Ariadne: Managing Fine-Grained Provenance on Data Streams

Boris Glavic¹ Kyumars Sheykh Esmaili² Peter M. Fischer³
Nesime Tatbul⁴

Illinois Institute of Technology¹
DBGroup

Nanyang Technological University²
SANDS

University of Freiburg³
Web Science

Intel Labs⁴
Intel Science and Technology Center for Big Data

DEBS 2013 - July 7, 2013 - Arlington, USA
Outline

1 Motivation

2 Reduced Eager Operator Instrumentation

3 Optimizations

4 Experiments

5 Conclusions
**Provenance**

- Information about the origin and creation process of data
- **Here:** On which inputs does a given output depend on

**Fine-grained Provenance**

- At the granularity of tuples
- Fix a tuple $t$ in an output or intermediate stream
- On which input tuples does it depend on?

**Example**

- $S_1$: 1 2 3
- $S_2$: a b
- $S_{out}$: 14
Provenance

- Information about the origin and creation process of data
- **Here**: On which inputs does a given output depend on

**Fine-grained Provenance**

- At the **granularity** of tuples
- Fix a tuple \( t \) in an output or intermediate stream
- On which input tuples does it depend on?

**Example**
Why (Fine-Grained) Stream Provenance?

Use cases

• Ad hoc inspection
  • DSMS generates alarm event based on sensor readings
  • Understand why alarm was raised to act appropriately

• Stream query debugging
  • Trace back and forward erroneous data items

• Auditing
  • Proof of correct operation
  • No access policies violated
Challenges and Opportunities

- **Online and infinite data arrival**
  - Traditional methods that reconstruct provenance retroactively not applicable

- **Ordered data model**
  - Order can be exploited for compressing provenance

- **Window-based processing**
  - Aggregation prevalent operation that has large provenance

- **Low-latency requirements**
  - Provenance overhead should not violate latency requirements

- **Non-determinism**
  - Sources: Load-shedding, windowing on system-time
  - Provenance computation can not assume reproducibility of results
## Related Work

### Database Provenance
- Large body of work on provenance models
- Query rewrite-based approaches

### Workflow Provenance
- Many systems support provenance tracking
- Mostly coarse-grained provenance

### Data Stream Provenance
- Coarse-grained provenance
  - Not detailed enough for most use-cases
- Fine-grained provenance
  - Stream debugging: One-step *(one operator at a time)*
  - Approximation techniques *(strong assumptions)*
How to generate provenance?

Inversion

- Infer provenance from query outputs
- Usually tracing back one operator at a time
- No storage overhead
- Only works for trivial operators
How to generate provenance?

Annotation Propagation

- Annotate data with provenance
- Propagate and manipulate annotations during query processing

Propagation - Query rewrite

- Rewrite query network to propagate annotations
- Use original DSMS operators
- Easy to implement, applicable to all DSMS*
- Often results in complex and inefficient networks
- Only applicable to deterministic networks
### How to generate provenance?

**Propagation - Operator instrumentation**

- Modify DSMS operators to propagate provenance information
- More efficient than query rewrite
- Tolerates certain types of non-determinism
- Keeps structure of the original query network intact
- Requires modification of the engine (DSMS source code needed)
Plan of attack

1) Provenance model
   - Annotated tuples
   - Annotated streams

2) Provenance generation and querying
   - Operator instrumentation
   - Buffer inputs + reconstruction for querying
   - Implementation in Ariadne (extension of Borealis)

3) Optimizations
   - Compression
   - Laziness
   - Decoupling provenance computation from query processing
Outline

1 Motivation

2 Reduced Eager Operator Instrumentation

3 Optimizations

4 Experiments

5 Conclusions
Overview

Operator Instrumentation

- For each operator of the system create new versions
  - Provenance generator (PG): Generates provenance annotations
  - Provenance propagator (PP): Propagates provenance annotations
- Instrument query (or parts thereof) to generate provenance
  - Replace operators with annotating versions
  - Only in the part of the network we want to track provenance

Reduced Eager

- Eager
  - Propagate provenance annotations during query processing
- Reduced Eager
  - Propagate sets of tuple-IDs (TIDs) annotations
  - Temporarily store inputs of relevant streams for restoring full tuples in provenance
Overview

Querying Provenance

- Translate provenance annotations into regular streaming data
- For one output tuple $t$ annotated with provenance $\{t_1, \ldots, t_n\}$
  - output $n$ tuples $(t, t_1), (t, t_2), \ldots$
- New operator (p-join) joins buffered input data with provenance annotations
- Use operators of the DSMS for querying
Overview

Instrumented Network

S1

S2

PG

PG

PP

Sout

Instrumented Network
Overview

- Instrumented Network
- Reconstruct Provenance
- Temporary Input Storage
- PG
- PP

Ariadne: Reduced Eager Operator Instrumentation
Stream Provenance Model

Provenance Model

- Provenance is modelled as a set of contributing tuples
  - From input of intermediate streams
- Provenance set $P(t, I)$
  - tuple $t$ in intermediate or result stream
  - set of upstream streams $I$
- Which tuples belong to provenance?
  - Declarative definition that models provenance for single operators
  - Transitivity
Example

- Query over stream of temperature readings
  - Filter out outliers (temperature above threshold)
  - Compute average temperature over every consecutive window of two readings
- \( P(3:2, \{2\}) = \{2:2, 2:3\} \)
  - 3:2 generated by aggregating values from 2:2 and 2:3
- \( P(3:2, \{1\}) = \{1:2, 1:4\} \)
  - 2:2 and 2:3 derived from 1:2 and 1:4
Provenance Annotated Streams

- Each tuple in a **provenance annotated stream (PAS)** is annotated with its provenance set
  - according to a set of input streams $I$
- PAS $P(O, I)$
  - Provenance annotated stream for stream $O$ according to set of upstream streams $I$
Example

- Query over stream of temperature readings
  - Filter out outliers (temperature above threshold)
  - Compute average temperature over every consecutive window of two readings
- Provenance annotated stream (PAS) \( P(3, \{1\}) \)

<table>
<thead>
<tr>
<th>TID</th>
<th>time</th>
<th>temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:1</td>
<td>0</td>
<td>102</td>
</tr>
<tr>
<td>1:2</td>
<td>5</td>
<td>105</td>
</tr>
<tr>
<td>1:3</td>
<td>10</td>
<td>399</td>
</tr>
<tr>
<td>1:4</td>
<td>15</td>
<td>85</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TID</th>
<th>time</th>
<th>temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:1</td>
<td>0</td>
<td>102</td>
</tr>
<tr>
<td>2:2</td>
<td>5</td>
<td>105</td>
</tr>
<tr>
<td>2:3</td>
<td>15</td>
<td>85</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TID</th>
<th>avg_temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:1</td>
<td>103.5</td>
</tr>
<tr>
<td>3:2</td>
<td>95</td>
</tr>
</tbody>
</table>

\{1:1, 1:2\} \{1:2, 1:4\}
Approach

- Three instrumented versions for each operator
- **Provenance generators**: Initialize provenance annotations
- **Provenance propagators**: Propagate provenance annotations
- **Provenance dropper**: Drop provenance from input
Implementing Instrumented Operators

Provenance Generator

- Computes one-step provenance for an operator’s output based on its input
- Attach input TIDs as provenance
- For windowing operators merge TIDs from window
- Before instrumentation: \( I \rightarrow op \rightarrow O \)
- After instrumentation: \( I \rightarrow op^{PG} \rightarrow P(O, I) \)
Implementing Instrumented Operators

Provenance Propagator

- Compute output provenance set by combining provenance sets from the input
- Before instrumentation: \( P(I, I') \rightarrow op \rightarrow O \)
- After instrumentation: \( P(I, I') \rightarrow op^{PP} \rightarrow P(O, I') \)

Provenance Dropper

- Removes provenance annotations
- Instrumentation: \( P(I, I') \rightarrow op^{PD} \rightarrow O \)
Instrumenting a Network

- User provides
  - query network \( q \)
  - stream \( O \)
  - set of streams \( I \) upstream from \( O \)
- Output is a network that computes \( P(O, I) \)

**InstrumentNetwork**\((q, O, I)\)

for all \( op \) on paths between \( I \) and \( O \)

if \( op \) is connected to \( I \)
  replace \( op \) with \( PG \) version

else
  replace \( op \) with \( PP \) version
Instrumenting a Network

Example

- Network that computes $P(3, \{1\})$

<table>
<thead>
<tr>
<th>TID</th>
<th>time</th>
<th>temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:1</td>
<td>0</td>
<td>102</td>
</tr>
<tr>
<td>1:2</td>
<td>5</td>
<td>105</td>
</tr>
<tr>
<td>1:3</td>
<td>10</td>
<td>399</td>
</tr>
<tr>
<td>1:4</td>
<td>15</td>
<td>85</td>
</tr>
</tbody>
</table>

1

2

PG

α

PP

TID avg_temp
3:1 103.5 {1:1, 1:2}
3:2 95 {1:2, 1:4}

Example

- Network that computes $P(2, \{1\})$

<table>
<thead>
<tr>
<th>TID</th>
<th>time</th>
<th>temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:1</td>
<td>0</td>
<td>102</td>
</tr>
<tr>
<td>1:2</td>
<td>5</td>
<td>105</td>
</tr>
<tr>
<td>1:3</td>
<td>10</td>
<td>399</td>
</tr>
<tr>
<td>1:4</td>
<td>15</td>
<td>85</td>
</tr>
</tbody>
</table>

1

2

PG

α

PD

TID avg_temp
3:1 103.5 {1:1, 1:2}
3:2 95 {1:2, 1:4}
Ariadne Implementation

Based on Borealis

- Set of standard DSMS operators
- Operators connected through queues
- Fixed-length tuples

Changes to Borealis code

- Implement annotations
- Implement operator instrumentation
- Implement p-join and input buffering
Approach

1. Translate provenance annotations into regular stream data to query using the host language
Querying Provenance

**Approach**

1. Translate provenance annotations into regular stream data to query using the host language.

**P-join**

- Joins a PAS $P(O, I)$ with buffered input tuples from $I$.
- Combine output $t$ with each tuple in its provenance $P(t, I)$. 
Querying Provenance

Example

- Tuple 3:2 with $P(3:2, \{1\}) = \{1:2, 1:4\}$

<table>
<thead>
<tr>
<th>TID</th>
<th>avg_temp</th>
<th>p_time</th>
<th>p_temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>4:1</td>
<td>103.5</td>
<td>0</td>
<td>102</td>
</tr>
<tr>
<td>4:2</td>
<td>103.5</td>
<td>5</td>
<td>105</td>
</tr>
<tr>
<td>4:3</td>
<td>95</td>
<td>5</td>
<td>105</td>
</tr>
<tr>
<td>4:4</td>
<td>95</td>
<td>15</td>
<td>85</td>
</tr>
</tbody>
</table>
Outline

1 Motivation
2 Reduced Eager Operator Instrumentation
3 Optimizations
4 Experiments
5 Conclusions
Compression
- Reduce load on queues by using concise provenance representation
- Reduce memory imprint

Lazy Retrieval
- Avoid reconstructing unneeded provenance

Replay-Lazy
- Avoid generating unneeded provenance
- Decouple provenance computation from regular stream processing
Replay-Lazy

Idea

- Avoid provenance computation if provenance is not needed
- Propagate concise representation of a super-set of the provenance
- If provenance is requested **replay** super-set through provenance generating network
Replay-Lazy

Idea

- Avoid provenance computation if provenance is not needed
- Propagate concise representation of a super-set of the provenance
- If provenance is requested, replay super-set through provenance generating network

Covering Intervals

- Interval:
  - Minimal TID in provenance
  - Maximal TID in provenance
- Constant size super-set of provenance (piggy-back!)

- Provenance: \{1, 2, 4, 5, 10, 16, 65\}
- Covering interval: [1, 65]
Enabling Replay

- Covering interval operators:
  - CG corresponds to PG
  - CP corresponds to PP
- **C-join** operator ⊗
  - For each input get covering interval
  - Retrieve all tuples from covering interval from connection point
Replay-Lazy

**Approach**

1. Instrument original network to produce covering interval
2. Filter out results and c-join
3. Feed result into provenance generating copy of the network
Outline

1 Motivation
2 Reduced Eager Operator Instrumentation
3 Optimizations
4 Experiments
5 Conclusions
Send large batch of 100,000 tuples
Measure completion time
  - Without Retrieval/With Retrieval
  - No Provenance/Instrumentation/Rewrite
Using the Basic Network
  - Vary Aggregation Window Size
  - Slide fixed to 1
  - Increase window size \( \rightarrow \) increase amount of provenance
Completion Time - Basic Network

**Completion Time (sec)**

- No Provenance
- Instrumentation (Generation)
- Instrumentation (Retrieval)
- Replay-Lazy (Covering Interval)
- Replay-Lazy (Retrieval)
- Rewrite

**Window Size**

- 10
- 20
- 40
- 60
- 80
- 100

---

*Glavic, Sheykh Esmaili, Fischer, Tatbul - Ariadne: Experiments*
Outline

1 Motivation
2 Reduced Eager Operator Instrumentation
3 Optimizations
4 Experiments
5 Conclusions
Conclusions

Reduced-Eager Operator Instrumentation

- Novel propagation provenance method for DSMS
- Replace operators in a query network with provenance-aware (instrumented) versions
  - Keeps original network structure intact
- Deals well with non-determinism
- Flexible:
  - Single- and multi-hop provenance
  - Instrument parts of a network
Conclusions

Optimizations

- **Replay-Lazy**
  - Propagate only covering intervals - replay through provenance generating copy of network
  - Reduce cost for low retrieval rates
  - Option to decouple provenance computation

- **Lazy-Retrieval**
  - Push filters through p-joins
  - Avoids provenance reconstruction

- **Compression**
  - Interval
  - Delta
  - Dictionary
Future Work

### Optimizing Temporary Input Storage
- Exploiting additional knowledge in purging temporary input storage
  - Static query network analysis
  - Self-optimizing purging strategies
  - Compression and Approximations

### Integration with Distributed Storage and Scaling-out
- Using distributed write-optimized storage for provenance?

### Order-aware Provenance Model
- Integrate order into the provenance model
- “Tuple $t$ is after $u$ in the result, because $t$’s provenance is before $u$’s provenance in the input.”
<table>
<thead>
<tr>
<th>Name</th>
<th>Affiliation</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boris Glavic</td>
<td>IIT DBGroup</td>
<td><a href="mailto:bglavic@iit.edu">bglavic@iit.edu</a></td>
</tr>
<tr>
<td>Kyumars Sheykh Esmaili</td>
<td>Nanyang University SANDS</td>
<td><a href="mailto:kyumarss@ntu.edu.sg">kyumarss@ntu.edu.sg</a></td>
</tr>
<tr>
<td>Peter M. Fischer</td>
<td>University of Freiburg Web Science</td>
<td><a href="mailto:peter.fischer@cs.uni-freiburg.de">peter.fischer@cs.uni-freiburg.de</a></td>
</tr>
<tr>
<td>Nesime Tatbul</td>
<td>Intel Labs Intel Science and Technology Center for Big Data</td>
<td><a href="mailto:tatbul@csail.mit.edu">tatbul@csail.mit.edu</a></td>
</tr>
</tbody>
</table>

http://cs.iit.edu/~dbgroup/research/ariadne.php
Outline

6 Implementation

7 Additional Experiments

8 Optimizations
How to Implement Annotations?

Alternatives for sending annotations

- Borealis models streams as queues of fixed-length tuples with header and payload

1. **Variable-length tuples:** Negates optimizations based on fixed-length
2. **Split provenance sets into fix-length tuples**
3. **New information channels:** Complex interactions with normal query processing
How to Implement Annotations?

Alternatives for sending annotations

- Borealis models streams as queues of fixed-length tuples with header and payload

1. Variable-length tuples: Negates optimizations based on fixed-length
2. Split provenance sets into fix-length tuples
   - Send provenance set after each output tuple
   - First tuple has small header to tell downstream operators how many provenance tuples will follow
   - Additional provenance tuples only store payload (TIDs)
3. New information channels: Complex interactions with normal query processing
Implementing Instrumented Operators

Approach

- Factor out common functionality
  - Serializing/Deserializing provenance to/from queues
  - Caching provenance of windows
  - Merging provenance sets
- Implemented in Provenience Wrapper
  - Operators access queues through the wrapper
  - Reduces code changes to each operator
- LOC
  - Provenience wrapper: ~8000 LOC
  - Instrumented operators: aggregation (largest) 200 LOC
Temporary Input Storage

Connection Points

- Borealis feature for storing tuples from a stream
- Time-based or count based eviction strategies
- Content can be joined with streams
Outline

6 Implementation

7 Additional Experiments

8 Optimizations
Varying Window Size

- Send large batch of 100,000 tuples
- Measure completion time
  - No Retrieval
  - No Provenance/Single/Optimized/Covering Interval
- Using the Basic Network
  - Vary Window Size
  - Slide fixed to 1

Basic Query Network

Rewritten Version

Instrumented Version

Replay-Lazy Version
Varying Window Size

Completion Time (sec) vs. Window Size for different provenance schemes:
- No Provenance
- Single
- Optimized
- Covering Interval

Graph showing the completion time for different window sizes and provenance schemes.
Vary Retrieval Frequency

- Send large batch of 100,000 tuples
- Measure completion time
  - Instrumentation with Retrieval
  - Replay-Lazy with Retrieval
- Using a sequence of aggregations
  - Vary Retrieval Frequency - selectivity of filter before p-join
  - Slide fixed to 1, Window size fixed to 100
Vary Retrieval Frequency

Completion Time / Completion Time without Provenance

Completion Time (%)

Retrieval Frequency

- Instrumentation with Retrieval (Optimized)
- Replay-Lazy with Retrieval (Optimized)

Instrumentation with Retrieval (Optimized)
Replay-Lazy with Retrieval (Optimized)
Latency

- In Ariadne inputs are send in batches with a fixed number of tuples (parameter)
- Measure latency
  - Using the Basic Network
- Vary the load
  - Vary batch size and fix the delay between consecutive batches
Latency

- Ariadne - Appendix: Additional Experiments
## Nested Aggregations

- Sequence of aggregation operators
- Measure Completion time
- Vary Number of aggregation operators

<table>
<thead>
<tr>
<th>Method</th>
<th>Number of Aggregations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>No Provenance</td>
<td>3.1</td>
</tr>
<tr>
<td>Instr.</td>
<td>3.9</td>
</tr>
<tr>
<td>Generation</td>
<td>3.0</td>
</tr>
<tr>
<td>Retrieval</td>
<td>3.1</td>
</tr>
<tr>
<td>Cov. Inter.</td>
<td>5.2</td>
</tr>
<tr>
<td>Retrieval</td>
<td>7.2</td>
</tr>
<tr>
<td>Rewrite</td>
<td></td>
</tr>
</tbody>
</table>
Outline

6 Implementation

7 Additional Experiments

8 Optimizations
Compression

Rationale

- Reduce amount of data that is shipped between operators
- Requirements
  - Fast compression and decompression
  - Operations such as merging sets on compressed data
## Rationale

- Reduce amount of data that is shipped between operators
- Requirements
  - Fast compression and decompression
  - Operations such as merging sets on compressed data

## Interval Compression

- **Input:** \( \{1, 2, 4, 5, 6, 7, 9\} \)
- **Output:** \( \{[1 - 2], [4 - 6], [9 - 9]\} \)
Compression

Rationale

- Reduce amount of data that is shipped between operators
- Requirements
  - Fast compression and decompression
  - Operations such as merging sets on compressed data

Interval Compression

- Input: \{1, 2, 4, 5, 6, 7, 9\}
- Output: \{[1 – 2], [4 – 6], [9 – 9]\}

Dictionary Compression

- Input: \{1, 2, 4, 5\}
- Output: LZ77(\{1, 2, 4, 5\})
Delta Compression

• Express provenance as delta to down-stream provenance
  • Once in a while send full provenance
  • Express provenance as delta to previous full provenance
• Input: \{1, 2, 4, 7, 9\} ← \{4, 7, 9, 10\} ← \{7, 9, 10, 12, 15, 19\}
• Output: \{1, 2, 4, 7, 9\} ← 3 − \{10\} ← 2 − \{10, 12, 15, 19\}
Compression

**Delta Compression**

- Express provenance as delta to down-stream provenance
  - Once in a while send full provenance
  - Express provenance as delta to previous full provenance
- Input: $\{1, 2, 4, 7, 9\} \leftarrow \{4, 7, 9, 10\} \leftarrow \{7, 9, 10, 12, 15, 19\}$
- Output: $\{1, 2, 4, 7, 9\} \leftarrow 3 - \{10\} \leftarrow 2 - \{10, 12, 15, 19\}$

**Heuristic Adaptive Compression**

- No one fits all
- Combine compression methods
- Heuristic rules for when to apply which method
Retrieval-Lazy

Approach

- User query that filters out unneeded provenance

Slide 9 of 9  Glavic, Sheykh Esmaili, Fischer, Tatbul - Ariadne - Appendix: Optimizations
Retrieval-Lazy

Approach

- User query that filters out unneeded provenance
- Avoid expensive provenance reconstruction if provenance is not needed
  - If possible filter before reconstruction (p-join)
  - Push filters through p-joins