RDFPath
Path Query Processing on Large RDF Graphs with MapReduce

1st Workshop on High-Performance Computing for the Semantic Web (HPCSW 2011)

Martin Przyjaciel-Zablocki
Alexander Schätzle
Thomas Hornung
Georg Lausen

University of Freiburg
Databases & Information Systems
Overview

1. Motivation
2. MapReduce
3. RDFPath
4. Experimental Results
5. Summary
Large RDF Graphs

- **Semantic Web**
  - Amount of Semantic Data increases steadily
  - RDF is W3C standard for representing Semantic Data

- **Social Networks**
  - > 500 million active users in Facebook
  - Interacting with >900 million objects (pages, groups, events)
  - Social Graphs can also be represented in RDF

- How to explore, analyze such large RDF Graphs?

- **Our Approach:** Distributed analysis of large RDF Graphs by mapping RDFPath to MapReduce
MapReduce

- **MapReduce**
  - Popularized by Google & widely used
  - Automatic parallelization of computations
  - Fixed two-stage process: Map $\rightarrow$ Reduce $\rightarrow$ Map $\ldots$

- **Distributed File System**
  - Clusters of commodity hardware
    - Fault tolerance by replication
  - Very large files / write-once, read-many pattern

- **Apache Hadoop**
  - Well-known Open-Source implementation
  - Used by Yahoo, Facebook, Amazon, IBM, Last.fm, …
MapReduce (2)

Steps of a MapReduce execution

- **Signatures**
  - **Map:** \((\text{in\_key}, \text{in\_value}) \Rightarrow \text{list} (\text{out\_key}, \text{intermediate\_value})\)
  - **Reduce:** \((\text{out\_key}, \text{list} (\text{intermediate\_value})) \Rightarrow \text{list} (\text{out\_value})\)
3. RDFPath

Path queries on RDF Graphs
RDFPath

Idea
- Navigational queries on RDF Graphs
- Declarative path specification inspired by XPath
- Particularly designed with regard to MapReduce
  → Every location step is mapped to one MapReduce job
- Extendibility

Features
- Fixed-length paths, shortest paths
- Aggregate functions, degree of a node
- Adjacent nodes and edges
- ...

3. RDFPath
Path Language
Example (1)

- Allen :: knows [country=equals("DE")], > age

Results

- Allen (knows) Jacob [country="DE"] (age) 42
- Allen (knows) Sarah [country="CH"]
- Allen (knows) Chris [country="CH"]
Example (2)

- Allen :: knows(*3)

Results
- Allen (knows) Jacob
- Allen (knows) Jacob (knows) Emily
- Allen (knows) Chris
- Allen (knows) Sarah
Implementation

- System Architecture

- Storing RDF Graphs
  - RDF Graph is loaded in advance
  - Vertical Partitioning by predicate
  - Sequence Files stored in HDFS contain PathObjects
Implementation (2)

- Query Processing

![Diagram showing the query processing flow](image)
Implementation (3)

- Mapping to MapReduce Jobs
  - Allen :: knows(*2) > country [equals("DE")]

- Map task:
  - Tagging intermediate paths and country partition for join
  - Applying filter condition

- Reduce task:
  - Perform join and store resulting paths back to HDFS
4. Experimental Results

Properties of RDFPath
Experiments

- **Environment Setup**
  - Cluster of 10 machines (Dual Core CPU, 4GB RAM, 1GB HD)
  - Cloudera’s Distribution for Hadoop 3 Beta 1
  - Default configuration with 9 reducers (one per HD)

- **Real-World Last.fm dataset**
  - 225 million RDF triples

- **Different input-sizes** instead of varying the number of machines
Query 1

- The Disco Boys - I Surrender :: trackSimilar(*4) > trackAlbum

- **Linear scaling behavior** for execution times & amount of data
- Execution times mainly influenced by number of intermediate results stored locally as well as transferred data
Query 3

- Chris :: knows(*X), 1 ≤ X ≤ 10

- 98% of all reachable persons can be reached by at most 7 edges
  ➔ Corresponds to the well-known six degrees of separation paradigm

- Linear scaling behavior, mainly determined by number of joins
Query 3

- Chris :: knows(*X), 1 ≤ X ≤ 10

- 98% of all reachable persons can be reached by at most 7 edges
  ➔ Corresponds to the well-known six degrees of separation paradigm
- Linear scaling behavior, mainly determined by number of joins
Summary

- Amount of **Semantic Web** data is growing constantly!

- **RDFPath**
  - Intuitive syntax for path queries
  - Expression of interesting graph issues such as Friend of a Friend queries
  - Investigating small world properties like six degrees of separation
  - Designed with MapReduce in mind

- **Execution**
  - Direct mapping to MapReduce
  - Scaling properties of MapReduce enable analysis of large RDF Graphs

- **Evaluation**
  - Based on real-world data from Last.fm
  - Shows linear scaling behavior in the size of the graph
  - Confirms the effectiveness of our approach
Thanks for your attention.
Backup Slides

- Last.fm
- Examples cont.
- Evaluation cont.
- RDFPath Features
- Mapping
- Comparison
- Reduce–Side–Join
4. Evaluation

Last.fm

- artistSimilar: 26.49%
- listenedTo: 22.20%
- trackSimilar: 19.41%
- topTracks: 5.91%
- topArtists: 5.85%
- trackTopTracks: 5.84%
- artistTopTracks: 4.00%
- artistTopFans: 3.44%
- trackTopFans: 3.31%
- knows: 1.70%
- listenedTo+: 0.42%
- topTracks: 0.12%
- topFans: 0.12%
- topArtists: 0.12%
- artistSimilar: 0.12%
- userTopAlbums: 0.12%
- trackDuration: 0.12%
- trackPlaycount: 0.12%
- albumArtist: 0.12%

Realname
Country
Duration
Gender
Age
listenedTo+
topTracks
knows
Track
User
Album
Artist
Playcount
playcount
artistSimilar
trackSimilar
topTracks
topArtists
topFans
b) Example (3)

- Allen :: knows > knows [country=prefix("C")]
  > age [min(18)][max(80)] . avg()

- Result
  - 34
b) Example (4)

* :: knows [equals("Sarah")]

Results
- Allen (knows) Sarah
- Chris (knows) Sarah
- Jacob (knows) Emily
- Allen (knows) Jacob
- Allen (knows) Chris
b) **Example (5)**

* :: knows > knows [equals("Sarah")]

**Results**

- Allen (knows) Chris (knows) Sarah
- Allen (knows) Jacob (knows) Emily
b) Example (6)

- * :: * > * [equals("Sarah")]

Results
- Allen (knows) Chris (knows) Sarah
- Allen (knows) Jacob (knows) Emily
b) Example (7)

- Allen :: * .count()

- Result
  - 3
b) Example (Last.fm)

- Michael_Jackson :: artistTracks
  
  [trackAlbum = equals(Michael_Jackson_-_Thriller)]
  
  > trackSimilar [trackDuration = min(50000)]
  
  > trackTopFans [userCountry = equals(DE)].

Results

- Michael_Jackson (artistTracks)
  Michael_Jackson_-_Beat_It (trackSimilar)
  Michael_Jackson_-_Billie_Jean (trackTopFans) Mark

- Michael_Jackson (artistTracks)
  Michael_Jackson_-_Someone_in_the_Dark (trackSimilar)
  Rihanna_-_Russian_Roulette (trackTopFans) Megan
c) Query 2 (Backup)

- Michael_Jackson :: artistTracks
  [ trackAlbum = equals('Michael_Jackson_-_Thriller') ]
  > trackSimilar [ trackDuration = min(50000) ]
  > trackTopFans [ country = equals('DE') ]

---

4. Evaluation
Query 2
d) RDFPath Features by Example

- **Startnode**
  - Chris :: knows
  - * :: knows

- **Location Step**
  - Chris :: knows > knows > age
  - Chris :: knows (2) > age
  - Chris :: *

- **Filters:** equals(), prefix(), suffix(), min(), max()
  - Chris :: knows > age [min(18)] [max(67)]
  - Chris :: * > * [equals('Peter')]
  - Chris :: knows [age = min(30)] [country = prefix('D')] > name

- **Bounded Search**
  - Chris :: knows [country = equals('DE')] (*3)

- **Bounded Shortest Path**
  - Chris :: knows (*3).distance('Peter')

- **Aggregation Functions:** count(), sum(), avg(), min() and max()
  - Chris :: *.count()
  - Chris :: knows > age.avg()
Mapping to MapReduce Jobs

- **Composition**
  - Queries in RDFPath are composed of location steps
  - One location step is mapped to a join in MapReduce

- **Map tasks:**
  - Tagging for RSJs
  - Applying filter conditions

- **Reduce tasks:**
  - Performing RSJs
  - Aggregating values
  - Computing Shortest paths
  - Checking for Cycles
### RDF Query Languages

<table>
<thead>
<tr>
<th>Property</th>
<th>RQL</th>
<th>SeRQL, RDQL³, N3</th>
<th>Versa</th>
<th>RxPath</th>
<th>RPL</th>
<th>SPARQL (1.0)</th>
<th>RDFPath</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adjacent nodes</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Adjacent edges</td>
<td>±</td>
<td>±</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Degree of a node</td>
<td>±</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Path</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>±</td>
<td>±</td>
<td>±</td>
</tr>
<tr>
<td>Fixed-length Path</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>×</td>
<td>±</td>
<td>±</td>
<td>✓</td>
</tr>
<tr>
<td>Distance between 2 nodes</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>±</td>
</tr>
<tr>
<td>Diameter</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Shortest Paths</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>±</td>
</tr>
<tr>
<td>Aggregate functions</td>
<td>±</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>±</td>
<td>×</td>
<td>±</td>
</tr>
</tbody>
</table>

(×: not supported, ±: partially supported, ✓: fully supported)

**Table 1.** Comparison of RDF Query Languages (adapted from [4, 5])**
RDF Query Languages (2)

- **Adjacent nodes:**
  - All nodes adjacent to a start node

- **Adjacent edges:**
  - All predicates of statements involving a given start node

- **Degree of a node:**
  - Number of predicates involving a start node

- **Path:**
  - Find paths between a fixed start node and an end note of arbitrary length

- **Fixed-length paths:**
  - Find all paths of length 2 between start note and end note

- **Distance between two resources:**
  - (Length of shortest path without a max length!)

- **Diameter of a graph:**
  - Diameter of a given RDF Graph.
f) Comparison to SPARQL 1.0

- Fixed-length paths

```
SELECT ?tmp1, ?tmp2, ?tmp3,
     ?tmp4, ?tmp5
WHERE {
    Allen knows ?tmp1 .
    ?tmp5 knows ?tmp5
}
```

Allen :: knows(5)

SPARQL

RDFPath
Comparison to SPARQL 1.0

Filters

```
SELECT ?tmp1, ?tmp2, ?tmp3, ?tmp4
WHERE {
  Allen knows ?tmp1 .
  ?tmp1 age ?age1 .
  Filter (?age1 >= 20)
  ?tmp2 country "DE" .
}
```

```
Allen :: knows [age=min(20)] > knows [country=equals("DE")]
```
Comparison to SPARQL 1.0

- **Drawbacks of SPARQL**
  - Limited navigational capabilities
  - No Shortest Path Queries
  - No Aggregate functions (count, min, max, ...)
  - No Abbreviations for following same edges

- **Advantages of SPARQL**
  - Expressive general purpose query language for RDF
  - Intuitive Syntax, related to SQL
  - SPARQL 1.1 adds support for some path queries
g) Reduce–Side Join

Example:

* :: knows(*2) > knows

intermediate paths

- Peter (knows) Frank (knows) Chris
- Peter (knows) Simon (knows) Johan
- Klaus (knows) Simon (knows) Johan
- Frank (knows) Chris
- Peter (knows) Klaus
...

Reduce–Side Join

- Chris  Peter
- Johan  Frank
- Johan  Lukas
- Frank  Chris
- ...

RDFPath: Path Query Processing on Large RDF Graphs

36
reduce-side join (2)

mapper input

intermediate paths

- Peter (knows) Frank (knows) Chris
- Peter (knows) Simon (knows) Johan
- Klaus (knows) Simon (knows) Johan
- Frank (knows) Chris
- Peter (knows) Klaus

knows

- Chris Peter
- Johan Frank
- Johan Lukas
- Frank Chris

mapper output

key

- (Chris, 1) Peter (knows) Frank (knows) Chris
- (Johan, 1) Peter (knows) Simon (knows) Johan
- (Johan, 1) Klaus (knows) Simon (knows) Johan
- (Chris, 1) Frank (knows) Chris
- (Klaus, 1) Peter (knows) Klaus

-value

- (Chris, 0) Peter
- (Johan, 0) Frank
- (Johan, 0) Lukas
- (Frank, 0) Chris
- ...

...
g) Reduce–Side Join (3)

Sorting phase:
1. Partition according to the first keypair \( \% \) reducer
2. Sort within a partition according to the whole keypair

Consequences
- A reducer gets all values with the same first keypair
- The values within a partition contain at first all new nodes and thereafter all intermediate paths
g) Reduce–Side Join (4)

Reducer Input

Reduction Input

Reducer Output

Reduction Output

---

3. RDFPath
Reduce–Side–Join

RDFPath: Path Query Processing on Large RDF Graphs