Nested Pattern Queries Processing Optimization over Multi-Dimensional Event Streams

Fuyuan Xiao and Masayoshi Aritsugi
Computer Science and Electrical Engineering
Graduate School of Science and Technology, Kumamoto University
2-39-1 Kurokami, Chuo-ku, Kumamoto 860-8555, Japan
{*xiao@dbms., aritsugi@*}*cs.kumamoto-u.ac.jp*

***Abstract*—Recently, many modern applications have required complex event processing technology to analyze multi-dimensional stream big data available in real-time data feeds. To address the above requirement, we develop a novel real-time event stream processing method called multi-query optimization strategy (MQOS). MQOS aims at exploiting not only common sub-expressions among nested event pattern queries, but also replicas of the appropriate common operators' results for the queries needed to minimize recalculation and re-communication costs.**

We first design a triaxial hierarchy consisting of nested query pattern, nested query concept and operator type hierarchies to specify the relationship among the sub-expressions of queries. Next, based on the triaxial hierarchy, we devise a cost-based heuristic to find an optimized query execution plan with minimum costs of operators and communications. We then propose three reuse schemes of common sub-expressions: nested query pattern-based, nested query concept-based and operator type-based reuse schemes. By integrating the optimized query execution plan-find approach with the three reuse schemes, we present the MQOS to achieve nested pattern queries processing optimization. Finally, our experiments tested on StreamBase under different workload conditions demonstrate the superiority of MQOS.

***Keywords*-Complex event processing (CEP); Multi-dimensional event stream; Nested pattern query;**

I. Introduction

The complex event processing (CEP) over event streams called as event stream processing (ESP) has gained a lot of attentions in recent years, due to their expected capabilities to help business detection, online financial transactions, and sensor networks. Various systems have been proposed in recent years, such as Aurora [1], Borealis [2], STREAM [3], Telegraph [4], SASE [5], Cayuga [6], and PIPES [7] as academic research systems, and Coral8 & Aleri [8], StreamBase [9], and Oracle CEP [10] as products in the industry. However, there are still many open issues to be solved in the CEP and ESP fields [11].

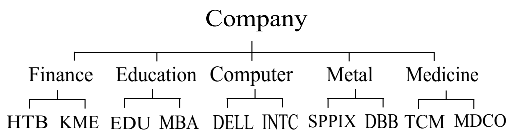
One of the issues is how to process complex pattern matching over stream big data efficiently. As described in [11], ESP systems have both characteristics of CEP and data stream processing (DSP) systems. CEP systems [12], [13], [14], [15] must be able to support sophisticated pattern matching over real time event streams, while DSP systems [4], [16], [17] must be able to handle high volume data and continuous data streams with low latency. Fig. 1 shows a concept hierarchy of stock companies and examples of complex event pattern queries, which are written in a query language based on related work [18], [12], [15] as explained in Section III. Such complex pattern queries should be able to involve nests of sequence (SEQ), conjunction (AND), negation (NEG), and combination of them, namely, SEQ_NEG and AND_NEG operators. Also, stream big data would have many such complex dimensions that objects, time, location, and event type, in nature. For example, time of sale may be organized as a day-month-quarter-year hierarchy and product may be organized as a product-category-industry hierarchy. We therefore propose optimization of nested pattern queries over multi-dimensional event streams in this paper.

Our proposal in this paper is based on online analytical processing (OLAP) systems [19] and multi-query optimization [20], and can optimize nested pattern queries over multi-dimensional event streams. Similar to E-cube [18], we design a new model to introduce OLAP functionality for multi-dimensional event stream analysis. We also design reuse schemes of sub-expressions among nested event pattern queries for high throughput of CEP systems. We attempt to exploit the sub-expressions of multiple queries in terms of not only hierarchical events but also SEQ, AND, NEG and their combination operators. To our knowledge, there is no optimization study of processing multiple nested event pattern queries in which all of the above operators can appear. Main contributions of this paper are as follows.

- We design a triaxial hierarchy consisting of nested query pattern, nested query concept and operator type hierarchies to specify the relationship among the sub-expressions of queries which can have nested SEQ, AND, NEG and their combination operators.
- On the foundation of the triaxial hierarchy, we design an integrated directed acyclic graph (IDAG) cost model to estimate relative costs of different reuse cases. This cost model allows us to find an optimized query exe-

0730-3157/13 $26.00 © 2013 IEEE
DOI 10.1109/COMPSAC.2013.14

Workload 1:
q_1=*SEQ*(HTB, EDU, MBA)
q_2=*SEQ_NEG*(HTB, !SPPIX, EDU, MBA)
q_3=*SEQ*(Medicine, DELL, KME)
q_4=*SEQ*(Medicine, Computer, Metal)
q_5=*SEQ*(HTB, EDU, Medicine, Computer)



Workload 2:
q_6=*AND*(*SEQ*(KME, MDCO), *SEQ*(DBB, DELL), *SEQ*(HTB, TCM))
q_7=*SEQ*(KME, MDCO, DBB, DELL, HTB, TCM)
q_8=*SEQ_NEG*(KME, MDCO, !INTC, DBB, DELL, HTB, TCM)

Workload 3:
q_9 =*AND*(*SEQ*(MDCO, EDU), HTB)
q_{10} =*SEQ*(MDCO, EDU, HTB)
q_{11} =*SEQ_NEG*(MDCO, EDU, !INTC, HTB)
q_{12} =*AND*(INTC, *SEQ*(MDCO, EDU, HTB))
q_{13} =*SEQ*(Finance, Metal, Education, Medicine)
q_{14} =*SEQ*(Finance, Metal, *AND*(*SEQ*(MDCO, EDU), HTB))
q_{15} =*SEQ*(Finance, Metal, Education, Medicine, DELL)
q_{16} =*SEQ*(Finance, Metal, Education, Medicine, Computer)

Figure 1. A concept hierarchy of stock companies and event pattern query workloads.

cution plan.

- We design three reuse schemes to alternatively get the replicas number of common sub-expressions and to prepare for computing the rest of queries. The schemes are nested query pattern-based reuse scheme (NQPRS), nested query concept-based reuse scheme (NQCRS), and operator type-based reuse scheme (OTRS).
- By integrating the optimized query execution plan approach with the three reuse schemes, a multi-query optimization strategy (MQOS) is proposed to achieve high throughput of nested pattern queries processing. MQOS reuses not only common sub-expressions among nested pattern queries but also replicas of the appropriate common operators' results for the queries needed to minimize recalculation and re-communication costs.
- We did experiments of the workloads shown in Fig. 1. The results show that MQOS can outperform other alternate processing strategies.

The rest of this paper is organized as follows. We review necessary related work in Section II. Nested pattern query processing basics are introduced in Section III. In Section IV, we propose a triaxial hierarchy in terms of nested query pattern, nested query concept and operator type hierarchies. In Section V, we first devise IDAG cost model to find an optimized query execution plan. Next, we design three reuse schemes: NQPRS, NQCRS and OTRS. Then, by integrating the cost model with the three reuse schemes, we propose MQOS. We compare MQOS with related approaches under different workload conditions in the StreamBase system and report the experiment results in Section VI. We conclude our paper in Section VII.

II. Related Work

Although there have been studies of CEP systems [5], [6], [18], [12], [13], they can only support flat queries with SEQ operators. For example, ZStream [12] focuses on searching for an optimal execution plan of a single pattern query, but it does not consider multiple nested queries. Cayuga [6] supports sub-queries in FROM clause, but it does not allow us to apply NEG operators over composite event types.

Liu et al. [15] support nested pattern detection queries with SEQ, AND, NEG, and OR operators. However, they focus on optimization of a single pattern query and do not treat with multiple pattern queries. In addition, they do not support OLAP functionality for multi-dimensional event analysis.

Cuzzocrea et al. [21], [22] try to support efficient OLAP analysis on multidimensional data streams. However, they do not focus on complex event pattern query processing.

E-cube [18] introduces OLAP for multi-dimensional event stream analysis. An event concept hierarchy and E-Cube hierarchy of queries with query concept refinement and query pattern refinement are introduced for supporting OLAP operations over multi-dimensional event streams. However, it focuses only on SEQ operators with NEG operators.

In contrast to these studies, our work supports OLAP functionality for multi-dimensional event stream analysis and focuses on optimization of multiple nested queries with SEQ, AND, NEG and their combination operators.

III. Nested Pattern Query Processing Basics

In this section, we introduce basic event model, nested pattern query language, types of operators and their formal semantics. They all are based on related studies, e.g., [18], [12], [13], [15].

A. Event Model

An event which represents an instance is an occurrence of interest at a point in time. Basically, events can be classified into primitive events and composite events.

Definition 1: A primitive event e_i is typically modeled multi-dimensionally denoted as e_i=e($e_i.t$, ($e_i.st = e_i.et$), $< a_1, \ldots, a_m >$), where $e_i.t$ is event type that describes the essential features of e_i, $e_i.st$ is the start time-stamp of e_i, $e_i.et$ is the end time-stamp of e_i, $< a_1, \ldots, a_m >$ are other attributes of e_i and the number of attributes in $e(\cdot)$ denotes the dimensions of interest.

Definition 2: Based on Definition 1, a composite event is denoted as e=e($e.t$, ($\min\limits_{1\leq i\leq n} e_i.st \neq \max\limits_{1\leq i\leq n} e_i.et$), $< a_1, \ldots, a_g >$).

B. Nested Pattern Query Language

We introduce the following language for specifying nested pattern queries which supports nests of SEQ, AND, NEG and their combination operators:

PATTERN (Event Expression: composite event expressed by nesting of SEQ, AND, NEG and their combination operators)
WHERE (Qualification: value constraint)
WITHIN (Window: time constraint)

C. Types of Operators and their Formal Semantics

We define operators that appear in the PATTERN clause of a query. In the following, E_i denotes an event type.

Definition 3: A SEQ operator [18], [15] specifies a particular order according with the start time-stamps in which the event must occur to match the pattern and thus form a composite event:

$$SEQ(E_1, \ldots, E_i, \ldots, E_n) = \{< e_1, \ldots, e_i, \ldots, e_n > | (e_1.st < \ldots < e_i.st < \ldots < e_n.st) \wedge (e_1.t = E_1) \wedge \ldots \wedge (e_i.t = E_i) \wedge \ldots \wedge (e_n.t = E_n)\}.$$

A $SEQ(E_1$, ..., $SEQ(E_i$, ..., $E_j)$, ..., $E_n)$ expression can be flattened as $SEQ(E_1$, ..., E_i, ..., E_j, ..., $E_n)$.

Definition 4: An AND operator [15] takes a set of event types as input and events occur within a specified time window without specified time order:

$$AND(E_1, \ldots, E_i, \ldots, E_n) = \{< e_1, \ldots, e_i, \ldots, e_n > | (e_1.t = E_1) \wedge \ldots \wedge (e_i.t = E_i) \wedge \ldots \wedge (e_n.t = E_n)\}.$$

An $AND(E_1$, ..., $AND(E_i$, ..., $E_j)$, ..., $E_n)$ expression can be flattened as $AND(E_1$, ..., E_i, ..., E_j, ..., $E_n)$

Definition 5: The symbol "!" before an event type E_i expresses the negation of E_i and indicates that E_i is not allowed to appear in the specified position. The operator with symbol "!" denotes as NEG operator [18], [13], [15].

Definition 6: A $SEQ\_NEG(E_{i-1}$, $!E_i$, $E_{i+1})$ operator specifies that no event of E_i can appear between E_{i-1} and E_{i+1} with specified time order:

$$SEQ\_NEG(E_{i-1}, !E_i, E_{i+1}) = \{< e_{i-1}, e_{i+1} > | (e_{i-1}.st < e_{i+1}.st) \wedge (e_{i-1}.t = E_{i-1}) \wedge (e_{i+1}.t = E_{i+1}) \wedge (\neg\exists e_i \; where \; (e_i.t = E_i) \wedge (e_{i-1}.et < e_i.st \leq e_i.et < e_{i+1}.st))\}.$$

This operator can be interpreted as $NEG(SEQ(E_{i-1}$, $E_{i+1})$, $!E_i)$ with time constraint $e_{i-1}.et < e_i.st \leq e_i.et < e_{i+1}.st$.

Definition 7: An $AND\_NEG(E_i$, $!E_k$, $E_j)$ operator specifies that no event of E_k can appear between E_i and E_j without specified time order:

$$AND\_NEG(E_i, !E_k, E_j) = \{< e_i, e_j > | (e_i.t = E_i) \wedge (e_j.t = E_j) \wedge (\neg\exists e_k \; where \; (e_k.t = E_k) \wedge ((e_i.et < e_k.st \leq e_k.et < e_j.st) \vee (e_j.et < e_k.st \leq e_k.et < e_i.st)))\}.$$

This operator can be interpreted as $NEG(AND(E_i$, $E_j)$, $!E_k)$ with time constraint $(e_i.et < e_k.st \leq e_k.et < e_j.st)$ $\vee$ $(e_j.et < e_k.st \leq e_k.et < e_i.st)$.

Table I
NOTATION

Notation	Meaning
EPQ(E_i, E_j)	abstract expression of event pattern query with event types E_i and E_j
R_{E_i}	input rate of source data of E_i
$\lambda(s_c)$	input rate of stream s_c
$L(s_c)$	network latency of stream s_c
TW_p	time window specified in a given pattern query
P_{E_i}	selectivity of all single-class predicates for E_i
ℓ	number of streams
N_{E_i}	number of events of E_i within TW_p
$P^t_{E_i,E_j}$	selectivity of the implicit time predicate between E_i and E_j, with $e_i.et < e_j.st$
P_{E_i,E_j}	selectivity of multi-class predicates between E_i and E_j
$C^i_{O_h}$	cost of operator O_h to access its input data
$C^o_{O_h}$	cost of operator O_h to generate its output data
C_{O_h}	total cost of operator O_h
C_{s_c}	communication cost used for stream s_c
P_k	candidate query execution plan
G_k	integrated directed acyclic graph P_k
$Total\ Cost(G_k)$	total operator cost of P_k
PT_{q_t}	expression or sub-expression of query q_t
$R_{PT_{q_t}}$	results of PT_{q_t}
P_{opt}	optimized query execution plan
PL_{P_k}	number of subsets of P_k
P^s	set of P_k with the smallest PL_{P_k}

IV. Triaxial Hierarchy Model

An event concept hierarchy is commonly used to summarize information at different levels of abstraction. Based on the hierarchies introduced in [18], for efficiently and directly reusing common operators' results, we define a novel triaxial hierarchy to specify the relationship among the sub-expressions of queries. Dissimilar to [18], in our new definitions of nested query pattern and concept hierarchies, (i) we focus on nests of SEQ, AND, NEG and their combination operators, not just pure SEQ operator, and (ii) the finer level nested query pattern can be formed from its coarser level directly, rather than after eliminating some event types from its coarser level. Here, we define event type E_{ni} is a finer level (resp. coarser level) of an event type E_n (resp. E_{nij}) in an event concept hierarchy expressed as $E_{nij} \subset E_{ni} \subset E_n$. Fig. 1 depicts a concept hierarchy of stock companies which is organized as (stock category)-(company category)-(company) hierarchy. And event type Finance is a finer level of Company and coarser level of HTB, expressed as HTB $\subset$ Finance $\subset$ Company. Table I shows key notations used in the rest of this paper. We introduce EPQ(E_i, E_j), which is an abstract event pattern query expression with event types E_i and E_j. This is only necessary for the discussions in this paper.

Definition 8: A nested query pattern hierarchy (NQPH) is a directed acyclic graph (DAG) where nodes correspond to expressions or sub-expressions of queries denoted as PT_{q_i}. A pattern query is formed by combining primitive

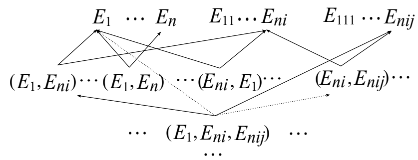


Figure 2. A directed acyclic graph of NQPH.

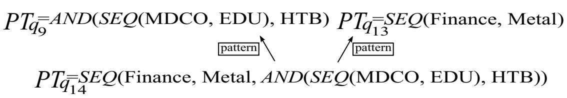


Figure 3. Relationship based on NQPH.

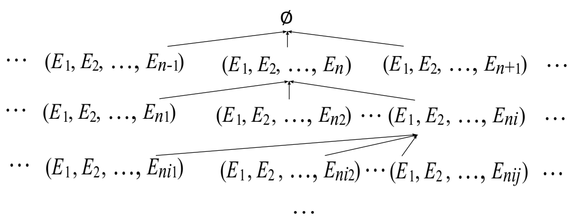


Figure 4. A tree of NQCH of event types.

or composite event types. The longest pattern queries reside at the bottom of NQPH model, while shorter pattern queries progressively reside higher (say EPQ(E_1, E_{ni}, E_{nij})) and much higher (say EPQ(E_1, E_{ni})) in NQPH model as shown in Fig. 2. PT_{q_i} is a coarser level of PT_{q_j}, denoted by $PT_{q_j} \subset_P PT_{q_i}$, if (i) PT_{q_i} has the same kind of expression with part of PT_{q_j}, (ii) $\forall$ EPQ(E_k, ..., E_{kp}, ..., E_{kpf}) $\in PT_{q_i}$ $(1 \le k \le n,\ 1 \le p \le i,\ 1 \le f \le j)$, $\exists$ EPQ(E_m, ..., E_{mp}, ..., E_{mpf}) $\in PT_{q_j}$ $(1 \le m \le n)$, such that E_k = E_m, E_{kp} = E_{mp}, E_{kpf} = E_{mpf}, and (iii) $\exists$ EPQ(E_h, ..., E_{hp}, ..., E_{hpf}) $\in PT_{q_j}$, and $\notin PT_{q_i}$, such that $E_h \neq E_k$, $E_{hp} \neq E_{kp}$, $E_{hpf} \neq E_{kpf}$ $(1 \le h \le n)$.

In other words, we can drill down PT_{q_i} to a finer level PT_{q_j} by adding one or more event types from PT_{q_i}. For example, as shown in Fig. 1, let us consider q_9 = AND(SEQ(MDCO, EDU), HTB), q_{13} = SEQ(Finance, Metal, Education, Medicine) and q_{14} = SEQ(Finance, Metal, AND(SEQ(MDCO, EDU), HTB)). The text in box of Fig. 3 specifies the relationship among those expressions or sub-expressions of pattern queries in terms of Definition 8. Queries q_9 and q_{13} has the same kind of expression with part of q_{14}, i.e., $AND(SEQ(\cdot))$ and $SEQ(\cdot)$, respectively. We can see that query q_{14} is a finer level of query q_9 and sub-expression of query q_{13}. Hence, q_{14} can be formed by reusing sub-expressions of q_{13} and q_9's results, SEQ(Finance, Metal) and AND(SEQ(MDCO, EDU), HTB), and joining them by SEQ operator.

Definition 9: A nested query concept hierarchy (NQCH) is a tree where the apex node has empty episode and the other nodes correspond to expressions or sub-expressions of queries. The pattern queries with the most specific event types reside at the leaves of the tree, while pattern queries with more general event types progressively reside higher (say EPQ(E_1, E_2, ..., E_{ni1})) and much higher (say EPQ(E_1, E_2, ..., E_{ni})) in NQCH model as shown in Fig. 4. PT_{q_i} is a finer level of PT_{q_j}, denoted by $PT_{q_i} \subset_C PT_{q_j}$, if (i) PT_{q_i} has the same kind of expression with part of PT_{q_j}, (ii) $\forall$ EPQ(E_{gp}, ..., E_{hp}, ..., E_{mp}) $\in PT_{q_i}$, $\exists$ EPQ(E_d, ..., E_k, ..., E_n) $\in PT_{q_j}$, such that $E_{gp} \subset E_d$, $E_{hp} \subset E_k$, $E_{mp} \subset E_n$, and (iii) $\exists$ EPQ(E_{ap}, ..., E_{bp}, ..., E_{cp}) $\in PT_{q_j}$, and $\notin PT_{q_i}$, such that $E_{ap} \neq E_{gp}$, $E_{bp} \neq E_{hp}$, $E_{cp} \neq E_{mp}$ $(1 \le a \le b \le c \le d \le g \le n)$.

In other words, we can roll up PT_{q_i} to a coarser level PT_{q_j} by merging PT_{q_i} with the rest expression that belongs to PT_{q_j} but is not captured by PT_{q_i}. For example, as shown in Fig. 1, let us consider q_3 = SEQ(Medicine, DELL, KME) and q_4 = SEQ(Medicine, Computer, Metal). The text in box of Fig. 5 specifies the relationship among those expressions or sub-expressions of queries in terms of Definition 9. We can see that the sub-expression of query q_3 is a finer level of sub-expression of query q_4. Hence, sub-expression of q_4, SEQ(Medicine, Computer), can be formed by reusing sub-expression of q_3's results, SEQ(Medicine, DELL), and merging it with the rest sub-expression, i.e., SEQ(Medicine, INTC).

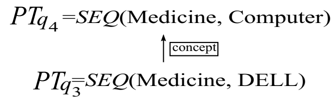


Figure 5. Relationship based on NQCH.

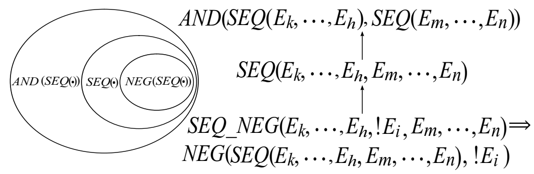


Figure 6. OTH and containment relationships among pattern queries based on it.

We describe three kinds of expressions of pattern queries based on the operator types it used as $AND(SEQ(\cdot))$, $SEQ(\cdot)$ and $NEG(SEQ(\cdot))$. Apart from mining relationships in terms of NQPH and NQCH, Fig. 6 depicts the containment relationships among the three kinds of expressions of pattern queries.

Definition 10: An operator type hierarchy (OTH) is shown in Fig. 6. $AND(SEQ(\cdot))$ PT_{q_i} is a coarser level of $SEQ(\cdot)$ or $NEG(SEQ(\cdot))$ PT_{q_j}, denoted by $PT_{q_j} \subset_O PT_{q_i}$, if for the positive part of pattern query, (i) PT_{q_j} has the same kind of expression with part of PT_{q_i}, (ii) $\forall$ $SEQ(E_k, \ldots, E_h, E_m, \ldots, E_n) \in PT_{q_j}$, $\exists$ $SEQ(E_a, \ldots, E_b, E_c, \ldots, E_d) \in PT_{q_i}$, such that $E_k = E_a$, $E_h = E_b$, $E_m = E_c$, $E_n = E_d$, and (iii) $\exists$ $SEQ(E_m, \ldots, E_n, E_k, \ldots,$

PT_{q_6}=AND(SEQ(KME, MDCO), SEQ(DBB, DELL), SEQ(HTB, TCM))

$\uparrow$ operator type

PT_{q_7}=SEQ(KME, MDCO, DBB, DELL, HTB, TCM)

$\uparrow$ operator type

PT_{q_8}=SEQ_NEG(KME, MDCO, !INTC, DBB, DELL, HTB, TCM)

Figure 7. Relationship based on OTH.

E_h) $\in PT_{q_i}$, and $\notin PT_{q_j}$, such that $E_m \neq E_a$, $E_n \neq E_b$, $E_k \neq E_c$, $E_h \neq E_d$.

$SEQ(\cdot)$ PT_{q_i} is a coarser level of $NEG(SEQ(\cdot))$ PT_{q_j}, denoted by $PT_{q_j} \subset_O PT_{q_i}$, if for the positive part of pattern query, (i) $\forall$ $SEQ(E_k, \ldots, E_h, E_m, \ldots, E_n) \in PT_{q_j}$, $\exists$ $SEQ(E_a, \ldots, E_b, E_c, \ldots, E_d) \in P_{PT_{q_i}}$, such that $E_k = E_a$, $E_h = E_b$, $E_m = E_c$, $E_n = E_d$, (ii) $\forall$ $SEQ(E_a, \ldots, E_b, E_c, \ldots, E_d) \in PT_{q_i}$, $\exists$ $SEQ(E_k, \ldots, E_h, E_m, \ldots, E_n) \in P_{PT_{q_j}}$, such that $E_a = E_k$, $E_b = E_h$, $E_c = E_m$, $E_d = E_n$.

In other words, we can drill down PT_{q_i} to a finer level PT_{q_j} by filtering out some data that are not captured by PT_{q_j}. For example, as shown in Fig. 1, let us consider q_6 = AND(SEQ(KME, MDCO), SEQ(DBB, DELL), SEQ(HTB, TCM)), q_7 = SEQ(KME, MDCO, DBB, DELL, HTB, TCM) and q_8 = SEQ_NEG(KME, MDCO, !INTC, DBB, DELL, HTB, TCM). The text in box of Fig. 7 specifies the relationship among those expressions or sub-expressions of queries in terms of Definition 10. We can see that query q_7 is a finer level of query q_6 and coarser level of query q_8. Thus, q_7 can be formed by reusing q_6's results, i.e., filtering some data that are not captured by q_7. After that, we can generate q_8 by reusing q_7's results, namely, filtering out some data in accordance with INTC negative event type.

Definition 11: A triaxial hierarchy is an integrated directed acyclic graph (IDAG), denoted by $G_k(P_k, S_k)$, where

(i) P_k is a finite set of expressions or sub-expressions of queries $\{PT_{q_1}, \ldots, PT_{q_t}, \ldots, PT_{q_r}\}$, denoted by a candidate query execution plan, which corresponds to a set of operators $\{O_1, \ldots, O_h, \ldots, O_n\}$ called G_k's vertices, and

(ii) S_k denotes a finite set of streams $\{s_1, \ldots, s_c, \ldots, s_\ell\}$ called G_k's edges, and the stream s_c labeled with a relationship between expressions or sub-expressions of queries is in terms of Definitions 8, 9 and 10 like as shown in Figs. 3, 5 and 7.

V. Multi-Query Optimization Strategy

In this section, we first devise an IDAG cost estimation model to generate an optimized query execution plan on the basis of the triaxial hierarchy. After getting the optimized query execution plan, we need to record each expression or sub-expression's reuse condition and prepare for computing the rest of sub-expression for the query needed. Therefore, we propose three reuse schemes below. Then, by integrating the above two proposals, we present the MQOS to achieve nested pattern queries processing optimization.

A. Finding an Optimized Query Execution Plan

R_{E_i} represents the number of events of E_i per unit time. P_{E_i} represents the selectivity of all single-class predicates for the event type E_i. T_{W_p} specifies the time window in a given pattern query. We denote N_{E_i} as the number of events of E_i that are within time constraint T_{W_p} as $R_{E_i} \times T_{W_p} \times P_{E_i}$. Let $C^i_{O_h}$ and $C^o_{O_h}$ be input and output costs of operator O_h, respectively. We measure CPU cost of O_h, C_{O_h}, as follows:

$$C_{O_h} = C^i_{O_h} + C^o_{O_h}. \tag{1}$$

Let $\lambda(s_c)$ be the input rate of stream s_c and $L(s_c)$ be the network latency of stream s_c. We define the communication cost of stream s_c, C_{s_c}, as follows:

$$C_{s_c} = \lambda(s_c) \times L(s_c), \quad 1 \leq c \leq \ell. \tag{2}$$

$$\lambda(s_c) = \begin{cases} N_{E_i}, & \text{if } s_c \text{ connects data source of } E_i \text{ with an operator} \\ C^o_{O_h}, & \text{if } s_c \text{ connects two operators} \end{cases} \tag{3}$$

Table II summarizes the input and output cost formulas for each operator. $P^t_{E_i,E_j}$ represents the selectivity of the implicit time predicate between E_i and E_j, with time constraint $e_i.et < e_j.st$. P_{E_i,E_j} represents the selectivity of multi-class predicates that can be defined in advance [12]. For example, we can set $e_i.price < e_j.price$ and the selectivity of multi-class predicates as 1/2 in advance.

Definition 12: We define $P = \{P_1, \ldots, P_k, \ldots, P_m\}$ as a set of candidate query execution plans where P_k represents a kind of way to form query q_j. Based on Definition 11, we define G = $\{G_1, \ldots, G_k, \ldots, G_m\}$ as a set of IDAGs corresponding to P. G_k is with a set of operators $\{O_1, \ldots, O_h, \ldots, O_n\}$ and a set of streams $\{s_1, \ldots, s_c, \ldots, s_\ell\}$. Each operator has a total cost C_{O_h} as (1). Each stream has a communication cost as (2). An IDAG, G_k, has the associated operator computation and communication cost, denoted by $Total\ Cost(G_k)$:

$$Total\ Cost(G_k) = \Sigma^n_{h=1} C_{O_h} + \Sigma^\ell_{c=1} C_{s_c}. \tag{4}$$

Optimized Query Execution Plan-Finder Problem: Given a set of expressions or sub-expressions of queries $\{PT_{q_1}, \ldots, PT_{q_i}\}$ and a provisioned query q_j, we can find some relationships between PT_{q_i} and PT_{q_j} in terms of NQPH, NQCH and OTH alternatively. Those expressions or sub-expressions can be grouped along with their remaining sub-expressions as a set of candidate query execution plans P = $\{P_1, \ldots, P_k, \ldots, P_m\}$ to perform q_j. We aim to find a P_{opt} from P to perform q_j where the P_{opt} is with $\{PT_{q_1}, \ldots, PT_{q_t}, \ldots, PT_{q_r}\}$, and its IDAG is with minimum $Total\ Cost(\cdot)$, denoted by G_{opt}. G_{opt} is the execution way,

Table II
COST FORMULAS FOR OPERATORS

Operator	**Meaning**	**Input Cost** $C^i_{O_h}$	**Output Cost** $C^o_{O_h}$
$SEQ(E_i, E_j)$	combine primitive e_i of E_i with e_j of E_j that $e_i.et < e_j.st$ within TW_p	$N_{E_i} \times N_{E_j} \times P^t_{E_i,E_j}$	$N_{E_i} \times N_{E_j} \times P^t_{E_i,E_j} \times P_{E_i,E_j}$
$AND(E_i, E_j)$	combine primitive e_i of E_i with any e_j of E_j within TW_p	$N_{E_i} \times N_{E_j}$	$N_{E_i} \times N_{E_j} \times P_{E_i,E_j}$
$SEQ\_NEG(E_i, !E_k, E_j) \Longrightarrow NEG(SEQ(E_i, E_j), !E_k)$	remove the results from SEQ_NEG operator where e_k of E_k appears between E_i and E_j	$C^i_{O_{seq}} + C^o_{O_{seq}}$	$C^o_{O_{seq}} + C^o_{O_{seq}} \times (1- P^t_{E_i,E_k} \times P^t_{E_k,E_j}) \times P^t_{E_i,E_j}$
$AND\_NEG(E_i, !E_k, E_j) \Longrightarrow NEG(AND(E_i, E_j), !E_k)$	remove the results from AND_NEG operator where e_k of E_k appears between E_i and E_j	$C^i_{O_{and}} + C^o_{O_{and}}$	$C^o_{O_{and}} + C^o_{O_{and}} \times (1- P^t_{E_i,E_k} \times P^t_{E_k,E_j} - P^t_{E_j,E_k} \times P^t_{E_k,E_i}) \times P^t_{E_i,E_j}$

such that $\forall\, k, Total\ Cost(G_{opt}) \leq Total\ Cost(G_k)$ defined in Definition 12.

A candidate P_k must satisfy the following constraints:

Constraint 1: Each PT_{q_t} in P_k satisfies the relationship with provisioned processing query in terms of Definitions 8, 9 and 10: $\forall\ PT_{q_t} \in P_k$, $\exists\ q_j$ such that $PT_{q_j} \subset_P PT_{q_t}$ or $PT_{q_t} \subset_C PT_{q_j}$ or $PT_{q_j} \subset_O PT_{q_t}$;

Constraint 2: Those expressions or sub-expressions are non-identical: $\forall\ (PT_{q_t}, PT_{q_r} \in P_k)$, $\exists\ PT_{q_t} \neq PT_{q_r}$.

In order to avoid exponential complexity of search space, we design a novel iterative refinement methodology adopting a cost-based heuristic for finding a good quality solution within reasonable time rather than enumerating entire search space. When processing query q_j, the optimized query execution plan-finder performs as follows:

Phase 1: Ordering and clustering candidate plans.

This phase first sorts candidate plans in ascending order according to the number of operators corresponding to P_k, denoted as PL_{P_k}, then clusters the candidate query execution plan P_k with the same and smallest number of operators as a set of P^s.

Phase 2: Estimating candidate plans' costs by the IDAG cost model.

On the context of Phase 1, we can search a small group space rather than enumerating all possible expressions or sub-expressions of queries. Then, we can estimate the costs of those candidate plans in P^s by the IDAG cost model as (4). We assume there exists at least one candidate plan in P^s. The reason for choosing P_k with the smallest number of operators to process given queries is that the less number of operators is used to perform query q_j, the less total cost will be consumed, because using more operators for processing the same nested pattern queries will result in the increase of the total cost according to the IDAG cost model described above.

Phase 3: Searching an optimized query execution plan based on the IDAG cost model.

In general, we can find an optimized query execution plan P_{opt}, and the total cost of its IDAG such that $Total\ Cost(G_{opt}) \leq Total\ Cost(G_k)$.

Theorem 1: For a given set of queries, if all sub DAG plans G_k in G is optimal for their corresponding expressions or sub-expressions as well, then an IDAG G must be optimal.

Proof: We prove this by contradiction. Suppose the theorem is not true; then it should be possible to find an IDAG G' with lower cost than G, but with the same output cardinality. Using G' as a substitute for G, we would then obtain the sub DAG plans G'_k with lower total cost for their corresponding expressions or sub-expressions, which contradicts the assumption that G_k is optimal. ∎

Algorithm 1. Nested query pattern-based reuse scheme

Input: P_{opt}: a set of $\{PT_{q_1}, \ldots, PT_{q_t}, \ldots, PT_{q_r}\}$ where each PT_{q_t} in P_{opt} has relationship with q_j based on NQPH model;
$R_{PT_{q_t}}$: the results of PT_{q_t};
Output: $Num\ R_{PT_{q_t}}$: the replicas number of $R_{PT_{q_t}}$;
for each PT_{q_t} in P_{opt}
  $Num\ R_{PT_{q_t}} + +$;
end for
for PT_{q_t} = EPQ($E_h, \ldots, E_{hp}, \ldots, E_{hpf}$) $\in q_j$ and $\notin PT_{q_i}$
  preparing for computing PT_{q_t} by efficient stack-based join;
  //assuming joining events between PT_{q_r} and PT_{q_t} are sorted and pointers exist
  if EPQ($E_h, \ldots, E_{hp}, \ldots, E_{hpf}$) is followed by PT_{q_r}
    q_j = EPQ($E_h, \ldots, E_{hp}, \ldots, E_{hpf}, PT_{q_r}$);
  else PT_{q_r} is followed by EPQ($E_h, \ldots, E_{hp}, \ldots, E_{hpf}$)
    q_j = EPQ($PT_{q_r}, E_h, \ldots, E_{hp}, \ldots, E_{hpf}$);
end for

Algorithm 2. Nested query concept-based reuse scheme

Input: P_{opt}: a set of $\{PT_{q_1}, \ldots, PT_{q_t}, \ldots, PT_{q_r}\}$ where each PT_{q_t} in P_{opt} has relationship with q_j based on NQCH model;
$R_{PT_{q_t}}$: the results of PT_{q_t};
Output: $Num\ R_{PT_{q_t}}$: the replicas number of $R_{PT_{q_t}}$;
for each PT_{q_t} in P_{opt}
  $Num\ R_{PT_{q_t}} + +$;
end for
for PT_{q_t} = EPQ($E_{ap}, \ldots, E_{bp}, \ldots, E_{cp}$) $\in q_j$ and $\notin PT_{q_i}$
  preparing for computing PT_{q_t} by efficient stack-based join;
  q_j is formed by merging PT_{q_r} with EPQ($E_{ap}, \ldots, E_{bp}, \ldots, E_{cp}$)
end for

B. The Algorithms of Reuse Schemes

In this subsection, we design three reuse schemes to alternatively get the replicas number of common sub-expressions

Algorithm 3. Operator type-based reuse scheme

Input: P_{opt}: a set of $\{PT_{q_1}, \ldots, PT_{q_t}, \ldots, PT_{q_r}\}$ where each PT_{q_t} in P_{opt} has relationship with q_j based on OTH model; $R_{PT_{q_t}}$: the results of PT_{q_t};
Output: $Num\ R_{PT_{q_t}}$: the replicas number of $R_{PT_{q_t}}$;
1 for each PT_{q_t} in P_{opt}
2 $\quad Num\ R_{PT_{q_t}}++$;
3 end for

Algorithm 4. Multi-query optimization strategy

1 **//Phase 1: Ordering Queries**
2 ordering queries in terms of NQPH, NQPH and OTH, then getting $Q = \{q_1, \ldots, q_i, \ldots, q_\hbar\}$
3 **//Phase 2: Setting expressions or sub-expressions of queries for the optimized query execution plan-finder**
4 listing $\{PT_{q_1}, \ldots, PT_{q_i}\}$ which are the most finer, and coarser level of provisioned queries in trems of NQCH, NQPH and OTH, then computing each PT_{q_i} and recording its intermediate operators
5 **//Phase 3: Processing the provisioned queries**
6 for $j = 1, \ldots, \hbar$ //classify provisioned query
7 $\quad$ invoking the optimized query execution plan-finder to get $P_{opt} = \{PT_{q_1}, \ldots, PT_{q_t}, \ldots, PT_{q_r}\}$
8 $\quad$ if $\forall\ PT_{q_t} \in P_{opt}$, $\exists\ PT_{q_j} \subset_P PT_{q_t}$ //satisfy NQPH
9 $\quad\quad$ invoking Algorithm 1;
10 $\quad$ if $\forall\ PT_{q_t} \in P_{opt}$, $\exists\ PT_{q_t} \subset_C PT_{q_j}$ //satisfy NQCH
11 $\quad\quad$ invoking Algorithm 2;
12 $\quad$ if $\forall\ PT_{q_t} \in P_{opt}$, $\exists\ PT_{q_j} \subset_O PT_{q_t}$ //satisfy OTH
13 $\quad\quad$ invoking Algorithm 3;
14 $\quad$ else
15 $\quad\quad$ for each PT_{q_t} in P_{opt}
16 $\quad\quad\quad Num\ R_{PT_{q_t}}++$;
17 $\quad\quad$ end for
18 $\quad\quad$ for $PT_{q_j} \subset_P PT_{q_t}$, and PT_{q_t} = EPQ($E_h, \ldots, E_{hp}, \ldots, E_{hpf}$) $\in q_j$ and $\notin PT_{q_i}$ //satisfy NQPH
19 $\quad\quad\quad$ preparing for computing PT_{q_t} by efficient stack-based join;
20 $\quad\quad\quad\quad$ if PT_{q_t} is followed by PT_{q_r}
21 $\quad\quad\quad\quad\quad PT_{q_j}$ = EPQ($E_h, \ldots, E_{hp}, \ldots, E_{hpf}, PT_{q_r}$);
22 $\quad\quad\quad\quad$ else PT_{q_r} is followed by PT_{q_t}
23 $\quad\quad\quad\quad\quad PT_{q_j}$ = EPQ($PT_{q_r}, E_h, \ldots, E_{hp}, \ldots, E_{hpf}$);
24 $\quad\quad$ end for
25 $\quad\quad$ for $PT_{q_t} \subset_C PT_{q_j}$, and PT_{q_t} = EPQ($E_{ap}, \ldots, E_{bp}, \ldots, E_{cp}$) $\in q_j$ and $\notin PT_{q_i}$ //satisfy NQCH
26 $\quad\quad\quad$ preparing for computing PT_{q_t} by efficient stack-based join;
27 $\quad\quad\quad PT_{q_j}$ is formed by merging PT_{q_r} with EPQ($E_{ap}, \ldots, E_{bp}, \ldots, E_{cp}$);
28 $\quad\quad$ end for
29 $\quad$ updating $\{PT_{q_1}, \ldots, PT_{q_i}\}$ by adding new ones into it
30 end for
31 storing $Num\ R_{PT_{q_t}}$ in $optArray$;
32 parallel processing multiple queries by $Num\ R_{PT_{q_t}}$'s in $optArray$

and prepare for computing the rest of sub-expression for the query needed. They are nested query pattern-based reuse scheme (NQPRS), nested query concept-based reuse scheme (NQCRS), and operator type-based reuse scheme (OTRS). The pseudocodes of them are shown in Algorithms 1, 2 and 3, respectively.

In Algorithm 1, given an optimized query execution plan P_{opt}, we first update the value of $Num\ R_{PT_{q_t}}$ for each PT_{q_t} in P_{opt} (lines 1-3), then prepare for computing sub-expression PT_{q_t} = EPQ($E_h, \ldots, E_{hp}, \ldots, E_{hpf}$) that belongs to q_j but is not captured by PT_{q_i}. Based on different conditions (lines 7 and 9), q_j can be formed by using EPQ($\cdot$) to join PT_{q_r} with PT_{q_t} as EPQ($E_h, \ldots, E_{hp}, \ldots, E_{hpf}$, PT_{q_r}) or EPQ(PT_{q_r}, $E_h, \ldots, E_{hp}, \ldots, E_{hpf}$) (lines 4-11). In Algorithm 2, we first update the value of $Num\ R_{PT_{q_t}}$ for each PT_{q_t} in P_{opt} (lines 1-3), then prepare for computing sub-expression PT_{q_t} = EPQ($E_{ap}, \ldots, E_{bp}, \ldots, E_{cp}$) that belongs to q_j but is not captured by PT_{q_i} (lines 4-7). q_j can be formed by merging PT_{q_r} with PT_{q_t}. In Algorithm 3, because we try to reuse coarser level query's replicate results to generate finer level query by filtering some data that are not captured by the finer level query, there is no rest sub-expression that needs to be prepared. Hence, we just need to update the value of $Num\ R_{PT_{q_t}}$ for each PT_{q_t} in P_{opt} (lines 1-3).

C. The MQOS Algorithm

Through integrating the optimized query execution plan-find approach with the three reuse schemes described in the above two subsections, respectively, we present here a multi-query optimization strategy (MQOS). MQOS is to produce query results quickly and improve computational efficiency by exploiting not only common sub-expressions among nested event pattern queries, but also replicas of the appropriate common operators' results for the queries needed. The steps of MQOS Algorithm are as follows (Algorithm 4):

Phase 1: Ordering queries.

This phase first sorts queries in ascending order based on the number of their positive event types in terms of NQPH. Next, this phase orders the queries with the same number of positive event types from finer level to coarser level in terms of NQCH. Then, this phase orders the queries with the same positive event types from coarser level to finer level in terms of OTH. The reason is that a query's results from coarser level in NQPH and OTH can be reused for a query from finer level, while a query's results with finer level event types in NQCH can be reused for a query with coarser level event types. After that, we get an ordered set of queries Q = $\{q_1, \ldots, q_i, \ldots, q_\hbar\}$ (lines 1-2).

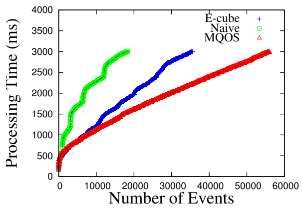
Phase 2: Setting expressions or sub-expressions of queries for the optimized query execution plan-finder.

This phase lists the expressions or sub-expressions of queries $\{PT_{q_1}, \ldots, PT_{q_i}\}$ which are the most finer, and coarser level of provisioned queries in terms of NQCH, NQPH and OTH alternatively. Then, this phase computes those expressions or sub-expressions of queries by efficient stack-based join [13] and records intermediate operators that belong to the set of operators corresponding to $\{PT_{q_1}, \ldots, PT_{q_i}\}$. The purpose is to provide a set of expressions or sub-expressions of queries for next phase to find an optimized query execution plan that can be used to perform the provisioned query q_j (lines 3-4).

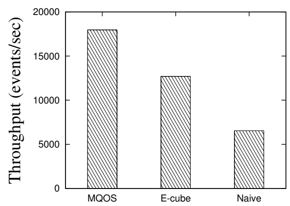
Phase 3: Processing the provisioned queries.

Table III
PARAMETERS FOR EVENT GENERATION

Parameter	Values
R_{E_i}	10^2-10^5 tuples/sec
TW_p	10 -10^{10} sec
$P^t_{E_i,E_j}$	1/2
P_{E_i,E_j}	1



(a) Processing Time



(b) Throughput

Figure 8. Workload 1 with pure SEQ pattern queries under NQPH and NQCH variations.

This phase goes through the ordered list of provisioned queries and iteratively processes them to an optimized query execution plan for given queries. We handle the ordered queries in Q iteratively. For $j = 1$, based on the IDAG cost estimation model (Subsection V-A), this phase tries to find an optimized query execution plan P_{opt} for generating q_j in terms of $\{PT_{q_1}, \ldots, PT_{q_i}\}$ (line 7). Henceforth, we can find an optimized plan P_{opt} = $\{PT_{q_1}, \ldots, PT_{q_t}, \ldots, PT_{q_r}\}$ with the minimum cost of operators and communications. If each PT_{q_t} in P_{opt} satisfies the NQPH, NQCH or OTH with query q_j, the system will invoke Algorithm 1, 2 or 3, respectively, to get $Num\ R_{PT_{q_t}}$ and prepare for computing the sub-expression that belongs to q_j but is not captured by PT_{q_i} (lines 8-9, 10-11, or 12-13, respectively). If the subsets of P_{opt} do not meet the above conditions, the system updates $Num\ R_{PT_{q_t}}$ for each PT_{q_t} in P_{opt} (lines 14-16) and prepares for computing the sub-expression that belongs to q_j but is not captured by PT_{q_i} based on NQPH or NQCH models (lines 18-28). Afterwards, the system updates expressions or sub-expressions of queries $\{PT_{q_1}, \ldots, PT_{q_i}\}$ by adding new ones into it (line 29). Finally, $Num\ R_{PT_{q_t}}$ is stored in $optArray$ (line 31) and the system processes the given nested pattern queries over multi-dimensional event streams by the final optimized query execution plan, namely, $Num\ R_{PT_{q_t}}$'s in $optArray$ (line 32).

VI. EXPERIMENTAL EVALUATION

By comparing MQOS with related approaches under different workload conditions alternately, we demonstrate the superiority and performance of our proposal. We implemented all approaches within StreamBase system [9]. We ran the experiments on an Intel®Core™duo CPU 3.33 GHz and 12.00 GB main memory. In each stream, we set stock ticker, timestamp and price information attributes. We define the processing time as the difference between the system time of the number of events output and the system time to process initial event contributing to the output. Table III summarizes the main parameters for event generation. We set the parameters and representative query workloads 1-3 shown in Fig. 1 based on [18], [12], [15].

A. Comparing over Pure SEQ Pattern Queries with NQPH and NQCH Variations

In this experiment, we evaluated query workload 1 with query pattern and query concept variations. Because E-cube is currently the most efficient method to treat with pure SEQ pattern queries under NQPH and NQCH variations, we attempt to compare MQOS with E-cube and Naive methods meanwhile. E-cube [18] is a kind of reuse-based scheme, that is, sharing query' results among given pure SEQ pattern queries. Naive [13] processes queries independently using stack-based query evaluation. Here, we set time window size of each operator as 10 sec and each input rate as 1000 tuples/sec.

Fig. 8 shows the processing time and throughput of the three methods, respectively. We observe that MQOS and E-cube generate results faster than Naive, because they avoid results re-computation by applying conditional computation. We also notice the speedup of MQOS over E-cube. The reason is that MQOS not only avoids query result re-computation, but also processes given queries by reusing common operators' results. In addition, because of cardinality of operator's output within TW_p, the relationship between the number of events and processing time is non-linear and, exactly, quadratic when processing time is less than time window size.

Specifically, q_2 can be performed by processing replicate results of q_1. The sub-expression of q_3's results can be replicated and reused for performing q_4 by processing replicate results from SEQ(Medicine, DELL) operator. The results of sub-expression of q_1 and q_4 can be reused for query q_5 by processing replicate results from SEQ(HTB, EDU) and SEQ(Medicine, Computer) operators, respectively. However, when performing queries q_4 and q_5, E-cube introduced extra delay due to removing partial event types from given queries and eliminating duplicated tuples, in order to satisfy the requirements of processing next queries. Consequently, MQOS used 9 operators, i.e., 7 SEQ, 1 SEQ_NEG and 1 UNION operators, E-cube used 15 operators, i.e., 7 SEQ, 1 SEQ_NEG, 3 MAP (removing event type KME from q_3, MBA from q_1 and Metal from q_4), 3 FILTER (filtering the duplicate tuples caused by MAP operator) and 1 UNION operators, while Naive used 12 operators, i.e., 11 SEQ and 1 SEQ_NEG operators for processing the given queries.

B. Varying Time Window Sizes of Operators

In this experiment, we examined the effect of time window sizes of operators for MQOS, E-cube and Naive on

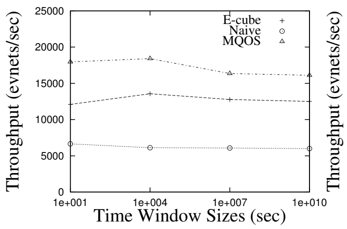


Figure 9. Varying time window sizes.

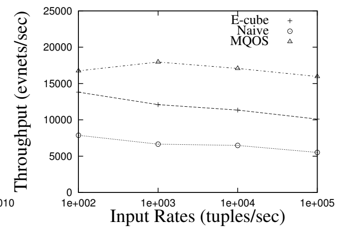


Figure 10. Varying input rates.

throughput based on Workload 1. We fixed each input rate as 1000 tuples/sec, and varied time window size from 10 up to 10^{10} sec.

The results are shown in Fig. 9. The throughput of MQOS is higher than the others no matter the time window size. As we set time window sizes from 10 to 10^4 sec, MQOS and E-cube can treat with queries more efficiently than Naive. However, when increasing time window sizes from 10^4 to 10^{10} sec, the runtime stack grows large, causing memory overhead so that the throughput of MQOS and E-cube decrease. Whereas, Naive cannot handle with the input rate variation as it increases from 10 up to 10^{10} tuples/sec. Consequently, we concluded that MQOS outperforms E-cube and Naive even under time window sizes variation.

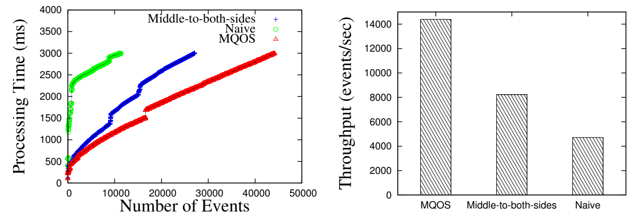
C. Varying Input Rates of Streams

In this experiment, we examined the effect of input rate of streams for MQOS, E-cube and Naive on throughput based on Workload 1. We fixed time window size of operators as 10 sec, and varied input rate of streams from 10^2 up to 10^5 tuples/sec.

The results are shown in Fig. 10, and the throughput of MQOS is more higher than the others as the variation of input rates. We can notice that as we set input rate of streams from 10^2 up to 10^3 tuples/sec, MQOS can treat with queries more efficiently than E-cube and Naive. As the input rate of streams vary from 10^3 up to 10^5 tuples/sec, the throughput of MQOS is decreasing. The reason is that when the input rate of streams are increased too much, the runtime stack grows large resulting in memory overhead so that the throughput of MQOS decreases. On the other hand, E-cube and Naive cannot handle with the input rate variation as it increases from 10^2 up to 10^5 tuples/sec. Consequently, we concluded that MQOS outperforms E-cube and Naive even under input rates variation.

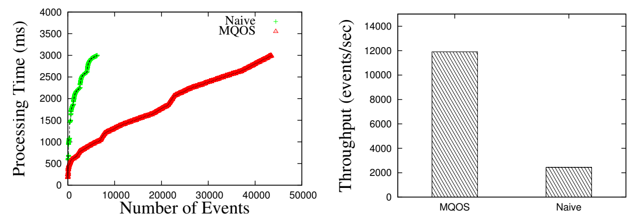
D. Comparing over Nested Pattern Queries with OTH Variation

In this experiment, we evaluated query workload 2 with only operator type variation. Since there are no related approaches that makes study of relationship among different operator types, we compare MQOS with Middle-to-both-sides which comes from part of our work, and Naive mean-



(a) Processing Time (b) Throughput

Figure 11. Workload 2 with nested pattern queries under OTH variation.



(a) Processing Time (b) Throughput

Figure 12. Workload 3 with nested pattern queries under triaxial mixed variation.

while. Based on OTH model, Middle-to-both-sides means $SEQ(\cdot)$ query will be processed first, then, its results are reused for $AND(SEQ(\cdot))$ and $NEG(SEQ(\cdot))$ queries, as shown in Fig. 6. Here, we set time window size of each operator as 10 sec and each input rate as 1000 tuples/sec.

Fig. 11 shows the processing time and throughput of the three methods, respectively. We observe that MQOS and Middle-to-both-sides generate results faster than Naive, because they avoid result re-computation. We also observe that MQOS produces results faster than Middle-to-both-sides. This is because MQOS uses Top-to-down method, namely, $AND(SEQ(\cdot))$ query will be processed first, then, its results can be reused for $SEQ(\cdot)$ query, whose results in turn can be reused for $NEG(SEQ(\cdot))$ query. Specifically, AND operator of q_6 does not care about the sequence of event types so that it can output more results than SEQ operator of q_7. Then, the results of q_6 can be replicated and reused for q_7. After that, we can perform q_8 by processing replicate results from q_7. On the contrary, because SEQ operator is related to the sequence of event types, it has implicit time predicate described in Subsection V-A. Consequently, Middle-to-both-sides works worse than MQOS.

E. Comparing over Nested Pattern Queries with Triaxial Mixed Variation

In this experiment, we evaluated a query workload 3 involving nested query pattern, nested query concept and operator type variations. Due to the complex of workload 3, there are no existing methods to efficiently treat with it under nested query pattern, nested query concept and operator type variations. We therefore attempt to compare MQOS with Naive method. Here, we set time window size of each operator as 10 sec and each input rate as 1000 tuples/sec.

Fig. 12 shows the processing time and throughput of the two methods. As expected, MQOS outperforms Naive. On closer analysis in MQOS in terms of workload 3, because the results of q_9 cannot be reused for query q_{13}, q_9 and q_{13} should be executed in parallel first. Next, the results of q_9 can be replicated and reused for performing q_{10}. Then, the results of q_{10} can be replicated and reused for q_{11} by adding predicate "!" and q_{12} can be performed by joining event type INTC with replicate results from q_{10} by $AND(\cdot)$ operator. The results of q_{14} can be generated by joining the sub-expression of q_{13}'s results, i.e., SEQ(Finance, Metal) with replicate results from q_9 by $SEQ(\cdot)$ operator. On the other hand, the replicate results of q_{13} can be reused for q_{15} and q_{16} in parallel.

VII. Conclusions

MQOS integrates OLAP with CEP functionalities for realizing (i) technologies that allow users to efficiently query large amounts of event stream data in multi-dimensions where each of which may be at different levels of abstraction, and (ii) technologies that allow CEP systems to process nested pattern queries by leveraging appropriate replicas of common operators' results. The experimental results showed that MQOS has faster processing time and higher throughput than E-cube, Naive and Middle-to-both-sides methods. In addition, the performance of MQOS keeps better than that of E-cube when varying time window sizes of operators or input rates of streams. Interesting future work includes considering optimizing nested pattern queries over live and archived multi-dimensional event streams.

References

[1] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: a new model and architecture for data stream management," *The VLDB Journal*, vol. 12, no. 2, pp. 120–139, Aug. 2003.

[2] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, "The design of the Borealis stream processing engine," in *CIDR '05*, 2005, pp. 277–289.

[3] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom, "Stream: The Stanford data stream management system," Stanford InfoLab, Technical Report 2004-20, 2004.

[4] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah, "TelegraphCQ: continuous dataflow processing," in *SIGMOD '03*, 2003, pp. 668–668.

[5] D. Gyllstrom, E. Wu, H.-J. Chae, Y. Diao, P. Stahlberg, and G. Anderson, "SASE: Complex event processing over streams," in *CIDR '07*, 2007, pp. 407–411.

[6] http://www.cs.cornell.edu/bigreddata/cayuga/.

[7] http://dbs.mathematik.uni-marburg.de/Home/Research/Projects/PIPES.

[8] http://www.coral8.com.

[9] http://www.streambase.com/.

[10] http://blogs.oracle.com/cep/.

[11] A. Margara and G. Cugola, "Processing flows of information: from data stream to complex event processing," in *DEBS '11*, 2011, pp. 359–360.

[12] Y. Mei and S. Madden, "Zstream: a cost-based query processor for adaptively detecting composite events," in *SIGMOD '09*, 2009, pp. 193–206.

[13] E. Wu, Y. Diao, and S. Rizvi, "High-performance complex event processing over streams," in *SIGMOD '06*, 2006, pp. 407–418.

[14] Y. Yan, J. Zhang, and M.-C. Shan, "Scheduling for fast response multi-pattern matching over streaming events," in *ICDE '10*, 2010, pp. 89–100.

[15] M. Liu, E. Rundensteiner, D. Dougherty, C. Gupta, S. Wang, I. Ari, and A. Mehta, "High-performance nested CEP query processing over event streams," in *ICDE '11*, 2011, pp. 123–134.

[16] J.-H. Hwang, U. Çetintemel, and S. Zdonik, "Fast and highly-available stream processing over wide area networks," in *ICDE '08*, 2008, pp. 804–813.

[17] F. Xiao, T. Kitasuka, and M. Aritsugi, "Economical and fault-tolerant load balancing in distributed stream processing systems," *IEICE Trans. Information and Systems*, vol. E95-D, no. 4, pp. 1062–1073, April 2012.

[18] M. Liu, E. Rundensteiner, K. Greenfield, C. Gupta, S. Wang, I. Ari, and A. Mehta, "E-cube: multi-dimensional event sequence analysis using hierarchical pattern query sharing," in *SIGMOD '11*, 2011, pp. 889–900.

[19] S. Chaudhuri and U. Dayal, "An overview of data warehousing and OLAP technology," *SIGMOD Record*, vol. 26, no. 1, pp. 65–74, 1997.

[20] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe, "Efficient and extensible algorithms for multi query optimization," in *SIGMOD'00*, 2000, pp. 249–260.

[21] A. Cuzzocrea, "CAMS: OLAPing multidimensional data streams efficiently," in *DaWaK '09*, 2009, pp. 48–62.

[22] A. Cuzzocrea and S. Chakravarthy, "Event-based lossy compression for effective and efficient OLAP over data streams," *Data Knowl. Eng.*, vol. 69, no. 7, pp. 678–708, Jul. 2010.