Poster: DejaVu – A Complex Event Processing System for Pattern Matching over Live and Historical Data Streams

Nihal Dindar, Peter M. Fischer, Nesime Tatbul
Systems Group, ETH Zurich, Switzerland
{dindarn, peter.fischer, tatbul}@inf.ethz.ch

ABSTRACT

This short paper provides an overview of the *DejaVu complex event processing (CEP) system*, with an emphasis on its novel architecture and query optimization techniques for correlating patterns across live and historical data streams.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems - Query Processing

General Terms

Performance, Languages, Theory

1. INTRODUCTION

CEP has proven to be an important technology for analyzing complex relationships over high volumes of data in many application domains. High-performance pattern matching over live event streams has been a central focus in CEP research to date. Though a less explored research direction, archiving streams and integrating them into the live stream processing pipeline offers many new key capabilities for CEP systems. In particular, longer-term data analysis, such as making predictions about future event occurrences or identifying causal relationships among complex events across multiple time scales is beyond the scope of existing CEP systems.

In this paper, we provide an overview of the DejaVu system developed at ETH Zurich, which provides declarative pattern matching capability over live and archived streams of events. DejaVu proposes a novel integrated CEP architecture and focuses on scalable data management techniques for processing various forms of pattern matching queries over event streams. We summarize the fundamental concepts behind DejaVu's architectural design, as well as its approach to optimized processing of a useful class of hybrid pattern matching queries, namely *pattern correlation queries (PCQ)*.

2. DEJAVU SYSTEM OVERVIEW

DejaVu is a CEP system that integrates declarative pattern matching over live and archived streams of events [4]. We have built DejaVu on top of the MySQL relational database system [1]. As such, we follow the basic architecture of MySQL, while making new extensions for supporting pattern matching queries. Figure 1 illustrates a high-level architecture of DejaVu. One of the key architectural features of MySQL that we exploit in our design is its pluggable storage engine API, introducing two new types of stores:

DEBS'11, July 11–15, 2011, New York, New York, USA.
ACM 978-1-4503-0423-8/11/07.

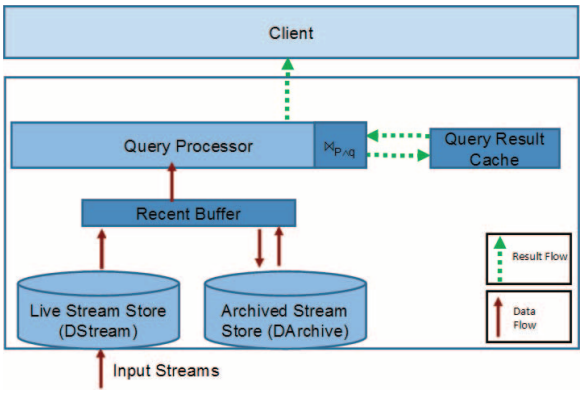


Figure 1: Architectural overview of the DejaVu CEP system

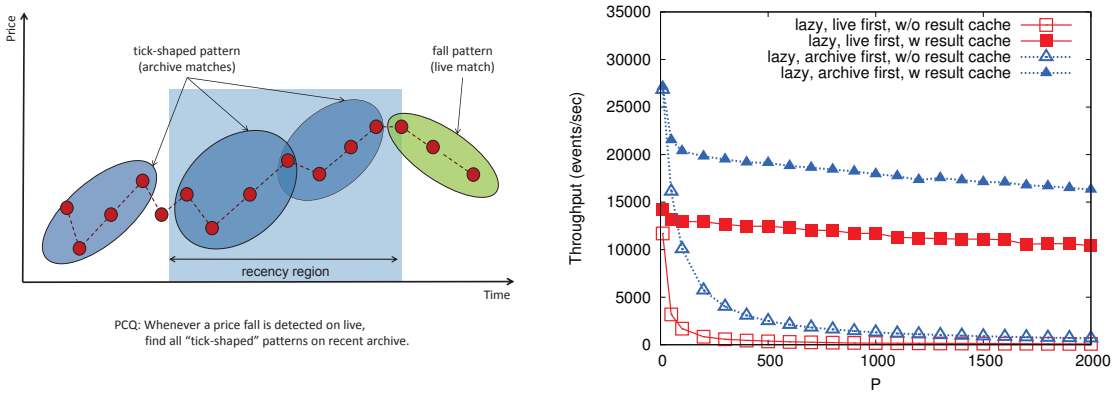
- **Live Stream Store (DStream)** is an in-memory store that accepts push-based inputs. It essentially acts like a tuple queue, providing live events into the Query Processor (QP) as they arrive. It supports both pull and push access by the QP.
- **Archived Stream Store (DArchive)** is a persistent stream store to materialize live events for long-term access. Given the predefined order of the events, it only support append updates.

Recent Buffer, an in-memory cache, mediates between the DStream, the DArchive, and the QP to provide efficient access to recent input tuples as well as handling bulk inserts into the DArchive.

Furthermore, we have extended the QP of MySQL with a finite state machine (FSM) implementation for pattern evaluation. More specifically, patterns are expressed with regular expressions and are represented with FSM operators, which are then integrated into the relevant parts of the MySQL query plan. New join algorithms are also implemented for handling pattern correlation queries. As a relational database system, MySQL is designed to work on one-time queries only. Thus, we have added query life-cycle management and continuous result reporting into the MySQL QP.

The last key component in our architecture is the **Query Result Cache**. Like materialized views in databases, it provides novel data structures to store previous pattern matches that can be reused later.

DejaVu follows a data model where streams are totally ordered sequences of relational tuples with start and end timestamps. Ensuring the total order for all streams provides DejaVu queries to be fully composable. As for the query model, in DejaVu, complex event patterns are expressed through an SQL-like declarative language that is based on a standard proposal for pattern matching over row sequences [5]. This proposal extends the `FROM` part of SQL with a new `MATCH_RECOGNIZE` clause that enables pattern specifications over the listed data sources. As an addition to the original proposal, we also allow streaming sources in the `FROM` part along with regular tables (see [3] or [4] for query examples). In



(a) Pattern correlation query

(b) Results with the NYSE TAQ dataset [2]

Figure 2: Financial use case

this case, the `MATCH_RECOGNIZE` clause in effect defines a "semantic window" over the live stream. This way, DejaVu's extended language can express a wider range of queries including hybrid pattern matching queries, such as pattern correlation queries.

3. PATTERN CORRELATION QUERIES

Correlating live and historical complex events for identifying causal dependencies between them or for predicting the reoccurrence of similar past events is a critical capability needed in many application domains such as medical diagnosis, algorithmic trading, travel time estimation for route planning. In DejaVu, we provide this capability via *Pattern Correlation Queries (PCQs)* [3].

Figure 2(a) illustrates a typical use case for PCQs from the algorithmic trading domain, where a trader would like to predict the stocks that could bring profits in the near future based on a query posed over market data events which does the following: Upon detecting a fall in the current price of stock X on the live data stream, look for a tick-shaped pattern for X within *recent* archive, where a fall in price was followed by a rise in price that went higher up than the beginning price of the fall. This use case requires evaluating complex events over live (falling) and archived streams (tick-shaped), and correlating them based on a recency criteria. The high-rate live stream and the high-volume archived stream, as well as the need for low-latency results for catching momentary trading opportunities render this use case a highly challenging one.

In current CEP systems, the only way to correlate two streams is using a join operator with time- or tuple-based windows. While PCQs can be implemented using such windows (forming all possible recency windows on live and archived streams, joining them, performing pattern processing on each joined result, and then joining the matched patterns), this approach would obviously be very costly. A better alternative would be to first apply pattern matching on both sources and then join the matched patterns using the recency window using a regular join. This *eager* pattern processing approach would be significantly cheaper, however, would still lead to processing some redundant patterns that will never contribute to the final result. In DejaVu, we instead propose a *lazy* pattern processing approach, where a pattern on one source (live or archive) is only computed if a corresponding pattern that falls in the recency window is found on the other one (archive or live). Moreover, on top of this lazy approach, we introduce three further optimizations:

- **Recent input buffering** enables the caching of the most recent stream tuples in memory for efficient access for both live and historical pattern matching.
- **Query result caching** avoids the redundant recomputation of patterns on one source that are correlated with multiple patterns on the other one.
- **Join source ordering** ensures that the source with the more selective pattern is used as the outer source to the hybrid nested loops join algorithm that implements our lazy approach, thus further reducing the total number of pattern computations.

We have implemented all the algorithms and optimization techniques summarized above in the DejaVu system and studied their performance through detailed experiments. Here, we provide a selected result for the financial PCQ illustrated above, run over a real stock market dataset from NYSE [2]. Figure 2(b) shows how throughput (i.e., input events consumed per second) changes with increasing recency window size (P). There are two important observations to make: (i) Since the historical pattern is more selective on this dataset, archive-first outperforms live-first (thus, also showing that join source ordering can yield significant performance gains); (ii) For both live-first and archive-first, using a query result cache leads an improve by more than order of magnitude.

A detailed description and evaluation of DejaVu's PCQ processing and optimization techniques is provided in a recent paper [3].

4. FUTURE DIRECTIONS

DejaVu presents a rich platform for exploring further research problems. Short-term directions include studying other forms of PCQs with correlation criteria based on, e.g., context similarity or other spatio-temporal relationships across complex event patterns, as well as providing suitable storage management techniques for efficient archive access (e.g., indexing). In the longer term, we would like to explore query rewrite-based optimizations for `MATCH_RECOGNIZE`.

5. REFERENCES

[1] MySQL. `http://www.mysql.com/`.
[2] NYSE Data Solutions. `http://www.nyxdata.com/nysedata`.
[3] N. Dindar, P. M. Fischer, M. Soner, and N. Tatbul. Efficiently Correlating Complex Events over Live and Archived Data Streams. In *ACM DEBS Conference*, New York, NY, July 2011.
[4] N. Dindar, B. Güç, P. Lau, A. Özal, M. Soner, and N. Tatbul. DejaVu: Declarative Pattern Matching over Live and Archived Streams of Events (Demo). In *ACM SIGMOD Conference*, Providence, RI, June 2009.
[5] F. Zemke, A. Witkowski, M. Cherniack, and L. Colby. Pattern Matching in Sequences of Rows. Technical Report ANSI Standard Proposal, July 2007.