# MUSE: Multi-query Event Trend Aggregation

Technical Report
May 2020

Anonymous Author(s)

## Contents

**Abstract**

Streaming analytics deploy Kleene pattern queries to detect and aggregate event trends on high-rate data streams. Despite increasing workloads, most state-of-the-art systems process each query independently, thus missing cost-saving sharing opportunities. Sharing event trend aggregation poses several technical challenges. First, Kleene patterns are in general very difficult to share due to complex nesting, predicates, and arbitrarily long matches. Second, not all sharing opportunities are beneficial because sharing Kleene patterns incurs non-trivial overhead in order to ensure the correctness of final aggregation results. We propose MUSE (Multi-query Shared Event trend aggregation), the first framework that shares aggregation queries with Kleene patterns while avoiding expensive trend construction. To find the beneficial sharing plan, the MUSE optimizer effectively selects robust sharing candidates from the exponentially large search space. Our experiments demonstrate that MUSE increases throughput by 4 orders of magnitude compared to state-of-the-art approaches.

# 1 Introduction

**Motivation**. Industries such as transportation, finance, and advertising use Complex Event Processing (CEP) technologies to extract insights from high-velocity event streams using Kleene pattern queries [1, 14]. While traditional pattern queries return fixed-length sequences, Kleene pattern queries are more flexible, capturing matches of arbitrary length. These arbitrarily long matches, also known as *event trends*, can be aggregated to derive meaningful insights, driving increased profit, customer satisfaction, and safety [11]. To achieve near real-time responsiveness for these expensive Kleene pattern workloads, it is imperative to exploit both sharing optimization and online execution strategies.

**State-of-the-Art Approaches**. CEP has become an increasingly important area of research [1, 13, 14, 7, 6, 4, 12, 15]. A variety of solutions have been introduced to address the three dimensions of the multi-query event trend aggregation problem, namely, multi-query sharing, online aggregation, and Kleene closure processing.

State-of-the-art CEP systems that support *Kleene closure* while utilizing *shared processing* implement a two-step approach to tackle trend aggregation [3, 5]. That is, they first construct all event trends matching the query patterns (an extremely expensive process especially for Kleene patterns), and only thereafter aggregate them. In this case, sharing optimization is only applied to the first step of trend matching and construction. Consequently, sharing optimization on trend construction does not guarantee real-time responsiveness since the complexity of trend construction itself is exponential in the number of matched events in the worst case [14].

Recent work on event trend processing [10, 8, 9] addresses this performance bottleneck by pushing the aggregation computation into the pattern matching process. Such *online* methods succeed to skip the event trend construction step, reducing the time complexity from exponential to polynomial in the worst case [8, 9]. Among these online approaches, only Sharon [10] enables sharing among multiple queries. However, its shared execution strategy is rigid, resulting in missed beneficial sharing opportunities. It also lacks support for Kleene closure and expressive predicates.

**Challenges**. Designing a unified solution that covers all three dimensions of trend aggregation faces unique challenges.

*Sharing diverse nested Kleene patterns*. Kleene patterns in a workload can be nested and quite diverse. For example, the following two queries contain the same event types in the same order, yet it is not clear what, if anything, can be shared: $\mathsf{SEQ}(A+, B, \mathsf{SEQ}(C, D)+)$ and $\mathsf{SEQ}(\mathsf{SEQ}(A, B)+, C, D)+$. In spite of Kleene patterns being among the most expensive CEP operations, effectively sharing the execution of such patterns has not previously been tackled.

*Shared computations without trend construction*. Traditional sharing strategies are incompatible with the online computation of event trend aggregation. Namely, each shared sub-pattern is aggregated separately, and then must be stitched together to form final matches. However, the validation for such stitching across partial results of Kleene sub-patterns, considering both temporal order and predicate constraints, requires keeping the actual sub-trends – which is in direct conflict to online aggregation that avoids trend construction [10, 11], with the latter our target here.

*Optimizing the Kleene sharing plan*. The prevailing assumption that more shared sub-patterns result in larger performance improvement does not always hold in the context of online trend aggregation due to the sharing overhead incurred for the correct stitching of partial aggregation results. Given the exponential complexity of the sharing plan search space, we thus need effective optimization algorithms that are supported by a cost-benefit model to select an efficient execution plan.

**Contributions**. We design a unique mu̲l̲ti-query s̲hared e̲vent trend aggregation (MUSE) approach to tackle the challenges above.

1. We introduce the MUSE optimizer that represents the nested Kleene pattern workload as a compact data structure called the global plan. The MUSE optimizer refines the global plan to select only beneficial sharing candidates in quadratic time by leveraging our cost-based benefit model.

2. We design the MUSE executor to avoid event trend construction by saving materialized states that contain intermediate aggregation results per query prior to executing shared sub-pattern aggregation.

3. Our preliminary experiments on real and synthetic event streams demonstrate that MUSE achieves 4 orders of magnitude performance improvement over state-of-the-art approaches.

## 2 Preliminaries

An *event* is a data tuple describing an incident of interest. The *event type* $E$ of $e$ specifies the set of *event attributes* associated with $e$. A specific attribute of $e$ is referred to as $e.attr$. An *event stream* $I$ is a sequence of events that arrive in order of their occurrence time. An event consumer continuously monitors the stream with event queries. We adopt the commonly used query language and semantics from SASE [1, 13, 14].

**Definition 1 (Kleene Pattern).** *A pattern $p$ corresponds to an event type $E$, a Kleene plus operator $p_i+$, or an event sequence operator $\mathsf{SEQ}(p_i, p_j)$ applied to patterns $p_i$ and $p_j$. Patterns $p_i$ and $p_j$ are called sub-patterns of $p$. A Kleene pattern is a pattern with at least one Kleene operator. If a Kleene operator in $p$ is* *applied to an expression that contains another Kleene operator, the pattern $p$ is nested.*

**Definition 2 (Event Trend Aggregation Query).** *An event trend aggregation query $q$ consists of five clauses:*

- *Aggregation result specification (RETURN clause),*
- *Kleene pattern $p$ (PATTERN clause),*
- *Predicates $\theta$ (optional WHERE clause),*
- *Grouping $G$ (optional GROUPBY clause), and*
- *Window $w$ (WITHIN/SLIDE clause).*

*An event trend $tr = (e_1, \ldots, e_k)$ is a sequence of events that matches the structure specified by a pattern $p$. All events in the trend $tr$ satisfy predicates $\theta$, have the same values of grouping attributes $G$, and are within one window $w$.*

**Problem Statement**. Given a workload of event trend aggregation queries $Q$ and a high-velocity event stream $I$, the *Multi-query Event Trend Aggregation Problem* is to minimize the execution time needed to evaluate $Q$ over stream $I$.

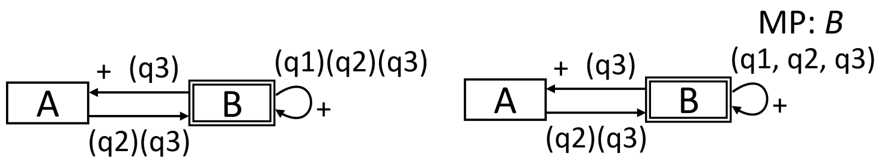
## 3 Multi-query Trend Aggregation

**MUSE Sharing Plan**. We represent each pattern as a Finite State Automaton (FSA) [1, 8, 13, 14], where each node is an event type and each edge indicates that events of these types are adjacent in a match. To expose all sharing opportunities, FSAs for each individual patterns are merged together into one integrated *global plan* (Figure 1(a)). In the global plan, each event type, shown as a rectangle, appears once. An event type shown as a double rectangle is the end type in a pattern. Each edge is labeled by the set of queries for which this edge holds. An edge that is labeled by $k$ queries corresponds to a sub-pattern that appears in those $k$ queries. If $k > 1$, we call such an edge *shareable*. An edge may also be marked by a "+" symbol if it corresponds to a Kleene pattern.

The MUSE optimizer analyzes the sharing opportunities in the global plan and produces the final MUSE sharing plan to guide runtime execution. In Figure 1(a), the label $(q1)(q2)(q3)$ on the Kleene edge $<B, B>$ denotes that $q1$, $q2$, and $q3$ are processed separately (not shared) in plan $P1$. In plan $P2$ in Figure 1(b), the label $(q1, q2, q3)$ denotes that the computation of $B+$ for all three queries is shared.

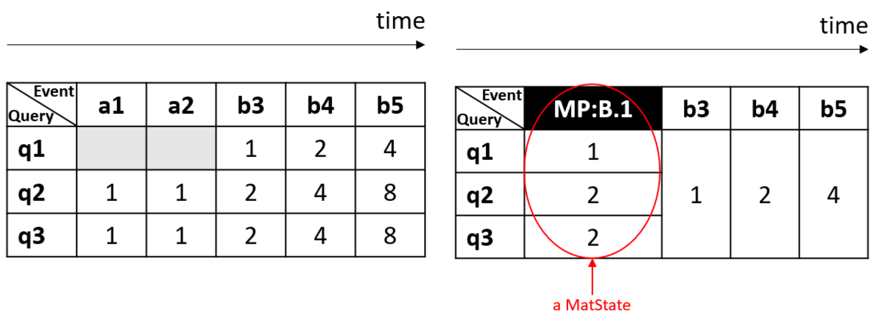
Whenever queries are shared, the sharing plan must also attach a *MatPoint* (materialization point). The MatPoint indicates the position in the plan where a shared sub-pattern begins. MatPoint candidates can be of any event type in a shareable sub-pattern. In Figure 1(b), the MatPoint for $(q1, q2, q3)$ is $B$ denoted by *MP:B*.

**MUSE Executor.** The design goal of the MUSE executor is to exploit sharing to eliminate re-computations while still maintaining online aggregation. This is achieved by introducing *MatStates* (Materialized States) for each MatPoint. A MatState stores a value for each query, corresponding to each query's intermediate trend aggregate. This allows us to share computation among relevant queries.



(a) Plan *P1* (b) Plan *P2*

**Figure 1:** Plans for query patterns *q1* = *B*+, *q2* = SEQ(*A*,*B*+), and *q3* = SEQ(*A*,*B*+)+: (a) Plan *P1* chooses no sharing; (b) Plan *P2* shares sub-pattern *B*+ for queries *q1*, *q2*, and *q3*.


time
time
a MatState

| Query \ Event | a1 | a2 | b3 | b4 | b5 |
|---|---|---|---|---|---|
| q1 | | | 1 | 2 | 4 |
| q2 | 1 | 1 | 2 | 4 | 8 |
| q3 | 1 | 1 | 2 | 4 | 8 |

| Query \ Event | MP:B.1 | b3 | b4 | b5 |
|---|---|---|---|---|
| q1 | 1 | | | |
| q2 | 2 | 1 | 2 | 4 |
| q3 | 2 | | | |

(a) Non-shared execution (b) Shared execution

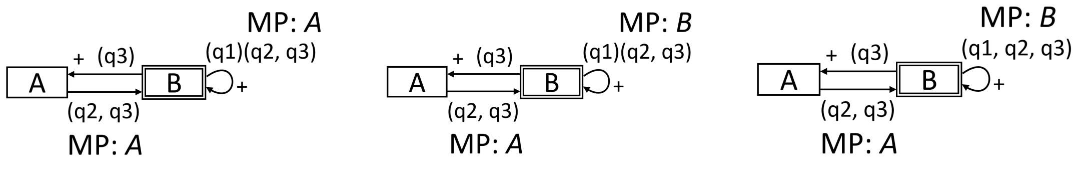
**Figure 2:** Intermediate aggregate count storage for stream *I* = *(a1, a2, b3, b4, b5)* under: (a) non-shared execution of plan *P1*; (b) execution sharing sub-pattern *B*+ for queries *q1*, *q2*, and *q3* as per plan *P2* from Figure 1.

Figure 2(a) depicts non-shared incremental count aggregates for each query and each event following the non-shared plan $P1$ in Figure 1(a). Additional structures for indexing are not shown for compactness. In Figure 2(b), we share $B+$ given the MUSE plan $P2$ in Figure 1(b). Since the MatPoint is event type $B$, a MatState is created when a $B$ event arrives following non-shared events. We can then share incremental aggregation of $B$ events and later lazily recover the actual counts by referring to the MatState. For example, the actual counts in event $b5$ are computed by multiplying the shared count in $b5$ by the value in its corresponding MatState. The actual count in $b5$ for query $q3$ is $4 \times 2 = 8$. In this simple example, we only show one MatState, but in general, there is a one-to-many correspondence between each MatPoint and its MatStates.

**MUSE Benefit Model**. Maintaining MatStates for shared execution causes computational overhead. Thus, we exclude sharing opportunities that are not beneficial as determined by the MUSE optimizer. At compile time, the optimizer computes the benefit of sharing based on stream statistics and produces the final MUSE sharing plan to guide the executor. We quantify the trade-off between sharing and not sharing in our MUSE benefit model. For our benefit model, we analyze the cost between two adjacent event types.

Consider a sub-pattern $(E', E)$ sharable in queries $Q$. Let $|E|$ ($|E'|$) denote the number of events of type $E$ ($E'$) per query window. When an event $e$ of type $E$ arrives in the stream, the following steps must be performed for a single non-shared query $q \in Q$.

(1) Locate all previously matched events $e'$ of type $E'$ such that the events $e$ and $e'$ satisfy the predicates in $q$. Assuming that we use common tree-structure indexing techniques, this cost is



(a) (b) (c)

**Figure 3:** More plan candidates for query patterns *q1* = *B*+, *q2* = SEQ(*A*,*B*+), and *q3* = SEQ(*A*,*B*+)+: (a) plan shares SEQ(*A*,*B*+) for *q2* and *q3* with MatPoint at *A*; (b) plan changes MatPoint to *B* for sub-pattern *B*+; (c) plan shares all three queries for *B*+

logarithmic in $|E'|$ [8].

(2) Compute the intermediate aggregate of $e$. In the worst case, all events previously matched by $q$ are predecessor events of $e$. The cost is linear in $|E'|$.

(3) Store $e$ in the data structure for $q$. Again, to support expressive predicates of $q$, the data structure is tree-based and sorted by the attribute values that are accessed by the predicates of $q$. The cost is logarithmic in $|E|$.

Then, the cost of non-shared execution for $(E', E)$ for all queries in $Q$ is quadratic in the number of events per window:

$$NonShared((E', E), Q) = |Q| \times |E| \times (\log |E'| + |E'| + \log |E|) \tag{1}$$

The main difference in shared trend aggregation is that we now maintain a set of MatStates for all queries, rather than a separate aggregate for each query. We denote $m$ as the maximum number of MatStates shared at the same time for event type $E$.

$$Shared((E', E), Q) = |E| \times (\log(|E'|) + m \times |E'| + \log(|E|)) \tag{2}$$

In addition, sharing introduces CPU overhead to compute MatStates, which includes the cost of evaluating counts. The cost of evaluating the counts for all shared $E$ events for all queries is linear in the number of MatStates.

$$Eval(E, Q) = |E| \times |Q| \times m \tag{3}$$

**Definition 3 (Benefit).** *The benefit of sharing edge* $<E', E>$ *for queries* $Q$ *is computed as the difference between the cost of the non-shared and shared execution plans. Let* $|E|$ *(*$|E'|$*) denote the number of events of type E (*$E'$*) per query window. The major cost factors of benefit are* $|E|$*,* $|E'|$*, the number of shareable queries* $|Q|$*, and the maximum number* $m$ *of MatStates shared at the same time.*

$$\begin{aligned} Benefit(<E', E>, Q) = {} & |Q| \times |E| \times (\log |E'| + |E'| + \log |E|) \\ & - |E| \times (\log(|E'|) + m \times |E'| + \log(|E|)) - |E| \times |Q| \times m \end{aligned}$$

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

Intuitively, if we maintain a fixed number of MatStates, the more queries that share a sub-pattern, the more beneficial it becomes.

**Lemma 1.** *Let $p$ be a shareable sub-pattern in a set of queries $Q$ with a fixed number of MatStates. For any edge $<E', E>$ in $p$, as the number of queries $|Q|$ increases, $Benefit(<E', E>, Q)$ increases.*

*Proof.* We take the partial derivative of $Benefit((E', E), Q)$ with respect to $|Q|$. Note that $m$ is not dependent upon $|Q|$.

$$\frac{\partial Benefit((E', E), Q)}{\partial |Q|} = |E| \times (\log(|E'|) + |E'| + \log(|E|) - m)$$

Since $|E'| \geq m$, $\partial Benefit((E', E), Q, \tau)/\partial |Q|$ is positive everywhere. □

Similarly, if the number of queries that share a sub-pattern remains constant, the sharing plan that generates less MatStates to be shared at the same time produces a higher benefit.

**Lemma 2.** *Let $p$ be a shareable sub-pattern in a fixed set of queries $Q$. For any edge $<E', E>$ in $p$, a smaller number of MatStates $m$ implies greater $Benefit(<E', E>, Q)$.*

*Proof.* We take the partial derivative of $Benefit((E', E), Q)$ with respect to $m$.

$$\frac{\partial Benefit((E', E), Q)}{\partial m} = -|E| \times |E'| - |E| \times |Q|$$

$\partial Benefit((E', E), Q)/\partial m$ is negative everywhere. □

**Search Space of Alternative MUSE Plans**. A MUSE sharing plan dictates which subsets of queries are shared for each of the subexpressions in the MUSE plan and where within the MUSE plan to place which MatPoint for each shared expression. The search space thus consists of all possible MUSE sharing plans, i.e., all combinations of (1) query subsets for each of the MUSE plan edges and (2) MatPoint candidates (sharable event types).

(1) *Query subset*. The queries on each edge of the plan are partitioned into non-overlapping subsets. A subset of size 1 indicates that the query is not shared. For example, in Figure 3(a), the label $(q1)(q2, q3)$ on edge $<B, B>$ denotes that queries $q2$ and $q3$ are shared, but $q1$ is not shared. The number of possible partitions increases exponentially in the number of queries.

(2) *MatPoint*. In addition, each shared query subset also selects a MatPoint. For example, in Figures 3(a) and 3(b), the optimizer can select either event type $A$ or event type $B$ as the MatPoint for the Kleene edge $<B, B>$ for the shareable sub-pattern $\mathsf{SEQ}(A, B+)$ for queries $q2$ and $q3$. The pool of MatPoint candidates for each query subset on each edge is linear in the number of event types.

**MUSE Optimizer**. Next, we tackle the challenge of finding an optimized sharing plan that balances the trade-off between its benefit and overhead. Aggressively sharing all possible shareable sub-patterns is not likely to achieve this. Hence, our optimizer instead takes as input the global plan, then refines and annotates it with sharing decisions to produce the final beneficial sharing plan.

Fortunately, many arrangements in the search space can be pruned based on Lemmas 1 and 2. For example, we can prune the plan in Figure 3(b) since by Lemma 1, benefit will increase by adding $q1$ to the shared query set given that sharing $q1$ does not introduce new MatStates. Rather, we thus ought to consider the plan in Figure 3(c) instead.

To leverage our pruning principles, we process decisions sequentially from the start to the end of each shareable sub-pattern. Thus, the MUSE optimizer traverses the global plan in a modified topological sort order. When we process a node, we select the most beneficial arrangement for each of its outgoing edges.

Note that Kleene closure introduces cycles to the global plan, but the topological sorting algorithm assumes a DAG. Fortunately, the query input has already identified which edges cause cycles, i.e., the Kleene+ edges, and they have been marked in the global plan. To modify the algorithm, we restrict the topological sorting algorithm to exclusively use the SEQ edges. Further, to then ensure a beneficial plan, we may have to revoke previously made sharing decisions and re-analyze event types for certain Kleene sub-patterns. Our optimizer algorithm stipulates that if an event type $E$ is nested in $t$ cycles in the global plan, then $E$ is analyzed at most $t + 1$ times. The number of edges in the global plan is an upper bound for $t$.

## 4 Preliminary Evaluation

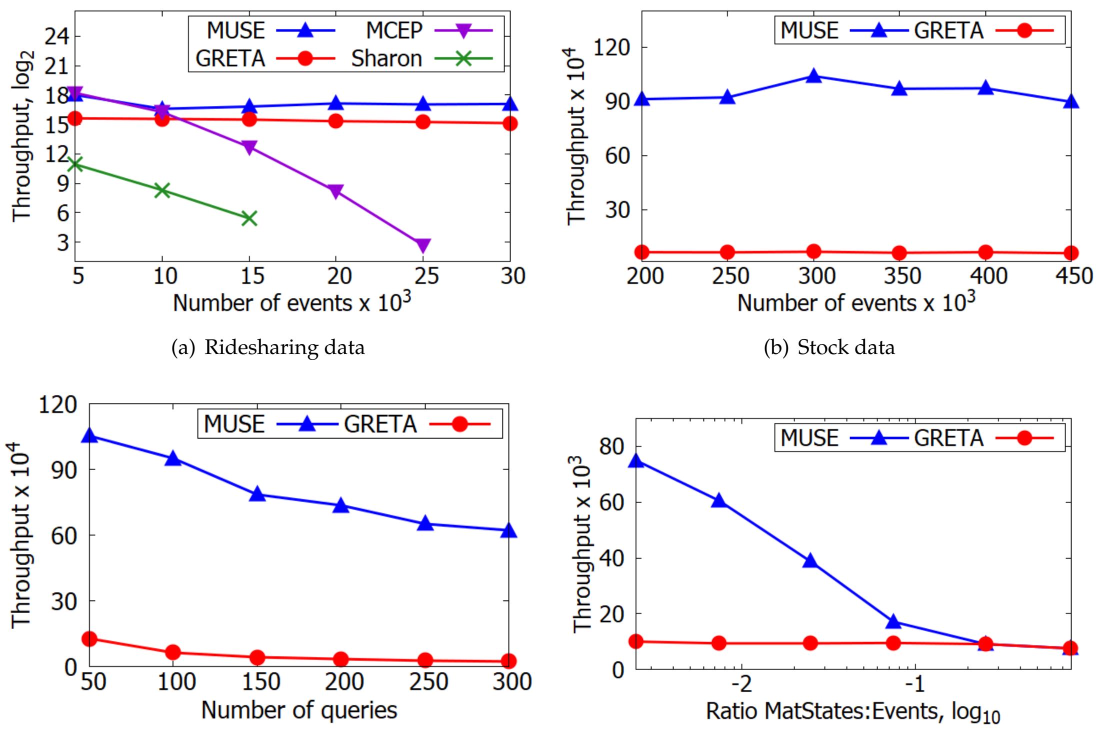
We implemented MUSE in Java with JRE 1.7.0_80 running on Ubuntu 16.04.6 with 503GB of RAM. We evaluate our MUSE approach using the following data sets. The *NASDAQ Stock Market Real Data Set* [2] contains transaction records of more than 3200 companies for one month. The *Ridesharing Data Set* was generated by our stream simulator to control the rate of event types in a stream that models transactions between riders and drivers in 20 districts.

For each data set, we generate four Kleene pattern workloads. We vary three major cost factors based on our benefit model: the number of events per window, the number of queries, and the ratio of MatStates to the number of events per window. Unless stated otherwise, each query workload consists of 100 queries. We execute each experiment on each workload three times and report the average.

We compare MUSE to three state-of-the-art approaches introduced in Section 1: *MCEP* [5], *Sharon* [10], and *GRETA* [8]. We use *throughput* as the metric to evaluate the efficiency of the MUSE approach. Throughput is the number of events processed by all queries per second, which is commonly reported for streaming systems [5, 7, 9, 13].

**Varying the Number of Events**. In Figure 4(a), we vary the number of events per window for all four approaches.

Sharon [10] does not support Kleene closure. Thus, it evaluates a set of flattened event sequence queries to detect all trends matched by one Kleene query. Due to this increased workload, MUSE has a throughput gain of 3 orders of magnitude over Sharon at 15k events per window. Sharon does not terminate for more than 15k events per window, so we do not report Sharon's results for 20k, 25k, and 30k events per window.



(a) Ridesharing data

(b) Stock data

(c) Stock data

(d) Ridesharing data

**Figure 4:** Throughput varying (a-b) number of events per window, (c) number of queries, and (d) MatState ratio

MCEP [5] is a two-step approach that utilizes sharing and reordering optimizations for the trend construction step. It achieves 2 orders of magnitude speed-up compared to Sharon. However, its throughput decreases exponentially in the number of events. Thus, MCEP does not terminate for several hours if the number of events per window exceeds 25k. GRETA [8] is an online approach that lacks sharing optimizations. At 25k events per window, GRETA outperforms MCEP by 3 orders of magnitude, while MUSE outperforms MCEP by 4 orders of magnitude (Figure 4(a)).

We evaluate performance on a higher rate event stream with over $10^5$ events per window in Figure 4(b). Measurements for Sharon and MCEP are not shown because these approaches fail to return results for several hours. While MUSE processes each event once for all queries, GRETA repeats computations for each query. Thus, MUSE achieves 14-fold increase in throughput over GRETA.

**Varying the Number of Queries**. In Figure 4(c), we observe that the more queries are shared, the higher the benefit compared to the non-shared online approach. This confirms Lemma 1. Specifically, MUSE achieves from 7-fold to 25-fold throughput gain over GRETA when the number of queries increases from 50 to 300 for 200K events per window (epw) for the stock data set.

**Varying MatState Ratio**. We observe increased sharing benefit with less number of MatStates, confirming Lemma 2. In Figure 4(d), we report throughput with varying ratios of the number of

MatStates to the number of matched events. For streams with very few MatStates, MUSE achieves nearly 7-fold increase in throughput compared to the non-shared system, GRETA. Throughput gain decreases as more MatStates must be maintained. Eventually, it becomes no longer beneficial to share when the number of MatStates grows too large. Under these conditions, the MUSE optimizer decides not to share, and MUSE performs similarly to GRETA.

## 5 Conclusions

Our MUSE approach tackles shared aggregation of event trends matched by diverse nested Kleene pattern queries over high speed streaming data. The MUSE optimizer exposes all sharing opportunities in the global plan and selects the beneficial ones based on the MUSE benefit model. The MUSE executor minimizes query re-computation and trend construction overhead by deploying our lightweight MatState sharing technique. Our preliminary evaluation shows that the MUSE achieves several orders of magnitude performance improvement over state-of-the-art solutions on both real and synthetic data sets.

## References

[1] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman. Efficient pattern matching over event streams. In *SIGMOD*, pages 147–160, 2008.

[2] EODData. Historical Price Data. `https://www.eoddata.com`, 2019.

[3] M. Hong, M. Riedewald, C. Koch, J. Gehrke, and A. Demers. Rule-based multi-query optimization. In *EDBT*, pages 120–131, 2009.

[4] I. Kolchinsky and A. Schuster. Join query optimization techniques for complex event processing applications. In *VLDB*, pages 1332–1345, 2018.

[5] I. Kolchinsky and A. Schuster. Real-time multi-pattern detection over event streams. In *SIGMOD*, pages 589–606, 2019.

[6] I. Kolchinsky, I. Sharfman, and A. Schuster. Lazy evaluation methods for detecting complex events. In *DEBS*, pages 34–45, 2015.

[7] Y. Mei and S. Madden. ZStream: A cost-based processor for adaptively detecting composite events. In *SIGMOD*, pages 193–206, 2009.

[8] O. Poppe, C. Lei, E. A. Rundensteiner, and D. Maier. GRETA: Graph-based Real-time Event Trend Aggregation. In *VLDB*, pages 80–92, 2017.

[9] O. Poppe, C. Lei, E. A. Rundensteiner, and D. Maier. Event trend aggregation under rich event matching semantics. In *SIGMOD*, pages 555–572, 2019.

[10] O. Poppe, A. Rozet, C. Lei, E. A. Rundensteiner, and D. Maier. Sharon: Shared online event sequence aggregation. In *ICDE*, pages 737–748, 2018.

[11] Y. Qi, L. Cao, M. Ray, and E. A. Rundensteiner. Complex event analytics: Online aggregation of stream sequence patters. In *SIGMOD*, pages 229–240, 2014.

[12] M. Ray, C. Lei, and E. A. Rundensteiner. Scalable pattern sharing on event streams. In *SIGMOD*, pages 495–510, 2016.

[13] E. Wu, Y. Diao, and S. Rizvi. High-performance Complex Event Processing over streams. In *SIGMOD*, pages 407–418, 2006.

[14] H. Zhang, Y. Diao, and N. Immerman. On complexity and optimization of expensive queries in CEP. In *SIGMOD*, pages 217–228, 2014.

[15] S. Zhang, H. T. Vo, D. Dahlmeier, and B. He. Multi-query optimization for complex event processing in SAP ESP. In *ICDE*, pages 1213–1224, 2017.