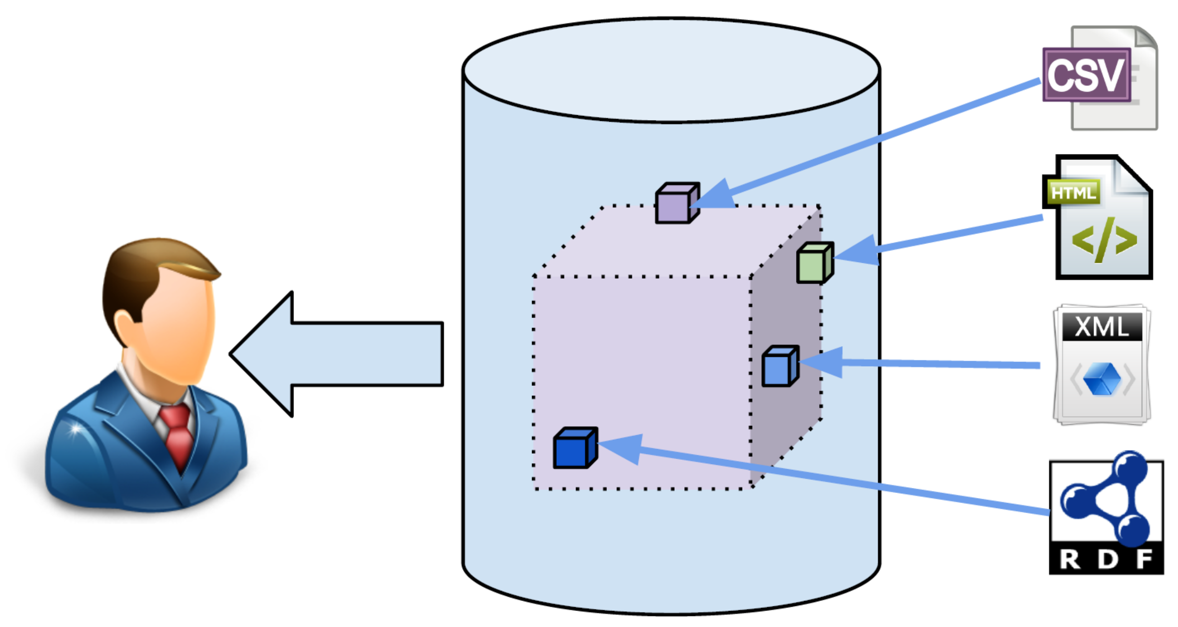
Towards OLAP over Federated RDF Sources

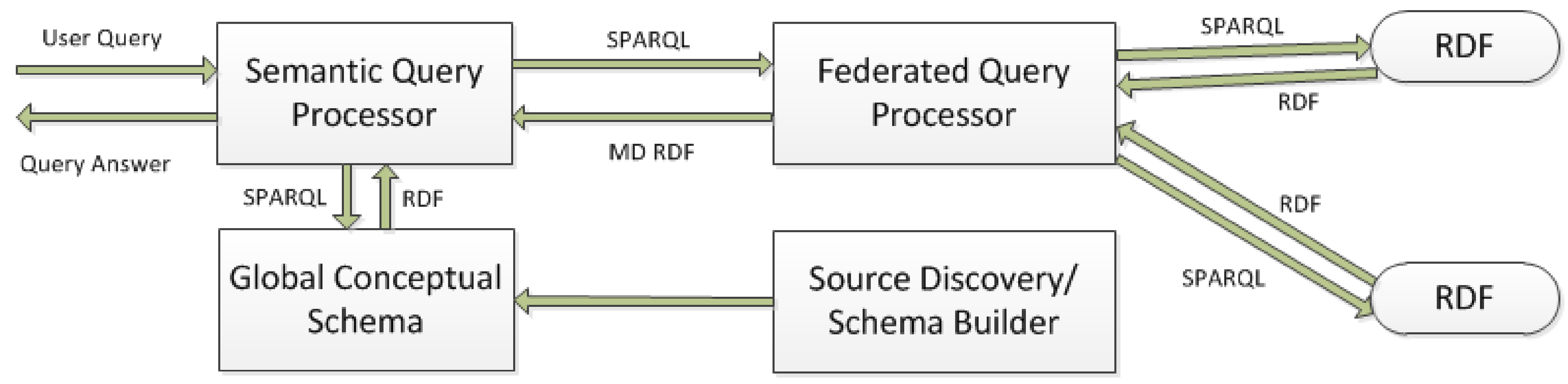
Dilshod Ibragimov (Supervisors Esteban Zimányi, Torben Bach Pedersen)
dibragim@ulb.ac.be, diib@cs.aau.dk
Université Libre de Bruxelles (ULB), Aalborg University (AAU)

1. BI and the Semantic Web

- Business Intelligence tools need to analyze data published on the Web
- OLAP-style analysis of Linked Data may help in better decision making



2. System Architecture



- *Global Conceptual Schema* (GSC) – high-level view of the system (expressed in QB4OLAP)
- *Source Discovery/Schema Builder* – discovery of data sources and construction of the GCS
- *Federated Query Processor* (FQP) – retrieval, in parallel, data from several federated data sources
- *Semantic Query Processor* – conversion of a user query to SPARQL which is sent to the FQP

3. Processing Aggregate Queries in a Federation of SPARQL Endpoints

Motivating Example

Data Structure

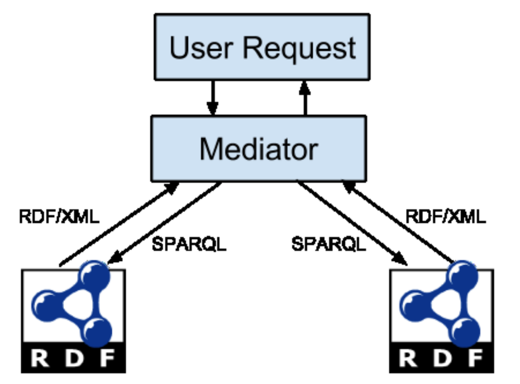
```
#observation
<http://www.kanzaki.com/works/2011/stat/ra/20110414/p13/t08>
   rdf:value "0.079"^^ms:microsv ;
   ev:place <http://sws.geonames.org/1852083/> ;
   ev:time
<http://www.kanzaki.com/works/2011/stat/dim/d/20110414T08PT1H> ;
   scv:dataset <http://www.kanzaki.com/works/2011/stat/ra/set/moe> .
#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 .
```

Federated Query

```
SELECT ?regName (AVG (?rValue ) AS ?avgSUM)
WHERE {
   ?s ev:place ?placeID . ?s ev:time ? time .
   ?s rdf:value ? rValue .
   SERVICE <http://lod2.openlinksw.com/sparql> {
      ?placeID gn:parentFeature ?regionID .
      ?regionID gn:name ?regName .  }
} GROUP BY ?regName
#Query times out because of inefficient strategy
```

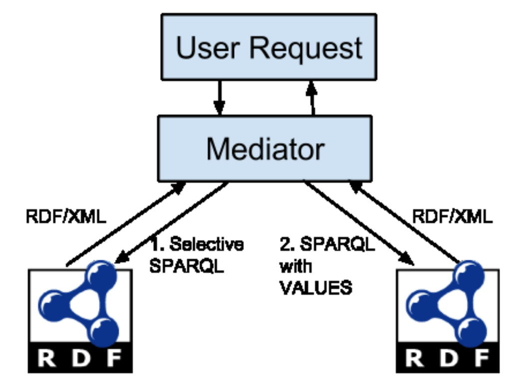
Basic Strategies

Mediator Join



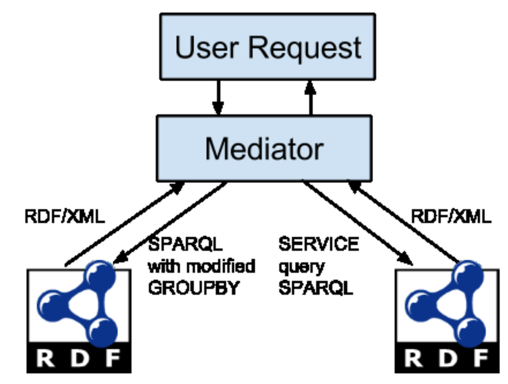

```
SELECT ?placeID ?regName
WHERE {
  ?placeID gn:parentFeature/gn:name
?regName .
}
```

Semi-Join




```
SELECT ?placeID ?regName WHERE {
 ?placeID gn:parentFeature/gn:name
 ?regName .
 VALUES (?placeID) {    ....
  (<http://sws.geonames.org/182083/>) }
}
```

PartialAgg




```
SELECT ?placeID ?regName WHERE {
 ?placeID gn:parentFeature/gn:name
 ?regName .
 VALUES (?placeID) {    ....
  (<http://sws.geonames.org/182083/>) }
}
```

```
SELECT ?placeID ?rValue WHERE {
  ?s ev:place ?placeID .
  ?s rdf:value ?rValue .
}
```

```
SELECT ?placeID ?rValue WHERE {
  ?s ev:place ?placeID .
  ?s rdf:value ?rValue .
  FILTER(rValue < 0.08)
}
```

```
SELECT ?placeID (SUM (?floatRV ) AS
?avgSUM) (COUNT (?floatRV ) AS
?avgCNT ) WHERE {
   ?s ev:place ?placeID . ?s ev:time ?
time .
   ?s rdf:value ? radioValue .
}
GROUP BY ?placeID
```

CODA – Cost-based Optimizer for Distributed Aggregate Queries

Overview

- Decomposes the original query into multiple subqueries (query Q_M and SERVICE queries $Q_{e1} \dots Q_{eN}$)
- Estimates query execution costs for different query execution plans
- Chooses the one with minimum costs

Cost Model

Overall costs C_Q: $C_Q = C_P + C_C$

Communication costs C_C for subquery S_i:

$C_C(S_i) = C_O + c_{S_i} * C_{map}$; C_O - communication establishing overhead , c_{S_i} - result size, and C_{map} - single result transfer cost

Processing costs

$C_P = c_{agg_i} * C_{AGG}$; c_{agg_i} - number of aggregated observations, C_{AGG} - cost for processing a single observation

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"

The goal is to find efficient plan (not to estimate the execution time)

Estimating Result Size

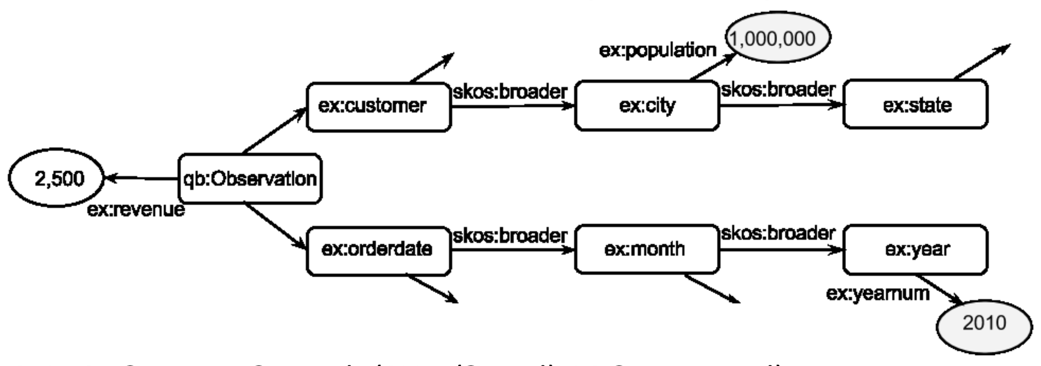
Result size estimation - VoID statistics (dataset statistics)

- 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 $c_t/(c_s * c_o)$
- Joins - estimates depend on shape (star vs path). Formulas from "Resource Planning for SPARQL Query Execution on Data Sharing Platforms"

type	pattern	cardinality $card(join, partition)$
subject – subject	?s predA ?o . ?s predB ?o2	$\frac{card(pat_1) \cdot card(pat_2)}{\max(cp_{predA,s}, cp_{predB,s})}$

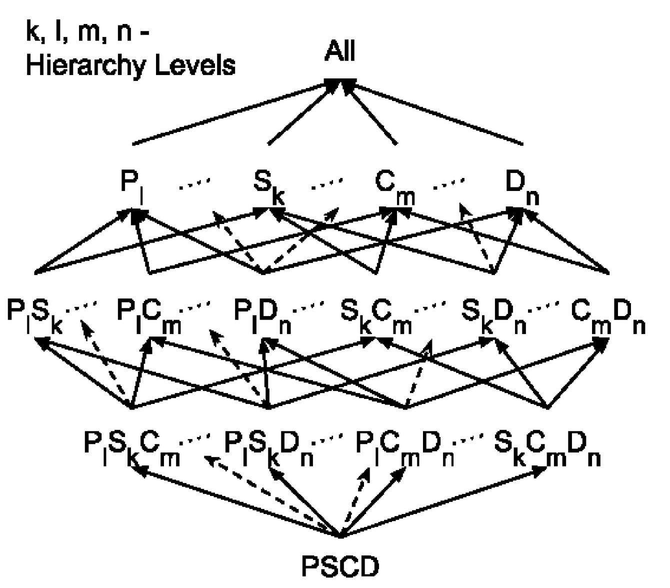
4. Improving Performance of Aggregate Queries using Materialized RDF Views

Data Cube Query Example




```
SELECT ?c_state ?month (SUM(?total) AS ?sum_total)
FROM <http://ex.com>
WHERE {
   ?obs ex:OrderDate ?lo_orderdate ; ex:Customer ?customer ;
   ex:Revenue ?total . ?customer skos:broader ?c_city .
   ?c_city skos:broader ?c_state . ?c_city ex:population ?pop .
   ?lo_orderdate skos:broader ?month . ?month skos:broader ?year .
   ?year ex:value ?yearNum .
   FILTER(?yearNum=2010 && ?pop > 1000000)
}
GROUP BY ?c_state ?month
```

Materializing RDF Data Cube



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

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 {
   SELECT ?id ?vCity ?vMonth (SUM(?rev) AS ?srev) (COUNT(?rev) AS
   ?crev)
   WHERE {
      ?li ex:OrderDate ?odate ; ex:Customer ?cust ;
         ex:Revenue ?rev . ?cust skos:broader ?city .
      ?odate skos:broader ?vMonth .
      BIND(IRI('http://ex.org/id#', CONCAT(?vCity, ?vMonth)) AS ?id) .
   }
    GROUP BY ?id ?vState ?vMonth
}
```

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) * 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.

Storing Views

- Storing each materialized RDF view in a separate named graph is a better option
- Benefits include:
  - Easier maintenance (easier to update)
  - Faster retrieval (scanning is faster)
  - Correctness of aggregation

5. Publications

Published:

D. Ibragimov, K. Hose, T. B. Pedersen, E. Zimányi. Towards Exploratory OLAP over Linked Open Data – A Case Study. BIRTE 2014

D. Ibragimov, K. Hose, T. B. Pedersen, E. Zimányi. Executing Aggregate SPARQL Queries over Federated Endpoints

Future Work:

Analyzing the Performance of Complex Aggregate SPARQL Queries with Intermediate Results Materialization

Answering Aggregate SPARQL Queries over Materialized Views with Inferred Knowledge