E-Cube: Multi-dimensional Event Sequence Processing Using Concept and Pattern Hierarchies

Recommended Citation


This paper is brought to you by eResearch@Ozyegin. For more information, please contact eresearch-help@ozyegin.edu.tr

eResearch@Ozyegin
Increasing the impact of OzU research
E-Cube: Multi-Dimensional Event Sequence Processing Using Concept and Pattern Hierarchies

Mo Liu *, Elke Rundensteiner #, Kara Greenfield #, Chetan Gupta *, Song Wang *, Ismail Ari, Abhay Mehta *

# Worcester Polytechnic Institute, USA
{lumo|rundenst|kgreenfield}@cs.wpi.edu

* Hewlett-Packard Labs, USA
{chetan.gupta|songw|abhay.mehta}@hp.com

Ozyegin University, Turkey
Ismail.Ari@ozyegin.edu.tr

Abstract— Many modern applications including tag based mass transit systems, RFID-based supply chain management systems and online financial feeds require special purpose event stream processing technology to analyze vast amounts of sequential multi-dimensional data available in real-time data feeds. Traditional online analytical processing (OLAP) systems are not designed for real-time pattern-based operations, while Complex Event Processing (CEP) systems are designed for sequence detection and do not support OLAP operations. We will demonstrate a novel E-Cube model that combines CEP and OLAP techniques for multi-dimensional event pattern analysis at different abstraction levels. A London transit scenario will be given to demonstrate the utility and performance of this proposed technology.

I. INTRODUCTION

A. Motivating Examples

Today, in many metropolitan areas from London to Moscow, mass transit agencies (MTA) issue their passengers near-field contactless (NFC) or contact based smart cards for fast payment and convenient access to transportation. The collected event data, including time and location information, continuously flow to a central location in the form of structured event streams. However, their analysis keeps lagging, thus missing critical opportunities for real-time analysis. Officials are demanding tools that will help them analyze the current status of these complex systems at different dimensions and in real-time, so that they can make strategic decisions and act on them quickly. These decisions include resource scheduling, route planning, variable pricing, fraud detection, revenue distribution, and planning mass evacuation for disaster scenarios.

For example, in Figure 1, we show related continuous queries that may be written by different officials to analyze the mass transit system operations. One MTA personnel may want to monitor movement of people from the greater London area to inner London within 24 hours in order to estimate how many people commute to work from outside of the city as shown in $Q_1$. Some officials might then look to see how many people are also returning to Outer London within 24 hours in order to differentiate between commuters and people staying in Inner London for an extended period of time. Local authorities in Zone 1 would be interested in how many people come into Zone 1 from Outer London every day. In such real-time monitoring and decision scenarios, our proposed system, E-Cube, can be coupled with the data middleware which collects and routes the data streams from various sources (metro stations, buses, etc.) to provide complex event stream processing functions.

Another interesting use case is derived from our collaboration with the University of Massachusetts Hospital, where for health reasons, employees need to be continuously tracked throughout their workdays for paths of exposure to different diseases and unhygienic conditions. They would be alerted if, for instance, they want to enter operation rooms without visiting the disinfection room first. To comply with federal health and safety regulations and prevent the spread of contagious diseases, hospital officials need the capability to conduct path analysis determining who went into an operating room, left for a break, and returned without disinfecting themselves. On finding matches, officials may then need to drill down to find out which operating rooms and/or patients may be affected.

B. Contributions of this Demonstration Proposal

In this demonstration, we present our E-Cube system, capable of online multi-dimensional event pattern analysis for complex scenarios such as the ones described above.

Traditional OLAP aims to quickly provide answers to analytical queries that are multi-dimensional in nature [1], [2], [3] by “navigating” the data at different abstraction levels. However, the state-of-the-art OLAP technology tends to be set-based instead of sequence based, eliminating the opportunity to look at individual matches. Also, such systems tend to aggregate over scalar values [3]. Hence in the context of event streams, it is insufficient in supporting multi-dimensional event sequence analysis. Complex Event Processing (CEP) systems demonstrate sophisticated capabilities for pattern matching [4].
[5], [6]. However, state-of-the-art CEP systems do not yet support OLAP operations for multi-dimensional stream analysis although such streams occur abundantly in real life. To the best of our knowledge, no prior work combines CEP and OLAP techniques for multi-dimensional pattern analysis over event streams as achieved in the E-Cube system that we propose to demonstrate. It is clearly not feasible to materialize the full cube over the space of multi-level sequences [7] since we are dealing with streaming data. Instead we only materialize cuboids corresponding to the user queries. We make the following contributions in this work:

- We define concepts needed to build a meaningful event pattern query hierarchy that integrates both pattern and concept refinement relationships between CEP queries.
- We identify OLAP like operations on E-Cube to allow users to navigate from one E-cuboid to another in the E-Cube space.
- We design several alternative E-cuboid query evaluation strategies, including specific-to-general and general-to-specific which evaluates coarser (resp. finer) level pattern queries in the E-Cube hierarchy by maximally reusing intermediate results from patterns at finer (resp. coarser) levels of abstraction.
- We design a cost-driven query optimizer, called Chase, that selectively exploits alternate E-cuboid evaluation strategies and determines an optimal global ordering of E-cuboids in the E-Cube hierarchy to achieve maximal re-use and optimal performance.
- We build the E-Cube system along with other alternate processing strategies within a uniform platform. Our comparative experimental study demonstrates the superior performance of our method, while our case study illustrates the utility of this proposed technology.

II. E-CUBE MODEL

An event pattern such as one specified by the SEQ operator [6] extracts sequences of event instances matching particular event types with order constraints from an event stream.

A concept hierarchy defines a sequence of mappings from a set of specific low-level concepts to high-level more general concepts [8]. Figure 2 shows a concept hierarchy for primitive events, e.g., the location attribute used in our MTA passenger tracking scenario. Due to domain dependencies, concept hierarchies for primitive event types and attributes need to be predefined by system administrators for the specific domain.

enforces the existence of more event types and associated sequential event relationships than \( Q_1 \).

An event pattern query hierarchy [9] then is the composition of concept hierarchies and pattern hierarchies into one integrated structure. Referring to Figure 1, an edge in an event pattern query hierarchy indicates that one pattern query \( q_j \) can be rolled up into another pattern query \( q_i \) by either changing the event type to a coarser level in the concept hierarchy, changing the pattern hierarchy to a coarser level, or both. Each edge is labeled with either the label “pattern”, “concept” or both to indicate the method by which relationships among the two queries are relaxed (or restricted) along this branch of the hierarchy.

An E-cuboid corresponds to a pattern query \( q_i \) and its results. E-Cube is a collection of smaller E-cuboids integrated based on pattern and concept sharing among those queries into an event pattern query hierarchy. Here we focus on the processing of E-cuboid continuous queries specified by \( < P_i, C_i, G_1, \text{agg} > \) where \( P_i \) refers to an event pattern, \( C_i \) refers to a concept abstraction level, \( G_1 \) refers to a GROUPBY clause, and \( \text{agg} \) refers to an aggregation function.

Multiple sequential pattern queries can be registered with the E-Cube system and their relations to each other can be represented as a directed-acyclic graph or DAG as shown in Figure 1. Pattern-relation denotes obtaining one sequence from another by deletion or addition of one or more event types and concept-relation denotes obtaining a sequence from another by moving up or down in the concept hierarchy.

E-OLAP Operations. We adopt the core set of OLAP operations for attributes, namely, roll-up, drill-down, slice, and dice. We also support a similar set of OLAP operations for patterns by pattern-roll-up, pattern-drill-down, concept-roll-up and concept-drill-down. In an event pattern query hierarchy, pattern-roll-up can be realized by deleting one or more event types. Pattern-drill-down is the reverse of pattern-roll-up. For example, in Figure 1, the pattern in \( Q_4 \) is rolled-up into the pattern in \( Q_3 \) by truncating the event types \( \text{zone2} \) and \( \text{zone1} \). Concept-roll-up can be realized by stepping up a concept hierarchy of event types. Concept-drill-down is the reverse of concept-roll-up.

III. CONDITIONAL PATTERN EVALUATION STRATEGIES

The evaluation of each pattern query is not independent in E-Cube. Rather, the event pattern query hierarchy helps us to achieve better performance in multi-pattern evaluation because it provides a blueprint of online pattern filtering and rapid result sharing. We now analyze and design strategies for the conditional processing of one continuous event pattern query from another.

A. General-to-specific Query Evaluation Method

The general-to-specific method proceeds by evaluating the pattern at a coarser level of abstraction first and exploiting the partial results for determining the results of the finer level patterns down the event pattern query hierarchy. For example, in Figure 1, \( Q_1 \) is evaluated first. To avoid re-evaluating the same subpattern, the results of \( Q_2 \) can be computed by joining the results of \( Q_1 \), OuterLondon and GreaterLondon events.

Besides computational sharing, by using this general-to-specific evaluation method, we can also achieve online pattern filtering. The idea is that if pattern \( Q_1 \) is at a coarser level
than pattern $Q_2$, and a matching attempt with $Q_1$ fails, then there is no need to carry out the evaluation for $Q_2$. Our cost analysis and experimental evaluation have shown that this is the most efficient method when the queries being evaluated differ only in pattern level and not in concept level [9].

B. Specific-to-general Query Evaluation Method

The specific-to-general method evaluates the lower level pattern query first, then these results can be reused for the upper level pattern evaluation. In addition, the events of the higher level event concept type that have not been captured by the lower queries must also be constructed as delta results. For example, in Figure 1, from $Q_1$ to $Q_3$ only the event concept hierarchy level is changed. Results for $Q_3$ are computed first before $Q_1$ and returned for $Q_1$. So we don’t need to recompute such results during evaluation of $Q_1$. Our cost analysis and evaluation have shown that this is the most efficient method when the queries being evaluated differ only in concept level and not in pattern [9].

C. CHASE: Cost-based Hybrid Adaptive Sequence Evaluation

When evaluating each pattern in an event pattern query hierarchy, three choices are considered (1) Compute a query independently from other queries by stack-based join method; (2) Compute a query from one of the parent queries by the general-to-specific method; (3) Compute a query from one of its children queries by the specific-to-general method. Given an event pattern query hierarchy $H$, we define a directed weighted graph $G = (V, E)$ where $V$ denotes the set of vertices that one to one correspond to the queries in the event query hierarchy $H$. A directed edge $e(v_i, v_j) \in E$ with the weight $w(v_i, v_j)$ denotes the computation cost of computing vertex $v_i$ (query $q_i$) from vertex $v_j$ (query $q_j$). After the weighted directed graph is constructed, the optimum execution ordering can be calculated by finding the minimum cost spanning tree (MST) in the graph. The forest representing the optimum execution ordering and the minimum spanning tree coincide, since both trees are required to reach all the query nodes in the event query hierarchy with minimum total edge cost.

D. Execution of Optimized Evaluation Plan

E-cube executor evaluates the queries in the order determined by our cost-based optimizer. The logical event pattern query hierarchy, which is a directed labeled graph representing the pattern queries registered in the system and their inter-relationships, is encoded as an efficient hash look-up structure. This structure acts as the blueprint for the construction of the run-time data structure, called the hierarchical instance stacks, which hold the event instances being processed in a compact manner, namely, each event instance is only stored once even though it may match event types in multiple queries. This facilitates automaton-based complex event processing, in particular, the application of the efficient stack-based join strategy to evaluate pattern queries – while avoiding unnecessary eager state enumeration, as is found in most CEP based systems [6], [5], [10].

Furthermore, this run-time structure also encodes the reuse-based execution strategies as determined by the optimizer. If a query $Q_i$ is executed independently from other queries, i.e., using the stack-based join strategy, the execution is triggered to follow the edges of the hierarchical pattern graph annotated with the assertions by the same query identifier $Q_i$. If a query $Q_i$ is executed based upon its parent (child) query $Q_j$, the blueprint instead instructs the executor to reuse the results from the $Q_i$’s parent (child) query $Q_j$, and to construct delta results using stack-based traversal, as needed. For example, if the query $Q_i$ is computed from its parent query $Q_j$ with only pattern changes, the delta results are constructed by joining the buffered parent results with the relevant event instances from the stacks whose types differ between the two queries. Overall, this optimized execution assures maximum result reuse and avoidance of unnecessary eager state enumeration.

E. System Architecture

We have implemented our proposed E-Cube framework shown in Figure 3 in a prototype data stream management system called CHAOS [11]. E-Cube features a powerful interface for interacting with the users for pattern query specification and refinement, for E-Cube hierarchy navigation, for result viewing, and for performance monitoring. As explained above, the E-Cube Optimizer utilizes a cost-based approach to determine ordering among and reuse of results among pattern queries. These instructions are then passed to the E-Cube Executor, where they are instantiated into a runtime infrastructure which exploits compact hierarchical stacks for shared storage and computation and the Hierarchical Pattern Graph for triggering the optimized ordering of execution and result reuse among queries. The E-Cube Executor receives event streams from the stream feeder, and passes its output, along with run-time statistics collected by the statistics gatherer, up to the Interaction Server for preparation for display to the user.

IV. SOFTWARE DEMONSTRATION

Our demonstration will demonstrate the London transit scenario. The audience of our demonstration will be able to see and perform the following actions using our E-Cube system:

A. Define and Select a Concept Hierarchy

First, the attendees will be able to act as a system administrator to create a concept hierarchy through an interactive Graphical User Interface (GUI) where the user simply selects a parent node and then enters in the concept name for one of its children. Upon clicking the AddChild button, the page
will be automatically updated to reflect the changes in the tree. Once a concept hierarchy has been created, it will be available from a drop down list for future references. Switching to the role of an end user, attendees will then be able to select and display their concept hierarchy.

B. Create, Submit, and Re-use Pattern Queries

Next, the users will be able to create, compile, and execute a query via the query console using the E-Cube language as shown in Figure 1. An alternate way to add a new query is to right-click on a query that has already been created, bringing up a menu of extended OLAP operations and selecting one of which will result in the corresponding query being created (See Figure 4). With either method, the updated event pattern query hierarchy including the new query (appearing highlighted) will be shown. Once all of the queries have been entered, E-Cube optimizer will analyze them and organize them into a pattern event hierarchy as shown in Figure 5. The bold arrows depict the sequence of reuse. E-Cube also allows users to save their event pattern hierarchies, so that they can reload the hierarchical state later in time.

C. Hierarchy Navigation

Optionally, the attendees will be able to navigate E-Cuboids through the hierarchy and retrieve only the corresponding sequence results instead of all results being displayed concurrently. After selecting the E-Cuboid for an initial query, the attendees can navigate through the hierarchy to get the results at different abstraction and refinement levels.

D. Explore Results of Our Optimizer

The query optimizer applies cost-based optimization strategies to select both the global order of execution among the queries within a hierarchy (indicated by circled numbers in each query in Figure 5) as well as the local decision of pattern reuse versus native stack-based join computation (indicated by a dashed line surrounding the query in Figure 5). The attendees can learn about the decisions made by the optimizer.

E. View Results

The attendees will have several options for viewing the results, either as a continuous output stream of matches, as a graph showing the number of matching results found over time (see Figure 6), and other forms of visualization. If the user rolls over any portion of the graph of results, a pop-up box will appear, telling them the number of matches that have been found as of that moment in the evaluation.

F. Performance Monitoring

Finally, our demonstration will measure and visually illustrate the statistics at run time from the monitoring component in E-Cube. This includes size of input over time, memory consumption, storage reuse, filtering percentage, and filtering time.

ACKNOWLEDGEMENTS

This work was supported by HP Innovations Award, IIS NSF SGER No. 0633930, IIS NSF No. 041456, NSF REU.

REFERENCES