Shared Online Event Trend Aggregation

Olga Poppe, Chuan Lei, Lei Ma, Allison Rozet, Elke A. Rundensteiner

Best short paper in CIKM 2020
Full paper in SIGMOD 2021
Motivation

What are event trends?
Algorithmic Trading

Goal:
Reliable actionable insights about the stream

Solution:
Each event is considered in the context of other events in the stream

Algorithmic Trading

Single event =
Single stock value

Event sequence =
Stock down trend of fixed length

Event trend =
Stock up trend of any length
Algorithmic Trading

Single event =
Single stock value

Event sequence =
Stock down trend of fixed length

Event trend =
Stock up trend of any length

Algorithmic Trading

Single event =
Single stock value

Event sequence =
Stock down trend of fixed length

Event trend =
Stock up trend of any length

Event Trends

Infection spread

Path of infection spread

Ridesharing

Trajectory of shared ride

Financial fraud

Circular check kite

Performance optimization

Increasing load of a system component
Complexity of Event Trend Analytics
Under Skip-Till-Any-Match Semantics [SIGMOD’08]
Complexity of Event Trend Analytics

Under Skip-Till-Any-Match Semantics [SIGMOD’08]

Existing event trends

New event trends

New event trends

Existing event trends
Event Trend Aggregation Queries

Ridesharing

q1: RETURN T.district, COUNT(*), SUM(T.duration)
    PATTERN Request R, Travel T+, NOT Pickup P
    WHERE [driver, rider]
    GROUP-BY T.district
    WITHIN 30 min SLIDE 1 min

Number and duration of trips in which driver drove to pickup location but did not pick up the rider.
Event Trend Aggregation Queries

Ridesharing

q1: RETURN T.district, COUNT(*), SUM(T.duration)

PATTERN Request R, Travel T+, NOT Pickup P

WHERE [driver, rider]

GROUP-BY T.district

WITHIN 30 min SLIDE 1 min

Number and duration of trips in which driver drove to pickup location but did not pick up the rider
Event Trend Aggregation Queries

Ridesharing

q1: 
- **RETURN**: T.district, COUNT(*), SUM(T.duration)
- **PATTERN**: Request R, Travel T+, NOT Pickup P
- **WHERE**: [driver, rider]
- **GROUP-BY**: T.district
- **WITHIN**: 30 min SLIDE 1 min

Number and duration of trips in which driver drove to pickup location but did not pick up the rider
Event Trend Aggregation Queries

Ridesharing

q1: RETURN T.district, COUNT(*), SUM(T.duration)
PATTERN Request R, Travel T+, NOT Pickup P
WHERE [driver, rider]
GROUP-BY T.district
WITHIN 30 min SLIDE 1 min

Number and duration of trips in which driver drove to pickup location but did not pick up the rider
Event Trend Aggregation Queries

Ridesharing

q1: 

```
RETURN T.district, COUNT(*), SUM(T.duration)
PATTERN Request R, Travel T+, NOT Pickup P
WHERE [driver, rider]
GROUP-BY T.district
WITHIN 30 min SLIDE 1 min
```

Number and duration of trips in which driver drove to pickup location but did not pick up the rider.
**Problem Statement**

**Event trend aggregation queries**

<table>
<thead>
<tr>
<th>Query</th>
<th>RETURN</th>
<th>PATTERN</th>
<th>WHERE</th>
<th>GROUP-BY</th>
<th>WITHIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1:</td>
<td>T.district, COUNT(*), SUM(T.duration)</td>
<td>Request R, Travel T+, NOT Pickup P</td>
<td>[driver, rider]</td>
<td>T.district</td>
<td>30 min SLIDE 1 min</td>
</tr>
<tr>
<td>q2:</td>
<td>T.district, COUNT(*), AVG(T.speed)</td>
<td>Request R, Travel T+, Dropoff D</td>
<td>[driver, rider] AND R.type=Pool</td>
<td>T.district</td>
<td>30 min SLIDE 5 min</td>
</tr>
<tr>
<td>q3:</td>
<td>T.district, COUNT(*), SUM(T.duration)</td>
<td>Request R, Travel T+, Cancel C</td>
<td>[driver, rider] AND T.speed&lt;10</td>
<td>T.district</td>
<td>20 min SLIDE 1 min</td>
</tr>
</tbody>
</table>

**High-rate event stream**

Average query latency of all queries is minimal
Challenges

1. **Exponential complexity vs real-time response**

   **Online**
   
   Event trend aggregation without event trend construction reduces complexity from exponential to quadratic [VLDB’17, SIGMOD’19]

   **Shared**
   
   Event trend aggregation among multiple queries requires construction of shared sub-trends to ensure correctness

⇒ Correct yet efficient shared online event trend aggregation strategy
Challenges

1. Exponential complexity vs real-time response
2. Benefit vs overhead of sharing

**Benefit**
Due to avoided re-computations for similar queries in the workload

**Overhead**
Due to maintenance of intermediate results per query to ensure correctness

⇒ Light-weight yet accurate sharing benefit model
Challenges

1. Exponential complexity vs real-time response
2. Benefit vs overhead of sharing
3. **Bursty event streams vs light-weight sharing decisions**

**Static sharing optimizer**
Can do more harm than good if **event rate and data distribution fluctuate**

**Dynamic sharing optimizer**
Must **adjust its decisions** to the changing cost factors **at runtime**

⇒ Runtime yet light-weight sharing decisions
State-of-the-Art

<table>
<thead>
<tr>
<th>Approach</th>
<th>Kleene closure</th>
<th>Online aggregation</th>
<th>Sharing decisions</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCEP [SIGMOD'19]</td>
<td>✓</td>
<td>-</td>
<td>static</td>
</tr>
<tr>
<td>Sharon [ICDE’18]</td>
<td>-</td>
<td>✓</td>
<td>static</td>
</tr>
<tr>
<td>Greta [VLDB’17]</td>
<td>✓</td>
<td>✓</td>
<td>not shared</td>
</tr>
</tbody>
</table>
## State-of-the-Art

<table>
<thead>
<tr>
<th>Approach</th>
<th>Kleene closure</th>
<th>Online aggregation</th>
<th>Sharing decisions</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCEP [SIGMOD'19]</td>
<td>✓</td>
<td>-</td>
<td>static</td>
</tr>
<tr>
<td>Sharon [ICDE'18]</td>
<td>-</td>
<td>✓</td>
<td>static</td>
</tr>
<tr>
<td>Greta [VLDB'17]</td>
<td>✓</td>
<td>✓</td>
<td>not shared</td>
</tr>
</tbody>
</table>

Hamlet dynamically decides to share or not to share online event trend aggregation.
Hamlet Framework

Hamlet Optimizer

Static Workload Analysis
Sharing Benefit Model
Decision to Split & Merge Graphlets
Choice of Query Set

Hamlet Executor

Stream Partitioning
Graph Construction
Sharing Benefit Monitoring
Split & Merge of Graphlets

Query workload
Event stream

Sharing benefit
Runtime configuration

Aggregation results

21
Sharable Queries

q1: RETURN T.district, COUNT(*), SUM(T.duration)
PATTERN Request R, Travel T+, Pickup P
WHERE [driver, rider]
GROUP-BY T.district
WITHIN 30 min SLIDE 1 min

q2: RETURN T.district, COUNT(*), AVG(T.speed)
PATTERN Request R, Travel T+, Dropoff D
WHERE [driver, rider] AND R.type=Pool
GROUP-BY T.district
WITHIN 30 min SLIDE 5 min

q3: RETURN T.district, COUNT(*), SUM(T.duration)
PATTERN Request R, Travel T+, Cancel C
WHERE [driver, rider] AND T.speed<10
GROUP-BY T.district
WITHIN 20 min SLIDE 1 min

Queries are sharable if their
○ Patterns contain at least one sharable
Kleene sub-pattern,
Sharable Queries

q1: RETURN T.district, COUNT(*), SUM(T.duration)
PATTERN Request R, Travel T+, Pickup P
WHERE [driver, rider]
GROUP-BY T.district
WITHIN 30 min SLIDE 1 min

q2: RETURN T.district, COUNT(*), AVG(T.speed)
PATTERN Request R, Travel T+, Dropoff D
WHERE [driver, rider] AND R.type=Pool
GROUP-BY T.district
WITHIN 30 min SLIDE 5 min

q3: RETURN T.district, COUNT(*), SUM(T.duration)
PATTERN Request R, Travel T+, Cancel C
WHERE [driver, rider] AND T.speed<10
GROUP-BY T.district
WITHIN 20 min SLIDE 1 min

Queries are sharable if their
○ Patterns contain at least one sharable Kleene sub-pattern,
○ Aggregation functions can be shared,
Sharable Queries

q1: RETURN T.district, COUNT(*), SUM(T.duration)  
   PATTERN Request R, Travel T+, Pickup P  
   WHERE [driver, rider]  
   GROUP-BY T.district  
   WITHIN 30 min SLIDE 1 min

q2: RETURN T.district, COUNT(*), AVG(T.speed)  
   PATTERN Request R, Travel T+, Dropoff D  
   WHERE [driver, rider] AND R.type=Pool  
   GROUP-BY T.district  
   WITHIN 30 min SLIDE 5 min

q3: RETURN T.district, COUNT(*), SUM(T.duration)  
   PATTERN Request R, Travel T+, Cancel C  
   WHERE [driver, rider] AND T.speed<10  
   GROUP-BY T.district  
   WITHIN 20 min SLIDE 1 min

Queries are sharable if their
○ Patterns contain at least one sharable Kleene sub-pattern,
○ Aggregation functions can be shared,
○ Windows overlap, and
Sharable Queries

Queries are sharable if their
○ Patterns contain at least one sharable
Kleene sub-pattern,
○ Aggregation functions can be shared,
○ Windows overlap, and
○ Grouping attributes are the same.

q1: RETURN T.district, COUNT(*), SUM(T.duration)
PATTERN Request R, Travel T+, Pickup P
WHERE [driver, rider]
GROUP-BY T.district
WITHIN 30 min SLIDE 1 min

q2: RETURN T.district, COUNT(*), AVG(T.speed)
PATTERN Request R, Travel T+, Dropoff D
WHERE [driver, rider] AND R.type=Pool
GROUP-BY T.district
WITHIN 30 min SLIDE 5 min

q3: RETURN T.district, COUNT(*), SUM(T.duration)
PATTERN Request R, Travel T+, Cancel C
WHERE [driver, rider] AND T.speed<10
GROUP-BY T.district
WITHIN 20 min SLIDE 1 min
Hamlet Template

q1: RETURN T.district, COUNT(*), SUM(T.duration)
PATTERN Request R, Travel T+
WHERE [driver, rider]
GROUP-BY T.district
WITHIN 10 min SLIDE 5 min

q2: RETURN T.district, COUNT(*), AVG(T.speed)
PATTERN Pickup P, Travel T+
WHERE [driver, rider] AND P.type=Pool
GROUP-BY T.district
WITHIN 15 min SLIDE 5 min
Hamlet Framework

Hamlet Optimizer
- Static Workload Analysis
- Sharing Benefit Model
- Decision to Split & Merge Graphlets
- Choice of Query Set

Hamlet Executor
- Stream Partitioning
- Graph Construction
- Sharing Benefit Monitoring
- Split & Merge of Graphlets

Query workload

Event stream

Aggregation results

Sharing benefit

Runtime configuration

Decision to Split & Merge Graphlets

Choice of Query Set
Non-Shared Graph Construction

2 event trends:
- r1,t3
- r2,t3

Event of type Request
Event of type Travel
Non-Shared Graph Construction

2 event trends:
r1,t3
r2,t3

Event of type Request
Event of type Travel
Non-Shared Graph Construction

2 event trends:
- r1,t3
- r2,t3

Event types:
- Request
- Travel

Event markers:
- Event of type Request
- Event of type Travel
Non-Shared Graph Construction

2 event trends:
- r1,t3
- r2,t3

Event of type Request
Event of type Travel
Non-Shared Graph Construction

6 event trends:
- $r_1, t_3$
- $r_1, t_4$
- $r_1, t_3, t_4$
- $r_2, t_3$
- $r_2, t_4$
- $r_2, t_3, t_4$

$NonShared(Q) = O(n^2)$

where $n$ – # events in a window
Non-Shared Graph Construction

\[ \text{NonShared}(Q) = O(n^2 \times k) \]

where \( n \) - # events in a window,
\( k \) - # queries
Non-Shared Graph Construction

NonShared \( Q = O(n^2 \times k) = 14^2 \times 2 = 392 \)

where \( n \) - # events in a window,
\( k \) - # queries
The set of predecessor events is different for q1 and q2 due to:

- Different patterns
The set of predecessor events is different for q1 and q2 due to:

- Different patterns
- Predicates
Shared Graph Construction

<table>
<thead>
<tr>
<th>Snapshot</th>
<th>q1</th>
<th>q2</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>y</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>
**Shared Graph Construction**

\[
\text{Shared}(Q) = O(n^2 \ast s + s \ast k \ast g \ast t)
\]

where \(n\) – # events in a window,
\(k\) – # queries,
\(g\) – # events per graphlet,
\(s\) – # snapshots,
\(t\) – # types per query
Shared Graph Construction

\[ \text{Shared}(Q) = O(n^2 \cdot s + s \cdot k \cdot g \cdot t) = 14^2 \cdot 2 + 2 \cdot 2 \cdot 4 \cdot 2 = 424 \]

\[ \text{NonShared}(Q) = O(n^2 \cdot k) = 14^2 \cdot 2 = 392 \]

where \( n \) – # events in a window,
\( k \) – # queries,
\( g \) – # events per graphlet,
\( s \) – # snapshots,
\( t \) – # types per query
Hamlet Framework

Hamlet Optimizer
- Static Workload Analysis
- Sharing Benefit Model
- Decision to Split & Merge Graphlets
- Choice of Query Set

Sharing benefit

Runtime configuration

Hamlet Executor
- Stream Partitioning
- Graph Construction
- Sharing Benefit Monitoring
- Split & Merge of Graphlets

Aggregation results

Query workload

Event stream
Dynamic Sharing Decision

\[ \text{Shared}(T_3, Q_T) = 4 \times 7 \times 1 + 1 \times 2 \times 4 \times 2 = 44 \]

\[ \text{NonShared}(\{T_3, T_4\}, Q_T) = 2 \times 4 \times 7 = 56 \]

A burst is a set of consecutive events of type \( T \), the processing of which can be shared by queries \( Q_T \) that contain a Kleene sub-pattern \( T^+ \).

\[ |\text{Single event}| \leq |\text{Burst}| \leq |\text{Window}| \]
Dynamic Sharing Decision

Shared execution

R1

P2

T3
Dynamic Sharing Decision

\[ \text{Shared}(T_3, Q_T) = 4 \times 11 \times 2 + 1 \times 2 \times 8 \times 2 = 120 \]

\[ \text{NonShared}([T_4, T_5], Q_T) = 2 \times 4 \times 11 = 88 \]
Dynamic Sharing Decision

Non-shared execution
Dynamic Sharing Decision

Shared execution

\[ \text{Shared}(T_6, Q_T) = 4 \times 15 \times 1 + 1 \times 2 \times 4 \times 2 = 76 \]

Non-shared execution

\[ \text{NonShared}(\{T_4, T_5\}, Q_T) = 2 \times 4 \times 15 = 120 \]

- Merge creates one snapshot
  - Linear in # events per graphlet
- Split comes for free!
Experiments
Experimental Setup

Infrastructure
Java 8, Ubuntu 14.04, 16 cores, 128GB

Data sets
○ NYC taxi and Uber real data set
○ Smart home real data set
○ Stock real data set
○ Ridesharing data set

Metrics
○ Latency
○ Throughput
○ Peak memory

Cost factors
○ Number of events per minute
○ Number of queries

<table>
<thead>
<tr>
<th>Approach</th>
<th>Kleene closure</th>
<th>Online aggregation</th>
<th>Sharing decisions</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCEP [SIGMOD’19]</td>
<td>✓</td>
<td>-</td>
<td>static</td>
</tr>
<tr>
<td>Sharon [ICDE’18]</td>
<td>-</td>
<td>✓</td>
<td>static</td>
</tr>
<tr>
<td>Greta [VLDB’17]</td>
<td>✓</td>
<td>✓</td>
<td>not shared</td>
</tr>
<tr>
<td>Hamlet [SIGMOD’21]</td>
<td>✓</td>
<td>✓</td>
<td>dynamic</td>
</tr>
</tbody>
</table>
Hamlet vs State-of-the-Art

○ Hamlet outperforms Sharon by 3-5 orders of magnitude, Greta by 1-2 orders of magnitude, and MCEP by 7-76X
○ Hamlet terminates within 25 ms, Sharon – 50 min, Greta – 3 sec, MCEP – 1 sec
Dynamic vs Static Sharing Decisions

**Static optimizer**
- Shared execution during the entire window
- ⇒ Number of snapshots is 10K-20K
- ⇒ Sharing overhead

**Dynamic optimizer**
- 10% of bursts is not shared
- ⇒ Number of snapshots is reduced by 50% (4K-8K)
- ⇒ 21-34% speed-up compared to static optimizer

Overhead:
- 400-600 sharing decisions per window within 20ms
- 0.2% of total latency per window

Stock real data
- 120 events per shared burst of event on avg
- Number of graphlets is 400-600
- Number of shared graphlets is 360-500
Conclusions

Hamlet integrates:

- Shared online trend aggregation strategy
- Dynamic sharing optimizer
  - Makes fine-grained sharing decisions per each
    - Sharable Kleene sub-pattern,
    - Burst of events, and
    - Subset of queries.
  - Switches between shared and non-shared execution at runtime

Hamlet achieves substantial performance gains compared to state-of-the-art
Acknowledgements

Chuan Lei
Researcher

Lei Ma
PhD student

Allison Rozet
SWE

Elke A. Rundensteiner
Professor

Funding agencies:
- NSF grants IIS-1815866, IIS-1018443, CRI-1305258
- U.S. Department of Education grant P200A150306
- U.S. Department of Agriculture grant 1023720
Thanks!

Questions?