A Compressed, Inference-enabled Scheme for RDF Stream Processing

Jérémy LHEZ, Xiangnan REN, Badre BELABBESS, Olivier CURÉ

LIGM, Marne-la-Vallée, FRANCE
OUTLINE
Agenda of the Presentation

- CONTEXT
- LITEMAT
- PATBIN
- CONCLUSION
Importance of stream processing
DATA STREAMS EVERYWHERE
WAVES PROJECT

➤ **Smart water network management**
  - Data streams from sensors
  - Filtering errors in measures
  - Identify sources in external events

➤ **Main partner: Suez**
  - 650 collaborators in Europe
  - 73 billion $US in R&D

➤ **French project**
Why water management?

Water SUPPLIED to the network - Water BILLED to customers = NON-REVENUE WATER (NRW)

Billed water 65%  

NRW 35%

48.6 billion m³/year

Loss of US$14 billion/year

x2 the annual domestic water consumption of the USA
Why water management?

Physical losses
→ 32 billion m³/year

Water supplied

Commercial losses
→ 16 billion m³/year

Leakage

→ 90% of leaks are invisible...

Incorrect Billing

Frauds

Billed water

→ Inaccurate metering
→ Data handling errors

ENVIRONMENTAL STAKE

ECONOMICAL STAKE

Inaccurate metering and data handling errors lead to
physical losses of 32 billion m³/year, of which 90% are invisible leaks. This results in commercial losses of 16 billion m³/year.
WAVES PROJECT

➢ **Main aspects:**
  - Real-Time processing
  - RDF data streams/SPARQL queries
  - Reasoning Capabilities/Inferences

➢ **Objectives:**
  - Robust RSP engine
  - Modularity and flexibility
  - Distribution – Industrial Cluster

➢ **Other applications:**
  - Banking/payments, climate, energy, power consumption, etc
ARCHITECTURE

IOT DEVICES

RDFIZATION

NATIVE DATA CLEANSING

COMPRESSION

APPLYING NEW FILTER

RESULT OF FILTER 1

Visualization

Stream Input 1

Stream Input 2

Stream Input N

Filter 1

Filter 2

Filter N

External Triple store

In memory

Triple store

In memory

Key-value store

Data enrichment

Step

Operators

Window

In memory

Triple store

Stream Output

Visualization

WAVES ATOS SE
ARCHITECTURE

IOT DEVICES

RDFIZATION

COMPRESSSION

NATIVE DATA CLEANSING

LiteMat

Filter N

FILTER 2

FILTER 1

RESULT OF FILTER 1

Visualization

Stream Input 1

Stream Input 2

Stream Input N

Step

Operators

Window

In memory Triple store

In memory Key-value store

Data enrichment

External Triple store

RESULT OF FILTER 1

LiteMat

PatBin

APPLYING NEW FILTER

FILE

EXTERNAL TRIPLE STORE

EVENT STREAMING
ISSUES OF STREAM PROCESSING

➢ Compression efficiency
  • No distribution
  • Decompression process
ISSUES OF STREAM PROCESSING

➢ **Compression efficiency**
  - No distribution
  - Decompression process

➢ **Solutions specific to a use case**
  - **Materialization:** huge amounts of data
  - **Query rewriting:** execution time
ISSUES OF STREAM PROCESSING

➢ **Compression efficiency**
  - No distribution
  - Decompression process

➢ **Solutions specific to a use case**
  - **Materialization**: huge amounts of data
  - **Query rewriting**: execution time

➢ **Lack of performance for heavy data load**
LITEMAT

Encoding the concepts and properties,
a compressed form conserving the hierarchy
ENCODING ALGORITHM

➢ **Principle:**
  - Binary structure conserving the semantics of the ontology
  - Each identifier is prefixed by its parent’s
  - Conversion as integer identifiers
  - Supports RDFS
ENCODING ALGORITHM

➢ **Principle:**
  - Binary structure conserving the semantics of the ontology
  - Each identifier is prefixed by its parent’s
  - Conversion as integer identifiers
  - Supports RDFS

➢ **Advantages:**
  - Compression maintaining hierarchy
    ✓ No need for ontology at query runtime
    ✓ Execution performance
  - Easy query rewriting
Extract from SSN

- **Thing**
  - **PhysicalObject**
  - **Sensor**
  - **SensingDevice**

Transitivity

- `waves:QBE04 rdf:type ssn:SensingDevice` (explicit)
  - `waves:QBE04 rdf:type ssn:Sensor`
  - `waves:QBE04 rdf:type DUL:PhysicalObject` (implicit)
## Concepts

<table>
<thead>
<tr>
<th>Concepts</th>
<th>Compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>owl:Thing</td>
<td>1</td>
</tr>
<tr>
<td>DUL:PhysicalObject</td>
<td></td>
</tr>
<tr>
<td>ssn:Sensor</td>
<td></td>
</tr>
<tr>
<td>ssn:SensingDevice</td>
<td></td>
</tr>
<tr>
<td>dbo:Engine</td>
<td></td>
</tr>
<tr>
<td>DUL:SocialObject</td>
<td></td>
</tr>
</tbody>
</table>
# LITEMAT EXAMPLE

<table>
<thead>
<tr>
<th>Concepts</th>
<th>Compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>owl:Thing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
</tr>
<tr>
<td>DUL:PhysicalObject</td>
<td></td>
</tr>
<tr>
<td></td>
<td>101</td>
</tr>
<tr>
<td>ssn:Sensor</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ssn:SensingDevice</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>dbo:Engine</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>DUL:SocialObject</td>
<td></td>
</tr>
<tr>
<td></td>
<td>110</td>
</tr>
</tbody>
</table>
LITEMAT EXAMPLE

Concepts

owl:Thing

DUL:PhysicalObject

ssn:Sensor

ssn:SensingDevice

dbo:Engine

DUL:SocialObject

Compression

1

10100

10101

10110

110
## LITEMAT EXAMPLE

### Concepts

<table>
<thead>
<tr>
<th>owl:Thing</th>
<th>Compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUL:PhysicalObject</td>
<td>1</td>
</tr>
<tr>
<td>ssn:Sensor</td>
<td>101</td>
</tr>
<tr>
<td>DUL:SocialObject</td>
<td>101010</td>
</tr>
<tr>
<td>ssn:SensingDevice</td>
<td>101011</td>
</tr>
<tr>
<td>dbo:Engine</td>
<td>10110</td>
</tr>
<tr>
<td></td>
<td>110</td>
</tr>
</tbody>
</table>
Identify the subclasses specific to a concept in a specific interval

- e.g. $[40, 48] = \text{all physical objects}$
Query reformulation:

- SELECT ?x WHERE { ?x rdf:type DUL:PhysicalObject . }
  FILTER (?v >= 40 && ?v < 48) }

Usage in WAVES:
- Encoding of streams and static knowledge bases
- Integration in PatBin
Materialization and query rewriting over compressed forms
WHY PATBIN?

➢ Motivation
  ● Exploit repetitive information in streams & queries
  ● Querying compressed data

➢ Principle
  ● Combination of materialization and query rewriting
  ● Decomposition of the triples in Pattern - Binding:
    ■ Patterns compressed, using Litemat
    ■ Bindings holding the objects
PATBIN PRINCIPLE

➢ **Pattern**
  - Use the predicates’ identifiers from LiteMat
  - Format preserving the triples interlinking
    - example: 16:21:37:(29)

➢ **Binding**
  - Storage of the objects
    - example: “Q250”;”Hubies haut”; ;”Pascal”

➢ **Advantages**
  - Pattern repetitive for streams (no regeneration)
  - No need for decompression
PATBIN EXAMPLE: STREAM

➢ Stream (in blue):
  
  _:x1  id  “Q250”
  _:x1  date  30/03/2017
  _:x1  pressureMeasure  _:x2
  _:x2  value  4.5

➢ Pattern:

  Ps = 38:40:56:(42)

➢ Binding:

  Bs = 30/03/2017;"Q250"; ;4.5

LiteMat dictionary

  date  =  38
  id    =  40
  value =  42
  hLocation =  54
  pMeasure =  56
PATBIN EXAMPLE: QUERY

Query (in red):

```sql
SELECT ?x ?v
FROM ...
WHERE {
  ?s id ?x .
  ?s hasLocation "Platform1" .
  ?s pressureMeasure ?y .
  ?y value ?v .
}
```

Pattern:

\[ Pq = 40:54:56:(42) \]

Binding:

\[ Bq = ;"Platform1"; ; \]
Need of materialization (in green):

- $Ps \cap Pq = Pi$
  - $38:40:56:(42) \cap 40:54:56:(42) = 40:56:(42)$
- $Pq - Pi = Pm$
  - $40:54:56:(42) - 40:56:(42) = 54$

Fetch the missing information:

- SELECT ?y
WHERE {
  ?x id “Q250” .
  ?x hasLocation ?y .
  ?x pressureMeasure ?z .
}
PATBIN EXAMPLE

➢ Enrich the stream triples (in light blue):
  ● $38:40:56:(42) + 54 = 38:40:54:56:(42)$ $\Rightarrow$ P’s
  ● $30/03/2017;”Q250”; ;4.5 + “Platform1” = 30/03/2017;”Q250”;”Platform1”; ;4.5$ $\Rightarrow$ B’s

➢ Verify the compatibility:
  ● $Pq \subseteq P’s = true$
  ● $Bq \subseteq B’s == true$?
    ○ $;”Platform1”; ; \subseteq 30/03/2017;”Q250”;”Platform1”; ;4.5$ = true

➢ Give the result:
  ● “Q250” 4.5
CONCLUSION
CONCLUSION

➢ Waves project
  • RSP engine
  • Use case: drinkable water network management

➢ LiteMat
  • Compression with identifiers
  • Entity identifiers represent the ontology semantics

➢ PatBin
  • Efficient reasoning scheme
  • Combines materialization, query rewriting and compression based on LiteMat
FUTURE WORK

➢ LiteMat
  - Support of RDFS++
    - *owl:sameAs*, transitive and inverse properties

➢ PatBin
  - Support of SPARQL operators
    - ✓ aggregates (MIN/MAX, SUM, AVG), ORDER, GROUP, HAVING
    - ✗ OPTIONAL, FILTER, ...
# LITEMAT PERFORMANCES

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Duration (sec.)</th>
<th>Added triples (%)</th>
<th>Deleted triples (%)</th>
<th>throughput (triples/sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUBM1K</td>
<td>172.7</td>
<td>0</td>
<td>0</td>
<td>466,552</td>
</tr>
<tr>
<td>LUBM10K</td>
<td>1,755.8</td>
<td>0</td>
<td>0</td>
<td>307,139</td>
</tr>
<tr>
<td>DBPedia</td>
<td>29.2</td>
<td>0</td>
<td>0</td>
<td>478,129</td>
</tr>
<tr>
<td>Wikidata</td>
<td>859.2</td>
<td>0</td>
<td>0.04</td>
<td>77,240</td>
</tr>
</tbody>
</table>

**Ontology-based encoding with Lite materialization**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Duration (sec.)</th>
<th>Added triples (%)</th>
<th>throughput (triples/sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUBM1K</td>
<td>176.6</td>
<td>38.15</td>
<td>460,917</td>
</tr>
<tr>
<td>LUBM10K</td>
<td>3,136.8</td>
<td>38.2</td>
<td>272,796</td>
</tr>
<tr>
<td>DBPedia</td>
<td>57</td>
<td>13.6</td>
<td>356,348</td>
</tr>
<tr>
<td>Wikidata</td>
<td>1,482</td>
<td>57.9</td>
<td>123,571</td>
</tr>
</tbody>
</table>

**Ontology-based encoding with full materialization**

<table>
<thead>
<tr>
<th>Query</th>
<th>Lite mat.</th>
<th>Full mat.</th>
<th>No mat.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>15.6</td>
<td>20.2</td>
<td>21.8</td>
</tr>
<tr>
<td>Q2</td>
<td>15.6</td>
<td>17.2</td>
<td>21.6</td>
</tr>
<tr>
<td>Q3</td>
<td>72.2</td>
<td>99.2</td>
<td>81.4</td>
</tr>
<tr>
<td>Q4</td>
<td>50.2</td>
<td>79.8</td>
<td>27.2</td>
</tr>
</tbody>
</table>

**Query times (in sec.) for Lite/Full/No materialization**
### PATBIN PERFORMANCES

<table>
<thead>
<tr>
<th>5 triples</th>
<th>10 triples</th>
<th>25 triples</th>
<th>50 triples</th>
<th>100 triples</th>
</tr>
</thead>
<tbody>
<tr>
<td>383</td>
<td>387</td>
<td>394</td>
<td>397</td>
<td>401</td>
</tr>
<tr>
<td>PatBin time</td>
<td>PatBin time</td>
<td>PatBin time</td>
<td>PatBin time</td>
<td>PatBin time</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>RDSZ time</td>
<td>RDSZ size</td>
<td>PatBin size</td>
<td>PatBin size</td>
<td>PatBin size</td>
</tr>
<tr>
<td>312</td>
<td>370</td>
<td>425</td>
<td>523</td>
<td>750</td>
</tr>
<tr>
<td>13</td>
<td>29</td>
<td>89</td>
<td>184</td>
<td>382</td>
</tr>
</tbody>
</table>

Signature generation performance for RDSZ and PatBin, time of compression in microseconds, size of the signature in bytes.
PATBIN PERFORMANCES

Graph matching performance

Materialized RDF Stream and reformulated continuous query