A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data

Danh Le-Phuoc, Minh Dao-Tran, Josiane Xavier Parreира, Manfred Hauswirth
Outline

1. Motivation
2. Processing model
3. CQELS Engine
4. Experiment
5. Conclusion & Future work
Integrate Highly Dynamic data into Linked Data Cloud
Processing Linked Stream Data In A Nutshell

A. Linked Open Data cloud

B. Query

SELECT ?person
FROM ... [NOW]
WHERE {
?person ...
}

C. Sensor Stream Data

Continuous Query

Static RDF datasets

pre-processing

optimization

execution

answer

Stream Data in RDF
Black-box approach and limitations
EP–SPARQL and C–SPARQL

Motivation

EP–SPARQL

Prolog Engine

EP–SPARQL to Prolog

Query Rewriter

Orchestrator

RDF to Prolog facts

Data transformation

CSPARQL to SPARQL

Query Rewriter

SPARQL Engine

Data transformation

Data transformation

RDF to Java objects

Orchestrator

CSPARQL to EPSE EPL

Query Rewriter

ESPER

Enabling networked knowledge.
White-box approach: Enable native and adaptive processing
Enabling the adaptivity

- Incoming data will continuously change the costs of query plans

- Data elements are adaptively routed to processing operators on equivalent data flows (aka, routing policies)
Different data flows for different data

- Notify two people who are co-authors of a paper if they are in the same location (within the last 30 seconds)
Processing Model – Operators

- **Triple-based window operators** extracts triples from RDF stream or dataset that
  - Match a given triple pattern
  - Valid within in a time window

- **Relational operators** enable employing relational algebras in the processing model

- **Streaming operators** generate new streams from output of other operators based on graph templates
Continuous Query Evaluation over Linked Streams (CQELS)

CQELS Engine

Digital Enterprise Research Institute

Query

Adaptive Optimizer

Adaptive Executor

Dynamic Routing Policy

CQELS language (an extension of SPARQL 1.1)

Caching and Indexing

Operator Implementations

Adaptive Execution

SPO index scheme
Dictionary
Ring Triple-based indices for windows

Native Access methods

RDF dataset

Linked datastream

Enabling networked knowledge.
Techniques

- **Encoding with dictionary**
  - Smaller memory for representing triples
  - Avoid lookup & decoding overhead for numeric RDF nodes

- **Caching and Indexing**
  - Caching: avoid re-computing of intermediate results of sub-queries over non-stream data.
  - Indexing: allow faster access on caches and window data.

- **Dynamic Routing Policy**
  - Enable incoming data can be executed in multiple equivalent data flows adapt the changes
  - Easy & flexible to implement a routing policy
CQELS query language

CQELS Language – an extension to SPARQL

1.1

Construct new RDF stream

Stream pattern
Experiment setup

- **Conference scenario**: combine linked stream from RFID tags (physical relationships) with DBLP data (social relationships)

- **Setup**
  - **Systems**: CQELS vs ETALIS and C-SPARQL
  - **Datasets**
    - Replayed RFID data from Open Beacon deployments
    - Simulated DBLP by SP²Bench
  - **Queries**: 5 query templates with different complexities
    - Q1: selection,
    - Q2: stream joins, Q3, Q4: Stream and non-stream joins
    - Q5: aggregation
  - **Experiments**
    - Single query: generate 10 query instances of each template by varying the constants
    - Vary size of the DBLP ($10^4$–$10^7$ triples)
    - Multiple queries: register $2^M$ instances at the same time ($0 \leq M \leq 10$)
### Experiment results – Query execution time

- **Perform faster by orders of magnitudes**

<table>
<thead>
<tr>
<th></th>
<th>Query 1</th>
<th>Query 2</th>
<th>Query 3</th>
<th>Query 4</th>
<th>Query 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQELS</td>
<td>0.47</td>
<td>3.90</td>
<td>0.51</td>
<td>0.53</td>
<td>21.83</td>
</tr>
<tr>
<td>C-SPARQL</td>
<td>332.46</td>
<td>99.84</td>
<td>331.68</td>
<td>395.18</td>
<td>322.64</td>
</tr>
<tr>
<td>ETALIS</td>
<td>0.06</td>
<td>27.47</td>
<td>79.95</td>
<td>469.23</td>
<td>160.83</td>
</tr>
</tbody>
</table>

- **Simple selection:** ETALIS performs best
- **Stream join:** 25 times faster than C-SPARQL, 8 times faster than ETALIS
- **Aggregation:** 15 times faster than C-SPARQL, 8 times faster than ETALIS
- **Join and non-stream join:** 600 times faster than C-SPARQL, 150–850 times faster than ETALIS
Experiment results – Scalability (non-stream data size)

Non-stream data size: logarithmic to size of static intermediate results
Experiment results – Scalability (number of queries)

- Q1
- Q3
- Q4
Future work

- **Optimization**
  - Adaptive cost-based query optimization
  - Inter-query optimization

- **Smart and dynamic caching**
  - Adaptive caching
  - Materialized view maintenance for dynamic data

- **Scalability**: scale CQELS engine on multiple machines
Conclusion

- Native and Adaptive approach to process Linked Stream Data and Linked Data
  - Native and Adaptive Processing Model
  - Performs faster by orders of magnitude
  - Scales better: data size and number of queries
  - CQELS engine supports CQELS language compatible with SPARQL 1.1
Adaptive Cost-based optimization

- Cost-based query optimization with uniform data assumption ➔ 2–3 times faster than heuristic optimization, 5% overhead on optimization
Building blocks of a query processor

- Query
- Optimizer
- Executor
- Execution
- Access methods
- Database
- Datastream
- Nested-loop or hash joins
- Aggregation
- Sliding windows
- BTree++, hash table
- Triple-based indices

Motivation – Query Processor
Background – Linked Data Stream

Linked Data Stream

- **Stream data sources**
  - **Billions** of sensor data sources: mobile phones, weather observation stations, location tracking, etc
  - **Torrents** of dynamic web data sources: Twitter, Facebook, web feeds, etc

- Data heterogeneity ➔ Integration problem

- **Solution:** lifting sensor data into RDF
  - Semantic Sensor Web/W3C Semantic network Incubator Group
  - Semantic Streams
  - RDF wrappers for news feeds, Twiteer, Facebook, etc

- **Linked Data Stream**
Techniques

- **Encoding with dictionary**
  - RDF nodes are encoded as integer identifiers
  - Bypass the dictionary for numeric nodes

- **Caching and Indexing**
  - Caching: precompute intermediate results of sub-queries over non-stream data.
  - Indexing: build indexes on caches and window data.

- **Dynamic Routing Policy**
  - Choose different plans for data coming from different streams
  - Use heuristic optimization based on sizes of data in windows at run-time