Strider: A Hybrid Distributed RDF Stream Processing Engine

Xiangnan Ren and Olivier Curé

ATOS, Bezons, France
LIGM, CNRS, UMR-8049, France
Stream processing, Why?

Stream is everywhere
 CONTEXT

 ❑ Project
   - Sponsored by WAVES FUI 17
   - Industrial Water Resource Management

 ❑ Streaming data as RDF, Why?
   - Facilitate heterogeneous data integration
   - Support stream reasoning
RDF Stream Processing (RSP)

- **Centralized Systems:** C-SPARQL, CQELS, SparqlStream, ETALIS, etc

- **Distributed Systems:** CQELS-Cloud, Katts

None of them cover scalability, reasoning and adaptivity
Adaptivity, why?

- Data structure may change over time
Adaptivity, why?

- Data structure may change over time
SYSTEM DESIGN

ARCHITECTURE

SYNTAX
SYSTEM DESIGN

Data Source → RDF Event Converter → kafka → Broker 1, Broker 2, Broker 3 → RDF Stream to DataFrame Converter → Spark → Streaming → Request Layer

Query 1, Query 2, Query 3

STRIDER
SYSTEM DESIGN

SYNTAX


REGISTER { QueryId [Q8] Reasoning [FALSE] SPARQL [
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn>
prefix cuahsi: <http://www.cuahsi.org/waterML/>
SELECT ?s ?o2 ?o3
WHERE {
  ?s ssn:hasValue ?o1; ssn:hasValue ?o2
  ssn:hasValue ?o3.
  ?o1 rdf:type cuahsi:flow.
  ?o2 rdf:type cuahsi:temperature.
  ?o3 rdf:type cuahsi:chlorine.
} ]
}
QUERY PROCESSING

HYBRID STRATEGY

AQP

F-AQP & B-AQP
HYBRID STRATEGY

- Rule-Based
  - Static Optimization
- Cost-Based
  - Dynamic Optimization
QUERY PROCESSING

HYBRID STRATEGY

- Rule-Based
  - Static Optimization
  - Dynamic Optimization

- Cost-Based
  - Decision Maker
    - Forward
    - Backward

Optimizer
SELECT ?s ?o2 ?o3
WHERE {
  ?s ssn:hasValue ?o1;  (tp1)
  ssn:hasValue ?o2  (tp2)
  ssn:hasValue ?o3.  (tp3)
  ?o1 rdf:type cuahsi:flow. (tp4)
  ?o2 rdf:type cuahsi:temperature. (tp5)
  ?o3 rdf:type cuahsi:chlorine. (tp6)
}

Undirected Connected Graph (UCG)
QUERY PROCESSING

(a) UCG initialization of $w_n$

(a) UCG initialization of $w_{n+1}$
QUERY PROCESSING

(a) UCG initialization of $w_n$

(a) UCG initialization of $w_{n+1}$
QUERY PROCESSING

(a) UCG initialization of $w_n$

(b) find UCG path of $w_n$

(a) UCG initialization of $w_{n+1}$

(b) find UCG path of $w_{n+1}$
QUERY PROCESSING

(a) UCG initialization of $w_n$
(b) find UCG path of $w_n$
(c)

(a) UCG initialization of $w_{n+1}$
(b) find UCG path of $w_{n+1}$
QUERY PROCESSING

(a) UCG initialization of \( w_n \)

(b) find UCG path of \( w_n \)

(c) Logical plan of \( w_n \)

(a) UCG initialization of \( w_{n+1} \)

(b) find UCG path of \( w_{n+1} \)

(c) Logical plan of \( w_{n+1} \)
QUERY PROCESSING

- Forward & Backward

**Forward** Adaptive Query Processing (F-AQP)

*Query plan generation based on currently collected statistic*

**Forward**
- Generate query logical plan $p_i$ by dynamic programming
- Query Processing at $w_i$

$W_i$
Forward & Backward

Backward Adaptive Query Processing (B-AQP)

Query plan generation based on previously collected statistic
## QUERY PROCESSING

### B-AQP & F-AQP

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Advantage</th>
<th>Drawback</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-AQP</td>
<td>No dynamic programming overhead</td>
<td>Query plan generation through previously-collected statistics</td>
</tr>
<tr>
<td>F-AQP</td>
<td>Query plan generation through real-time collected statistics</td>
<td>Overhead for dynamic programming, side-effects caused by pipeline interruption</td>
</tr>
</tbody>
</table>
B-AQP & F-AQP

- **Adaptive optimization**
  - **Forward**
    - Generate query logical plan $p_0$ by dynamic programming
    - Query Processing at $w_0$
  - **Backward**
    - Statistics gathering over data stream at $w_0$
    - Generation of query logical plan $p_1$ at $w_1$
  - **Forward**
    - Apply query logical plan $p_1$ at $w_1$
    - Trigger Forward for $w_2$
  - **Forward**
    - Generate query logical plan $p_2$ by dynamic programming
    - Query Processing at $w_2$
    - Trigger Forward for $w_3$

- **Static optimization**
  - Static optimization is triggered, cancel adaptive optimization
  - Apply query logical plan $p_{stat}$
  - Statistics gathering over data stream at $w_3$
  - Generation of query logical plan $p_4$ at $w_4$

**Events**
- $w_0$: $\delta_0 < \varepsilon$
- $w_1$: $\delta_1 > \varepsilon$
- $w_2$: $\delta_2 > \varepsilon$
- $w_3$: $\delta_3 < \varepsilon$
- $w_4$, ...
EVALUATION
EVALUATION

- **Metrics**: Throughput and Latency

- **Datasets**: 9 queries, SRBench, Waves Datasets

- **Hardware**: 3 Nodes of dataflow (c4Xlarge)  
  6 nodes of Spark Cluster
## EVALUATION

### Throughput: triples/second

<table>
<thead>
<tr>
<th></th>
<th>Q1 (SR)</th>
<th>Q2 (SR)</th>
<th>Q3 (SR)</th>
<th>Q4 (W)</th>
<th>Q5 (W)</th>
<th>Q6 (W)</th>
<th>Q7 (W)</th>
<th>Q8 (W)</th>
<th>Q9 (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strider (D-S)</td>
<td>2620005</td>
<td>1431808</td>
<td>495380</td>
<td>3116147</td>
<td>939506</td>
<td>325328</td>
<td>1555542</td>
<td>422591</td>
<td>194225</td>
</tr>
<tr>
<td>Strider (D-A)</td>
<td>2620005</td>
<td>1164471</td>
<td>838531</td>
<td>3116147</td>
<td>1148896</td>
<td>886769</td>
<td>2005617</td>
<td>524181</td>
<td>387742</td>
</tr>
<tr>
<td>Strider (L-S)</td>
<td>837860</td>
<td>510231</td>
<td>382972</td>
<td>965986</td>
<td>370904</td>
<td>264665</td>
<td>538524</td>
<td>127276</td>
<td>91710</td>
</tr>
<tr>
<td>Strider (L-A)</td>
<td>837860</td>
<td>608228</td>
<td>329683</td>
<td>965986</td>
<td>481945</td>
<td>310039</td>
<td>538524</td>
<td>98573</td>
<td>68345</td>
</tr>
<tr>
<td>C-SPARQL</td>
<td>13142</td>
<td>12008</td>
<td>10934</td>
<td>56179</td>
<td>40469</td>
<td>34882</td>
<td>54917</td>
<td>49985</td>
<td>33988</td>
</tr>
<tr>
<td>CQELS</td>
<td>16817</td>
<td>14211</td>
<td>10637</td>
<td>16541</td>
<td>0</td>
<td>19574</td>
<td>19534</td>
<td>19472</td>
<td>6527</td>
</tr>
</tbody>
</table>

*Million-Level Throughput*


**EVALUATION**

- **Latency:** millisecond

![Query Latency Chart]

<table>
<thead>
<tr>
<th>Query</th>
<th>Strider (D-S)</th>
<th>Strider (D-A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1 (SR)</td>
<td>442</td>
<td>442</td>
</tr>
<tr>
<td>Q2 (SR)</td>
<td>835</td>
<td>1088</td>
</tr>
<tr>
<td>Q3 (SR)</td>
<td>2455</td>
<td>1502</td>
</tr>
<tr>
<td>Q4 (W)</td>
<td>399</td>
<td>399</td>
</tr>
<tr>
<td>Q5 (W)</td>
<td>1337</td>
<td>1100</td>
</tr>
<tr>
<td>Q6 (W)</td>
<td>3736</td>
<td>1338</td>
</tr>
<tr>
<td>Q7 (W)</td>
<td>832</td>
<td>631</td>
</tr>
<tr>
<td>Q8 (W)</td>
<td>6004</td>
<td>3754</td>
</tr>
<tr>
<td>Q9 (W)</td>
<td>6996</td>
<td>3267</td>
</tr>
</tbody>
</table>

**Second-Level Delay**
Adaptive Optimization

Throughput Comparison

Throughput (triples/second)

Time (minutes)
OTHER FEATURES

☐ SUPPORTED SPARQL OPERATORS

**Supported Query Types:**
Select, Construct, Ask

**Supported Ops:**
Projection, Join (Binary), Optional (Left-Outer-Join), BGP, Union, Filter, Distinct, Group

**Supported Exprs:**
Aggregation, LogicalAnd, LogicalOr, Bound, LogicalNot, Equals, NotEquals, GreaterThan, GreaterThanOrEqual, LessThan, LessThanOrEqual, NodeValue, ExprVar, etc.
OTHER FEATURES

REASONING SUPPORT (RDFS & OWL:sameAs)

LiteMat: a scalable, cost-efficient inference encoding scheme for large RDF graphs, IEEE, BigData, 2015

Strider: An Adaptive, Inference-enabled Distributed RDF Stream Processing Engine, VLDB Demo, 2017
CONCLUSION & FUTURE WORK

- Available on github: https://github.com/renxiangnan/strider
- Reduce query latency
- Multiple queries optimization
- Reactive system
Thank you