How to Build a Stream Reasoning Application
D. Dell'Aglio, E. Della Valle, T. Le-Pham, A. Mileo, and R. Tommasini
http://streamreasoning.org/events/streamapp2017
Share, Remix, Reuse — Legally

- This work is licensed under the Creative Commons Attribution 3.0 Unported License.

- **Your are free:**
  - **to Share** — to copy, distribute and transmit the work
  - **to Remix** — to adapt the work

- **Under the following conditions**
  - **Attribution** — You must attribute the work by inserting a credits slide stating
    - These slides are partially based on “How to Build a Stream Reasoning Application 2017” by D. Dell'Aglio, E. Della Valle, T. Le-Pham, Mileo, and R. Tommasini available online at http://streamreasoning.org/events/streamapp2017

- To view a copy of this license, visit http://creativecommons.org/licenses/by/3.0/
A model to describe stream processing

## Solutions vs. requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>DSMS CEP</th>
<th>Sem Web</th>
</tr>
</thead>
<tbody>
<tr>
<td>volume</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>velocity</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>variety</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>incompleteness</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>noise</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>reactive answers</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>fine-grained information access</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>complex domain models</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>high-level languages</td>
<td>X</td>
<td>✓</td>
</tr>
</tbody>
</table>
• Research question
  – is it possible to make sense in real time of multiple, heterogeneous, gigantic and inevitably noisy and incomplete data streams in order to support the decision processes of extremely large numbers of concurrent users?

Is this feasible?

- Proposed approach: **cascading** Stream Reasoning

A model to describe stream reasoning

Stream Reasoning

- Stream Processing (DSMS)
- Graph-level entailment
- Window merge
- Event Processing (CEP)
- Window-level entailment
- Window operator
- Stream-level entailment

Streams

Application


http://streamreasoning.org/events/streamapp2017
A model to describe stream reasoning

continuous deductive reasoning

• DL Ontology Stream $S^T$
  – A ontology stream with respect to a static Tbox $T$ is a sequence of Abox axioms $S^T(i)$

• A Windowed Ontology Stream $S^T(o,c]$
  – A windowed ontology stream with respect to a static Tbox $T$ is the union of the Abox axioms $S^T(i)$ where $o<i\leq c$

• Reasoning on a Windowed Ontology Stream $S^T(o,c]$ is as reasoning on a static DL KB

Example of Stream Reasoning 1/2

- **Query:** measure the impact of Alice's microposts
  **MEMO:** our running example data model

- **For example**

  Alice posts $p_1$. Bob posts $p_2$.

  - $p_1$ discusses $p_2$.
  - $p_2$ discusses $p_3$.
  - $p_3$ discusses $p_4$.
  - $p_4$ discusses $p_7$.
  - $p_7$ discusses $p_8$.
  - $p_6$ discusses $p_5$.

  Timeline:
  - 50 min ago
  - 40 min ago
  - 30 min ago
  - 20 min ago
  - 10 min ago
  - Now
What impact has been my micropost $p_1$ creating in the last hour? Let’s count the number of microposts that discuss it …

REGISTER STREAM ImpactMeter AS
SELECT (count(?p) AS ?impact)
FROM STREAM <http://.../fb> [RANGE 60m STEP 10m]
WHERE {
  :Alice posts [ sr:discusses ?p ]
}

Transitive property
Alice posts $p_1$.

7!
MEMO: forms of reasoning for Q/A

- **Data-driven (a.k.a. forward reasoning)**
  - RDF data → Reasoner → Inferred data → SPARQL
  - ontology

- **Query-driven – backward reasoning**
  - RDF data → Reasoner → SPARQL
  - ontology

- **Query-driven – query rewriting (a.k.a. ontology based data access)**
  - data → Rewritten query → Reasoner → SPARQL
  - ontology
Naïve Stream Reasoning

- **Data-driven (a.k.a. forward reasoning)**
  - S2R → RDF data → Reasoner → Inferred data → SPARQL
  - ontology

- **Query-driven – backward reasoning**
  - S2R → RDF data → Reasoner → SPARQL
  - ontology
Backward and forward naïve Stream Reasoners

- **Streaming knowledge base**

- **C-SPARQL**

- **Sparkwave**

- **Dynamite**

- **Yasper**
  - *You will use it in the next hands on session*
Naïve Stream Reasoning

- **Data-driven (a.k.a. forward reasoning)**
  - S2R → RDF data → Reasoner → Inferred data → SPARQL
  - ontology

- **Query-driven – backward reasoning**
  - S2R → RDF data → Reasoner → SPARQL
  - ontology

- **Query-driven – query rewriting (a.k.a. ontology based data access)**
  - S2R → data → Rewritten query → Reasoner → SPARQL
  - ontology
Naïve query-driven stream reasoning by query rewriting

- **MEMO**

  ![Diagram](http://streamreasoning.org/events/streamapp2017)

- It is not that straight forward :-(
  - Lack of a standard query language for DSMS and CEP
  - Lack of a well-understood operational semantics for DSMS and CEP (cf. SECRET by I. Botan et al., PVLDB 3(1), 2010)
  - Lack of expressiveness in OWL2QL
Query rewriting naïve Stream Reasoners


Not so naïve stream reasoning

- Naïve data-driven approach

- From snapshots to changes
  - What has just been inserted?
  - What has just been deleted?

```
S2R → RDF data → Reasoner → Inferred data → SPARQL
```

```
S2R → insertions → Reasoner → Inferred data → SPARQL
S2R → deletions → Reasoner → Inferred data → SPARQL
```
Not so naïve stream reasoning

- MEMO

- The problem is that materialization (the result of data-driven processing) are very difficult to decrement efficiently.
  - State-of-the-art: DReD algorithm
    - Over delete
    - Re-derive
    - Insert

The Intuition of DRed Algorithm

- Let’s assume that we have the following materialized graph

- While inserts are not problematic, deletion are difficult to handle. If we delete $p_2$ discusses $p_1$ ($p_2 \rightarrow p_1$), we have
  - **overestimate the impact of the deletion** and mark for deletion $p_4 \rightarrow p_1$ that can be derived by $p_4 \rightarrow p_2$ and $p_2 \rightarrow p_1$

- **look for alternative derivation** of $p_4 \rightarrow p_1$ and eventually find the chain $p_4 \rightarrow p_3$ and $p_3 \rightarrow p_1
DReD-based stream reasoners

- **TROWL**
  - How: DRed in the context of approximate reasoning

- **The Backward/Forward Algorithm**
  - How: optimizing DRed
Is DReD needed?

- **DReD** works with **random insertions** and **deletions**
- **In a streaming setting**, when a triple enters the window, given the size of the window, the reasoner knows already when it will be deleted!

- **E.g.,**
  - if the window is 40 minutes long, and,  
  - it is 10:00, the triple(s) entering now  
  - will exit on 10:40.

- **Conclusion**  
  - **deletions are predictable**

<table>
<thead>
<tr>
<th>Time</th>
<th>Enter window</th>
<th>Exit window</th>
<th>Explicitly in window</th>
</tr>
</thead>
<tbody>
<tr>
<td>10:00</td>
<td>A←B</td>
<td>A ← B</td>
<td></td>
</tr>
<tr>
<td>10:10</td>
<td>B←C</td>
<td>A ← B ← C</td>
<td></td>
</tr>
<tr>
<td>10:20</td>
<td>A←E</td>
<td></td>
<td>E</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A ← B ← C</td>
<td></td>
</tr>
<tr>
<td>10:30</td>
<td>E←C</td>
<td></td>
<td>E</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A ← B ← C</td>
<td></td>
</tr>
<tr>
<td>10:40</td>
<td>A←B</td>
<td></td>
<td>E</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A ← B ← C</td>
<td></td>
</tr>
<tr>
<td>10:50</td>
<td>B←C</td>
<td></td>
<td>E</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A ← C</td>
<td></td>
</tr>
<tr>
<td>11:00</td>
<td>A←E</td>
<td></td>
<td>E</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>
**IMaRS algorithm**

- **Idea:**
  - add an expiration time to each triple and
  - use an hash table to index triples by their expiration time

- **The algorithm**
  1. deletes expired triples
  2. Adds the new derivations that are consequences of insertions annotating each inferred triple with an expiration time (the min of those of the triple it is derived from), and
  3. when multiple derivations occur, for each multiple derivation, it keeps the max expiration time.

---


http://streamreasoning.org/events/streamapp2017
IMaRS algorithm

- Incremental Reasoning on RDF streams (IMaRS): new reasoning algorithm optimized for reactive query answering

- Re-materialize after each window slide
- Use DRed
- IMaRS

% of deletions w.r.t. the content of the window
A model to describe stream reasoning

Graph-level entailment considers data item contents, but it does not use the temporal annotations.

Window-level entailment applies the inference process on the non-merged stream items.

E.g.,

- A door cannot be open and close at the same time.
- A window contains: door A is open @1, door A is close @2.
- At graph-level the reasoner tells that there is an inconsistency in the window because it ignore the parts in *italics*.
- At window-level the reasoner does not.

Best approaches I saw so far

A model to describe stream reasoning


Stream Reasoning

- Stream Processing (DSMS)
- Graph-level entailment
- Window merge
- Event Processing (CEP)
- Window-level entailment
- Window operator
- **Stream-level entailment**

Streams

Application

http://streamreasoning.org/events/streamapp2017
Stream-level entailment

- Window-level entailment only considers a recent portion of the stream
- Stream-level entailment aims at considering the entire stream
- This is not just a theoretical dream, CEP does so
  - E.g., rise C for every A that follows a B without a C in the middle
- Best approaches I saw so far
  - ETALIS
  - LARS
How to Build a Stream Reasoning Application

D. Dell’Aglio, E. Della Valle, T. Le-Pham, A. Mileo, and R. Tommasini

http://streamreasoning.org/events/streamapp2017

DL-based Stream Reasoning

Emanuele Della Valle