Enabling Deep Analytics in Stream Processing Systems

Milos Nikolic[1], Badrish Chandramouli[2], and Jonathan Goldstein[2]

[1] University of Oxford, milos.nikolic@cs.ox.ac.uk,
[2] Microsoft Research, badrishc@microsoft.com, jongold@microsoft.com

Abstract. Real-time applications often analyze data coming from sensor networks using relational and domain-specific operations such as signal processing and machine learning algorithms. To support such increasingly important scenarios, many data management systems integrate with numerical frameworks like R. Such solutions, however, incur significant performance penalties as relational engines and numerical tools operate on fundamentally different data models with expensive inter-communication mechanisms. In addition, none of these solutions supports efficient real-time and incremental analysis. In this work, we advocate a deep integration of domain-specific operations into general-purpose query processors with the goal of providing unified query and data models for both online and offline processing. Our proof-of-concept system tightly integrates relational and digital signal processing operations and achieves orders of magnitude better performance than existing loosely-coupled data management systems.

1 Introduction

Increasingly many applications analyze data that comes from large networks of sensor devices, commonly known as the Internet of Things (IoT). Typical IoT applications combine relational and domain-specific operations in processing sensor signals: for example, the former group signals by different sources or join signals with historical data; the latter use Fast Fourier Transform (FFT) to do spectral analysis, digital filters to recover noisy signals, or online machine learning to classify signal values. Reconciling these two seemingly disparate worlds, especially for real-time analysis, is challenging.

The database community has recognized the need for a tighter integration of data management systems and domain-specific algorithms. Numerical computing environments such as MATLAB and R provide efficient domain-specific algorithms but remain unsuitable for general-purpose data processing. In order to enable specialized routines in complex workflows, many data management systems nowadays integrate with numerical frameworks, R in particular: Spark [4] and SciDB [1] provide R packages; SQL Server, MonetDB, and StreamBase support queries that can invoke R code.

However, the existing integration mechanisms between database systems and numerical frameworks are suboptimal performance-wise as they treat both sides as independent systems. Such *loose system coupling* comes with significant processing overheads – for instance, executing R programs requires exporting data from the database, converting into R format, running R scripts, converting back into a relational format, and importing into the database. Sending data back and forth between the systems often dominates the execution time and increases latency, which makes this approach particularly unsuitable for real-time processing.

2 Domain-specific Computation in Stream Processing Engines

We advocate a *deep integration* of domain-specific operations with general-purpose stream processors. This approach aims to bring specialized operations closer to data and eliminate the need for expensive communication with external numerical tools. Enabling in-situ domain-specific computation in stream engines empowers users to express end-to-end workflows more succinctly, inside one system and using one language.

This tight integration poses several requirements and challenges:

1. *Query and data model reconciliation.* General-purpose stream processing engines and numerical tools use different query and data models. The former support relational and streaming queries over relational or tempo-relational data; the latter support domain-specific, mostly offline, computations on arrays. The key challenge is how to seamlessly unify these disparate models instead of simulating them on top of each other, and yet provide data practitioners and domain experts with familiar abstractions.

2. *Extensibility.* A query processor supporting deep analytics should allow domain experts to implement custom operators in a way that feels natural to them – by writing algorithms against arrays without worrying about the format of the underlying data. Exposing arrays to operator writers would enable easy integration of existing highly optimized algorithms, for instance, implementations using SIMD instructions.

3. *Online and incremental computation.* Stream processors loosely coupled with MATLAB or R cannot incrementalize domain-specific tasks that operate over hopping (overlapping) windows of data. The stateless nature of routines in MATLAB and R forces stream engines to redundantly recompute over overlapping subsets of data. On the other hand, deep integration admits more efficient data management (e.g., via fixed-size circular arrays) and incremental computation in user-defined stateful operators.

4. *Performance.* High performance is always a critical requirement for analytics. The deep integration approach can bring more expressiveness to the query language but carries a risk of completely giving up on performance. In order to be welcomed by data scientists and domain experts, a deeply integrated system should preserve the performance of existing relational operators while being competitive with best-of-breed systems for domain-specific tasks. For mixed workflows, a tightly-coupled system should exhibit better performance than existing loosely-coupled alternatives.

3 Proof of Concept: Signal Processing over Data Streams

Our recent work presents a system that deeply integrates relational and digital signal processing (DSP) while satisfying the above requirements [3]. The system, called TRILLDSP, is built on top of Trill [2], a high-performance incremental analytics engine that supports processing of streaming and relational queries.

Trill uses a tempo-relational data model to uniformly represent offline and online datasets as stream data. Each stream event is associated with a time window that denotes its period of validity. Such stream events form snapshots of valid data versions across time. The user query is executed against these snapshots in an incremental manner. Trill's query language provides standard relational operators (e.g., selection, projection, join, etc.) with temporal interpretation. Each operator is a function from stream to stream, which allows for elegant functional composition of queries.

TRILLDSP provides a unified query language for processing tempo-relational and signal data. We next show a query that combines these seemingly disparate worlds.

Example 1. An IoT application receives a stream of temperature readings from different sensors in time order. Each reading has the same format `<SensorId,Time,Value>`. The application runs the same query on every signal coming from a different source.

```
var q = stream.Map(s => s.Select(e => e.Value), e => e.SensorId)
              .Reduce(s => s.Window(512, 256,
                w => w.FFT().Select(a => f(a)).InverseFFT(), a => a.Sum()));
```

The query combines relational and DSP operators to express several processing phases: 1) *Grouping*: `Map` specifies a sub-query (projection with `Select`) to be performed in parallel on the stream and a grouping key (`SensorId`) to be used for shuffling the result streams; `Reduce` specifies the query to be executed per each group; 2) *Windowing*: Most DSP algorithms operate over windows of data defined by window size and hop size. The `Window` operator allows users to express such algorithms as a series of transformations of fixed-size arrays (`w`); 3) *Spectral Analysis*: The processing pipeline starts with a Fast Fourier Transform (FFT) that computes the frequency representation of a 512-sample window at each hopping point. A user-defined function f modifies the computed spectrum (e.g., retains top-k spectrum values and zeros out others) before invoking an inverse FFT; 4) *Unwindowing*: To restore the signal form, the final phase projects output arrays back to the time axis and sums up the overlapping values.

This query captures the fundamental technique of windowing in DSP and serves as a blueprint for a large class of IoT workflows. The declarative query model of TRILLDSP offers high-level operators capable of expressing such deep analytics succinctly. □

We design and implement the DSP functionality in TRILLDSP as a layer running on top of the unmodified Trill relational engine. The layer enriches the query model with a "walled garden" that provides signal abstractions and clear transitions between stream and signal operations. TRILLDSP makes no changes in the underlying tempo-relation data model (i.e., has no arrays as first-class citizens). Instead, it exposes arrays only through designated DSP operators and abstracts away the complexity of on-the-fly transformations between the relational and array models. In that way, the existing relational API and the performance of non-DSP queries remain unaffected.

For DSP experts, TRILLDSP provides an extensible framework for defining new user-defined window operations (e.g., variants of FFT, windowing functions, correlation functions, element-wise product, etc.), which can be seamlessly interleaved with relational operators. The framework internally exposes array abstractions to ease the integration of new black-box DSP operators. It also allows users to implement incremental versions of operators to be used in computation with hopping windows.

The unified query model of TRILLDSP supports both online and offline analysis. As offline datasets are streams consisting of events with an infinite lifetime, each tempo-relational operator is transferable between real-time and offline by construction. Thus, users can build queries from offline data and then put them unmodified in production.

Deep integration of signal and relational processing is key to achieving high performance. In pure DSP tasks, TRILLDSP is competitive or even faster than state-of-the-art numerical frameworks like MATLAB and R. This comes at no surprise as this

framework exposes array abstractions to operator writers for easy adoption of highly-optimized black-box implementations of DSP operations (e.g., those exploiting SIMD instructions). TRILLDSP shows its full potential when processing data coming from a large number of sensor devices using the query logic that combines relational and signal operation. All signal operations natively support grouped processing, that is, one operator can simultaneously process multiple groups by maintaining the state of each group. Coupling these operators with Trill's streaming temporal MapReduce operator enables efficient grouped signal processing. TRILLDSP's in-situ execution model achieves up to two orders of magnitude better performance than modern relational and array data management systems with loose R integration, such as SPARKR [4] and SCIDB-R [1].

4 Future Work

This proof-of-concept system [3] demonstrates that one can extend general-purpose query processors with domain-specific functionality and get the best of both worlds: a more powerful, declarative query language and high performance of both general-purpose and domain-specific operations. This result opens up the question whether the "one-size-doesn't-fit-all" paradigm from database systems also holds for streaming systems because of their dynamic nature that allows them to adapt data on-the-fly to meet different processing requirements.

The "walled garden" approach to enabling deep analytics over data streams can also be applied in domains other than signal processing, such as machine learning, scientific computing, computer vision, physical modeling, computational finance, etc. The window-based query template shown in the above example can support arbitrary computations over arrays: for instance, applications analyzing accelerometer data from wearable devices can build pipelines of machine learning operators to classify sequences of signal samples into classes of physical activities (e.g., walking, running, sitting, standing, etc.); algorithmic trading applications can implement trading strategies using pipeline operators over windows of stock prices, which can be possibly correlated with historical data. In general, enhancing relational engines with powerful capabilities from various domains enables data processing to be placed closer to data sources (i.e., towards network edges), which soon will be essential in order to cope with growing amounts of sensor data and ensure scalability of IoT applications.

References

1. P. G. Brown. Overview of SciDB: Large Scale Array Storage, Processing and Analysis. In *SIGMOD*, pages 963–968, 2010.
2. B. Chandramouli, J. Goldstein, M. Barnett, R. DeLine, J. C. Platt, J. F. Terwilliger, and J. Wernsing. Trill: A High-Performance Incremental Query Processor for Diverse Analytics. *PVLDB*, 8(4):401–412, 2014.
3. M. Nikolic, B. Chandramouli, and J. Goldstein. Enabling Signal Processing over Data Streams. In *SIGMOD*, 2017.
4. M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI*, pages 15–28, 2012.