Outline

- Intro and Objectives
- Brief Intro to Technologies
- Our Approach and Progress
- Future Work
More and more data are published on the Web

Business Intelligence tools need to analyze these data

OnLine Analytical Processing (OLAP) style analysis of Linked Data may help in better decision making

Expected challenges are data collection, integration, data aggregation...
Objectives

Design, develop, and evaluate an approach for performing OLAP over federated RDF sources

- Develop a framework for defining a multidimensional schema of a data cube expressed in RDF vocabularies in a global-as-view manner for further retrieval of data from different data sources
- Develop an approach for executing aggregate SPARQL queries over federated RDF sources
- Design an approach to support partial/result keeping materialization to store the results of previous requests and allow subsequent queries to execute faster
- Extend the query processing capabilities to include the inferred knowledge in materialized aggregate views of the linked data.
Language and Vocabularies

RDF (Resource Description Framework) is a standard model for data interchange on the Web

SPARQL is a query language for RDF. Queries defined in terms of graph patterns that are matched against the directed graph representing the RDF data

QB4OLAP is a special RDF vocabulary for OLAP Cubes on the Semantic Web.

VoID is an RDF Schema vocabulary for expressing metadata about RDF datasets
RDF

Statements about resources in the form of subject-predicate-object expressions

<TurtleExample>

\[ \text{\textless P1\textgreater \ is called \ Jimmy Wales } \]

TurtleExample : <P1> rdf:type foaf:Person .

\[ \text{\textless P1\textgreater \ foaf:name "Jimmy Wales" } \]

\[ \text{\textless P1\textgreater \ foaf:mbox \textless mailto:jwales@bomis.com\textgreater } \]

RDF extends the linking structure of the Web to use **URIs to name the relationship** between things
SPARQL Query Language

Developed by W3C Data Access Working Group

Queries defined in terms of graph patterns that are matched against the directed graph representing the RDF data

Ex: Show name and email of a person

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
WHERE {
  ?person a foaf:Person.
  ?person foaf:name ?name.
}

<table>
<thead>
<tr>
<th>Name</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jimmy Wales</td>
<td><a href="mailto:jwales@bomis.com">mailto:jwales@bomis.com</a></td>
</tr>
</tbody>
</table>
SPARQL Query Language

4 types of queries to retrieve (read) data:
- SELECT
- CONSTRUCT
- ASK
- DESCRIBE
QB4OLAP

QB4OLAP is a special RDF vocabulary for OLAP Cubes on the Semantic Web

-- Data structure definition and dimensions
exqb:NorthwindDW a qb:DataStructureDefinition ;
qb:component [qb:dimension exqb:Employee] ;

-- Definition of measures

-- Attributes
exqb4o:CompanyName a qb:AttributeProperty ;
rdfs:comment "Company Name"@en .

-- Dimension
exqb4o:CustomerDim a rdf:Property, qb:DimensionProperty .
VoID

- Vocabulary of Interlinked Datasets for expressing metadata about RDF datasets
  - General metadata (following the Dublin Core model)
  - Access metadata – access to RDF data using various protocols
  - Structural metadata – structure and schema of datasets
  - Description of links between datasets – relation among multiple datasets
Source Discovery/Schema Builder is responsible for the discovery of data sources and construction of the Global Conceptual Schema.
Global Conceptual Schema defines the high-level view of the system - expressed in QB4OLAP, VoID
Semantic Query Processor (using the Global Conceptual Schema) converts a user query to the new format and passes it to the Federated Query Processor.
Proposed System

Federated Query Processor retrieves data from several federated data sources
Federated Query Processor – Motivating Example

- Earthquake in the Pacific in March 2011 → tsunami → a nuclear accident
- Hourly observation of radioactivity statistics at 47 prefectures
- Observations (March 16, 2011 – March 15, 2012) converted to RDF data (places represented by URI from GeoNames)
- Interesting analyses:
  - AVG radioactivity separately for each prefecture in Japan
  - The MIN and MAX radioactivity for each prefecture (changes within one-year observations)
Motivating Example - Observation

```sparql
#observation
  rdf:value "0.079"^^ms:microsv ;
  ev:place <http://sws.geonames.org/1852083/> ;
  ev:time <http://www.kanzaki.com/works/2011/stat/dim/d/20110414T08PT1H> ;
#dimension - place

<http://sws.geonames.org/1852083/>  
  vcard:region "Tokyo"@en ;
  vcard:locality "Shinjuku"@en ;
  gn:lat "35.69355" ;
  gn:long "139.70352" .
#dimension - time
<http://www.kanzaki.com/works/2011/stat/dim/d/20110414T08PT1H>  
  rdfs:label "2011-04-14T08";
  tl:at "2011-04-14T08:00:00+09:00"^^xsd:dateTime ;
  tl:duration "PT1H"^^xsd:duration .
```
Motivating Example - Query

Ex: Show average radioactivity values for each prefecture

```
SELECT ?regName ( AVG (?floatRV) AS ?average ) WHERE {
  ?s ev:place ?placeID ;
  ev:time ?time ;
  rdf:value ?radioValue .
SERVICE <http://lod2.openlinksw.com/sparql> {
  ?regionID gn:name ?regName .
}
BIND (xsd:float (?radioValue) as ?floatRV ) .
}
GROUP BY ?regName
```
Motivating Example - Results

- Virtuoso v07.10.3207, Sesame v2.7.11, and Jena Fuseki v1.0.0 (based on ARQ) timed out.

- Network traffic analyzer showed that:
  - Virtuoso and Fuseki query GeoNames for each radioactivity observation (more than 400,000 requests).
  - Sesame is trying to download all triples that match the SERVICE query pattern (more than 7.8 million triples).
Basic Strategies - Mediator Join

- The mediator/federator receives the query from the user.
- The query optimizer sends separate queries to endpoints and merges the results.
- Strong point – parallelization.
- Weak point – expensive for large intermediate results/datasets.

```
SELECT ?placeID ?radioValue WHERE {
}
```

```
SELECT ?placeID ?regName WHERE {
    ?placeID gn:parentFeature ?regionID.
    ?regionID gn:name ?regName.
}
```
Basic Strategies - Semi-Join

- Main principle is to execute the subquery with the smallest result first and use the retrieved results as bindings for the join variable in other subqueries (SPARQL structure).
- Efficient for highly selective subqueries (with FILTER statement)

```
SELECT ?regName ( AVG (?radioValue) AS ?average ) WHERE {
  SERVICE <http://lod2.openlinksw.com/sparql> {
  }
  FILTER(?radioValue < 0.08) .
} GROUP BY ?regName
```
SELECT ?placeID ?radioVal
WHERE {
    ?s rdf:value ?radioVal ;
    ev:place ?placeID; ev:time ?time.
    FILTER (?radioValue < 0.08) .
}

SELECT ?placeID ?regName
WHERE {
    ?placeID gn:parentFeature ?rgID.
    ?rgID gn:name ?regName.
    VALUES (?placeID) {
        <http://sws.geonames.org/1852083/>...
    }
}

- Weak point - VALUES is not yet widely adopted in existing endpoints. SPARQL 1.0 compliant alternatives of UNION (or FILTER) must often be used.
Basic Strategies - Partial Aggregation

- If results are grouped by SERVICE query variables, further optimization is possible (motivating query example)

1) First group by the observation place (?placeID)

```sparql
SELECT ?placeID (SUM (?floatRV) AS ?avgSUM) (COUNT (?floatRV) AS ?avgCNT)
WHERE {
  BIND (xsd:float (?radioValue) as ?floatRV) .
}
GROUP BY ?placeID
```
Basic Strategies - Partial Aggregation

- *Then execute SERVICE query*

  ```sparql
  SELECT ?placeID ?regName WHERE {
    ?regionID gn:name ?regName .
  }
  VALUES (?placeID) {
    (<http://sws.geonames.org/1852083/>) ....
  }
  ```

- *Final step – join the intermediate results and compute the final result (distributed/algebraic functions)*
CODA – Cost-based Optimizer for Distributed Aggregate Queries

- Decomposes the original query into multiple subqueries (query $Q_M$ and SERVICE queries $Q_{e1} \ldots Q_{eN}$)

- Estimates query execution costs for different query execution plans

- Chooses the one with minimum costs
CODA - Costs

- **Overall costs** $C_Q$

  $$C_Q = C_P + C_C$$

- **Communication costs** $C_C$ for subquery $S_i$:

  $$C_C(S_i) = C_O + c_{S_i} \times C_{map};$$

  $C_O$ - communication establishing overhead, $c_{S_i}$ - result size, and $C_{map}$ - single result transfer cost

- **Processing costs**

  $$C_P = c_{agg_i} \times C_{AGG};$$

  $c_{agg_i}$ - number of aggregated observations, $C_{AGG}$ - cost for processing a single observation
CODA - Estimating Constants

- $C_{map}$ - estimated using “SELECT * WHERE { ?s #p ?o . FILTER(?o = #o) } LIMIT #L”; different values for #L, #o and #p

- $C_O$ - estimated with multiple “ASK {}” or “SELECT (1 AS ?v) {}”

- $C_{AGG}$ - estimated based on multiple “SELECT COUNT(?s) WHERE {?s ?p ?o } GROUP BY ?o”

Not perfectly accurate but the aim is to find out which execution plan is more efficient (not to predict the execution costs)
CODA - Result Size Estimation

- Result size estimation - **VoID statistics** (dataset, property partition, class partition)

- $c_t$ - total number of triples (void:triples), $c_s$ - total number of distinct subjects (void:distinctSubjects), $c_o$ - total number of distinct objects (void:distinctObjects)

- Single patterns - $C_{res}$ for (?s ?p ?o) is given by $c_t$, (s ?p ?o) estimated as $c_t/c_s$, (?s ?p o) as $c_t/c_o$, and (s ?p o) as $c_t/(c_s \times c_o)$; FILTER influence estimates

- Joins - estimates depend on shape (star vs path). Formulas taken from “Resource Planning for SPARQL Query Execution on Data Sharing Platforms”.

<table>
<thead>
<tr>
<th>type</th>
<th>pattern</th>
<th>cardinality</th>
</tr>
</thead>
<tbody>
<tr>
<td>subject – subject</td>
<td>?s predA ?o . ?s predB ?o2</td>
<td>$\frac{\text{card}(\text{join, partition})}{\text{card}(\text{pat}_1) \cdot \text{card}(\text{pat}_2)}$</td>
</tr>
</tbody>
</table>
CODA – Motivating Example

- Decomposed into 3 queries

```
SELECT ?placeID (AVG(?floatRV) AS ?average)
WHERE
{
  ?s ev:place ?placeID .
  BIND(xsd:float(?radioValue) AS ?floatRV) .
  ?s ev:time ?time .
}
GROUP BY ?placeID
```

```
SELECT ?placeID ?floatRV
WHERE
{
  ?s ev:place ?placeID .
  BIND(xsd:float(?radioValue) AS ?floatRV) .
  ?s ev:time ?time .
}
```

```
SELECT ?placeID ?regName
WHERE
{
  ?regionID gn:name ?regName .
}
```
CODA – Motivating Example

- Estimates for Radioact query:
  - number of aggregated triples: 405384
  - estimated cost: 15
  - number of returned triples: 405384
  - estimated cost: 129

- Estimates for GeoNames query:
  - number of returned triples: 7877627
  - estimated cost: 1956

- Selected plan – Partial Aggregation
Test Case – SSB as RDF

- Star Schema Benchmark converted to RDF (strongly resembling SSB tabular structure)

- We generated data for different scale factors (1 to 5 - 6M to 30M observations, 110,5M to 547,5M triples)

- Different configurations
  - two endpoints (one endpoint containing main observation data and one SERVICE endpoint containing supporting data)
  - three endpoints (two SERVICE endpoints containing supporting data)
  - four endpoints (three SERVICE endpoints containing supporting data)

- All datasets and queries are available at http://extbi.cs.aau.dk/coda/
Aggregate Queries in Federations of Endpoints

- Efficiently processing aggregate queries in a federation of SPARQL endpoints
- Processing strategies (MedJoin, SemiJoin, PartialAgg)
- Cost-based Optimizer for Distributed Aggregate queries (CODA)
  - efficient and scalable
  - chooses the best query processing plan in different situations
  - significantly outperforms current state-of-the-art triple stores
Improving Performance of Aggregate Queries

- With much data to process, analytical queries need special techniques to improve the performance of user queries.

- One such technique is materializing the results of predefined queries and answering user queries based on these results – materialized views.
Materializing RDF Views – Data Cube Example
Materializing RDF Views – Data Cube Example

- The total sales by month and customer state in 2010, for customers living in cities with over 1M inhabitants

SELECT ?c_state ?month (SUM(?total) AS ?sum_total)
FROM <http://ex.com>
WHERE {
  FILTER(?yearNum=2010 && ?pop > 1 000 000)
}
GROUP BY ?c_state ?month
Materializing RDF Views – Materializing RDF Data Cube

- Materializing all views in a data cube is not efficient
- Only several views with max benefit are chosen for materialization
Materializing RDF Views – Defining Views

- View query consists of 2 parts: SELECT query specifies the desired lattice node; CONSTRUCT query creates RDF triples from SELECT query results

CONSTRUCT {
    ?id ex:DateMonth ?vMonth ; ex:CustomerCity ?vCity ; ex:RevenueCount ?crev ; ex:RevenueSum ?srev .
}
WHERE {
    WHERE {
        ?odate skos:broader ?vMonth .
    }
    GROUP BY ?id ?vState ?vMonth
}
Materializing RDF Views – Cost Model

- The cost of answering a query – number of triples contained in the materialized view used to answer the query
- Observation is described by its $n$ dimensions and contains $m$ measures.
- The total number of triples in a view – $(n + m) \times N$, where $N$ is the number of observations
- In each step the algorithm selects a view with maximum benefit taking into account previously materialized views
Materializing RDF Views – Further Steps

- Answering Aggregate SPARQL Queries over Materialized Views with Inferred Knowledge
- Analyzing the Performance of Complex Aggregate SPARQL Queries with Intermediate Results Materialization
- Improving the Performance of OLAP Queries in a Federation of SPARQL Endpoints
Conclusion

- **OnLine Analytical Processing** (OLAP) style analysis of Linked Data may help in better decision making (e.g. analytics applications that integrate private data with web RDF datasets)

- The goal of the project is to **improve** the **performance** of **analytical SPARQL** queries over federated RDF sources