Realtime Healthcare Services Via Nested Complex Event Processing Technology*

Mo Liu, Medhabi Ray, Dazhi Zhang, Elke A. Rundensteiner, Daniel J. Dougherty
Worcester Polytechnic Institute, Worcester, MA 01609, USA
($liumo|medhabi|jasonzhang|rundenst|dd$)@cs.wpi.edu
Chetan Gupta, Song Wang, Ismail Ari‡
USA Hewlett-Packard Labs, USA
‡Ozyegin University, Turkey
($chetan.gupta|songw$)@hp.com ‡$Ismail.Ari$@ozyegin.edu.tr

ABSTRACT

Complex Event Processing (CEP) over event streams has become increasingly important for real-time applications ranging from healthcare to supply chain management. In such applications, arbitrarily complex sequence patterns as well as non existence of such complex situations must be detected in real time. To assure real-time responsiveness for detection of such complex pattern over high volume high-speed streams, efficient processing techniques must be designed. Unfortunately the efficient processing of complex sequence queries with negations remains a largely open problem to date. To tackle this shortcoming, we designed optimized strategies for handling nested CEP query. In this demonstration, we propose to showcase these techniques for processing and optimizing nested pattern queries on streams. In particular our demonstration showcases a platform for specifying complex nested queries, and selecting one of the alternative optimized techniques including sub-expression sharing and intermediate result caching to process them. We demonstrate the efficiency of our optimized strategies by graphically comparing the execution time of the optimized solution against that of the default processing strategy of nested CEP queries. We also demonstrate the usage of the proposed technology in several healthcare services.

1. BACKGROUND AND MOTIVATION

Complex Event Processing (CEP) has become increasingly important in modern applications, ranging from supply chain management with RFID tracking to real-time intrusion detection [1, 2, 3]. CEP must be able to support sophisticated pattern matching on real-time event streams including the arbitrary nesting of sequence (SEQ), AND, OR operators and the flexible use of negation (!) in such nested patterns. In our work we focus in particular on delivering healthcare services. For example, consider the scenario of a healthcare hygiene control system which is equipped with sensors and hygiene control is monitored and regulated throughout the hospital facilities by running pattern queries on continuous sensor data. The services range from reporting contaminated medical equipments in a hospital [4] to tracking hygiene violations made by healthcare workers [5]. In a real scenario running such a system, every healthcare worker wears an RFID badge. Surgical and non-surgical equipment are tagged. Sensors are located in every patient's room, Intensive Care Units, Emergency Rooms, Operation Theaters and near Sanitizing equipments. All these sensors continuously sense the environment and send collected events to a centralized system. Not only can the total number of events occurring in a second be very large, but the complexity of the patterns can also be very high and very urgent actions might be necessary in certain cases. Thus running pattern queries over such high input rate systems in real time is a challenge and we need an efficient underlying system to do so. Let us assume that the tools for medical operations are RFID-tagged. The system monitors the histories of the equipment (such as, records of surgical usage, washing, sharpening and disinfection). When a healthcare worker puts a box of surgical tools into a surgical table equipped with RFID readers, the computer would display warnings such as "The tool with id = "5" must be disposed". Query Q_1 (Figure 1) expresses this critical condition that after being recycled and washed, a surgery tool is being put back into use without first being sharpened, disinfected and then checked for quality assurance. Query Q_2 (Figure 2) reports hygiene violations caused by a healthcare worker who did not perform necessary sanitizing actions at the right time in proper sequences based on severity of threat to spread infection. This is a query written in *NEEL* language which was our prior work [6].

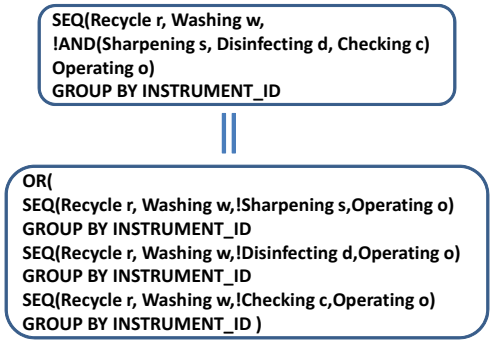

Figure 1: Example Query Q_1

One important feature of any query language, as learned from the community's experience with SQL, is the flexible nesting of query expressions. Nested CEP queries provide users with an intuitive way of expressing their requirements. Without this capability, users are severely restricted in forming complex patterns in a convenient and succinct manner [7]. In fact some nested queries cannot be

*This work is supported by HP Labs Innovation Program and NSF grants 1018443 and 0917017.

EDBT 2012, March 26–30, 2012, Berlin, Germany.

```
SEQ(Enter_Patient_Room en, Touch_Patient tp,
Exit_Patient_Room ex,
!OR(Sanitize s, Patient_Critical pc)
Enter_Patient_Room en1)
GROUP BY EMPLOYEE_ID
WITHIN 5 MINS
```

Figure 2: Example Query Q_2

expressed as flat queries or might result in an exponential number of flat queries. However, the state-of-art CEP systems including SASE [1], ZStream [3] and Cayuga [2] do not support nested CEP queries with negation.

Traditionally, an iterative execution strategy is adopted for processing nested queries [8]. Namely, first all component events matching the outer query would be identified. Thereafter, for each outer match such as SEQ(Recycle, Washing, Operating) in Q_1, the results for the nested inner subsequences would be iteratively computed, in this case, AND(Sharpening, Disinfection, Checking). Finally, each outer candidate sequence result would be constrained by the non-existence of the inner subsequence match between event pairs of Washing and Operating readings. As our prior experiments in [9] confirm, this classic process of first constructing the outer sequences and then iteratively the inner sequences can be prohibitively inefficient, missing critical opportunities for optimization as illustrated below.

Problem 1: In traditional iterative processing of nested CEP, a top-down ordering is forced upon the execution of the CEP subexpressions thus missing out on huge gains potentially achievable by decorrelating them and processing them simultaneously.

Problem 2: In traditional CEP processing, candidate sequence results generated may later simply be discarded – thus wasting precious resources. For example, in the above query Q_1, the generation of the sequence results for the outer subexpression SEQ(Recycle r, Washing w, Operating o) would be wasted, since during medical procedures inner sequences of type AND(Sharpening s, Disinfection d, Checking c) may not exist. This unnecessary outer event generation to be later discarded wastes precious memory and CPU processing resources.

Problem 3: On the other hand while the traditional processing strategy [8] produces full results satisfying the nested *negated* subexpression are processed completely, the full computation of negated sub-expression might not be needed because the existence of only one result should be able to filter an outer sub-expressions results.

Problem 4: Iterative execution strategy, with the sub-expression execution being triggered by an outer sub-expression result often leads to re-computations of the same result, thus causing an undesirable wastage of CPU resources.

To address these problems we set out to develop technology for the specification and execution of a wide range of nested CEP queries with negation appearing at any level of nesting. We will demonstrate a system which flattens nested CEP queries and executes these rewritten queries in an efficient manner [9]. In addition we also propose an effective caching technique to avoid re-computation of already computed pattern results. Caching is in particular an effective technologies for complex queries, for which no rewriting is possible. This nested CEP technology work is orthogonal to our parallel efforts of providing OLAP capabilities on streaming multi-dimensional data [10] and of adding "active rules and concurrency" into CEP engines [5].

In particular, we will demonstrate the following:

- Allow audience to enter queries in the nested CEP language *NEEL*[1], designed for specifying nested CEP queries.
- Demonstrate the caching based optimized processing of Nested CEP queries.
- Illustrate the step-wise application of rewriting technology to unnest *NEEL* queries whenever possible, resulting in a new rewritten query plan.
- Audience can then choose between bit-encoded shared execution processing strategy or continuous caching techniques for the optimized processing of *NEEL* query plan.
- Showcase a case-study in healthcare for infection prevention using our *NEEL* processing technology.

2. THE NESTED CEP TECHNOLOGY

2.1 E-Analytics System Architecture

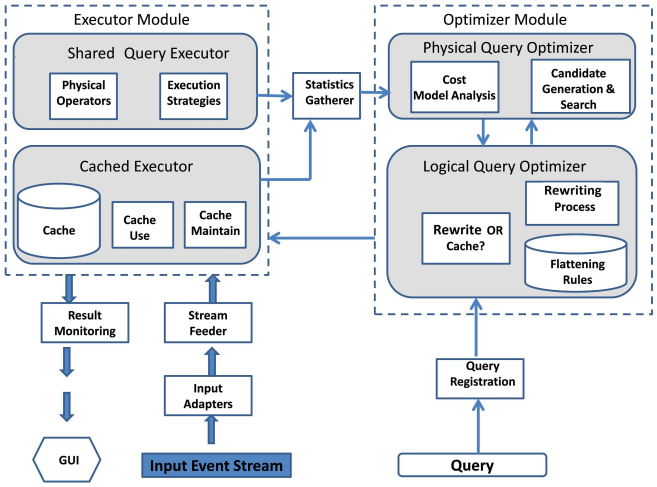

Figure 3: E-Analytics Event Processing System Architecture

The architecture of our *NEEL* query processing system is depicted in Figure 3. It can be seen broadly as comprised of two units namely, the Executor and the Optimizer. The Optimizer consists of the Logical and the Physical Optimizer. The query is first registered and goes into Logical Query Optimizer which applies the Rewriting Rules to a given nested CEP query to reduce the nesting of the NEEL query plan. The Physical Optimizer makes a cost based decision for grouping multiple rewritten sub-queries obtained from the Logical Optimizer based on input statistics. Lastly the Executor which is driven by the Optimizer has two components. The Query Executor uses a novel bit-marking technique [9] for shared expression execution and continuous caching for leveraging previously computed sub-expression.

2.2 *NEEL* Sub-expression Caching Strategies

Many *NEEL* queries cannot be completely flattened even after application of the rewriting rules. Thus for such queries we have to execute them iteratively. However as mentioned in Problem 4 of Section 1, the iterative execution of Nested CEP queries results in the re-computation of the results for inner sub-queries every time an outer triggering event arrives. Thus to avoid re-computations of

[1]NEEL stands for **Ne**sted Complex **E**vent Query **L**anguage.

intermediate results, we cache them. The cache is continuous and needs to be updated and maintained as the window moves on. For this we selectively associate a cache with each nested query subexpression. Each cache is attached with semantic descriptors which act as indicators to foretell what content have already been loaded in the cache. Given an outer query result triggered by an event e_n, we calculate the constraint window for each sub-query given by the timestamps of the events in the outer query bounding the sub query. For a given constrained window if a matching semantic descriptor is found in the cache, the results are directly retrieved from the cache. If however an exact match is not found, the cache is updated. We design a novel technique to avoid storing duplicate results for overlapping windows. The newly added content is now updated into the cache and attached with the appropriate Semantic Descriptor. Several enhancements of this basic caching will be demonstrated including maintaining flags for negated sub-queries and partitioning the cache for handling predicates.

2.3 *NEEL* Logical Query Optimization

This module handles the problem of reducing the nesting levels of a nested *NEEL* query when it is possible to do so. We have a set of rules to unnest nested *NEEL* queries with Sequence and Negation operators to produce equivalent queries without nesting under certain preconditions. We also have a procedure of applying these rules to a given *NEEL* query. Our proposed rewriting rules fall into three categories: flattening rules, distributive rules and negation push down rules. We briefly discuss this technique [9] based on the example shown below in figure 4.

We will now walk through a complete example for unnesting a *NEEL* query. Given a query Q_3 as shown in Figure 4(a) we first apply the Negation Push Down Rule resulting in the intermediate rewritten query shown in Figure 4(b). We then apply the Distributive Rule on Figure 4(b) resulting in the rewritten query shown in Figure 4(c). This corresponds to the final flat query in this case.

2.4 *NEEL* Physical Query Optimization

A rewritten *NEEL* query may consist of multiple expressions which may share common sub-expressions. One of the challenges for efficient computation is to avoid recomputation of common sub-expressions. This module handles the critical task of deciding shared sub-expressions processing. The decision is based on a cost model described in detail in [9] which uses statistics supplied by the Executor engine. This problem of optimally grouping sub-expressions has an exponential search space. We provide strategies for traversing the search space to find high-quality solutions efficiently. Q_2 in its rewritten form shown in Figure 4(c) consists of three conjuncts that share the same positive patterns SEQ(Recycle, Washing, Operating), except with different negative interleaved events. The first two share the common prefix SEQ(Recycle, Washing, ! Sharpening) while the last two share the common prefix SEQ(Recycle, Washing, ! Sharpening, Sharpening, ! Disinfection). Our optimizer finds a logical grouping of the sub-expressions with the minimum overall execution cost.

2.5 *NEEL* Shared Query Executor

We introduce a shared expression physical operators assisted by a dynamic bit-marking scheme that achieves early termination of evaluating negated sub-expressions to tackle problem 3 listed in Section 1. Unlike existing systems [1, 2, 3], we share event expressions when subpatterns contain the same positive event types while their negative event types may differ in their types as well as their position within the respective sequence. We observe that event expressions with common positive event types return the same results yet apply different filters about the required non-existence of certain events. The main idea of our strategy is to record the constraints of non-occurrence for each expression at compile time using an efficient bit-encoding based methodology. At run time, as we construct each sequence result, we keep track of which of the given constraints are satisfied. We stop the evaluation early for unsatisfied event expressions.

3. DEMONSTRATION

Our demonstration will introduce the audience to two distinct views, namely an internal and an external view. The first demonstration showcases the Nested CEP core technology focussing in particular on the alternate optimization and processing innovations we have developed. The second demonstration will showcase a real life scenario using our Nested CEP processing engine - The E-Analytics Engine.

3.1 The E-Analytic Engine

Entering a query and selecting Processing Technique. Figure 5 shows the console where the user can enter *NEEL* queries and submit them to generate a Query Plan. Following this a user may choose from among the alternate methods of executing the query. He can choose to execute the query without rewriting it, i.e using the caching based iterative technique or choose to rewrite it first and then optionally apply the shared sub-expressions based optimization to execute the rewritten query. The system can also use the two techniques in conjunction for partially rewritten queries.

Rewriting and Viewing Query Plan. The Rewriter traverses the parse tree and applies the rewriting rules based on our rewriting procedure until no more rule can be applied. The audience can visualize all intermediate stages of rewriting. Figure 6 shows the

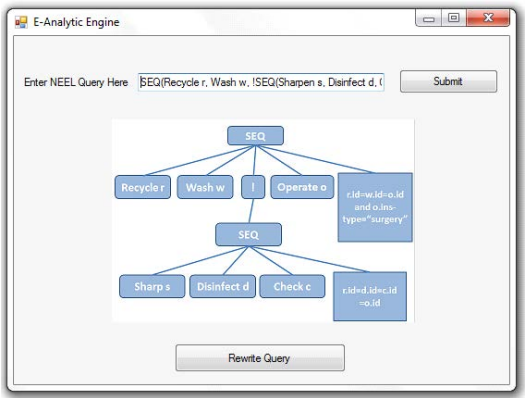

Figure 5: E-Analytic Query Entry Console

order of rule application for query Q_2. The audience can see for instance what rules are applied in what order. Here first the Negation Push Down rule has been applied followed by the Distributive Rule. Figure 7 shows the rewritten query plan.

Visualizing Performance Comparisons. The audience can choose to see how the different optimization techniques performed compared to each other and also against the default iterative processing technique. Once a query has been run using each of the strategies, a chart comparing the execution time for each of the techniques is generated and can be viewed.

3.2 The Healthcare Hygiene Monitoring System

```
PATTERN SEQ(Recycle r, Wash w,
            ! SEQ(Sharpen s, Disinfect d, Check c)
            Operate o)
WITHIN 1 hour
```

(a) Nested Query Q3

Applying Negation Push Down Rule:
SEQ(Recycle r, Wash w, !Sharpen s V
∃SEQ(!Sharpen s1, Sharpen s2, !Disinfect d) V
∃SEQ(!Sharpen s3, Sharpen s4, !Disinfect d1,
Disinfect d2, !Check c), Operate o)

(b) Applying Negation Push Down Rule to 4(a)

Applying Distributive Rule:
SEQ(Recycle r, Wash w, !Sharpen s, Operate o) OR
SEQ(Recycle r1, Wash w1,∃SEQ(!Sharpen s1, Sharpen s2, !Disinfect d), Operate o1) OR
SEQ(Recycle r2, Wash w2, ∃SEQ(!Sharpen s3, Sharpen s4, !Disinfect d1, Disinfect d2, !Check c), Operate o2))

(c) Applying Distributive Rule to 4(b)

Figure 4: Rewriting a Nested Query into its Rewritten forms

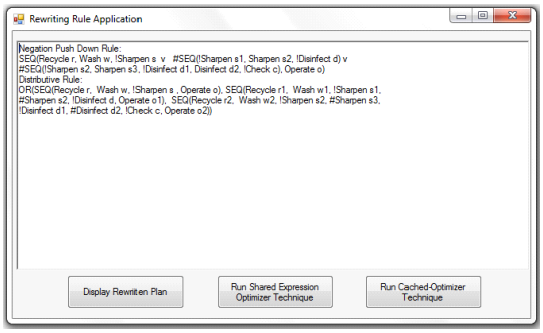

Figure 6: E-Analytic Rule Application Console

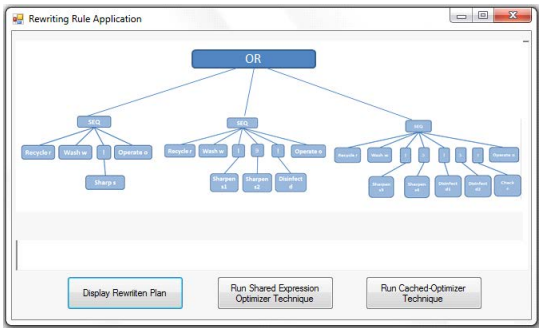

Figure 7: E-Analytic Console Showing Rewritten Query Plan

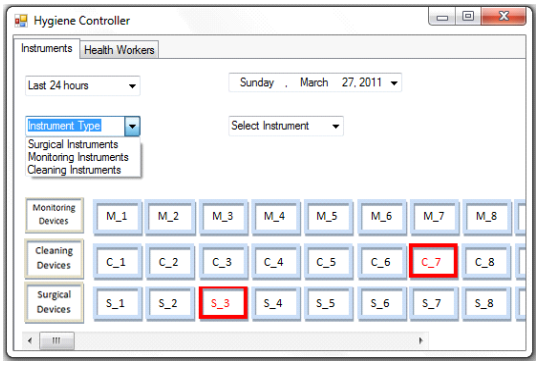

Figure 8: Equipment Status Monitoring Application

Demonstration Scenario: We will also demonstrate our technology in the context of several health care applications. Tools that do not follow a set pattern of cleansing processes should be considered unfit for reuse. This is a real challenge in hospitals where infections are often spread by reuse of unsterilized instruments.

Our system issues warnings and shows hygiene status of all the instruments and doctors in the inventory. Different colors indicate the status of the instruments. The system administrator can select from a dropdown any instrument and observe its history for some specified time. A snapshot of the application tracking instruments can be seen in Figure 8. The system also gives warnings to health care workers via portable devices such as PDAs or cell phones about particular instrument. Thus if the status of a scalpel turned dangerous the clean up crew should be issued a warning saying: "Warning: Remove instrument $scalpel_{023}$"

4. CONCLUSION

In this paper we showcase our work on Nested CEP processing including two alternative strategies for executing *NEEL* queries, namely expression sharing and expression caching in the continuous event sequence context. We also illustrated a critical healthcare application for infection prevention using our technology. In our future work, we will explore the question of how to bring the advances in expressive convenience of this proposed NEEL technology back into the other strands of CEP in particular ECUBE [10] and Hyreminder [5].

5. REFERENCES

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

[2] A. J. Demers, J. Gehrke, and et. al., "Cayuga: A general purpose event monitoring system." in *CIDR*, 2007, pp. 412–422.

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

[4] J. M. Boyce and D. Pittet, "Guideline for hand hygiene in healthcare settings," *MMWR Recomm Rep.*, vol. 51, pp. 1–45, 2002.

[5] D. Wang, H. Wang, and E. A. Rundensteiner, "Active complex event processing: Applications in real-time health care," in *VLDB (demo)*, 2010, pp. 1545–1548.

[6] M. Liu, E. A. Rundensteiner, D. Dougherty, C. Gupta, S. Wang, I. Ari, and A. Mehta, "NEEL: The nested complex event language for real-time event analytics," pp. 116–132, BIRTE, 2010.

[7] P. Seshadri, H. Pirahesh, and T. Y. C. Leung, "Complex query decorrelation," in *ICDE*, 1996, pp. 450–458.

[8] M. Liu, M. Ray, E. A. Rundensteiner, C. Gupta, S. Wang, I. Ari, and A. Mehta, "Processing strategies for nested complex sequence pattern queries over event streams," in *DMSN*, 2010, pp. 14–19.

[9] M. Liu, E. A. Rundensteiner, D. Dougherty, C. Gupta, S. Wang, A. Mehta, and I. Ari, "High-performance nested CEP query processing over event streams," in *ICDE*, 2011, pp. 122–134.

[10] M. Liu, E. Rundensteiner, K. Greenfield, C. Gupta, and et. al., "E-Cube: Multi-dimensional event sequence analysis using hierarchical pattern query sharing," in *SIGMOD conference*, 2011, pp. 889–900.