MonetDB/DataCell: Online Analytics in a Streaming Column-Store

Erietta Liarou, Stratos Idreos, Stefan Manegold, Martin Kersten
CWI, Amsterdam, The Netherlands
erietta@cwi.nl idreos@cwi.nl manegold@cwi.nl mk@cwi.nl

ABSTRACT

In DataCell, we design streaming functionalities in a modern relational database kernel which targets big data analytics. This includes exploitation of both its storage/execution engine and its optimizer infrastructure. We investigate the opportunities and challenges that arise with such a direction and we show that it carries significant advantages for modern applications in need for online analytics such as web logs, network monitoring and scientific data management. The major challenge then becomes the efficient support for specialized stream features, e.g., multi-query processing and incremental window-based processing as well as exploiting standard DBMS functionalities in a streaming environment such as indexing.

In this demo, we present the DataCell system, an extension of the MonetDB open-source column-store for online analytics. The demo gives the user the opportunity to experience the features of DataCell such as processing both stream and persistent data and performing window based processing. The demo provides a visual interface to monitor the critical system components, e.g., how query plans transform from typical DBMS query plans to online query plans, how data flows through the query plans as the streams evolve, how DataCell maintains intermediate results in columnar form to avoid repeated evaluation of the same stream portions, etc. The demo also provides the ability to interactively set the test scenarios regarding input data and various DataCell knobs.

1. INTRODUCTION

Numerous applications nowadays require online analytics over high rate streaming data. For example, emerging applications over mobile data can exploit the big mobile data streams for advertising and traffic control. In addition, the recent and continuously expanding massive cloud infrastructures require continuous monitoring to remain in good state and prevent fraud attacks. Similarly, scientific databases create data at massive rates daily or even hourly. In addition, web log analysis requires fast analysis of big streaming data for decision support.

A new processing paradigm is born [16, 9, 12] where incoming *streams* of data need to quickly be analyzed and possibly combined with existing data to discover trends and patterns. Subsequently, the new data may also enter the data warehouse and be stored as normal for further analysis if necessary. This new paradigm requires scalable query processing that can combine continuous querying for fast reaction to incoming data with traditional querying for access to the existing data. However, neither pure database technology nor pure stream technology are designed for this purpose. Database systems do not qualify for continuous query processing, while data stream systems are not built to scale for big data analysis.

DataCell Motivation. MonetDB/DataCell aims to provide scalable online analytics. We begin from a state of the art column-store design for big data analytics, MonetDB, and extend it with online functionality. The goal is to fully exploit the generic storage and execution engine of the DBMS as well as its complete optimizer stack. Stream processing then becomes primarily a query *scheduling* task, i.e., make the proper queries see the proper portion of stream data at the proper time. A positive side-effect is that our architecture supports SQL'03, which allows stream applications to exploit sophisticated query semantics. Numerous research and technical questions immediately arise. The most prominent issues are the ability to provide specialized stream functionality and hindrances to guarantee real-time constraints for event handling.

Contributions and Demo. Paper [16] illustrates the basic DataCell architecture and sets the research path and critical milestones. In addition, DataCell ships together with the MonetDB open-source system. Here, we present a demo based on MonetDB/DataCell. The demo showcases several of the key aspects of the MonetDB/DataCell design such as the ability to do both stream processing and normal queries and the ability to provide window based processing. Through a graphical user interface the user of the demo can observe the query execution inside DataCell, e.g., how the various columnar structures are populated, how intermediate results are kept around to avoid reevaluation for sliding window queries and how the shape of a normal query plan changes to a continuous query plan through the optimizer. In addition, the user can interact with the system, posing continuous or one-time queries or choosing some pre-defined scenarios as well as varying both input parameters and DataCell knobs.

Paper Organization. The rest of the paper is organized as follows. Section 2 briefly discusses related work while Section 3 gives an overview of the main design of DataCell. Then, Section 4 presents the demonstration scenarios as well as the ways the user can interact with the system and how the demo visualizes query execution and system status. Finally, Section 5 concludes the paper.

2. RELATED WORK

DataCell fundamentally differs from existing stream efforts [2, 3, 4, 7, 8, 10, 13, 21, 5, 11], etc. by building on top of the storage and execution engine of a DBMS kernel. It opens a very interesting path towards exploiting and merging technologies from both worlds.

Compared to even earlier efforts on *active* databases, e.g., [20], DataCell adds support for specialized stream functionalities, such as incremental processing. Incremental processing in a DBMS has been studied in the context of updating materialized views, e.g., [6, 14], but there the scenario is very different given that it targets read-mostly environments whereas an online scenario is by definition a write-only one.

Truviso Continuous Analytics system [12], a commercial product of Truviso, is another recent example that follows the same approach as DataCell. They extend the open source PostgreSQL database [19] to enable continuous analysis of streaming data, tackling the problem of low latency query evaluation over massive data volumes. TruCQ integrates streaming and traditional relational query processing in such a way that ends-up to a *stream-relational* database architecture. It is able to run SQL queries continuously and incrementally over data while they are still coming and before they are stored in *active database tables* (if they need to be stored). TruCQ's query processing significantly outperforms traditional *store-first-query-later* database technologies as the query evaluation has already been initiated when the first tuples arrive. It allows evaluation of one-time queries, continuous queries, as well as combinations of both types.

Another recent work, coming from the HP Labs [9], also confirms the strong research attraction for this trend. It defines an extended SQL query model that unifies queries over both static relations and dynamic streaming data, by developing techniques to generalize the query engine. It also extends the PostgreSQL database kernel [19], building an engine that can process persistent and streaming data in a uniform design. First, they convert the stream into a sequence of *chunks* and then continuously call the query over each sequential chunk. The query instance never shuts down between the chunks, in such a way that a cycle-based transaction model is formed.

The main difference of DataCell over the above two related efforts lies in the underlying architecture. DataCell builds over a column-store kernel using a columnar algebra instead of a relational one, bulk processing instead of volcano and vectorized query processing as opposed to tuple-based.

3. MONETDB/DATACELL

In this section, we discuss the DataCell architecture. We build the DataCell on top of MonetDB [17], an open-source column-oriented DBMS.

A Column-oriented DBMS. MonetDB is a full-fledged column-store engine. Every relational table is represented

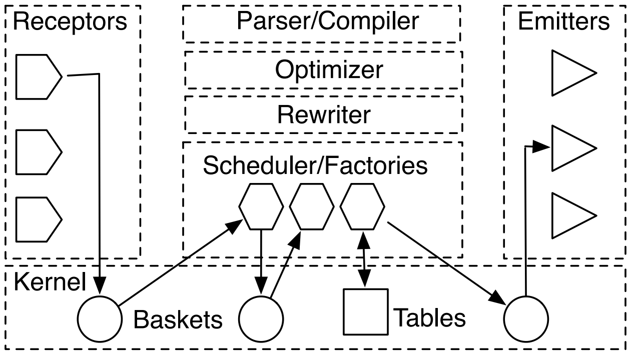


Figure 1: The DataCell Architecture

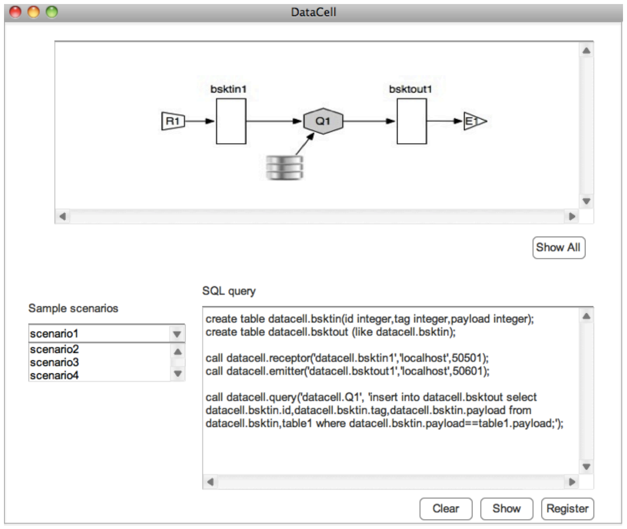
as a collection of *Binary Association Tables* (*BATs*), one for each attribute. Advanced column-stores process one column at a time, using *late* tuple reconstruction, discussed in, e.g., [1, 15]. Intermediates are also in column format. This allows the engine to exploit CPU- and cache-optimized vector-like operator implementations throughout the whole query evaluation, using an efficient bulk processing model instead of the typical tuple-at-a-time volcano approach. This way, a select operator for example, operates on a single column, filtering the qualifying values and producing an intermediate that holds their tuple IDs. This intermediate can then be used to retrieve the necessary values from a different column for further actions, e.g., aggregations, further filtering, etc. The key point is that in DataCell these intermediates can be exploited for flexible incremental processing strategies, i.e., we can selectively keep around the proper intermediates at the proper places of a plan for efficient future reuse.

DataCell. DataCell [16] is positioned between the SQL compiler/optimizer and the DBMS kernel. The SQL compiler is extended with a few orthogonal language constructs to recognize and process continuous queries. The query plan as generated by the SQL optimizer is rewritten to a continuous query plan and handed over to the DataCell scheduler. In turn, the scheduler handles the execution of the plan.

Figure 1 shows a DataCell instance. It contains *receptors* and *emitters*, i.e., a set of separate processes per stream and per client, respectively, to listen for new data and to deliver results. They form the edges of the architecture and the bridges to the outside world, e.g., to sensor drivers.

Baskets/Columns. The key idea is that when an event stream enters the system via a receptor, stream tuples are immediately stored in a lightweight table, called *basket.* By collecting event tuples into baskets, DataCell can evaluate the continuous queries over the baskets as if they were normal one-time queries and thus it can reuse any kind of algorithm and optimization designed for a DBMS. Once a tuple has been seen by all relevant queries/operators, it is *dropped* from its basket.

Factories/Queries. Continuous query plans are represented by *factories*, i.e., a kind of co-routine, whose semantics are extended to align with table producing SQL functions. Each factory encloses a (partial) query plan and produces a partial result at each call. For this, a factory continuously reads data from the input baskets, evaluates its query plan and creates a result set, which it then places in its output baskets. The factory remains active as long as the continuous query remains in the system, and it is always



```
create table datacell.bsktin(id integer,tag integer,payload integer);
create table datacell.bsktout (like datacell.bsktin);

call datacell.receptor('datacell.bsktin1','localhost',50501);
call datacell.emitter('datacell.bsktout1','localhost',50601);

call datacell.query('datacell.Q1', 'insert into datacell.bsktout select
datacell.bsktin.id,datacell.bsktin.tag,datacell.bsktin.payload from
datacell.bsktin,table1 where datacell.bsktin.payload==table1.payload;');
```

Figure 2: Posing queries.

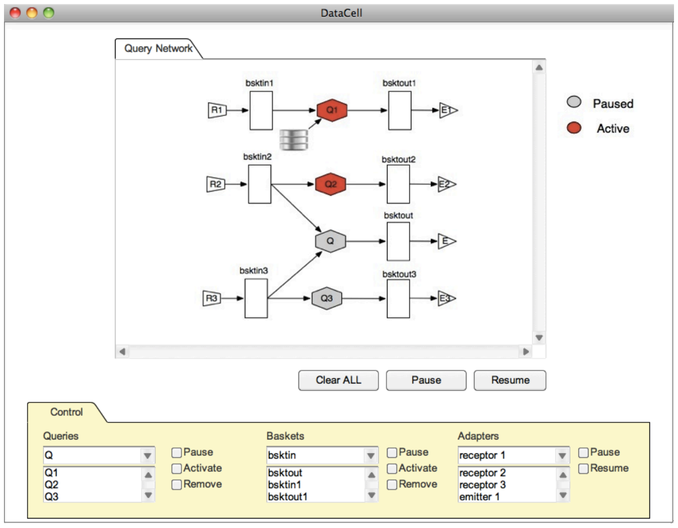


Figure 3: Monitoring and controlling the network of continuous queries.

alert to consume incoming stream data when they arrive.

Scheduler. The execution of the factories is orchestrated by the DataCell scheduler, which implements a Petri-net model [18]. The firing condition is aligned to arrival of events; once there are tuples that may be relevant to a waiting query, we trigger its evaluation. Furthermore, the scheduler manages the time constraints attached to event handling, which leads to possibly delaying events in their baskets for some time.

Two Query Paradigms. One important merit of the DataCell architecture, is the natural integration of baskets and tables within the same processing fabric. As we show in Figure 1, a single factory can interact both with tables and baskets. In this way, we can naturally support queries interweaving the basic components of both processing models. DataCell is shown to perform extremely well, easily meeting the requirements of the Linear Road Benchmark in [16], without also losing any database functionality.

Sliding Window Processing. Conceptually, DataCell achieves incremental processing by partitioning a window into n smaller parts, called *basic windows*. Each basic window is of equal size to the sliding step of the window and is processed separately. The resulting partial results are then *merged* to yield the complete window result. We design and develop the incremental logic at the query plan level, leaving the lower level intact and thus being able to reuse the complete storage and execution engine of a DBMS kernel. Proper optimizer rules, scheduling and intermediate result caching and reuse, allow us to modify the DBMS query plans for efficient incremental processing. In essence, query plans are split such as as many operators as possible can run independently on each portion of a sliding window stream. Then, when blocking operators occur, the plan merges intermediates from the active slides. A plan may split and merge multiple times depending on the query type.

4. DEMONSTRATION

In this section, we describe the demonstration of MonetDB/DataCell. The demonstration focuses on the main features of DataCell and on the specific scenarios that it enables. Below we describe the main demonstration scenarios

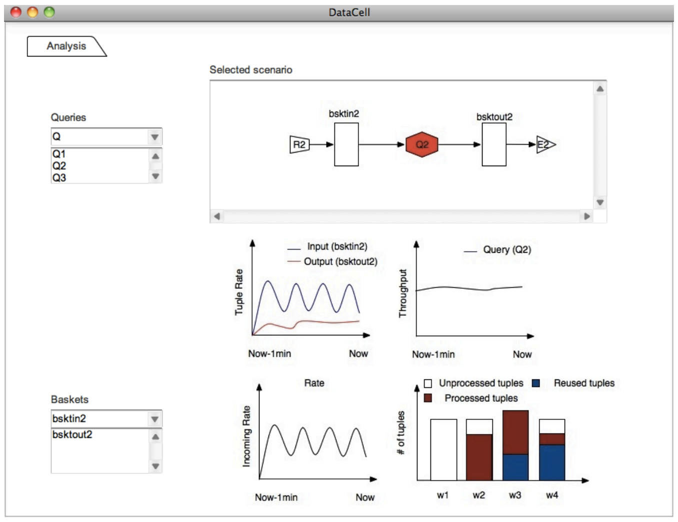


Figure 4: Analyzing performance.

and the main ways that the audience may interact with the system.

High-level Description. The main scenario involves one or more continuous queries waiting for incoming streams. As data arrives, queries are triggered by possibly combining and analyzing both stream data and persistent data. Figures 2, 3 and 4 depict part of the demo features.

Input Streams and Standing Queries. The audience is able to select from a predefined set of queries and data streams or alternatively insert new queries using the SQL extensions of MonetDB/DataCell. In addition, queries may be removed at any time. The predefined scenarios include the definition of several queries as well as the initialization of streams and the appropriate baskets. The demo includes various predefined data files which can be streamed in the system at rates which are configurable by the interface. Figure 2 shows some examples of such DataCell scenarios and how one can pose queries.

Query Network Characteristics. Once queries have been posed and the streams have been initialized, the user interface allows for visual inspection of the network of queries. For example, we can monitor which query waits for which stream, which baskets/columns it binds and how the various queries relate to each other regarding their input/output properties. Figure 3 shows some examples of such query networks in DataCell.

Pause and Resume. The audience may pause individual queries or even individual streams to inspect their status and see how the system instantly reacts to those changes. Resuming queries and/or streams is again possible at any time via the user interface. Figure 3 shows part of the control functionalities.

Detailed Status Inspection. The interface allows for a detailed inspection of the continuous query plan status, i.e., we can monitor where tuples live at any point in time, i.e., in which intermediate columns wait or which operators they feed. By pausing and resuming execution we can observe how tuples flow through the query network.

Simple Re-evaluation Scenarios. MonetDB/DataCell supports two modes of execution. The audience may trigger and experiment with both modes. With the first mode, queries are evaluated fully every time new relevant data arrive. In general, this is sufficient for non sliding window queries, i.e., for queries that do not include any window function or for simple tumbling window queries.

Sliding Window Processing. Using the second mode, called Incremental, MonetDB/DataCell enables partial and incremental computation of continuous queries. This is efficient for sliding window queries. The audience will be able to compare the two execution modes both in terms of elapsed time and in terms of investigating where the benefits of incremental processing come from. The user interface allows for continuous monitoring of inputs sizes and intermediate result sizes and consumption. The incremental mode continuously operates on smaller intermediates results by appropriately caching and reusing intermediates during sliding window queries.

Window Sizes. Users may define window sizes and step sizes for sliding window queries and visually observe how query plans and performance change with each change in those parameters. For different kinds of windows, the MonetDB/DataCell optimizer will produce different plans to properly divide the data and processing and to avoid recomputation as the windows slide.

Complex Queries. The audience will be able to see the difference that results from complex operators (e.g., joins) in continuous query plans with sliding windows as opposed to simple select project aggregation queries.

Analysis. An analysis pane allows for aggregation of performance metrics to observe elapsed time, incoming data rate for given baskets and other parameters over a period of time. Such parameters can be reported both for individual queries as well as for the complete query network. Figure 4 shows an example of the possible analysis tracking features.

5. CONCLUSIONS

MonetDB/DataCell shows that online continuous query processing can efficiently and elegantly be supported over an extensible DBMS kernel. These results open the road for scalable data processing that combines both stored and streaming data in an integrated environment in modern data warehouses. This is a topic with strong interest over the last few years and with a great potential impact on data management, in particular for business intelligence and science.

In this paper, we present a demonstration of MonetDB / DataCell, showcasing the main components and functionalities of the system such as the ability to perform online analytics, the combination of streaming data with persistent data and the ability for specialized stream functionalities such as sliding window query processing. The graphical interface of the demo provides both the means to monitor the system and the data flow inside the DataCell engine as well as the ability to interact and vary the scenarios tested.

6. REFERENCES

[1] D. Abadi et al. Materialization Strategies in a Column-Oriented DBMS. In *ICDE*, 2007.

[2] A. Arasu et al. STREAM: The Stanford Stream Data Manager. In *SIGMOD*, 2003.

[3] B. Babcock et al. Operator Scheduling in Data Stream Systems. *The VLDB Journal*, 13(4):333–353, 2004.

[4] H. Balakrishnan et al. Retrospective on Aurora. *The VLDB Journal*, 13(4):370–383, 2004.

[5] A. Biem et al. IBM infosphere streams for scalable, real-time, intelligent transportation services. In *SIGMOD*, 2010.

[6] J. A. Blakeley et al. Efficiently updating materialized views. In *SIGMOD*, 1986.

[7] S. Chandrasekaran et al. TelegraphCQ: Continuous Data- flow Processing for an Uncertain World. In *CIDR*, 2003.

[8] J. Chen et al. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *SIGMOD*, 2000.

[9] Q. Chen and M. Hsu. Experience in extending query engine for continuous analytics. In *DaWaK*, 2010.

[10] C. D. Cranor et al. Gigascope: A Stream Database for Network Applications. In *SIGMOD*, 2003.

[11] M. H. A. et al. Microsoft CEP Server and Online Behavioral Targeting. *PVLDB*, 2(2):1558–1561, 2009.

[12] M. J. Franklin et al. Continuous analytics: Rethinking query processing in a network-effect world. In *CIDR*, 2009.

[13] L. Girod et al. The Case for a Signal-Oriented Data Stream Management System. In *CIDR*, 2007.

[14] T. Griffin and L. Libkin. Incremental maintenance of views with duplicates. In *SIGMOD*, 1995.

[15] S. Idreos et al. Self-organizing Tuple-reconstruction in Column-stores. In *SIGMOD*, 2009.

[16] E. Liarou, R. Goncalves, and S. Idreos. Exploiting the power of relational databases for efficient stream processing. In *EDBT*, 2009.

[17] MonetDB. http://www.monetdb.com.

[18] J. L. Peterson. Petri nets. *ACM Comput. Surv.*, 9(3), 1977.

[19] PostgreSQL. http://www.postgresql.org/.

[20] U. Schreier et al. Alert: An Architecture for Transforming a Passive DBMS into an Active DBMS. In *VLDB*, 1991.

[21] M. A. Sharaf, P. K. Chrysanthis, A. Labrinidis, and K. Pruhs. Algorithms and metrics for processing multiple heterogeneous continuous queries. *ACM TODS*, 33(1), '08.