Elastic and scalable processing of Linked Stream Data in the Cloud

Danh Le-Phuoc*, Hoan Nguyen Mau Quoc, Chan Le Van, and Manfred Hauswirth
INSIGHT NUI Galway (DERI),
National University of Ireland
Processing Linked Stream Data In A Nutshell

A. Linked Open Data cloud

B. Query

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

C. Sensor Stream Data

Continuous Query

Pre-processing

Optimization

Execution

Answer

Static RDF datasets

Stream Data in RDF
Limitations of standalone engines

• Scalability issues of standalone Linked Stream Data Processing engines
  – Slow throughput (100 triples/second for 100-1000 concurrent queries)
  – Small RDF datasets (1-10 million triples)
  – Small sliding windows (1000 records)

• Disadvantages of processing on a single computer
  – Upper limitation of hardware configuration
  – Total cost of ownership
Scale Processing Linked Stream Data in the Cloud

• Big data requirements
  – **Velocity**: up to 100k triples/second for 10k concurrent queries
  – **Volume**: big windows (millions of records) and big RDF static datasets (billions of triples)

• Elasticity requirements
  – Add processing power online
  – Horizontal scalability

• Use “pay-per-use” networked computer infrastructure: Amazon EC, Google Cloud, Microsoft Azure, etc.
As a result, similar to MJOIN \([\text{MJOIN}]\), the evaluation of a new mapping requires the upper operator which consumes the MJOIN output as its input buffer. The negative tuple approach, of in the event of expiration, the outputs of MJOIN are stored in a buffer that is then recursively used to probe the rest to generate final mappings. Each probing sequence will stop at a new mapping buffer. The counter from the index is then used to stop the scan operation earlier.

Because the multiway join is symmetric, without loss of generality, we extend the incremental equations as shown above, this operator is supported by the high throughput in the method of Algorithm 6.2. INCREMENTAL EVALUATION FOR SLIDING WINDOWS

\[ R \cap n \]

\[ \sigma \]

\[ \text{Global Scheduler} \]

\[ \text{Local Scheduler} \]

\[ \text{Operator Containers} \]

**Logical continuous query network**

**Abstract elastic execution architecture**

**CQELS** : Continuous Query Evaluation over Linked Stream

**CQELS-QL** : CQELS Query Language
Parallelizing approach

- Parallelize sub operator-pipelines
- Parallelize I/O access: lookup on static data, dictionary
- I/O locality: co-locate data operations with data
- Use compact processing states
  - Encode processing states in fixed size integers
  - Compress and transfer encoded data in batches with processing “guidelines”
- Splitting and grouping scheduling strategy
  - Split incoming stream in batches of processing tasks
  - Grouping data operations in a same processing node
CQELS Cloud architecture

Coordinate pipelines of continuous tasks

Parallelize I/O access
CQELS Cloud building blocks

Decoder

Dynamic Executor

Operator pipelines

Adaptive Optimiser

Window Buffer Manager

Cache Manager

Encoder

Input Manager

Cache Fetcher

New RDF stream data

Static data
Parallel incremental computing algorithms

- Stateless operators: selection, filter, etc
- Dictionary encoding/decoding operators
- Stateful windowing operators
  - Multiway join
  - Aggregation
  - ...

Parallel multiway join algorithm: splitting to symmetric parallel join tasks

\[ \mu_1 \bowtie (R^2 \cdots \bowtie R^n) \]

\[ \mu_1 \circ \mu_2^i \]

\[ \{\mu_1\} \times \sigma_{\mu_1}(R^2) \]

\[ \mu_1 \]

\[ R^1 \]

\[ \mu_2 \]

\[ R^2 \]

\[ \mu_n \]

\[ R^n \]
Parallel multiway join algorithm: grouping on shared windows

Consider the multiple join operator for $R^4$. Each query over a stream $R$ contains a window size equal to the maximum window size over all $R$. Therefore, the join part only has to execute a single multiway query for $j=1..k$. For $j=2$, the routing part, each resulting mapping has to be routed to the query containing it. The multiple join algorithm only provides a fixed probing sequence for processing new mapping $W_i$.

$Q_1 = W_1^1 \bowtie W_1^2$

$Q_2 = W_2^1 \bowtie W_2^2$

$Q_3 = W_3^1 \bowtie W_3^2$
Experimental setup

• **Amazon EC2 Cluster:**
  – “Medium" EC2 instances, i.e., 3.5 GB RAM, 1 virtual core with 2 EC2 Compute Units, 410 GB instance storage, 64 Bit platform, moderate I/O performance.
  – 1 Nimbus node, 2 Zookeeper nodes, 1HBase Master node and 2-32 OC nodes

• **Stream data : use LSBench to simulate Social Network streams**

• **Experiments**
  – Operator scalability
  – Concurrent queries
Operator scalability

Throughput (Triples/s) vs No. of OC nodes

Throughput of Operators

Peak network traffic

Bandwidth (bytes)

4GBit/sec

5/7 20:00
5/7 21:00
5/7 22:00
Concurrent queries

Q1: simple matching

Q4: 3-way join on streams

Q5: 3-way join on streams and static data

Q10: Join and Aggregation
Conclusions

• CQELS Cloud
  – Horizontal and elastic scalability
  – Throughputs: 100,000s of triples per second for 10,000s of concurrent queries on a cluster of 32 medium EC2 nodes (100k-10millions records/window)

• Future work
  – Optimization
  – Adaptive load balancing