Tutorial on RDF Stream Processing 2016
M.I. Ali, J-P Calbimonte, D. Dell'Aglio, E. Della Valle, and A. Mauri
http://streamreasoning.org/events/rsp2016

RSP Optimisation Techniques
M.I. Ali
ali.intizar@insight-centre.org
http://intizarali.org
@intizarali
Data Streams are Everywhere

- Smart Cities and IoT are leading to an era of streaming world
- Sensors and mobile devices are producing an enormous amount of data
- Mostly in streaming fashion
Introducing Semantics in Data Streams

- Why RDF Data Streams?
  - Interoperable (easy integration)
  - Machine Readable
  - Reasoning
  - On-demand discovery
  - Ideal for the web
  - Dereferencing
The Goal
CityPulse: Real-time IoT Data Analytics and Large Scale Data Analytics for Smart Cities Applications

- CityPulse aims to support the integration of dynamic data sources and context-dependent on-demand adaptations of processing chains during run-time.

- CityPulse aims to bridge the gap between the application technologies on the IoT and real world data streams.

- It will use Cyber-Physical and Social data and will employ big data analytics and intelligent methods to aggregate, interpret and extract meaningful knowledge and perceptions from large sets of heterogeneous data streams.
CityPulse: Real-time IoT Data Analytics and Large Scale Data Analytics for Smart Cities Applications
Smart City Applications

Alert Systems
Connected Cars
Mobile-Centric
Public Transit
Smart Automation

The Smart City

http://streamreasoning.org/events/rsp2016
Is RSP Ready for Action?

- **Available Engines**
  - CQELS
  - C-SPARQL
  - SPARQLStream
  - ...

- **Processing capabilities tests**
  - Benchmarks
    - LS
    - SR
    - CSR

- **Performance and Scalability**

http://streamreasoning.org/events/rsp2016
Is RSP Ready for Action?

- RSP is still in its cradle
- On-going work for query language and semantics
- Existing RSP engines are not more than prototypes
- Benchmarking for performance and scalability testing in control environment
Challenges for RSP Optimisation

- Data Distribution
  - Data produced by streams is highly distributed

- Unpredictable Data Rate
  - Stream observation rate is variable
  - Stream Bursts
Challenges for RSP Optimisation

- Number of Concurrent queries
  - A large number of audience or end users e.g. Citizens of a smart city

- Background Data Integration
  - Streaming queries process a combination of streaming and static knowledge
  - Currently static knowledge base is processed in memory
Challenges for RSP Optimisation

- **Quasi-static Data**
  - Fetch and locally process can result into outdated results for quasi-static data

- **On-demand Discovery**
  - Stream Processing operate in a frequently changing world
  - Data and applications change quite frequently

- **Adaptation**
  - Streaming queries in dynamic environment need continuous monitoring
How can we optimise RSP?

- Benchmarking
- Resource Optimisation
- Resource Sharing/Join Optimisation
- Scalability
- Load Balancing
- Hybrid Reasoning
Benchmarks

- SR Bench
- LS Bench
- CSR Bench

Benchmarking Infrastructure

- CityBench
- YABench
- Heaven
CityBench Benchmarking Suite - CTI

Configurable Testbed Infrastructure (CTI)

CityBench Queries

Smart City Data Streams

Dataset Configuration Module

Query Configuration Module

Performance Evaluator

Static Datastore

RSP Engine

Benchmark Results

Smart City Applications

CityBench Queries

Smart City Applications
CityBench Benchmarking Suite

- CityBench is designed to evaluate RSP engines for Smart City Applications

- It comprises of
  - 7 real-time smart city data sets containing live RDF streams
  - Configurable Testbed Infrastructure with 6 parameters
  - 13 queries for 3 smart city applications e.g. Travel Planner, Parking Finder and CityDashboard
CityBench Benchmarking Suite

- CityBench Datasets
  - Vehicle Traffic
  - Parking
  - Weather
  - Pollution
  - Cultural Events
  - Library Events
  - User Location Stream
CityBench Benchmarking Suite- CTI

- Configuration Parameters
  - Changes in Input Streaming Rate
  - Play Back Time
  - Variable Background Data Sizes
  - Number of Concurrent Queries
  - Number of Streams within a Single Query
  - Selection of the RSP Engine
CityBench Evaluation

- We evaluated 2 state of the art RSP engines
  - CQELS
  - C-SPARQL

- Both engines were tested for their
  - Latency
  - Memory Consumption
  - Completeness

- Different settings by fine tuning CTI Parameters
  - Number of queries, users, background data size etc.
CityBench Evaluation: Latency

- Latency over Increasing Number of Input Streams

![Graph showing latency over increasing number of input streams. The x-axis represents experiment time in minutes, ranging from 0 to 15. The y-axis represents latency in milliseconds, ranging from 0 to 6000. The graph includes lines for Q10_8-csparql, Q10_2-csparql, Q10_2-cqels, Q10_5-csparql, and Q10_5-cqels. Each line represents different query execution scenarios.]
CityBench Evaluation : Latency

- Latency over Increasing Number of Concurrent Queries
  - CQELS: Q1, Q5 and Q8

![Graph showing latency over experiment time for different queries](chart1.png)

![Graph showing latency over experiment time for different queries](chart2.png)
CityBench Evaluation: Latency

- Latency over Increasing Number of Concurrent Queries
  - C-SPARQL: Q1, Q5 and Q8
CityBench Evaluation: Memory Consumption

- Memory Consumption over Increasing the Number of Concurrent Queries
CityBench Evaluation: Memory Consumption

- Memory Consumption over Increasing the Size of Background Data

![Graph showing memory consumption over experiment time](image-url)
CityBench Evaluation: Completeness

- Memory Consumption over Increasing the Size of Background Data
RDF Stream Processing (RSP) : Challenges

- Optimal Data Source Discovery
  - Streams are everywhere
  - Multiple data streams can answer the same query
  - Optimal data stream selection
  - Catering for user-defined constraints and preferences

- On-Demand Stream Federation
  - Automated composition of primitive data streams to answer complex queries

- Adaptation
  - Data source properties can change over time
  - Make sure selected sources remain “optimal” throughout life cycle of the query
Stream Discovery, Federation and Adaptation

- **Stream Discovery**
  - Data interoperability:
    - Semantic descriptions (ontologies and annotations)
  - Interface interoperability:
    - Streams as event services (service discovery)

- **Stream Federation**
  - Efficient processing of complicated event logics
    - Data Stream Management Systems
    - Complex Event Processing

- **Adaptation**
  - Continuous monitoring to observe constraints violations
  - Trigger adaptation mechanism to select new optimal data stream

Semantic Web
Service Oriented Architectures
DSMS and CEP
Continuous constraint checking

Semantic Web + Service Oriented Architecture + Complex Event Processing

http://streamreasoning.org/events/rsp2016
Motivation – Smart City Applications

- Detecting complex events in real-time by answering continuous queries over data streams.

- **Input data heterogeneity**
  + Background knowledge integration

- **Output data reusability**
  + Platform independency

- **Platform independence**

---

**Complex Event Processing (CEP)**

- Event engine
- Event Consumer
- Event Providers
- Complex Event (CE)
- RDF Stream Processing (RSP)
- Semantic Event Service (SES)

**RDF Stream Processing (RSP)**

- Ontologies & Service Descriptions
- Service Wrapper
- E₁ ∧ E₂ ∧ E₃
- E₁ subClassOf E₄
- C₁: Acc ≥ 90%

**Semantic Event Service (SES)**

- C₂: Acc ≥ 85%
- C₃: Acc ≥ 95%

**Motivation - Smart City Applications**

- Detecting complex events in real-time by answering continuous queries over data streams.

- **Input data heterogeneity**
  + Background knowledge integration

- **Output data reusability**
  + Platform independency

---

**Graphical Representation**

- Event Providers
- Internet of Things
- Internet of People
- Automatic adaptations?
Automated Complex Event Implementation System

Knowledge Base
- QoI/QoS
- Stream Description
- Data Mgmt, Indexing, Caching

Application Interface
- User Input
- Event Request
- ACEIS Core
  - Resource Discovery
  - Event Service Composer
  - Composition Plan
  - Resource Management

Data Federation
- Query
- Results
- Subscription Manager
  - Query Transformer
  - Query Engine
- Data Store

IoT Data Stream

Social Data Stream

Semantic Annotation

Constraint Validation
- Constraint Violation

Resource Management

Composition Plan

Adaptation Manager

Validation
Constraint
Violation
Adaptation
Manager

Data Store

IoT Data Stream

Social Data Stream

Knowledge Base
- QoI/QoS
- Stream Description
- Data Mgmt, Indexing, Caching
Summary of the Approach

- How to describe complex event services?
  - Create an Event Service Ontology with Event Patterns.

- How to determine if two event patterns are functionally equivalent?
  - Create and compare canonical event patterns to find substitutes.

- How to create event compositions and choose the optimal?
  - Top-down traverse to find functionally-equivalent canonical patterns.

- How to derive event service compositions efficiently?
  - Construct and utilize an Event Reusability Hierarchy for event service composition.

- How to ensure best remains best?
  - Monitor user defined constraints and trigger adaptation mechanism if constraints are violated.
Complex Event Service Ontology

Namespaces:
default: <http://www.insight-centre.org/ces#>
rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
owls: <http://www.daml.org/services/owl-s/1.2/Service.owl#>
owls-sp: <http://www.daml.org/services/owl-s/1.2/ServiceParameter.owl#>

Legend:
- Class
- Object property
- subClassOf
- Data property
Annotation of sensor streams

• A sensor service description is annotated as:

\[ s_{\text{desc}} = (t_d, g, q_d, P_d, \text{FoI}_d, f_d) \]

  • type grounding QoS Observed Properties
  • Feature Of Interest

  \[ P_d \rightarrow \text{FoI}_d \]

• Similarly, a sensor service request is annotated:

\[ s_r = (t_r, P_r, \text{FoI}_r, f_r, \text{pref}, C) \]

  • type Requested Properties
  • Feature of Interest
  • NFP Constraint and Preferences

  \[ P_d \rightarrow \text{FoI}_d \]
On-Demand Stream Federation

- **Event Request:**
  - User/Application defines an event request using CES Ontology

- **Procedure:**
  - Derive canonical forms of event patterns of CESs.
  - Apply tree isomorphism algorithms over the canonical event patterns and the event request to identify reusable or equivalent event patterns.
  - Generate all possible composition plans.
  - Aggregate NFPs and compare aggregated NFP values against constraints on event request to filter out unsatisfied composition plans.
  - Optimization using Genetic Algorithm (GA)
  - Rank the remaining composition plans based on preferences (soft constraints).
On-Demand Stream Federation

getCompletePattern()
- Create event reusability hierarchy
- Reusable relation: $R(ep_1, ep_2)$ holds if $R_d(ep_1, ep_2)$ or $R_i(ep_1, ep_2)$ holds.
Stream Federation: Composition Plan Generation

Event Service 1
- type = e1
- loc = loc1

Event Service 2
- type = e2
- loc = loc2

Event Service 3
- type = e3
- loc = loc3

Event Service 4
- type = e4
- loc = loc4

Composition Plan
- OR
  - SEQ
    - e4
    - loc = loc4
  - e3
    - loc = loc3

Query
- OR
  - SEQ
    - e1
    - e2
    - e3
Stream Federation: Composition Plan Generation

- **Brute-Force Enumeration**:  
  - global optimum, poor scalability.

  - near-optimal global planning, improved scalability,  
  - requires QoS metrics to be linear,  
  - operates on a fixed set of service classes in a composition plan,  
  - does not perform well in dynamic environments.

- **Genetic Algorithm** (Canfora et al. 2004, Wu et al. 2013):  
  - near-optimal global planning, good scalability,  
  - does not require linear QoS metrics,  
  - can provide composition plans with services with different granularity levels,  
  - can adapt to changes effortlessly,  
  - can achieve ~89% optimal results in 0-2 seconds using default settings in our approach.
Stream Federation: Composition Plan Generation

1. Define fitness function
2. Population initialisation
3. Selection of individuals based on fitness
   - Crossover genetic encodings of selected individuals
   - Mutation
4. Set derived results as the next generation
5. Termination

QoS aggregation schema based on patterns
ERH based
Stream Federation: Composition Plan Generation

- **ERF Space**
- **CCP Space**
- **Query**
- **Reusable Node**

![Diagram](http://streamreasoning.org/events/rsp2016)

**Cross Over**

- **chromosome for P_1**
- **chromosome for C_1**
- **chromosome for P_2**

**Fig.3: Example of genetic encoding and crossover operation**
Adaptation in Stream Processing

Why we need adaptation?

- Ensure that the “best” remains the “best”
- Improves robustness

Technical Adaptation in Stream Federation

- Monitoring quality updates of streaming sources
- Evaluate criticality of the update (based on query-related constraints and requirements)
- React to this change (discover new streaming sources)
- Monitoring quality updates of streaming sources
- Evaluate criticality of the update (based on query-related constraints and requirements)
- React to this change (discover new streaming sources)
Constraint Validation

- Degradation in stream quality is considered as constraints violation
- Better performance but can lead to the possibility of having better quality streams not considered

Adaptation Manager deals with constraint violations:

- Switching to alternative streams: only candidate streams selected by composition plan are considered as substitute for stream switching
- Re-generation of the composition plan and consideration of all the available (registered) stream.
Goal: transform the composition plan into a stream query evaluated by a stream reasoning engine over RDF data streams

- **Requirements:**
  - Matching event pattern operators to stream query operators
  - Transformation Algorithm

- **Alignments for CQELS, C-SPARQL and ETALIS:**

<table>
<thead>
<tr>
<th>ACEIS</th>
<th>E</th>
<th>And</th>
<th>Or</th>
<th>Seq</th>
<th>Rep₀</th>
<th>Repₙ</th>
<th>Sel</th>
<th>Filter</th>
<th>Window</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQELS</td>
<td>SGP</td>
<td>⊗</td>
<td>⊗</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>BGP+proj</td>
<td>Filter</td>
<td>Window</td>
</tr>
<tr>
<td>C-SPARQL</td>
<td>BGP</td>
<td>⊗</td>
<td>⊗</td>
<td>fₜ</td>
<td>⊗ +fₜ</td>
<td>fₜ⁺</td>
<td>BGP+proj</td>
<td>Filter</td>
<td>Window</td>
</tr>
<tr>
<td>ETALIS</td>
<td>BGP</td>
<td>⊗</td>
<td>⊗</td>
<td>SeqJoin</td>
<td>⊗ +SeqJoin</td>
<td>SeqJoin⁺</td>
<td>BGP+proj</td>
<td>Filter</td>
<td>getDuration()</td>
</tr>
</tbody>
</table>

- Sequence and Repetition not supported by CQELS.
- Sensor requests mapped to StreamGraphPattern(CQELS) and GroupGraphPattern(CQELS).
- AND operator mapped to stream join.
- OR operator mapped to OPTIONAL keyword (left-outer-join).
ACEIS in Practice

Input
• Complex Event Request (Function & non-functional properties)
• Sensor Metadata Repository (including quality updates)
• Sensor Data Streams (Semantically annotated data streams)

Process
• Discover relevant data stream (list of candidate data streams)
• Ranking the multiple candidate data streams (evaluate constraints & preferences)
• On-demand Stream Federation (composition plan using BF/GA)

Output
• Federated Output Data Stream

GitHub Source Code:
https://github.com/CityPulse/Stream-Discovery-and-Integration-Middleware
Query Scheduler in ACEIS

- Goal: Deploy multiple RSP engine instances and leverage load balancing techniques to increase the capacity of ACEIS server.

- Multiple load balancing techniques applied:
  - Equalised Queries (EQ): initialize multiple engines upfront and same number of queries deployed on each instance.

Data Federation

Composition Plan

Subscription Manager

Query Transformer

deploy(qid,engineID)

Query Engine

Scheduler

Dispatcher

Query Monitor

getEngine()

engineID

sendStats()

createQueryEngineInstance()
Query Scheduler in ACEIS

- Goal: Deploy multiple RSP engine instances and leverage load balancing techniques to increase the capacity of ACEIS server.

- Multiple load balancing techniques applied:
  - Elastic (EL): define maximum number of queries deployed on an instance and creates instances on demand

```
Data Federation

Composition Plan

Subscription Manager

Query Transformer

deploy(qid,engineID)

Scheduler

Dispatcher

Query Monitor

getEngine()

engineID

sendStats()

createEngineInstance()
```

Figure 9.25: Latency of CQELS engines using EQ

Figure 9.26: Latency of CSPARQL engines using EQ
Query Scheduler in ACEIS

- Goal: Deploy multiple RSP engine instances and leverage load balancing techniques to increase the capacity of ACEIS server.

- Multiple load balancing techniques applied:
  - Balanced Latency (BL): deploy the query on the instance with lowest average latency

```
Data Federation

Composition Plan

Subscription Manager

Query Transformer

deploy(qid,engineID)

Scheduler

Dispatcher

Query Monitor

senderStats()

createEngineInstance()

gEngine()

engineID

page

http://streamreasoning.org/events/rsp2016
```
Query Scheduler in ACEIS

- Goal: Deploy multiple RSP engine instances and leverage load balancing techniques to increase the capacity of ACEIS server.

- Multiple load balancing techniques applied:

![Diagram of query scheduler in ACEIS](image-url)
Query Scheduler in ACEIS

- Performance evaluation with EBL strategy

**Figure 9.32**: Latency of CQELS engines using EBL

**Figure 9.33**: Latency of CSPARQL engines using EBL

- Increased capacity of CQELS from 30 to about 1000
- Increased capacity of CSPARQL from 30 to about 90
- Further RSP Optimisation Opportunities
  - Sharing the resource among multiple concurrent queries
    - One stream buffer shared among multiple queries
    - Common triple patterns in multiple queries handled once
    - Join Optimisation for multiple concurrent queries
Pre-processing Input Queries
Further RSP Optimisation Opportunities

- Trade-off between latency and consistency
  - Querying remote endpoint can be resource intensive
  - SPARQL endpoint contain quasi-static knowledge base can impose restriction on number of service calls
  - Maintenance policy to keep the optimised outcomes from the tradeoff between latency and consistency.
Mediator system: Highest consistency with a latency threshold
Mediator system: Highest consistency with a latency threshold
Further RSP Optimisation Opportunities

- **Trade-off between expressivity and scalability**
  - RSP brings the power of reasoning
  - Reasoning tools need to be optimised to deal with large volume of data having high velocity and variety
  - RSP can support to improve RDF stream reasoning
  - Finding the trade-off between stream processing and stream reasoning can be key
Introduction – Stream reasoning

Stream reasoning

Stream processing
+ Process stream
- Lack complex reasoning tasks

Knowledge Representation & Reasoning
- Mainly on static data
+ Perform complex reasoning tasks
Scalability Vs Expressivity Tradeoff

Expressivity

Data Streams

Stream Reasoning

solution sets

Applications

Scalability

Data Streams

Semantic Complex Event Processing

complex events

Data Streams

Stream Query Processing

relevant events

http://streamreasoning.org/events/rsp2016
The StreamRule idea

- 2-tier approach: not all dynamic data streams are relevant for complex reasoning
- Enrich the ability of complex reasoning over data streams
- Keep the solution scalable
- Leverage existing engines from both stream processing and non-monotonic reasoning research areas
Approach

- Leverage existing engines from both stream processing and non-monotonic reasoning research areas
- Enable adaptation layer to enhance scalability

2-tier approach

Streams

RDF Stream Processing

Query

Filtered Stream

Rule-based Non-monotonic Reasoning

Logic Program

Adaptation

Answers

http://streamreasoning.org/events/rsp2016
Stream Rule

- Query Processing
  - RDF Files (e.g. maps)
  - Web of Data
  - Sensor Streams

- Controller
  - Rule-based Expressive Reasoning
  - Scalability requires adaptation!

- Application
  - LSD Wrappers
  - Stream Rule
    - C-SPARQL
    - CQELS
  - Clingo
    - Logic Program
  - Facts
  - Java

http://streamreasoning.org/events/rsp2016
Conclusions:

- RDF Stream Processing provides an exciting opportunity for applications dealing with heterogeneous data
- RSP engines are evolving and maturing
- Optimisation is critical for RSP to be considered for large-scale real-time applications.
- Distributed large-scale RSP engines
- Language semantics clarity necessary for query optimisation