An Extensible Test Framework for Microsoft StreamInsight

- Alex Raizman
- Asvin Ananthanarayan
- Anton Kirilov
- Badrish Chandramouli
- Mohamed Ali
Agenda

• Introduction to Microsoft StreamInsight
• Testing Challenges
• End to End test framework
  – Intent based testing (Declarative testing)
  – Event generation
  – Functional Verification
• Case study – Top K
• Conclusions
What is Stream Processing?

Stream Processing is the continuous and incremental processing of event streams from multiple sources based on declarative query and pattern specifications with near-zero latency.

<table>
<thead>
<tr>
<th></th>
<th>RELATIONAL</th>
<th>STREAMING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Paradigm</td>
<td>Ad-hoc queries or requests</td>
<td>Continuous standing queries</td>
</tr>
<tr>
<td>Latency</td>
<td>Seconds, hours, days</td>
<td>Milliseconds or less</td>
</tr>
<tr>
<td>Data Rate</td>
<td>Hundreds of events/sec</td>
<td>Tens of thousands of events/sec or more</td>
</tr>
<tr>
<td>Temporal semantics</td>
<td>Optional</td>
<td>Integral to streaming queries</td>
</tr>
</tbody>
</table>
Events and Event Streams

• An event is a collection of fields
• Streaming engine provisioned timestamp fields capture all the different temporal event characteristics
• Event sources populate timestamp fields

<table>
<thead>
<tr>
<th>Start Time</th>
<th>End Time</th>
<th>Long pumpID</th>
<th>String Location</th>
<th>Double flow</th>
<th>Double pressure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

• A stream is a sequence of events
  – Possibly infinite
• Stream characteristics:
  – Event/data arrival patterns (steady, bursty)
  – Out of order events: Order of arrival of events does not match the order of their application timestamps
  – Events with varying lifetimes
Relational Semantics Versus Streaming Semantics

**Relational**
- A join B on A.C1 = B.C1

<table>
<thead>
<tr>
<th>C1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>

**Streaming**
- A join B on A.C1 = B.C1

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>C1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>30</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>40</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>40</td>
<td>50</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>A.C1</th>
<th>A.C2</th>
<th>B.C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>25</td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>40</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>38</td>
<td>4</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

**Top 2 rows ORDER BY C1 DESC**
- Out of order event

<table>
<thead>
<tr>
<th>C1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>40</td>
<td>50</td>
</tr>
</tbody>
</table>

**Top 2 rows order by C1 DESC on a time window of size 10**
- Predicate matches but events do not overlap in time

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>A.C1</th>
<th>A.C2</th>
<th>B.C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
<td>15</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>15</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>C1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
</tr>
</tbody>
</table>

Out of order event
StreamInsight Query Examples

Example – JOIN, PROJECT, FILTER:

```csharp
from e1 in MyStream1
join e2 in MyStream2
on e1.ID equals e2.ID
where e1.f2 == "foo"
select new { e1.f1, e2.f4 };
```

Example – GROUP&APPLY, WINDOW:

```csharp
from e3 in MyStream3
group e3 by e3.i into SubStream
from win in SubStream.HoppingWindow(
    TenSeconds)
order by win.f).Take(10);
```
Testing Challenges

• Enable reuse of tests for different product languages
  – Multiple entry points to the query engine

• Event generation
  – Temporal characteristic.
  – More than payload generation

• Automated functional validation.
  – We have become very good at generating tests, but programmatic verification is still a challenge
Test Framework

Test reuse and Intent based testing

Intent tree (declarative representation)

Concrete tree (declarative representation)

StreamInsight Query Language Transformer

Stream Event Generator

SQL-equivalent Query Language Transformer

Stream Insight

LINQ Query

XML Query

Stream events

Event generation

SQL Server

SQL constructs

Automated functional validation

Output stream

Property Validator

Output Stream Canonical Results Verifier
Intent based testing

- Top K events on a window with an descending rank expression
  - Must have an operator as input to Top K
  - Must have a rank expression.

- Builds an intent tree.

```csharp
var rank = input.CreateExpression();

var topK = input
    .Window()
    .OrderBy(rank.Desc())
    .Top(K);
```
Intent tree to Concrete tree

- Building out the rest of the tree.
  - Finalize tree by binding to stream
  - Extend tree with another operator
  - Run model to generate subtree

- Repository of event types

- Expression builder

- Concrete tree = test

- Test suite = Multiple concrete trees
Stream Event Generator

• Two components
  – Temporal generation
  – Payload generation

• Supported temporal facets
  – event count
  – out of order
  – overlap
  – maximum duration of event
  – boundary cases
  – repository of pre-defined patterns

• Payload generation
  – leveraged existing payload generators

• Tester specifies as many facets or as little as they want.
Query Language Transformer

- Concrete tree to XML transform generates XML language for the test
- Concrete tree to LINQ transform generates C# code for the same test

```csharp
public static class QueryBuilder
{
    public static object CreateStream()
    {
        CepStream<integral> stream1 = CepStream<integral>.Create("Import");
        return (from window1 in stream1.HoppingWindow(
            new TimeSpan(1000000000),
            new TimeSpan(1000000000),
            HoppingWindowOutputPolicy.ClipToWindowEnd
            )
            from event1 in window1
            orderby event1.tulong_integral descending
            select event1).Take(10);
    }
}

public class integral
{
    public int tint_integral { get; set; }
    public long? tlong_integral { get; set; }
    public short? tsshort_integral { get; set; }
    public uint tuint_integral { get; set; }
    public ulong? tulong_integral { get; set; }
    public ushort tushort_integral { get; set; }
}
```
Test Framework

Intent Transformer

Intent tree (declarative representation)

Concrete tree (declarative representation)

StreamInsight Query Language Transformer

Stream Event Generator

SQL-equivalent Query Language Transformer

LINQ Query

XML Query

Stream events

SQL constructs

SQL Server

Automated functional validation

Stream Insight

Output stream Property Validator

Output Stream Canonical Results Verifier
Functional validation using SQL

**Diagram:**
- **Event Stream**
  - EventType = integral
  - Import.stream_data
  - REPOSITORY
  - Count
  - MaxDuration
  - Cti
  - Event Order

**Code:**
```csharp

CREATE VIEW [QI_TopK_stream_4]
AS SELECT
CAExpr.EventType,
CAExpr.ValidStart as ValidStart,
ValidEnd = CASE
WHEN D.endSqlTime > CAExpr.ValidEnd THEN CAExpr.ValidEnd
WHEN D.endSqlTime < CAExpr.ValidEnd THEN D.endSqlTime
ELSE D.endSqlTime
END, NewValidEnd -
CASE
WHEN D.endSqlTime > CAExpr.ValidEnd THEN CAExpr.ValidEnd
WHEN D.endSqlTime < CAExpr.ValidEnd THEN D.endSqlTime
ELSE D.endSqlTime
END, tint_integral, tlong_integral, tshort_integral,
tuint_integral, tulong_integral, tshort_integral FROM [HoppingWindow]()
AS D
CROSS APPLY

SELECT TOP (10) WITH TIES
EventType,
startSqlTime AS ValidStart,
endSqlTime AS ValidEnd,
endSqlTime AS NewValidEnd,
tint_integral, tlong_integral, tshort_integral, tuint_integral,
tulong_integral, tshort_integral
FROM [Import_stream] WHERE ValidStart < endSqlTime AND ValidEnd > startSqlTime
ORDER BY
[RankExpr]([Import_stream].tulong_integral).Data DESC
AS CAExpr
```
Functional validation using SQL

- SQL input table is populated with the event stream from the generator
- Equivalent SQL Queries are run over the input table
- Output of the StreamInsight query is also piped into SQL Server using output adapter
- Compares SQL output with output from StreamInsight for equality.
- If comparison fails → BUG!
Conclusions

• Intent based testing allows testers to think about test scenarios leading to richer, more interesting tests.

• A relational database can work as a test oracle for validating streaming queries.

• Most of the bugs were due to interesting temporal aspects of the event stream.

• Not everything can / should be tested using intent based techniques.
  – Negative tests, boundary cases
QUESTIONS ?
For More Information


• StreamInsight blog: http://blogs.msdn.com/streaminsight/


• StreamInsight E-clinics on Microsoft e-learning https://www.microsoftotelearning.com/eLearning