DejaVu: Declarative Pattern Matching over Live and Archived Streams of Events *

Nihal Dindar, Barış Güç, Patrick Lau, Aslı Özal, Merve Soner, Nesime Tatbul
Systems Group, Department of Computer Science, ETH Zurich, Switzerland
{dindarn, baris, laup, aoezal, msoner}@student.ethz.ch, tatbul@inf.ethz.ch

ABSTRACT

DejaVu is an event processing system that integrates declarative pattern matching over live and archived streams of events on top of a novel system architecture. We propose to demonstrate the key aspects of the DejaVu query language and architecture using two different application scenarios, namely a smart RFID library system and a financial market data analysis application. The demonstration will illustrate how DejaVu can uniformly handle one-time, continuous, and hybrid pattern matching queries over live and archived stream stores, using highly interactive visual monitoring tools including one that is based on the Second Life virtual world.

Categories and Subject Descriptors: H.2.4 [Database Management]: Systems – *query processing, relational databases*; H.3.4 [Information Storage and Retrieval]: Systems and Software – *current awareness systems*; H.4 [Information Systems Applications]: Miscellaneous

General Terms: Design

Keywords: data streams, complex event processing, pattern matching, MySQL, RFID, Second Life

1. INTRODUCTION

Pattern matching over data sequences has recently become a key requirement due to the increasing number of complex event processing (CEP) applications. Well-known CEP applications include financial market data analysis, RFID-based asset tracking, operational business intelligence, and network intrusion detection. In all of these applications, it is important to be able to detect complex patterns over live as well as archived streams of events.

The existing CEP engines (e.g., [5], [6]) mostly focus on high-performance pattern matching over live event streams. Furthermore, coming up with the right language primitives for defining these patterns has been a key concern and each system has proposed its own custom pattern matching language. In spite of their different syntax, these languages in fact have several common concepts that are deemed essential for CEP, such as regular expressions with selection predicates.

In DejaVu, we have a different focus and approach than the existing CEP systems. First of all, we believe that seamless integration of pattern matching functionality over live and stored event streams behind a common declarative interface is crucial. This not only enables arbitrary event correlations within or across different time domains, but also paves the way for making predictions about future event occurrences, since an important class of interesting event patterns are the ones that repeat in time, creating a sense of "deja vu". Furthermore, we believe that functionality-wise, pattern matching over live and stored streams have a lot in common; and therefore, reusing language and processing constructs for both scenarios is the right approach to take. On the other hand, performance issues and optimization possibilities are rather different for these scenarios. Hence, our goal in DejaVu is to develop a coherent system that unifies interface and functionality, but allows for flexible customization of implementation and optimization methods across live and historical data sequences. To realize this goal, we propose a novel approach that builds on and significantly extends a relational database engine architecture, by following a recent proposal for an SQL-based declarative pattern matching language standard [10].

In this demonstration proposal, we first present an architectural overview of our DejaVu system. Then we summarize the application scenarios that we have been building on top of DejaVu in order to showcase its main features. Finally, we discuss the issues regarding the logistic setup of the proposed demonstration scenarios.

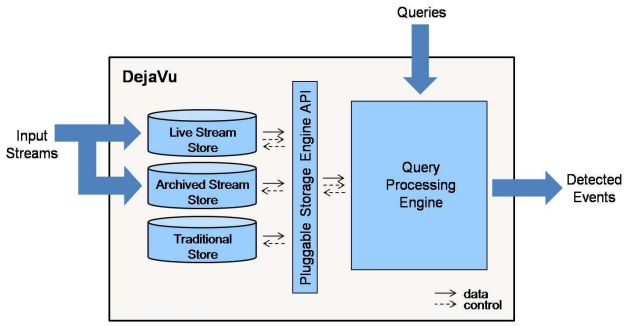


Figure 1: DejaVu System Architecture

2. DEJAVU SYSTEM OVERVIEW

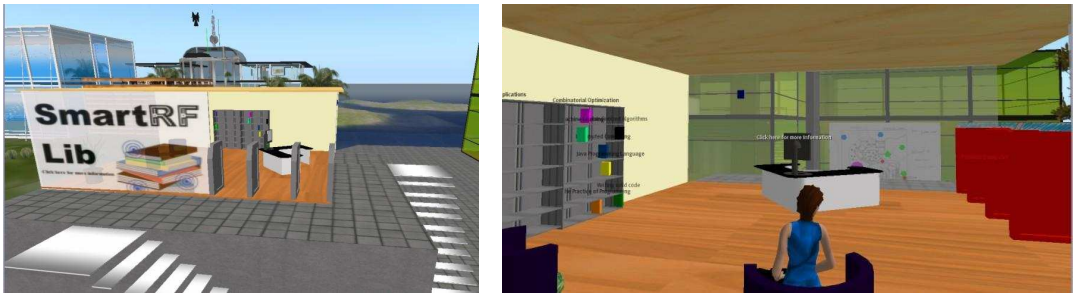
We have built DejaVu on top of the MySQL open-source database system [8]. As such, we follow the basic architectural skeleton of MySQL, while making new extensions for adding support for running different forms of pattern matching queries (one-time, continuous, and hybrid) as necessary.

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. Through this API, one can plug custom storage engines to work seamlessly with the MySQL query processing engine. In addition to the traditional relational store provided by MySQL, which we occasionally need for table lookups, we have also created two new types of storage engines:

- **Live Stream Storage:** This is an in-memory store that accepts push-based inputs. It essentially acts like a queue, providing live events into the query processing engine as they arrive. The

*This work has been supported by the following grants: Swiss NSF NCCR MICS 5005-67322, Swiss NSF ProDoc PDFMP2-122971/1, ETH matching funds, and a gift from Siemens.

SIGMOD'09, June 29–July 2, 2009, Providence, Rhode Island, USA.
ACM 978-1-60558-551-2/09/06.



(a) SmartRFLib on Second Life (b) Book theft event

Figure 2: Smart RFID Library application scenario

query processor can access the live stream store either in pull-mode or push-mode. We have added the latter option in our design to enable an adaptive switch between these two modes in order to flexibly cope with unpredictable bursts in input load inside the storage engine. In other words, under high load, we switch to pull mode and let the storage handle the problem.

- **Archived Stream Storage:** Live input events can also be fully or selectively materialized into an archive store. Archive store respects the pre-defined order of the events, and only allows updates in the form of appends. It is also designed to provide features such as data compression and efficient access methods for historical pattern matching queries. Furthermore, since the archive is a persistent store, it can support the live store in dealing with bursts and failures.

In addition to creating two new storage engines, we have also made an important extension to the MySQL query processor. More specifically, we have introduced a finite state machine (FSM) implementation to drive the evaluation of the pattern matching queries, where the FSM runs as an integral part of the MySQL query plan.

On the language front, we have extended the MySQL language parser with the `MATCH_RECOGNIZE` clause [10]. In the standard SQL syntax, this clause follows a table name in the `FROM` part and enables the match of the specified pattern on that table. Thus, the original language proposal assumes that pattern matching queries will be applied over contiguous rows in a given relational table. In DejaVu, we follow a similar syntax, but allow the `MATCH_RECOGNIZE` clause to be attached to both archived stream tables as well as live stream tables. In fact, we interpret that in the latter case, the `MATCH_RECOGNIZE` clause defines a "semantic window" over the live stream. This interpretation is in perfect agreement with the traditional SQL syntax for time- and count-based windows [4], [3]. We will show various features of the `MATCH_RECOGNIZE` clause with examples in the next section.

Finally, we have added new DDL statements and corresponding metadata into the MySQL catalogs to define continuous queries and tables of new store types. For continuous queries, we had to also add new mechanisms into the MySQL engine for query life-cycle management and continuous result reporting.

3. DEMONSTRATION DETAILS

In this section, we first describe two application scenarios, that will demonstrate different features of our system. Then we briefly discuss issues related to the logistic setup of our demonstration.

3.1 App #1: Smart RFID Library

Our first application is from the RFID-based asset tracking domain. We consider a library with books and users, each tagged with RFID labels. Based on continuous RFID readings from books and people, we would like to detect important library events such as book check-ins, check-outs, illegal check-outs (e.g., reference books or quota violation), and thefts. A similar library scenario was also used by the HiFi Project before, where the main focus was rather on the RFID event acquisition problem using a different architecture and a much simpler event language than ours [9]. Furthermore, we have also built a virtual library at the ETH Island on Second Life called *SmartRFLib* [2] (see Figure 2(a)). Through this virtual library, we can interactively visualize the detected events in real time. As such, our system also connects the real world events detected via an RFID sensor network with their representations on the virtual world of Second Life.

We will use this application scenario to demonstrate continuous pattern matching queries over the DejaVu live stream store. In the actual demonstration, we will show queries for all the complex events listed above. Here we only show the query for the book theft event as an example. Given a live stream of book readings Books(Tstamp, ReaderId, TagId) from the readers, the book theft event can be expressed as follows:

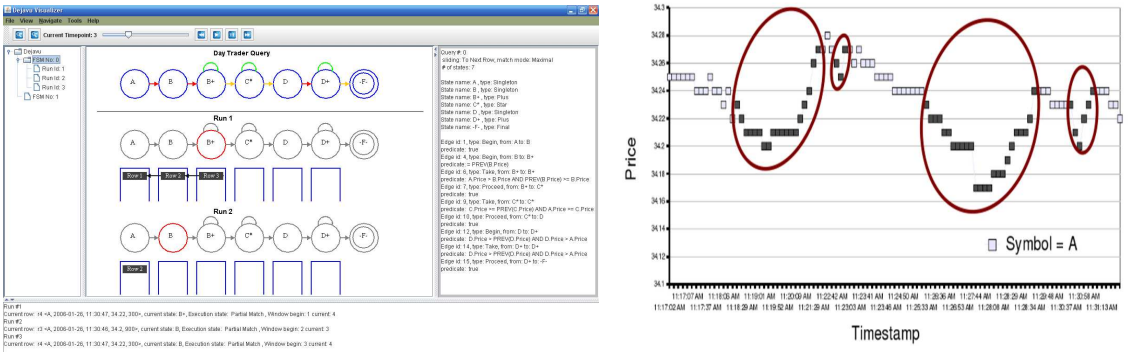
```
SELECT notify_theft(tstamp, book_tag, book_name)
FROM Books MATCH_RECOGNIZE(
  PARTITION BY TagId
  MEASURES B.Tstamp AS tstamp,
           B.TagId AS book_tag,
           get_name(B.TagId) AS book_name,
  ONE ROW PER MATCH
  AFTER MATCH SKIP PAST LAST ROW
  INCREMENTAL MATCH
  PATTERN(A B)
  DEFINE A AS (A.ReaderId = 'Shelf')
         B AS (B.ReaderId = 'Exit')
  );
```

Figure 2(b) shows how the resulting theft alert is visualized in Second Life (exit gates turn red + an alarm sounds).

3.2 App #2: Financial Market Data Analysis

Our second application is from the financial services domain. We took a real financial dataset from NYSE TAQ (Trade and Quote) database [1] and loaded this dataset into our archived stream store. We can also replay the same data as a continuous stream through our live stream store. Furthermore, we defined a set of common financial pattern matching queries (e.g., [7]).

We will use this application scenario to demonstrate one-time and hybrid pattern matching queries over the DejaVu archived and live stream stores. Here we give one example for each. Let ArchivedStock(Tstamp, Symbol, Price) be an archived stream table and LiveStock(Tstamp, Symbol, Price) be a live stream table. The following query is a simple day trader query that looks for a "tick-shape" pattern (i.e., a fall in price, followed by a rise in price that went higher up than the beginning price) for each company symbol over the historical NYSE market events in ArchivedStock:



(a) DejaVu FSM Visualizer and Debugger (b) "Tick patterns" detected over NYSE TAQ archive

Figure 3: Financial Market Data Analysis application scenario

```
SELECT symbol, initial_price, min_price, max_price
FROM ArchivedStock MATCH_RECOGNIZE(
  PARTITION BY Symbol
  MEASURES A.Symbol AS symbol,
           A.Price AS initial_price,
           MIN(B.Price) AS min_price,
           LAST(D.Price) AS max_price
  ONE ROW PER MATCH
  AFTER MATCH SKIP PAST LAST ROW
  MAXIMAL MATCH
  PATTERN(A B+ C* D+)
  DEFINE /* A matches any row */
    B AS (B.Price<A.Price AND B.Price<=PREV(B.Price))
    C AS (C.Price>=PREV(C.Price) AND C.Price<=A.Price)
    D AS (D.Price>PREV(D.Price) AND D.Price>A.Price)
  );
```

Figure 3(a) shows a run-time snapshot of this query on our interactive FSM visualizer and debugger tool, while Figure 3(b) shows how we visualize the result patterns on a graph.

The following query is a hybrid version of the day trader query. We replay a portion of the TAQ data as a live stream. Whenever a fall in price is detected on this live stream, we would like to look for historical tick patterns where the fall was followed by an increase that went up higher than the beginning price of the fall. The goal is to identify the stocks that could bring profits in the near future.

```
SELECT min_tstamp_l, symbol_l, min_price_l,
       initial_price_a, min_price_a, max_price_a
FROM LiveStock MATCH_RECOGNIZE(
  PARTITION BY Symbol
  MEASURES A.Symbol AS symbol_l,
           MIN(B.Tstamp) AS min_tstamp_l,
           MIN(B.Price) AS min_price_l
  ONE ROW PER MATCH
  AFTER MATCH SKIP PAST LAST ROW
  INCREMENTAL MATCH
  PATTERN (A B+)
  DEFINE /* A matches any row */
    B AS (B.Price<A.Price AND B.Price<=PREV(B.Price))
  ), ArchivedStock MATCH_RECOGNIZE(
  PARTITION BY Symbol
  MEASURES A.Symbol AS symbol_a,
           A.Price AS initial_price_a,
           MIN(B.Price) AS min_price_a,
           LAST(D.Price) AS max_price_a
  ONE ROW PER MATCH
  AFTER MATCH SKIP PAST LAST ROW
  MAXIMAL MATCH
  PATTERN (A B+ C* D+)
  DEFINE /* A matches any row */
    B AS (B.Price<A.Price AND B.Price<=PREV(B.Price))
    C AS (C.Price>=PREV(C.Price) AND C.Price<=A.Price)
    D AS (D.Price>PREV(D.Price) AND D.Price>A.Price)
  )
WHERE symbol_l = symbol_a;
```

3.3 Logistic Setup

In our demonstration, the DejaVu server runs on a Linux laptop. For the first application scenario, we will use three RFID readers, placed at different locations in the demonstration room. We also have a two-reader version of the same demonstration in case physical space becomes an issue. A second laptop acts as the base station for the RFID readings. Finally, a third laptop acts as the client interface, on which we also run the Second Life (SL) client. Note that the SL interface requires internet connectivity to communicate with the SL servers. In case of poor or no connectivity at the demonstration venue, we can still display the output event logs (we are also planning to make in advance a video of the full demonstration (including the SL part) as a precaution). The demonstration audience can interact with SmartRFLib by using the RFID tags that we will make available for them with which they can then create check-in, check-out, and theft events themselves. They can also play with our SL interface by registering their own avatars or by using one of our pre-defined avatars. For the second application scenario, internet connectivity is not needed. The audience can interact with DejaVu by defining their own financial pattern matching queries, watching or debugging the FSM execution on the visualizer, and finally seeing the results plotted on a graph.

Acknowledgments. We thank Çağrı Balkesen, Gautier Boder, Florian Keusch, Katinka Kromwijk, and Ali Şengül for their contributions to an earlier version of SmartRFLib, and Michele De Lorenzi and Julien Vocat for their help with Second Life.

4. REFERENCES

[1] NYSE Data Solutions. `http://www.nyxdata.com/nysedata/`.
[2] Second Life. `http://www.secondlife.com/`.
[3] StreamSQL. `http://www.streamsql.org/`.
[4] A. Arasu, S. Babu, and J. Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. *VLDB Journal*, 15(2), 2006.
[5] A. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, and W. White. Cayuga: A General Purpose Event Monitoring System. In *CIDR Conference*, Asilomar, CA, January 2007.
[6] D. Gyllstrom, E. Wu, H. Chae, Y. Diao, P. Stahlberg, and G. Anderson. SASE: Complex Event Processing over Streams (Demo). In *CIDR Conference*, Asilomar, CA, January 2007.
[7] A. Lerner and D. Shasha. The Virtues and Challenges of Ad Hoc + Streams Querying in Finance. *IEEE Data Engineering Bulletin*, 26(1), March 2003.
[8] S. Pachev. *Understanding MySQL Internals*. O'Reilly, 2007.
[9] S. Rizvi, S. R. Jeffery, S. Krishnamurthy, M. J. Franklin, N. Burkhart, A. Edakkunni, and L. Liang. Events on the Edge (Demo). In *ACM SIGMOD Conference*, Baltimore, MD, June 2005.
[10] F. Zemke, A. Witkowski, M. Cherniack, and L. Colby. Pattern Matching in Sequences of Rows. Technical Report ANSI Standard Proposal, July 2007.