Enabling Signal Processing over Stream Data

Milos Nikolic*, University of Oxford
Badrish Chandramouli, Microsoft Research
Jonathan Goldstein, Microsoft Research

*Work performed during internship at MSR
Signals in Streams

- Lots of “signals” in stream data
  - Internet-of-things devices, app telemetry (e.g., ad clicks)

- IoT workflows combine relational & signal logic
  - Ex: Real-time app

<table>
<thead>
<tr>
<th>ID</th>
<th>Time</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0:42:19</td>
<td>67</td>
</tr>
<tr>
<td>1</td>
<td>0:42:22</td>
<td>80</td>
</tr>
<tr>
<td>2</td>
<td>0:42:22</td>
<td>85</td>
</tr>
</tbody>
</table>

Which tools to use to build such apps?
How to reconcile two worlds?

**Data processing expert**

- **Engines**: stream engines, DBMS, MPP systems
- **Data model**: (tempo)-relational
- **Language**: declarative (SQL, LINQ, functional)
- **Scenarios**: real-time, offline, progressive

**Digital signal processing expert**

- **Engines**: MATLAB, R
- **Data model**: array
- **Language**: imperative (array languages, C)
- **Scenarios**: mostly offline, real-time

**Our solution**:  
- high-performance (2 OOM faster)  
- one query language  
- familiar abstractions to both worlds
Equally-spaced samples stored in array

1. **Window**
   - window size & hop size

2. **Per window: pipeline DSP ops**
   - array to array
   - Example: spectral analysis
     - FFT → user-defined function → IFFT

3. **Unwindow**
   - sum overlapping segments
Loose Systems Integration

Stream Processing Engine + R

- Stream engine for relational queries
  - Per-group computation, windowing, joins, etc.
- R for highly-optimized DSP operations
- Problem: impedance mismatch
  - High communication overhead (up to 95%)
  - Impractical for real-time analysis
  - Disparate query languages
Trill: Fast Streaming Analytics Engine

- **Performance**
  - 2-4 OOM faster than today’s SPE

- **Query model**
  - Based on temporal query model (relational with time)
  - Real-time, offline, progressive queries

- **Language integration**
  - Built as .NET library
  - Works with arbitrary C# data-types

[VLDB 2014 paper]

DSP Library

- **Unified query model**
  - Non-uniform & uniform signals
  - Type-safe mix of stream & signal operators

- **Array-based extensibility framework**
  - DSP operator writer sees arrays
  - Supports incremental computation

- **“Walled garden” on top of Trill**
  - No changes in data model
  - Inherits Trill’s efficient processing capability (e.g., grouped computation)
Tempo-Relational Model

- Uniformly represents offline and online datasets as stream data

**Relational Model**

**INPUT**
- t1
- t2
- t3
- t4

**Q = COUNT(*)**

**OUTPUT**
- 4

**Tempo-Relational Model**

**INPUT**
- e1
- e2
- e3
- e4
- e5

**OUTPUT**
- 4

Logical time

**snapshots**
Trill Example (Simplified)

• Define event data-type in C#

```csharp
struct SensorReading { long SensorId; long Time; double Value; }
```

• Define ingress

```csharp
var str = Network.ToStream(e => e.Time);
```

• Write query (in C# app)

```csharp
var query = str.Where(e => e.Value < 100) .Select(e => e.Value)
```

• Subscribe to result

```csharp
query.Subscribe(e => Console.Write(e)); // write results to console
```
Signal = stream w/o overlapping events

• Transition to signal domain
  • E.g., result of an aggregate query
    ```
    var signal = stream.Where(e => e.Value < 100).Count()
    ```

• Using stream operators to build signal operators
  • E.g., adding two signals as a temporal join of two streams
    ```
    left.Join(right, (l, r) => l + r)
    ```

Type-safe operations
Uniformly-sampled signals

• Sampling with interpolation

```javascript
var uniformSignal = signal.Sample(30, 0, ip => ip.Linear(60));
```

Period, offset, interpolation policy

Interpolation window
Bringing Array Abstractions to DSP Users

• **Initial idea: Window & Unwindow sample operators**
  - Window() creates a *stream of arrays*
    
    ```
    var s = uniformSignal.Window(5,3).FFT(...)
    ```
  - Unwindow() projects arrays back in time

• **Performance problems**
  - Creates dependencies between window semantics and system performance
  - No data sharing across overlapping arrays

• **Unclear language semantics**
  - e.g., stream of arrays: is it a signal or not?
Windowing Operator for DSP Users

• Expose arrays **only inside** the windowing operator

```javascript
var query = uniformSignal
    .Window(512, 256,
        w => w.FFT().Select(a => f(a)).IFFT(),
        a => a.Sum())
```

• **DSP pipeline & arrays instantiated only once** → better data management
User-Defined Operator Framework

- **DSP experts write array-array operators**
  - Matches their expectations
  - Allows optimized array-based logic (e.g., SIMD)

- **Incremental DSP operators**
  - Framework uses circular arrays to avoid data copying with hopping windows
  - New & old data available for incremental computation
Grouped Computation

- **Group-aware operators**
  - Online processing of intertwined signals
  - One state per each group
    - E.g., interpolator keeps a history of samples for each group

- **Streaming MapReduce in Trill**
  - Parallel execution on each sub-stream corresponding to a distinct grouping key

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

Performance: FFT with tumbling window

Window → FFT → Unwindow

RUNNING TIME (secs)

Pre-loaded datasets in memory

Pure DSP task
  • TrillDSP uses FFTW library

Comparable to best DSP tools
Performance: Grouping + DSP

Per sensor: Windowed FFT → Function → Inverse FFT → Unwindow

NORMALIZED TIME TO TRILLDSP ON 16 CORES

Pre-loaded datasets in memory
• 100 groups in stream

Up to 2 OOM faster than others

Performance benefits from:
• Efficient group processing, group-aware DSP windowing
• Using circular arrays to manage overlapping windows
• TrillDSP uses FFTW library
Conclusion

• **Apps mix relational & signal logic**
  - Per device: find periodicity in signals, interpolate missing data, recover noisy data
  - Different data models: relational vs. array

• **Existing query processors integrated with R**
  - Impedance mismatch $\rightarrow$ high performance overhead $\rightarrow$ not suitable for real-time

• **TrillDSP = Relational processing + Signal processing**
  - Unified query model for relational and signal data, for both real-time and offline
  - Gives users the view they are comfortable with
  - Avoids impedance mismatch between components
  
  ![Graph](image.png)

**Up to 2 OOM faster than systems integrated w/ R**