The case for Fine-Grained Stream Provenance

Boris Glavic*, Kyumars Sheykh Esmaili, Peter M. Fischer, Nesime Tatbul

Systems Group
Dept. of Computer Science
ETH Zurich – Switzerland

* CS Department
University of Toronto – Canada
Outline

- What is provenance?
- Stream Provenance Use Cases + Classification
- Challenges and Opportunities
- Design Space
- Combinations and Tradeoffs
- Conclusion and next steps
What is provenance?

- We see a results out a DBMS/DSMS
- What caused it?
- What contributed to it?
Source Provenance

- Where does the data come from?
- Currently state of the art in event/stream systems
- Low implementation/runtime overhead
Item-Level Provenance

- Which data items contributed to the result?
- Main goal of our research
- Significant implementation/runtime overhead
Attribute-Level Provenance

• Which attributes of data items contributed to the result?
• Which operations were performed for the result?
• Longer-Term research goal
• Possible to «piggyback» on item provenance [TRAMP]
Novel Provenance for Streams?

Order Provenance

- Why a particular output tuple was generated before/after another particular one?
- Problem statement and usefulness not yet fully understood
Outline

- What is provenance?
- Stream Provenance Use Cases + Classification
- Challenges and Opportunities
- Design Space
- Combinations and Tradeoffs
- Conclusion and next steps
Sensor Alerting

- Requirements
  - Granularity & Scope: item level, all items
  - Lifetime: time-bound, typically minutes to hours
  - Retrieval Strategy: ad-hoc drilldown
  - Retrieval Response time: at most few seconds

(80°), (93°), (120°), (260°), (387°)
Web Recommender Systems

Buy a Passat!

Requirements

- Granularity & Scope: individual items, all items & queries
- Lifetime: action+time bound (recommendation view)
- Retrieval Strategy: ad-hoc drilldown
- Retrieval Response time: some milliseconds

petfisch
Really funny:
youtu.be/...
Indicator-based Assurance

- Give summarized information on performance or compliance
- Customer/Auditor wants proof on these indicators
- Requirements
  - Granularity & Scope: item level, selected items & queries
  - Lifetime: indicator retention period
  - Retrieval Strategy: direct indicator computation or ETL
  - Retrieval Response time: offline

<table>
<thead>
<tr>
<th>KPI/KSI</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delivery on time</td>
<td>97.5%</td>
</tr>
<tr>
<td>SoD maintained</td>
<td>99.1%</td>
</tr>
<tr>
<td>Anonymization maintained</td>
<td>99.99%</td>
</tr>
</tbody>
</table>
Other Use Cases

- Stream Query Debuggers
  - Facilitate Replay
  - Attribute-level provenance for analysis

- Event Warehousing
  - Analytic queries
  - Arbitrary scope
  - Full materialization

- Uncertainty Management
  - Varying requirements for granularity, lifetime

- All need item-level (or better) provenance
- Requirements on lifetime, access pattern, response time vary considerably
Outline

- What is provenance?
- Stream Provenance Use Cases + Classification
- Challenges and Opportunities
- Design Space
- Combinations and Tradeoffs
- Conclusion and next steps
Problem Statement

- Correctly expressing provenance
  - Semantics of data & operations
  - Non-determinism (?)
  - Capturing and Representing

- Scalability and Performance
  - Infinite, Transient Data
  - (Possibly) High Data Rates
  - Aggregation
  - Response Times + Cost for Query and Provenance Operations
Approach

- Leverage existing provenance techniques
- Extend and improve for
  - Efficient Representation
  - Efficient Computation
  - Scalable Storage Management
Goals and Results

- Identify «right» methods
  - TIDs with upstream data storage
  - Propagation
  - Model/operator-specific compression
  - Application-driven data discarding

- Investigate important tradeoffs
  - Lazy/Eager computation
  - Runtime overhead vs retrieval overhead
  - Precision/Accountability vs Cost
Outline

- What is provenance?
- Stream Provenance Use Cases + Classification
- Challenges and Opportunities
- Design Space
- Combinations and Tradeoffs
- Conclusion and next steps
Stream Provenance Semantics

- Build on “contribution semantics“ formalism [CWW00]
- Uses Influence-CS (Why Provenance)
- Depends on specific stream model and operators, here
  - Selection+Projection
  - Window+Aggregates
  - Stream Join (using windows)
- Infinity and its management mostly solved by stream models
- Data Model and Operator semantics allow specific optimizations
External Provenance Representation

- Extend data model, custom query language [Trio]
- Duplicate data with provenance, standard query [PERM]
Internal Provenance Representation

- Full tuples
- TIDs
  - Store data at some point upstream
  - Use TID Set as annotations of provenance
  - Look up actual tuples when retrieving provenance
Computing Provenance (I)

- Investigate methods from existing provenance work
- Query Rewrite [PERM]
  - Complement query plan/network with additional (standard) operators to compute provenance
  - No changes to operators needed
  - Less precise, limited optimization potential
  - High runtime overhead

Join (windowcond)

Average size= 4
slide= 4
Computing Provenance (II)

- **Propagation**
  - Extend data model and operators to handle provenance annotations
  - Trace execution of query and generate provenance on the fly
  - Precise, flexible
  - High implementation overhead

- **Inversion**
  - Invert operation to generate provenance
  - Low runtime overhead
  - Limited to subset of deterministic operators, only execution lazy possible
  - Example: Join without projection, selection
Storage

- Eager, full tuple approaches do not need any storage.
- Lazy and/or TID approaches need to retain data until it is needed.
- Problem: which data to store for how long.
- Unless full archiving is used, one needs provenance for that!
Compression

- Large aggregation create high provenance/data ratio
  Window size=1000 with aggregation:
  1 data, 1000 provenance item

- Reduce overhead during propagation and computation

Strategies

- (TID)
- Interval compression
- Huffman Encoding/Gzip compression
- Delta-Encoding between Windows

- Enable new architecture option
  - Covering Intervals with Lazy Retrieval
Outline

- What is provenance?
- Stream Provenance Use Cases + Classification
- Challenges and Opportunities
- Design Space
- Combinations and Tradeoffs
- Conclusion and next steps
Possible instantiations

- Many variants possible, mostly for comparison:
  Everything with full tuples, Query Rewrite full tuples …

- Two promising candidates
  - Fully Eager Propagation
  - Lazy Retrieval with Covering Intervals
Fully Eager Propagation

- Use TIDs
- Use compression where applicable
- Permanently generate results with exact provenance TIDs
- Retrieve/join on demand

Suitable if

- Most events require provenance
- Precise recording is required
- Fast retrieval needed
Lazy retrieval with Covering Intervals

- Run normal query network to create covering intervals
- Create a second network for full provenance computation
- Keep and replay data according to covering intervals
- Suitable for
  - Low runtime overhead
  - A small subset of events require provenance
  - Deterministic operators needed or slight deviation allowed
  - Scaleout
Todo

- Implementation: mostly complete
  - Based on Aurora/Borealis
  - Propagation for most operators
  - Query Network Rewriter based on [PERM]
  - Not all compression methods yet implemented
- Performance Results: ongoing
  - Cost of individual operators and combination: expected results
  - Compression: partially done
  - Retrieval, scenarios: to be done
Summary and next steps

- Many use cases for fine-grained Provenance in DSMS
- Transient nature and high data rates major problems
- Leverage and extend existing provenance theory and implementation

(Likely) common core approach:
- TID/Upstream Storage/Propagation
- Model/operator-specific compression
- Application-driven data discarding

Important tradeoffs:
- Runtime overhead vs retrieval overhead
- Precision/Accountability vs Cost