Complete Event Trend Detection in High-Rate Event Streams

Olga Poppe*, Chuan Lei**, Salah Ahmed*, and Elke A. Rundensteiner*

*Worcester Polytechnic Institute, **NEC Labs America

SIGMOD
May 16, 2017

Funded by NSF grants CRI 1305258, IIS 1343620
Real-time Event Trend Analytics

Event trend = event sequence of any length

Traffic control
- Event trend: Aggressive driving

Health care
- Event trend: Irregular heart rate

Cluster monitoring
- Event trend: Uneven load distribution

E-commerce
- Event trend: Items often bought together

Stock market
- Event trend: Head-and-shoulders

Financial fraud
- Event trend: Circular check kite
Check Kiting Fraud

Account 1: $10k

Account 2: $10k

$110k

$100k

$100k

$110k
Check Kiting Fraud

- In 2013, a bank fraud scheme netted $5 million from six New York City banks [FBI]
- In 2014, 12 people were charged in a large-scale “bust out” scheme, costing banks over $15 million [The Press Enterprise]
Complete Event Trend Detection

CETs: Complete Event Trends

PATTERN Check+ C [ ]
WHERE C.type = not-covered AND
C.destination = Next(C).source
WITHIN 12 hours SLIDE 1 minute

Event Stream
Check deposit
C: Event type
1: Time stamp
A: Source bank
B: Destination bank

Cash withdrawal
W: Event type
9: Time stamp
B: Source bank
Problem Statement & Challenges

Problem Statement

CET optimization problem is to detect all CETs matched by Kleene query q in stream I with minimal CPU processing costs while staying within memory M

Challenges

1. Expressive yet efficient
   Exponential number of event trends of arbitrary length

2. Real-time yet lightweight
   Common event sub-trend storage versus their re-computation

3. Optimal yet feasible
   NP-hard event stream partitioning problem
State-of-the-Art Approaches

1. **Limited expressive power**
   Neither Kleene closure nor the skip-till-any-match semantics are supported [1,2,3]

2. **Delayed system responsiveness**
   Common event sub-trends are re-computed [1,2,3,4]

1) Flink. https://flink.apache.org/
Base-Line CET Detection

Cases of the base-line algorithm:

1. Start a new CET
Base-Line CET Detection

Cases of the base-line algorithm:
1. Start a new CET
2. Append to an existing CET
Base-Line CET Detection

Cases of the base-line algorithm:
1. Start a new CET
2. Append to an existing CET
3. Replicate the prefix of an existing CET and append to it
Base-Line CET Detection

Problem: Exponential time & space complexity
Overview of Our CET Approach

Event trend output stream

↑

**Step 2:**
Graph-based CET Detection
Trade-off between time & space complexity

↑ CET graph

**Step 1:**
Compact CET Encoding as CET Graph
Quadratic time & space complexity

↑

Input event stream
Step 1: CET Graph Construction

Cases of the graph construction algorithm:
1. Start a new CET
Step 1: CET Graph Construction

Cases of the graph construction algorithm:
1. Start a new CET
2. Append to an existing CET
Step 1: CET Graph Construction

Cases of the graph construction algorithm:
1. Start a new CET
2. Append to an existing CET
3. Append to the prefix of an existing CET
Step 1: CET Graph Construction

- **Compact CET encoding** = CET graph
  - Matched event = vertex
  - Event adjacency relation = edge
  - CET = Path through the graph

Quadratic time & space complexity
Step 2: Graph-based CET Detection

Spectrum of CET Detection Algorithms

**T-CET**: Time-optimal BFS-based algorithm

1, 3, 5
2, 3, 5
2, 4, 5

**M-CET**: Memory-optimal DFS-based algorithm

1, 3, 5, 7

Is a middle ground possible?
Step 2: Graph-based CET Detection

Our Proposed **H-CET** (Hybrid) Algorithm

**How do we partition the graph?**
Graph partitioning search is *exponential* in # of atomic graphlets

**Goal:** Optimal graph partitioning plan
Balanced Graph Partitioning

Theorem. The closer a graph partitioning is to balanced, the lower are CPU & memory costs of the CET detection.

**CPU**: 27 connect operations
**Memory**: 42 events

**CPU**: 27 connect operations
**Memory**: 36 events
Graph Partitioning Algorithm

Pruning principles:
1. Unbalanced node pruning
Number of Graphlets

2 Graphlets

3 Graphlets

CPU: 27 connect operations
Memory: 42 events

CPU: 38 connect operations
Memory: 18 events

Theorem. If we add a cut to the graph, memory costs of CET detection goes down, while CPU processing time goes up.
Graph Partitioning Algorithm

Pruning principles:
1. Unbalanced node pruning
2. Infeasible level pruning
Graph Partitioning Algorithm

Pruning principles:
1. Unbalanced node pruning
2. Infeasible level pruning
3. Inefficient branch pruning
Experimental Setup

Execution infrastructure:
Java 7, 1 Linux machine with 16-core 3.4 GHz CPU and 128GB of RAM

Data sets:
• Stock real data set (ST) [1]
  \textbf{CETs} = Stock trends
• Physical activity monitoring real data set (PA) [2]
  \textbf{CETs} = Behavioral patterns per person
• Financial transaction synthetic data set (FT)
  \textbf{CETs} = Circular check kites

Experimental Setup

CET detection algorithms:

- Base line (BL) maintains a set of CETs
- SASE++ is memory-optimized [1,2]
- Flink is a popular open-source streaming engine that supports event pattern matching but not Kleene closure. Thus, we flatten our queries [3]

CET graph partitioning algorithms:

- Exhaustive (Exh)
- Greedy
- Branch and bound (B&B)

CET Detection Algorithms

CET

• utilizes available memory to achieve **42-fold** speed-up compared to **SASE++**
• is **2 orders of magnitude** faster and requires **2 orders of magnitude** less memory than **Flink**
CET Graph Partitioning

**Graph partitioning algorithms**

- **Exh**
- **Greedy**
- **B&B**

- **B&B** is
  - 2 orders of magnitude faster than **Exhaustive** but 3-fold slower than **Greedy**
  - CET detection in a greedily partitioned CET graph is almost 3-fold slower than in an optimally partitioned CET graph
Conclusions

We are the first to enable real-time Kleene closure computation over event streams under memory constraints

1. **CET graph** compactly encodes all CETs and defines the spectrum of **CET detection algorithms**

2. **Hybrid CET detection algorithm** utilizes available memory to achieve **42-fold speed-up**

3. **Graph partitioning algorithm** prunes large portions of search to efficiently find an optimal graph partitioning
Acknowledgement

• DSRG group at WPI
• SIGMOD reviewers
• NSF grants CRI 1305258, IIS 1343620