring
  - E-commerce purchase tracking
  - …
Supply Chain Management

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

**Spoiled drugs**: drugs that were exposed to temperature > 100F for 24 hours.

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

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

Yanlei Diao, University of Massachusetts Amherst
Challenges

- **Rich languages**
  - Sequencing
  - Kleene closure
  - Negation
  - Complex predicates
  - Event selection strategies…

- **Efficient evaluation over streams**
  - Relational stream systems: selection-join-aggregation
  - Recent event systems

---

Significantly richer than regular languages!

Lacking support for key features. Not optimized.
Our Goal and Contributions

- A fundamental evaluation and optimization framework for the full set of event pattern queries
  - Formal evaluation model
    - Precise semantics of queries
    - Query evaluation plans
    - Formal results on expressive power
  - Runtime complexity analysis
  - Runtime algorithms and optimizations
  - Performance evaluation results
Event Pattern Languages

- Recent proposals: SQL-TS, Cayuga, SASE+, CEDR, StreamSQL, Coral8...
- Language structure of SASE+:

  FROM <input stream>
  
  PATTERN <pattern structure>
  
  [WHERE <pattern matching condition>]
  
  [WITHIN <time window>]
  
  [RETURN <output specification>]
Q1: Stock Trend Monitoring

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

```
FROM InputStream
PATTERN SEQ(Stock+ a[], Stock b)
WHERE [symbol] AND
    a[1].volume > 1000 AND a[i].price > min(a[..i-1].price) AND b.volume < 80% * a[a.LEN].volume
WITHIN 1 hour
```
Q1 Using Partition Contiguity

- **Event Selection Strategy: Partition Contiguity**
  - Captures a continuous trend in each partition

```sql
FROM InputStream
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
```
Q1 Using Skip Till Next Match

- Event Selection Strategy: Skip Till Next Match
  - Captures a broad trend while ignoring local fluctuating values

FROM InputStream
PATTERN SEQ(Stock+ a[], Stock b)
WHERE skip_till_next_match(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
Q2: Contaminated Shipments

“In a food supply chain, detect contaminated shipments.”

FROM InputStream
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

Yanlei Diao, University of Massachusetts Amherst
Q2 Using Skip Till Any Match

- Event Selection Strategy: **Skip Till Any Match**
  - Can both select and skip relevant events, non-deterministic
  - Computes transitive closure over an event stream
A Formal Evaluation Model: NFA\(^b\)

NFA

e1  e2

Match Buffer

a[1]  a[i]  b

Automaton State

a[1]

Computation State

value 1
value 2
value 3
A Formal Evaluation Model: NFA$^b$

NFA:  
- $a[1]$ transitions to $a[i]$ via $\theta_{a[1]}\_begin$.
- $a[i]$ transitions to $\theta_{a[i]}\_take$.
- $a[i]$ transitions to $b$ via $\theta_{a[i]}\_proceed$.
- $b$ transitions to $F$ via $\theta_{b}\_begin$.

Match Buffer:  
- $a[1]$: $e_2$

Automaton State:  
- $a[i]$

Computation State:  
- value 1
- value 2
- value 3

Yanlei Diao, University of Massachusetts Amherst
A Formal Evaluation Model: NFA\textsuperscript{b}

**NFA**

- \( a[1] \)\( \xrightarrow{\theta_{a[1]}\_begin} \) \( a[i] \)
- \( \theta_{a[i]}\_take \)
- \( \theta_{a[i]}\_proceed \)
- \( \theta_{b}\_ignore \)
- \( \theta_{b}\_begin \)

**Automaton State**

- \( a[i] \)

**Match Buffer**

- \( e_2 \)
- \( e_3 \)

**Computation State**

- Value 1
- Value 2
- Value 3
A Formal Evaluation Model: NFA

NFA

\[ a[1] \xrightarrow{\theta_a[i]_{\text{begin}}} a[i] \xrightarrow{\theta_a[i]_{\text{proceed}}} b \xrightarrow{\theta_b_{\text{begin}}} F \]

Match Buffer

- \( e_1 \)
- \( e_2 \)
- \( e_3 \)
- \( e_4 \)
- \( e_5 \)
- \( e_6 \)

Automaton State

- \( a[i] \)

Computation State

- value 1
- value 2
- value 3

Yanlei Diao, University of Massachusetts Amherst
A Formal Evaluation Model: NFA\(^b\)

**Match Buffer**
- a[1]
- a[i]
- b

**Automaton State**
- F

**Computation State**
- value 1
- value 2
- value 3

**Accepting Run**
- a[1] → θ\(_{a[1]\_begin}\) → a[i] → θ\(_{a[i]\_proceed}\) → b → θ\(_{b\_begin}\) → F

- θ\(_{a[i]\_ignore}\)
- θ\(_{b\_ignore}\)

- θ\(_{a[i]\_take}\)
Expressibility of $\text{NFA}_b$

**Complexity Hierarchy**

- **Stream SQL w. recursion**
- **Stream SQL w.o. recursion**
- **Regular**
- **$\text{NFA}_b$ (SASE+)**

- $\text{P}$
- "truly feasible"
- $\text{NC}$
- $\text{NC}^1$
- $\log(\text{CFL})$, $\text{SAC}^1$
- $\text{NSPACE}[\log n]$
- $\text{DSPACE}[\log n]$
- $\text{NC}^1$
- $\text{ThC}^0$
- $\text{AC}^0$

Yanlei Diao, University of Massachusetts Amherst
Query Compilation into NFA

PATTERN WHERE

SEQ(Stock+ a[], Stock b)

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
Computation State of NFA\(^b\)

**Computation state:** a minimum set of values for edge evaluation, extracted from *parameterized predicates*.

\[
\begin{align*}
\theta_{a[i]}\_ignore &= \\
\theta_{a[i]}\_begin &= a[i].type = \text{‘Stock’} \land a[i].symbol = a[1].symbol \land a[i]\_price > a[i-1]\_price \\
\theta_{a[i]}\_proceed &= \\
\theta_{b}\_ignore &= \\
\theta_{b}\_begin &= b.type = \text{‘Stock’} \land b.symbol = a[1].symbol \land b.volume < 80\% \ast a[a.LEN]\_volume
\end{align*}
\]

### Computation State

<table>
<thead>
<tr>
<th>NFA(^b) state</th>
<th>Attribute</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[1]</td>
<td>symbol</td>
<td>set()</td>
</tr>
<tr>
<td>a[i]</td>
<td>price</td>
<td>setLast()</td>
</tr>
<tr>
<td>a[i]</td>
<td>volume</td>
<td>setLast()</td>
</tr>
</tbody>
</table>
Runtime Challenges

A Single Run of NFA

**Match Buffer**
- `a[1]`: `e_2`
- `a[i]`: `e_3`, `e_5`
- `b`: Empty

**Automaton State**
- `a[1]`
- `a[i]`
- `F`

**Computation State**
- `value 1`
- `value 2`
- `value 3`
Runtime Challenges

Simultaneous runs of $\text{NFA}^b$:
- A new run can start before an old run completes.
- A run can branch at an $\text{NFA}^b$ state due to *nondeterminism*.

<table>
<thead>
<tr>
<th>Match Buffer</th>
<th>Automaton State</th>
<th>Computation State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a[1]$</td>
<td>$a[i]$</td>
<td>value 1</td>
</tr>
<tr>
<td>$e_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>value 2</td>
</tr>
<tr>
<td>$e_3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$e_5$</td>
<td></td>
<td>value 3</td>
</tr>
<tr>
<td>$a[i]$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$b$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Runtime Complexity

How many runs can we have?
• Depends on *event selection strategy*, partition window size…
• Polynomial to exponential in the worse case.

A Single Run of NFA

<table>
<thead>
<tr>
<th>Match Buffer</th>
<th>Automaton State</th>
<th>Computation State</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[1]</td>
<td>a[i]</td>
<td>value 1</td>
</tr>
<tr>
<td>e₂</td>
<td></td>
<td>value 2</td>
</tr>
<tr>
<td>e₃</td>
<td></td>
<td>value 3</td>
</tr>
<tr>
<td>e₅</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Merging Match Buffers

Match Buffer for Run 1

Match Buffer for Run 2

Match Buffer for Run 3

Yanlei Diao, University of Massachusetts Amherst
Merging Match Buffers

Shared Buffer for Runs 1, 2, 3

Erroneous result!
A Shared, Versioned Match Buffer

Shared, Versioned Buffer for Runs 1, 2, 3
Merging Equivalent Runs

- **Equivalent runs**
  - Despite distinct history, two runs have the same *computation state* at present.
  - They will select the same events till completion, hence can be merged.
Merging Equivalent Runs

**PATTERN**

SEQ(Stock a[], Stock b)

**WHERE**

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

*(symbol, price and volume of the recent selected event)*

**Computation State of Run 1**

<table>
<thead>
<tr>
<th>NFA(^b) state</th>
<th>attribute</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[1]</td>
<td>symbol</td>
<td>XYZ</td>
</tr>
<tr>
<td>a[i]</td>
<td>price</td>
<td>last:121</td>
</tr>
<tr>
<td>a[i]</td>
<td>volume</td>
<td>last:1000</td>
</tr>
</tbody>
</table>

**Computation State of Run 2**

<table>
<thead>
<tr>
<th>NFA(^b) state</th>
<th>attribute</th>
<th>operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[1]</td>
<td>symbol</td>
<td>XYZ</td>
</tr>
<tr>
<td>a[i]</td>
<td>price</td>
<td>last:121</td>
</tr>
<tr>
<td>a[i]</td>
<td>volume</td>
<td>last:1000</td>
</tr>
</tbody>
</table>
Merging Equivalent Runs

**PATTERN**

SEQ(Stock+ a[], Stock b)

**WHERE**

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

(symbol, min price of all selected events, volume of last event)
Performance of Kleene Closure

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

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

```
[symbol]  AND
a[1].price % 500 = 0  AND
P(a[i])    AND
b.volume < 150
```

**WITHIN W**

**Parameters:**

**P** = (p1) true

- (p2) a[i].price > a[i-1].price
- (p3) a[i].price > aggr(a[..i-1].price)

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

**W** = 500

**Basic Algorithm:**

*shared buffer + separate run execution*

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

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

Basic vs. Backtrack:

- Basic 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 expensive queries.
- Higher performance gains when partition window size increases.
Summary

- NFA$^b$ automaton, a formal evaluation model for event pattern queries
  - Expressibility of NFA$^b$
  - Compilation techniques
- Runtime complexity and sharing techniques
- Performance results
  - Tens of thousands of events/sec for fairly expensive queries
  - Even higher throughput for cheaper queries
  - Sharing among runs offers 40%-110% performance gains
- Potential impact: a pattern matching operator to be integrated into relational stream systems
Future Work

- **Query complexity analysis**
  - What pattern queries can be evaluated using constant time per event?

- **Optimizations**
  - Negation
  - Composed queries

- **Robust event processing**
  - Uncertain events
  - Out of order events

- **Benchmark for event pattern matching**
Questions