Supported by NSF Grants IIS-1815866, CRI-1305258, and IIS-1018443, and the U.S. Department of Education grant P200A150306.

MUSE: Multi-query Event Trend Aggregation

Allison Rozet[1], Olga Poppe[2], Chuan Lei[3], and Elke A. Rundensteiner[1]

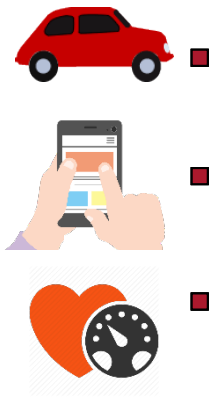
1. Worcester Polytechnic Institute
2. Microsoft Gray Systems Lab
3. IBM Research - Almaden

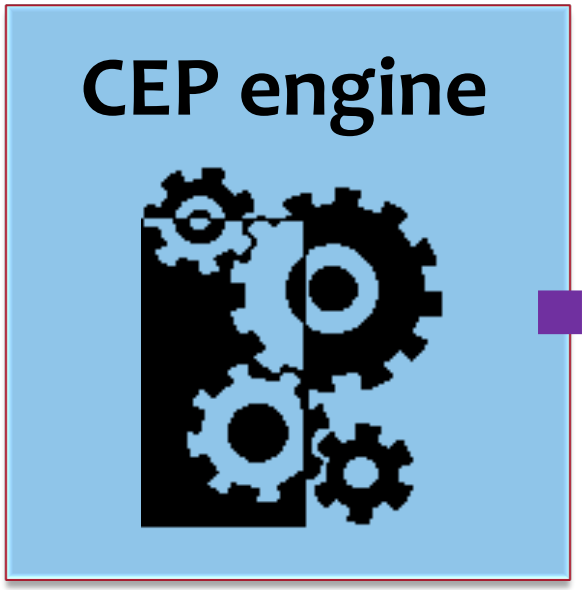
ACM International Conference on Information and Knowledge Management

October 2020

Complex Event Processing

Primitive events





Complex events

Input: High-rate, potentially unbounded event stream

Output: Reliable summarized insights about the current situation in real time

Problem

Objective: Near Instantaneous Responsiveness

Expensive Event Trend Aggregation Queries

High Volume, High Velocity Event Stream

Our goal is to identify, analyze, and exploit **sharing opportunities** in order to optimize workload processing.

Kleene Pattern Aggregation Query

Query q1:
RETURN COUNT(*)
PATTERN B+

Stream:
b1, b2, b3

Trends:
b1
b1, b2
b1, b2, b3
b1, b3
b2
b2, b3
b3

Final count: 7

A *trend* is an arbitrarily long sequence of events that matches the query.

COUNT(*) returns the number of trends.

A *two-step approach* constructs all matches prior to aggregation.

Exponential complexity.

Online Aggregation

Query q1:
RETURN COUNT(*)
PATTERN B+

Stream:
b1, b2, b3

Event	bi.count
b1	1
b2	2
b3	4

Final count: 7

An *online approach* maintains aggregates incrementally.

bi.count is the number of partial trends that end at event bi.

For example, b3.count tells us that there are 4 partial trends that end at b3. They are (b1,b2,b3), (b1,b3), (b2,b3), and (b3).

Quadratic complexity.

Multi-query Online Aggregation

Query q1:
RETURN COUNT(*)
PATTERN B+

Stream:
b1, a1, b2, b3

Event	bi.count
b1	1
b2	2
b3	4

Final count: 7

We have an identical sub-pattern…

…but the numbers are not the same. How could anything possibly be shared here?

Query q2:
RETURN COUNT(*)
PATTERN SEQ(A,B+)

Stream:
b1, a1, b2, b3

Event	ai.count	bi.count
a1	1	
b2		1
b3		2

Final count: 3

Challenges

Sharing diverse nested Kleene patterns:

SEQ(P, T+, D)  SEQ(SEQ(P, T+)+, D)

Shared computation without trend construction:

Sharing requires trend construction $\Rightarrow\Leftarrow$ Online skips trend construction

Optimizing the Kleene sharing plan:

Exponentially large search space 

Muse* Executor

time →

Event / Query	a1	a2	b3	b4	b5
q1			1	2	4
q2	1	1	2	4	8
q3	1	1	2	4	8

time →

Event / Query	MP:B.1	b3	b4	b5
q1	1	1	2	4
q2	2			
q3	2			

a MatState

Non-shared execution
q1 = B+
q2 = SEQ(A,B+)
q3 = SEQ(A,B+)+

Execution sharing B+
MatPoint (materialization point) is B.
A *MatState* (materialized state) stores each query's intermediate trend aggregate.

*__Muse__ = __Mu__lti-query __s__hared __e__vent trend aggregation

Benefit Model

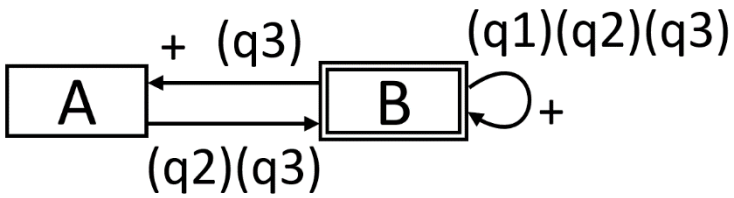
If Benefit(<E',E>, Q) > 0, we say it is **beneficial to share**.

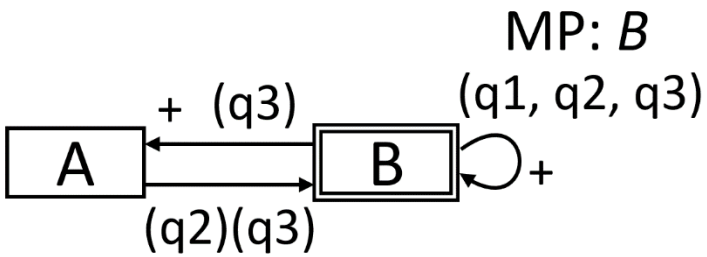
Lemma 3.2. The more queries that share a sub-pattern, the more beneficial it becomes.

Lemma 3.3. Reducing the number of MatStates increases the sharing benefit.

Muse Optimizer

- Begin with a global sharing plan
- Prune plans in the search space using Lemmas 3.2 and 3.3
- Optimizer follows a modified topological sort algorithm





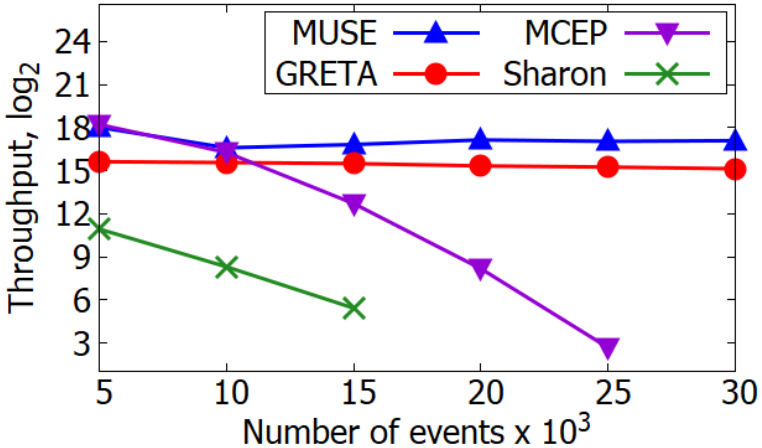
Experimental Setup

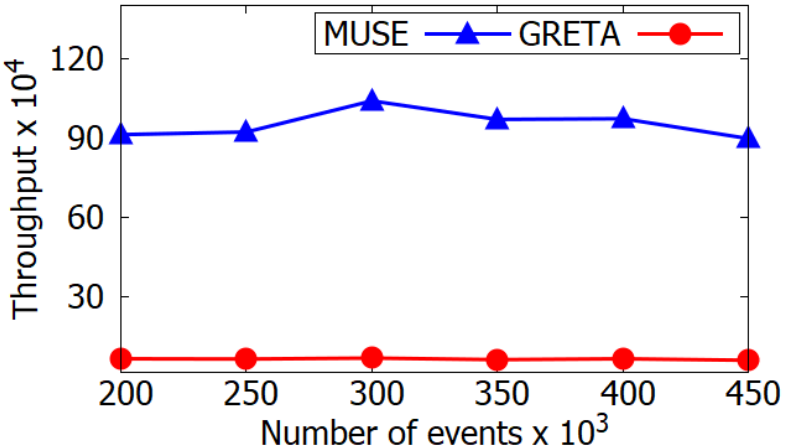
Data Sets:

- NASDAQ Stock Market Real Data Set
  - Transactions for over 3200 companies for one month
  - Stock ticker symbol, time stamp, price, volume
- Ridesharing Synthetic Data Set
  - Controls the rate of event types in the stream
  - 50 event types and 20 districts

NASDAQ Stock Market Real Data Set. EODData, Historical Price Data. https://www.eoddata.com, 2019.

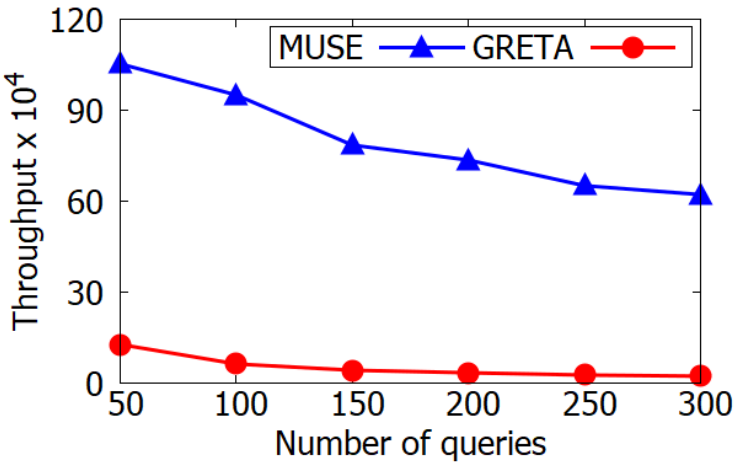
Experimental Results

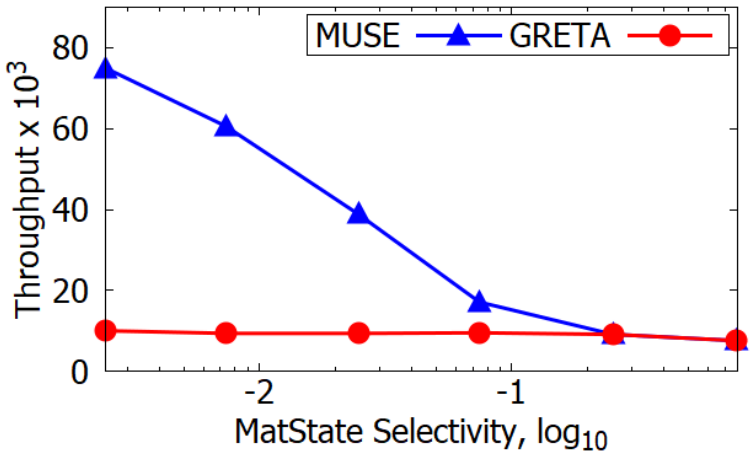




- Muse has a throughput gain of 3 orders of magnitude over Sharon at 15k events per window (left: ridesharing)
- Muse outperforms MCEP by 4 orders of magnitude at 25k events
- Muse achieves 14-fold increase in throughput over GRETA on a higher-rate event stream (right: stock)

Experimental Results





- Muse achieves from 7-fold to 25-fold throughput gain over GRETA when the number of queries increases from 50 to 300 (left: stock)
- For streams with very few MatStates, Muse sees nearly 7-fold increase in throughput compared to GRETA (right: ridesharing)

Conclusions

- Muse defines *shared* aggregation of event trends matched by *diverse nested Kleene* pattern queries over high speed streaming data in *real-time*
- Several orders of magnitude performance improvement over state-of-the-art