Distributed RDF Data Integration

Steven Lynden

National Institute of Advanced Industrial Science and Technology (AIST), Japan
AIST
National Institute of Advanced Industrial Science and Technology

- Approximately 2,500 researchers
- Multiple locations throughout Japan; headquarters in Tsukuba, Ibaraki
- Focus on R&D, developing standards, maintaining strong links with industry and academia.

ITRI – Information Technology Research Institute
RDF/Semantic Web related research within ITRI

- Storage schemes and index-based structures for efficient RDF storage and retrieval
- Using MapReduce for RDF query processing
  - Parallel RDF query processing

This presentation:

- Distributed RDF query processing
  - How to execute queries over multiple SPARQL endpoints
- Web service-based standards for RDF data access
  - Standardisation effort within the Open Grid Forum (OGF)
Distributed RDF query processing
RDF Distributed Query Processor
Motivation

• RDF data available on the Web has grown over the last few years
  – Linked Data and consistent URLs
  – SPARQL endpoints
• Many datasets are interlinked
  – Shared ontologies
  – owl:sameAs links
• Applications required federated queries over multiple SPARQL endpoints
  – Active research area
  – Query federation extensions present in SPARQL 1.1
SPARQL 1.1

• SERVICE construct for accessing remote endpoints
  – Access to remote endpoint explicitly specified in the query, e.g:

```sparql
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
FROM <http://example.org/myfoaf.rdf>
WHERE
{
  SERVICE <http://people.example.org/sparql> {
    ?person foaf:name ?name .
  }
}
```

From SPARQL 1.1. Federated Query Draft: http://www.w3.org/TR/sparql11-federated-query/

• Alternative:
  – SERVICE construct is not used and the query processor decides which services to access
Declarative, automatically optimised approach

- The query processor is configured with a set of available endpoints or discovers them automatically.
- Some metadata required to be able to select data sources.
- User issues a query:

  ```sparql
  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
  SELECT ?name ?mbox
  WHERE
  { ?x foaf:name ?name .
    ?x foaf:mbox ?mbox }
  ```

- Optimisation problem: how to select sources, how to minimise the amount of data retrieved and minimise query response times.
Query processing

**Input** = a declarative query
1. Translate the query into a tree of operators
2. Optimise the query
   - Logical (e.g. reorder operators)
   - Physical (e.g. algorithm selection)
3. Execution

**Output** = a query result

Relational algebra is very well studied with many efficient optimisation techniques known.

Efficient optimisation requires *statistics* to minimise response time. Join order is important.
Processing RDF data

• There have been non-relational approaches to storing RDF data, e.g. as a graph
• However, many RDF data storage solutions rely on relational DBMSs:
  – Jena, Oracle, Sesame, 3store
• Various approaches:
  – Single table with 3 columns (subject/predicate/object)
  – Multiple tables, e.g. vertical partitioning
Vertically partitioned RDF representation

- Re-write the data as two column tables, one table for each distinct property

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ID1&gt;</td>
<td>&lt;name&gt;</td>
<td>Evans</td>
</tr>
<tr>
<td>&lt;ID1&gt;</td>
<td>&lt;livesIn&gt;</td>
<td>Leeds</td>
</tr>
<tr>
<td>&lt;ID2&gt;</td>
<td>&lt;name&gt;</td>
<td>Johnson</td>
</tr>
<tr>
<td>&lt;ID2&gt;</td>
<td>&lt;livesIn&gt;</td>
<td>Hull</td>
</tr>
<tr>
<td>&lt;ID3&gt;</td>
<td>&lt;name&gt;</td>
<td>Davies</td>
</tr>
<tr>
<td>&lt;ID3&gt;</td>
<td>&lt;livesIn&gt;</td>
<td>Leeds</td>
</tr>
</tbody>
</table>

Vertically partitioned:

**name**

<table>
<thead>
<tr>
<th>Subject</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ID1&gt;</td>
<td>Evans</td>
</tr>
<tr>
<td>&lt;ID2&gt;</td>
<td>Johnson</td>
</tr>
<tr>
<td>&lt;ID3&gt;</td>
<td>Davies</td>
</tr>
</tbody>
</table>

**livesIn**

<table>
<thead>
<tr>
<th>Subject</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ID1&gt;</td>
<td>Leeds</td>
</tr>
<tr>
<td>&lt;ID2&gt;</td>
<td>Hull</td>
</tr>
<tr>
<td>&lt;ID3&gt;</td>
<td>Leeds</td>
</tr>
</tbody>
</table>

RDF:

- `<ID1> <name> “Evans”`<ID1> <livesIn> “Leeds”
- `<ID2> <name> “Johnson”`<ID2> <livesIn> “Hull”
- `<ID3> <name> “Davies”`<ID3> <livesIn> “Leeds”
Single ‘triples’ table:

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ID1&gt;</td>
<td>&lt;name&gt;</td>
<td>Evans</td>
</tr>
<tr>
<td>&lt;ID1&gt;</td>
<td>&lt;livesIn&gt;</td>
<td>Leeds</td>
</tr>
<tr>
<td>&lt;ID2&gt;</td>
<td>&lt;name&gt;</td>
<td>Johnson</td>
</tr>
<tr>
<td>&lt;ID2&gt;</td>
<td>&lt;livesIn&gt;</td>
<td>Hull</td>
</tr>
<tr>
<td>&lt;ID3&gt;</td>
<td>&lt;name&gt;</td>
<td>Davies</td>
</tr>
<tr>
<td>&lt;ID3&gt;</td>
<td>&lt;livesIn&gt;</td>
<td>Leeds</td>
</tr>
</tbody>
</table>

SELECT t1.subject, t1.object, t2.object
FROM triples as t1, triples as t2
WHERE t1.predicate='<name>' and
    t2.predicate='<livesIn>' and
    t1.subject=t2.subject

Vertically partitioned:

Vertically partitioning shown to be effective

- Other schemas and indexes often used
Distributed query processing

• More issues:
  – Communication between distributed locations
  – Parallel execution at multiple locations
  – Less predictability
  – More heterogeneity

Fig. 4. Distributed query plan.
Distributed query processing over SPARQL endpoints

- **Autonomous**
  - May not provide statistics, only support certain queries etc.

- **Unpredictable**
  - Varying data transfer rates, temporarily unavailable etc.

- **Constantly updated**
  - Generally growing in size

These issues further complicate the problem. It is difficult to generate static query plans that perform well.
Adaptive query processing

• An alternative to generating static query plans
  – Generate an initial plan
  – Modify the plan during execution

• Challenges
  – Avoid throwing away intermediate results
  – Efficiently monitoring query execution; deciding when to adapt

• Recent work on adaptive query processing RDBMS has produced useful results
  – Changing join order during execution
    • Useful when initial plans are sub-optimal

• Suitable for distributed query processing over SPARQL Web Services, which are inherently unpredictable
ADERIS

- Distributed query processor over SPARQL endpoints developed at AIST
- Suitable for executing joins over separate endpoints
- Uses adaptive query processing:
  - Change join order at runtime
  - Respond to the different characteristics of each service
Reordering example (index nested loop join)

• Now in a state where the join’s inputs can be exchanged without losing results
Adaptive reordering of a join sequence

1. Retrieve data from remote endpoints, create vertically partitioned data (table A, B, C, D) - Based on estimates of join selectivity to minimise execution time

2. Start executing the query. Monitor the actual selectivity of each join.

3. If estimated selectivity is incorrect, a cost model is used to estimate whether changing join order will result in lower response time.

Example: tables D and C switched in the join order.
PREFIX foaf: http://xmlns.com/foaf/0.1/
SELECT ?name ?email
WHERE {
  ?person a foaf:Person .
  ?person foaf:name ?name .
}
Query 1 (Result size = 150):
select * where {
  ?x dbp:reference ?ref . 777,679
  ?x rdf:comment ?comment . 10,000
  ?x skos:subject ?subj . 9971
  ?x foaf:page ?page . 10,000
  ?x rdfs:type ?type . 800,000
FILTER ( regex(str(?subj),"building") )
}

Query 2 (Result size = 8):
select * where {
  ?x dbp:reference ?ref . 777,679
  ?x rdf:comment ?comment . 10,000
  ?x skos:subject ?subj . 9971
  ?x foaf:page ?page . 10,000
  ?x rdfs:type dbp:book . 3105
}

Query 3 (Result size = 8):
select * where {
  ?x dbp:reference ?ref . 777,679
  ?x rdf:comment ?comment . 10,000
  ?x skos:subject ?subj . 9971
  ?x foaf:page ?page . 10,000
  ?x rdfs:type dbp:book . 3105
  ?x dbp:releaseDate ?date . (DBP) 126,737
}

Query 4 (Result size = 13):
select * {
}
Distributed RDF Query Processing - Summary

• ADERIS
  – Uses adaptive query processing to deal with unpredictable data sources
  – Translates SPARQL to relational operations for which optimisation approaches well-studied
  – Join reordering is one type of adaptivity
  – Adaptive interaction with data sources also used.

• Complementary techniques also important
  – Logical optimisation
    • Query re-writing

• An open problem, active area of research
Web service-based specifications for access to RDF data resources
Open Grid Forum

http://www.ogf.org

• Open community aimed at promoting applied distributed computing
• Development of scalable applications and infrastructures for use within enterprise/science community
• Open forums to:
  – Build focused communities
  – Explore trends
  – Share best practices
  – Consolidate into standards
Database Access and Integration Services (DAIS) Working Group

- A group comprising individuals from academia and industry interested in database technology in service-based computing architectures e.g. grids and clouds

Develop standards

- Provide consistent access to autonomously managed databases via web services
- Ease application development through the provision of components that can be seamlessly integrated with other standards
Web Services – Database Access and Integration (WS-DAI) Family of Specifications

- A core specification defining the basic properties and message patterns
- Extensions to the core specifications for:
  - Relational databases
  - XML databases
  - RDF databases

From Antonioletti et al. WS-DAI RDF(S) Realization: Introduction, Motivational Use Cases and Terminologies
WS-DAI approach
Data service and data resources

Interfaces to data resources act as a conduit to the underlying data management system

From Antonioletti et al. WS-DAI RDF(S) Realization: Introduction, Motivational Use Cases and Terminologies
Interface types

• Data Description
  – provide information about a service/data resource accessed through a service.
  – Property document, resource properties

• Data Access
  – provide access to data through a service interface.
    • e.g. SPARQL query

• Data Factory
  – provide indirect access to data through a client specified interface.
    • e.g. create a graph resource as the result of a SPARQL query
Motivation for an RDF-based specification

• Complement existing WS-DAI specifications defined in the context of service-based computing
  – Leverage existing specifications, e.g. WS-ResourceProperties, WS-ResourceLifetime
• Richer set of access mechanisms than the SPARQL protocol
• Querying is not the only way to access resources
  – e.g. the Jena Semantic Web Framework has an ontology API
  – Define ontology primitives for WS-based access
WS-DAI-RDF Querying & Ontology access

Execute queries using SPARQL

Manage the contents of an RDF repository

From Antonioletti et al. WS-DAI RDF(S) Realization: Introduction, Motivational Use Cases and Terminologies
SPARQL protocol & WS-DAI-RDF Indirect Data Access

- RDF specific interface supporting SPARQL

- Indirect-access and data resource descriptions may provide a better interface for distributed query processing
WS-DAI-RDF Indirect Access

From Antonioletti et al. WS-DAI RDF(S) Realization: Introduction, Motivational Use Cases and Terminologies
WS-DAI-RDF - Summary

• A Web Service-based specification for accessing RDF data resources
  – Querying: can be seen as an alternative to the SPARQL protocol
  – Ontology: API style access to RDF
• Currently working drafts, yet to become full standards
• Used in a variety of applications
  – Applications requiring seamless integration with other WS specifications
  – Distributed RDF query processing
• Provides indirect data access allowing separate resources to be created and results to be pulled back