SECRET: A Model for Analyzing the Execution Semantics of Stream Processing Engines

Nesime Tatbul
ETH Zurich
Stream Processing: A Decade Ago
[Aurora, VLDB’02]

- Monitoring applications require collecting, processing, disseminating, and reacting to real-time events from push-based data sources.
- “Store and Pull” model of traditional databases does not work well.

Traditional Database Systems

Stream Processing Engines
Stream Processing: Today

- Tens of commercial products
- Countless academic prototypes
- Open-source distributed platforms and Hadoop extensions (e.g., Yahoo! S4, Twitter Storm, HStreaming)
- Streams ⊂ Big Data

Integrated Stream Processing (ISP)

• Integrating information from multiple, heterogeneous data sources has both been a key enabler and a major challenge in the DB community over the last 20 years.

• Today, similar integration support for SPEs is needed in three main forms:
  1. across multiple streaming data sources
  2. between SPEs and traditional DBMSs
  3. over multiple SPEs
#1: Streaming Data Source Integration

- **Goal:** Integrated querying over multiple, potentially heterogeneous streaming data sources
- **Example:** Route planning based on information from news feeds, weather sensors, traffic cameras, etc.
- **Main challenges:**
  - Schemas of different sources can differ from one another and from the input schemas of the already running continuous queries (CQs).
  - Input sources/the network can introduce imperfections into the stream.
  - Adapters may become a bottleneck.
- **Current state of the art:**
  - Commercial SPEs: adapters + SDKs
  - Research: Mapping Data to Queries [Hentschel et al.], ASPEN [Ives et al.]
#2: SPE-DBMS Integration

- **Goal:** Integrated querying over SPEs and traditional database and data warehousing systems
- **Example:** Operational business intelligence, Continuous analytics, Stream warehousing
- **Main challenges:**
  - Bridge the “data vs. operation” gap between the two worlds.
  - Find the right language and architecture primitives for the required level of querying, persistence, and performance.
- **Current state of the art:**
  - Languages [STREAM CQL, StreamSQL, MATCH-RECOGNIZE]
  - Architectures [SPE-based (e.g., StreamBase);
    DBMS-based (e.g., Truviso, DejaVu, MaxStream)]
#3: SPE-SPE Integration

• **Goal:** Integrated querying over multiple, potentially heterogeneous SPEs
  – to exploit the advantages of distributed operation
  – to exploit specialized capabilities and strengths of SPEs
  – to provide higher-level monitoring over large-scale enterprises with loosely-coupled operational units

• **Example:** Supply-chain management

• **Main challenges:**
  – the need for functional integration
  – the need to deal with heterogeneity at different levels (e.g., query models, capabilities, performance)

• **Current state of the art:**
Overview of our ISP Research

• **SPE-DBMS integration**
  – MaxStream: a federation middleware for ISP
    [BIRTE’09, ICDE’10, NTII’10]
  – DejaVu: declarative pattern matching in ISP systems
    [SIGMOD’09, Pervasive’09, DEBS’11]
  – SMS: storage and transaction management techniques for ISP
    [EDBT’09, EDBT’12]

• **SPE-SPE integration**
  – MaxStream
  – **SECRET**: a model to describe SPE query execution semantics
    [VLDB’10, VLDBJ’12]
  – ExoEngine: an architecture for virtualizing SPEs
    [Middleware’11]
Problem: Heterogeneity of SPEs

- SPEs have differences in their query models:
  - Syntax heterogeneity:
    - Language clauses/keywords for common query constructs syntactically differ.
  - Capability heterogeneity:
    - Support for certain query types differs.
  - Execution model heterogeneity:
    - Underlying query execution models differ.
    - Not exposed to the application developer via language syntax.

- So, what?
  - Difficult to build, debug, port, integrate applications.

Syntax for time-based sliding windows:
- StreamBase: [SIZE x ADVANCE y TIME]
- Coral8: KEEP x SECONDS

Support for flexible slide:
- StreamBase allows an arbitrary value for y.
- Coral8 uses y=1msec by deafult.

You may get different results for:
- StreamBase: [SIZE x ADVANCE 1 TIME]
- Coral8: KEEP x SECONDS
Example #1: StreamBase vs. Coral8

• Input Stream: InStream(Time, Val)
  \{(30, 10), (31, 20), (36, 30), \ldots\}

• Continuous Query:
  “Compute average value of the tuples for the last 5 seconds.”

• Result Stream: OutStream(Val)
  – StreamBase: {(10), (15), (15), (15), (15), (20), (30), \ldots}\}
  – Coral8 : {(10), (15), (20), (30), \ldots\}

WHY?
Our Solution

• Goal:
  – To describe, predict, and compare the basic query execution behaviors of diverse SPEs (i.e., focus on execution model heterogeneity)

• Approach:
  – A formal model based on experimenting with real systems
  – Design principles: expressive, simple, orthogonal, extensible

• Scope:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>in-order streams (gaps, simultaneity)</td>
<td></td>
</tr>
<tr>
<td>Queries</td>
<td>time-based windows</td>
<td>+ tuple-based windows</td>
</tr>
<tr>
<td>Systems</td>
<td>Coral8, STREAM, StreamBase</td>
<td>+ Oracle CEP</td>
</tr>
</tbody>
</table>
SECRET Model Overview

• What affects query results produced by an SPE?
  Query Result = F(system, query, input)

• Tick -> Report -> Content -> Scope
Time & Order in SECRET

• Two notions of time:
  – $t^{app}$: application time of a tuple (t in SECRET)
  – $t^{sys}$: system time (arrival time) of a tuple (τ in SECRET)

• Order assumptions:
  – Tuples are partially ordered by their $t^{app}$ values.
  – Tuples are totally ordered by their $t^{sys}$ values.
  – Batch-id (bid) defines a further ordering among simultaneous tuples (i.e., tuples having the same $t^{app}$) [Jain et al.].
    ➢ Tuples are partially ordered by their bid values.
• Scope is based on
  – window specification (size (ω) and slide (β))
  – start of the first window (t₀)
• Scope(t) defines the interval of “active window” at application time t.
  – Active window at time t is the open window with the earliest start time.
Content in SECRET

• Content(t, τ) specifies the set of input tuples that are in Scope(t) as of system time τ.

• It can return different values at t, depending on the arrival of tuples (due to τ).
REport in SECRET

• Report(t, τ) defines the conditions under which the window contents become visible for further query evaluation and result reporting.

• It can take a logical combination of the following:
  – content change
  – window close
  – non-empty content
  – periodic
Example #1 with SECRET

- Input: \( \text{InStream}(\text{Time, Val}) = \{(30, 10), (31, 20), (36, 30), \ldots\} \)
- Query: “Compute average value of the tuples for the last 5 sec.”
- Result: \( \text{OutStream}(\text{Val}) \)
  - \( \text{StreamBase (window close&non-empty)} = \{(10), (15), (15), (15), (20), (30), \ldots\} \)
  - \( \text{Coral8 (content change&non-empty)} = \{(10), (15), (20),(30), \ldots\} \)
Tick in SECRET

• Tick(\(\tau\)) defines the condition which drives an SPE to take action on its input.

• It can be based on one of the following:
  – tuple-driven: react to individual tuples
  – time-driven: react to all tuples with the same \(t^{\text{app}}\) value
  – batch-driven: react to subsets of tuples with the same \(t^{\text{app}}\) value [Jain et al.]

• Note: tuple-driven and time-driven are in fact special cases of batch-driven.
Example #2: Coral8 - Difference in Tick

- Input: InStream(Time, Val) = {(3, 10), (5, 20), (5, 30), (5, 40), (5, 50), ...}
- Query: “Compute sum of the values for the last 4 seconds.”
- Result: OutStream(Val)
  - Coral8 (tuple-driven) = {(10), (30), (60), (100), (150), ...}
  - Coral8 (time-driven) = {(10), (150), ...}
Extending SECRET
From Time-based to Tuple-based

• Windowing domain is different.
  – tid: tuple-id of a tuple (i in SECRET)
  – Tuples are totally ordered by their tid values.
  – Windows (size and slide) are defined in terms of tid’s instead of t^{app}’s.

• Scope and Content:
  – t -> i and t_{0} -> i_{0}
  – t is not unique -> i is unique

• Tick:
  – maps from t^{sys} to t^{app} -> maps from t^{sys} to tid
  – no gaps nor a notion of simultaneity in the tid domain => simpler to formulate

• Report:
  – Content-change and non-empty-content always true => simpler to formulate
  – **Tick may invoke Report with multiple tid values** (when: time- or batch-driven tick + simultaneous input) => One of those that satisfy the Report condition is chosen (“evaporating tuples” [Jain et al.])
Example #3: Tuple-based Windows

- Input: \( \text{InStream}(\text{Time}, \text{Val}) = \{(1, 10), (2, 20), (2, 30), (2, 40), (2, 50), (4, 60), \ldots\} \)
- Query: “Compute sum of the values for a tuple-based tumbling window of size 2.”
- Result for an SPE \( S(\text{Tick}=\text{time-driven}, \text{Report}=\text{window-close}) \):
  - \( \text{OutStream}(\text{Val}) = \{(70), (110), \ldots\} \)
## SPEs in SECRET

<table>
<thead>
<tr>
<th>SPE</th>
<th>$t_0$, $i_0$</th>
<th>Report</th>
<th>Tick</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coral8</td>
<td>$\left[ \frac{tt_1 - \omega}{\beta} \right] \beta - 1, 0$</td>
<td>content change window close non-empty content periodic</td>
<td>tuple-driven time-driven batch-driven</td>
</tr>
<tr>
<td>Oracle CEP</td>
<td>$\left[ \frac{tt_1}{\beta} \right] \beta - \omega, \beta - \omega$</td>
<td>content change window close non-empty content periodic</td>
<td>tuple-driven time-driven batch-driven</td>
</tr>
<tr>
<td>STREAM</td>
<td>$tt_1 - \omega, \beta - \omega$</td>
<td>content change window close non-empty content periodic</td>
<td>tuple-driven time-driven batch-driven</td>
</tr>
<tr>
<td>StreamBase</td>
<td>$\left[ \frac{tt_1 - \omega}{\beta} \right] \beta - 1, 0$</td>
<td>content change window close non-empty content periodic</td>
<td>tuple-driven time-driven batch-driven</td>
</tr>
</tbody>
</table>
Summary

• SECRET uncovers the execution model heterogeneity of SPEs, paving the way for:
  – tools for building, debugging, and optimizing streaming applications on a given SPE
  – tools for porting streaming applications across SPEs
  – capability-based query processing across integrated SPEs
  – a formal basis for standardization

• SECRET is extensible to other types of inputs, queries, and SPEs.

• More information:
  – http://www.systems.ethz.ch/research/SECRET/
SECRET in Action (work in progress)
Query Equivalence for SPEs

• When is a query \( Q_1 \) under SPE query execution semantics \( S_1 \) is “equivalent” to another query \( Q_2 \) under SPE query execution semantics \( S_2 \)?

• Main challenges:
  – \( S_i \) expressed in SECRET => explore the SECRET design space
  – Equivalence definition?
  – Dependence to input knowledge
  – Completeness vs. Practicality
SECRET in Action (work in progress)
Example Use Case: Query Translation

Semantic Equivalence Checker
Rule Engine
Rule Base

Query Translator

Yes
Syntax Translator

<Yes, Q_d in S_d's dialect>

No
<No, NULL>
SECRET in Action (work in progress)
Example Use Case: Federated Query Processing

[Diagram of a federated query processing system with components labeled as Query Translator, Federator, Federator Client, SPE1, ..., SPEn.]
Future Directions
Streaming Data is part of Big Data

• Streaming Data is the Velocity dimension of Big Data
  – With stream processing, we can react to Big Data as it happens and tackle problems in a more incremental way.

• For ISP, we need to consider multiple dimensions together
  – streaming data source integration: Velocity + Variety
  – SPE-DBMS Integration: Velocity + Volume
  – SPE-SPE Integration: Velocity + “Variety”

• Lots of interesting challenges and opportunities
Managing Big Data in the Cloud
An Example Problem

Big Data:
Large amounts of data stored on the client

Big Processing Power:
High number of CPU cores available in the cloud

Bottleneck:
Limited network bandwidth between the client and the cloud
Managing Big Data in the Cloud
The “Stream As You Go” Approach [DMC’12, SSDBM’12]

• Key idea:
  – Data is streamed into the cloud.
  – Incremental processing starts right away.
  – Results are streamed back to the client as they become available.

• Benefits:
  – hides the data transfer latency, reducing the total round-trip time
  – enables in-memory processing, saving from data access time and cloud storage costs
  – enables pipelined parallelism (in addition to partitioned parallelism), leading to early results and shorter completion time

• Stream-as-you-go is a good fit for data- and compute-intensive cloud applications that are incrementally-processable.
  – E.g.: DNA sequence analysis
Managing Big Data in the Cloud
The “Stream As You Go” Approach [DMC’12, SSDBM’12]

- Implementation & Evaluation
  - DNA sequence analysis
    (read alignment (Bowtie) + SNP detection (SOAPsnp))
  - 1.4 GB E. Coli dataset
  - IBM InfoSphere Streams
  - Amazon EC2
- Compare against:
  - Hadoop-based Crossbow
  - UNIX-based Trivial

PoC in progress for testing with 200 GB human genome dataset from German Cancer Research Institute (DKFZ)