Web Data Management in RDF Age

M. Tamer Özsu

University of Waterloo
David R. Cheriton School of Computer Science
Acknowledgements

This presentation draws upon collaborative research and discussions with the following colleagues (in alphabetical order)

Güneş Aluç, University of Waterloo

Khuzaima Daudjee, University of Waterloo

Olaf Hartig, University of Waterloo

Lei Chen, Hong Kong University of Science & Technology

Lei Zou, Peking University
Web Data Management

A long term research interest in the DB community
Interest Due to Properties of Web Data

- Lack of a schema
  - Data is at best “semi-structured”
  - Missing data, additional attributes, similar data but not identical
- Volatility
  - Changes frequently
  - May conform to one schema now, but not later
- Scale
  - Does it make sense to talk about a schema for Web?
  - How do you capture “everything”?
- Querying difficulty
  - What is the user language?
  - What are the primitives?
  - Aren’t search engines or metasearch engines sufficient?
More Recent Approaches to Web Querying

- **Fusion Tables**
  - Users contribute data in spreadsheet, CVS, KML format
  - Possible joins between multiple data sets
  - Extensive visualization
More Recent Approaches to Web Querying

- Fusion Tables
  - Users contribute data in spreadsheet, CVS, KML format
  - Possible joins between multiple data sets
  - Extensive visualization

- XML
  - Data exchange language
  - Primarily tree based structure

```xml
<list title="MOVIES">
  <film>
    <title>The Shining</title>
    <director>Stanley Kubrick</director>
    <actor>Jack Nicholson</actor>
  </film>
  <film>
    <title>Spartacus</title>
    <director>Stanley Kubrick</director>
  </film>
  <film>
    <title>The Passenger</title>
    <actor>Jack Nicholson</actor>
  </film>
  ...
</list>
```
More Recent Approaches to Web Querying

- **Fusion Tables**
  - Users contribute data in spreadsheet, CVS, KML format
  - Possible joins between multiple data sets
  - Extensive visualization

- **XML**
  - Data exchange language
  - Primarily tree based structure

- **RDF (Resource Description Framework) & SPARQL**
  - W3C recommendation
  - Simple, self-descriptive model
  - Building block of semantic web & Linked Open Data (LOD)
RDF Data Volumes . . .

. . . are growing – and fast

- Linked data cloud currently consists of 3000 datasets with >84B triples
- Size almost doubling every year
RDF Data Volumes . . .

- . . . are growing – and fast
  - Linked data cloud currently consists of 3000 datasets with
    >84B triples
  - Size almost doubling every year

March '09: 89 datasets

Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch.
http://lod-cloud.net/
RDF Data Volumes . . .

. . . are growing – and fast

- Linked data cloud currently consists of 3000 datasets with >84B triples
- Size almost doubling every year

September ’10: 203 datasets

Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch.
http://lod-cloud.net/
RDF Data Volumes . . .

. . . are growing – and fast

- Linked data cloud currently consists of 3000 datasets with >84B triples
- Size almost doubling every year

Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch.
http://lod-cloud.net/

September ’11: 295 datasets, 25B triples
RDF Data Volumes . . .

. . . are growing – and fast

- Linked data cloud currently consists of 3000 datasets with >84B triples
- Size almost doubling every year

April ’14:
570 datasets, ??? triples

Linked Object Data – Closer Look

Linked Datasets as of April 2014
Globally Distributed Network of Data
Three Approaches

- Data warehousing
  - Consolidate data in a repository and query it
- SPARQL federation
  - Leverage query services provided by data publishers
- Live Linked Data querying
  - Navigate through LOD by looking up URIs at query execution time
Outline

1. LOD and RDF Introduction
2. Data Warehousing Approach
   - Relational Approaches
   - Graph-Based Approaches
3. SPARQL Federation Approach
   - Distributed RDF Processing
   - SPARQL Endpoint Federation
4. Live Querying Approach
   - Traversal-based approaches
   - Index-based approaches
   - Hybrid approaches
5. Conclusions
Outline

1. LOD and RDF Introduction
2. Data Warehousing Approach
   - Relational Approaches
   - Graph-Based Approaches
3. SPARQL Federation Approach
   - Distributed RDF Processing
   - SPARQL Endpoint Federation
4. Live Querying Approach
   - Traversal-based approaches
   - Index-based approaches
   - Hybrid approaches
5. Conclusions
Traditional Hypertext-based Web Access

Data exposed to the Web via HTML
Linked Data Publishing Principles

IMDb

World Book

Data model: RDF
Global identifier: URI
Access mechanism: HTTP
Connection: data links

(https://...linkedmdb.../Shining, releaseDate, 23 May 1980)
(https://...linkedmdb.../Shining, filmLocation, http://cia.../UK)
(https://...linkedmdb.../29704, actedIn, http://...linkedmdb.../Shining)

(https://cia.../UK, hasPopulation, 63230000)
RDF Introduction

- Everything is an uniquely named resource

http://data.linkedmdb.org/resource/actor/JN29704
RDF Introduction

- Everything is an **uniquely** named resource
- Prefixes can be used to shorten the names

```
xmlns:y=http://data.linkedmdb.org/resource/actor/
y:JN29704
y:JN29704:hasName "Jack Nicholson"
y:JN29704:BornOnDate "1937-04-22"
y:TS2014:title "The Shining"
y:TS2014:releaseDate "1980-05-23"
y:TS2014
```
RDF Introduction

- Everything is an **uniquely** named resource
- Prefixes can be used to shorten the names
- Properties of resources can be defined

XML:

```xml
<Y:resource y:JN29704=
    xmlns:y=http://data.linkedmdb.org/resource/actor/ y:JN29704
    y:JN29704:hasName "Jack Nicholson"
    y:JN29704:BornOnDate "1937-04-22"

y:TS2014:title "The Shining"
y:TS2014:releaseDate "1980-05-23"
```
RDF Introduction

- Everything is an uniquely named resource
- Prefixes can be used to shorten the names
- Properties of resources can be defined
- Relationships with other resources can be defined

```
xmlns:y=http://data.linkedmdb.org/resource/actor/
y:JN29704

y:JN29704:hasName “Jack Nicholson”
y:JN29704:BornOnDate “1937-04-22”

JN29704:movieActor

```
RDF Introduction

- Everything is an uniquely named resource
- Prefixes can be used to shorten the names
- Properties of resources can be defined
- Relationships with other resources can be defined
- Resource descriptions can be contributed by different people/groups and can be located anywhere in the web
  - Integrated web “database”

XMLNS:y=http://data.linkedmdb.org/resource/actor/
  y:JN29704

y:JN29704:hasName “Jack Nicholson”
y:JN29704:BornOnDate “1937-04-22”

JN29704:movieActor

y:TS2014


A MASTERCipe OF MODERN HORROR
THE SHINING
RDF Data Model

- Triple: Subject, Predicate (Property), Object \((s, p, o)\)
  - **Subject**: the entity that is described (URI or blank node)
  - **Predicate**: a feature of the entity (URI)
  - **Object**: value of the feature (URI, blank node or literal)
- \((s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)\)
- Set of RDF triples is called an **RDF graph**

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://...imdb.../29704">http://...imdb.../29704</a></td>
<td>movie:actor_name</td>
<td>“Jack Nicholson”</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
### RDF Example Instance

Prefixes:
- `mdb=http://data.linkedmdb.org/resource/`
- `geo=http://sws.geonames.org/`
- `bm=http://wifo5-03.informatik.uni-mannheim.de/bookmashup/`
- `lexvo=http://lexvo.org/id/`
- `wp=http://en.wikipedia.org/wiki/`

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mdb:film/2014</code></td>
<td><code>rdfs:label</code></td>
<td>&quot;The Shining&quot;</td>
</tr>
<tr>
<td><code>mdb:film/2014</code></td>
<td><code>movie:director</code></td>
<td><code>mdb:director/8476</code></td>
</tr>
<tr>
<td><code>mdb:film/2014</code></td>
<td><code>movie:music_contributor</code></td>
<td><code>mdb: music_contributor/4110</code></td>
</tr>
<tr>
<td><code>mdb:film/2014</code></td>
<td><code>foaf:based_near</code></td>
<td><code>geo:2635167</code></td>
</tr>
<tr>
<td><code>mdb:director/8476</code></td>
<td><code>movie:director_name</code></td>
<td>&quot;Stanley Kubrick&quot;</td>
</tr>
<tr>
<td><code>mdb:film/2685</code></td>
<td><code>movie:director</code></td>
<td><code>mdb:director/8476</code></td>
</tr>
<tr>
<td><code>mdb:film/2685</code></td>
<td><code>rdfs:label</code></td>
<td>&quot;A Clockwork Orange&quot;</td>
</tr>
<tr>
<td><code>mdb:film/424</code></td>
<td><code>movie:director</code></td>
<td><code>mdb:director/8476</code></td>
</tr>
<tr>
<td><code>mdb:actor/29704</code></td>
<td><code>movie:actor_name</code></td>
<td>&quot;Jack Nicholson&quot;</td>
</tr>
<tr>
<td><code>mdb:film/1267</code></td>
<td><code>movie:actor</code></td>
<td><code>mdb:actor/29704</code></td>
</tr>
<tr>
<td><code>mdb:film/1267</code></td>
<td><code>rdfs:label</code></td>
<td>&quot;The Last Tycoon&quot;</td>
</tr>
<tr>
<td><code>mdb:film/3418</code></td>
<td><code>movie:actor</code></td>
<td><code>mdb:actor/29704</code></td>
</tr>
<tr>
<td><code>mdb:film/3418</code></td>
<td><code>rdfs:label</code></td>
<td>&quot;The Passenger&quot;</td>
</tr>
<tr>
<td><code>geo:2635167</code></td>
<td><code>gn:name</code></td>
<td>&quot;United Kingdom&quot;</td>
</tr>
<tr>
<td><code>geo:2635167</code></td>
<td><code>gn:population</code></td>
<td>62348447</td>
</tr>
<tr>
<td><code>geo:2635167</code></td>
<td><code>gn:wikipediaArticle</code></td>
<td><code>wp:United_Kingdom</code></td>
</tr>
<tr>
<td><code>bm:books/0743424425</code></td>
<td><code>dc:creator</code></td>
<td><code>bm:persons/Stephen+King</code></td>
</tr>
<tr>
<td><code>bm:books/0743424425</code></td>
<td><code>rev:rating</code></td>
<td>4.7</td>
</tr>
<tr>
<td><code>bm:books/0743424425</code></td>
<td><code>scom:hasOffer</code></td>
<td><code>bm:offers/0743424425amazonOffer</code></td>
</tr>
<tr>
<td><code>lexvo:iso639-3/eng</code></td>
<td><code>rdfs:label</code></td>
<td>&quot;English&quot;</td>
</tr>
<tr>
<td><code>lexvo:iso639-3/eng</code></td>
<td><code>lvont:usedIn</code></td>
<td><code>lexvo:iso3166/CA</code></td>
</tr>
<tr>
<td><code>lexvo:iso639-3/eng</code></td>
<td><code>lvont:usesScript</code></td>
<td><code>lexvo:script/Latn</code></td>
</tr>
</tbody>
</table>
Web of Linked Data

Given a finite or countably infinite set $\mathcal{D}$ of Linked Documents, a Web of Linked Data is a tuple $W = (D, adoc, data)$ where:

- $D \subseteq \mathcal{D}$,
- $adoc$ is a partial mapping from URIs to $D$, and
- $data$ is a total mapping from $D$ to finite sets of RDF triples.
Linked Data Model

Web of Linked Data

Given a finite or countably infinite set $\mathcal{D}$ of Linked Documents, a Web of Linked Data is a tuple $W = (D, adoc, data)$ where:

- $D \subseteq \mathcal{D}$,
- $adoc$ is a partial mapping from URIs to $D$, and
- $data$ is a total mapping from $D$ to finite sets of RDF triples.

Data Links

A Web of Linked Data $W = (D, adoc, data)$ contains a data link from document $d \in D$ to document $d' \in D$ if there exists a URI $u$ such that:

- $u$ is mentioned in an RDF triple $t \in data(d)$, and
- $d' = adoc(u)$. 
RDF Query Model – SPARQL

- Query Model - **SPARQL Protocol and RDF Query Language**
- Given $U$ (set of URIs), $L$ (set of literals), and $V$ (set of variables), a SPARQL expression is defined recursively:
  - an atomic triple pattern, which is an element of
    $$(U \cup V) \times (U \cup V) \times (U \cup V \cup L)$$
    - $?x rdfs:label “The Shining”
    - $P$ FILTER $R$, where $P$ is a graph pattern expression and $R$ is a built-in SPARQL condition (i.e., analogous to a SQL predicate)
      - $?x rev:rating ?p FILTER(?p > 3.0)
    - $P1$ AND/OPT/UNION $P2$, where $P1$ and $P2$ are graph pattern expressions
- Example:
  - ```sql
SELECT ?name
WHERE {
  ?d movie:director_name "Stanley Kubrick".
  FILTER(?r > 4.0)
}
```
SPARQL Queries

SELECT ?name
WHERE {
?d movie:director_name "Stanley Kubrick".
FILTER(?r > 4.0)
}

FILTER(?r > 4.0)
Outline

1. LOD and RDF Introduction

2. Data Warehousing Approach
   Relational Approaches
   Graph-Based Approaches

3. SPARQL Federation Approach
   Distributed RDF Processing
   SPARQL Endpoint Federation

4. Live Querying Approach
   Traversal-based approaches
   Index-based approaches
   Hybrid approaches

5. Conclusions
Naïve Triple Store Design

```
SELECT ?name
WHERE {
  ?d movie:director_name "Stanley Kubrick".
  FILTER(?r > 4.0)
}
```

<table>
<thead>
<tr>
<th>Subject</th>
<th>Property</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>mdb:film/2014</td>
<td>rdfs:label</td>
<td>&quot;The Shining&quot;</td>
</tr>
<tr>
<td>mdb:film/2014</td>
<td>movie:director</td>
<td>mdb:director/8476</td>
</tr>
<tr>
<td>mdb:director/8476</td>
<td>movie:director_name</td>
<td>&quot;Stanley Kubrick&quot;</td>
</tr>
<tr>
<td>mdb:film/2685</td>
<td>movie:director</td>
<td>mdb:director/8476</td>
</tr>
<tr>
<td>mdb:film/2685</td>
<td>rdfs:label</td>
<td>&quot;A Clockwork Orange&quot;</td>
</tr>
<tr>
<td>mdb:film/424</td>
<td>movie:director</td>
<td>mdb:director/8476</td>
</tr>
<tr>
<td>mdb:film/424</td>
<td>rdfs:label</td>
<td>&quot;Spartacus&quot;</td>
</tr>
<tr>
<td>mdb:actor/29704</td>
<td>movie:actor_name</td>
<td>&quot;Jack Nicholson&quot;</td>
</tr>
<tr>
<td>mdb:film/1267</td>
<td>movie:actor</td>
<td>mdb:actor/29704</td>
</tr>
<tr>
<td>mdb:film/1267</td>
<td>rdfs:label</td>
<td>&quot;The Last Tycoon&quot;</td>
</tr>
<tr>
<td>mdb:film/3418</td>
<td>movie:actor</td>
<td>mdb:actor/29704</td>
</tr>
<tr>
<td>mdb:film/3418</td>
<td>rdfs:label</td>
<td>&quot;The Passenger&quot;</td>
</tr>
<tr>
<td>geo:2635167</td>
<td>gn:name</td>
<td>&quot;United Kingdom&quot;</td>
</tr>
<tr>
<td>geo:2635167</td>
<td>gn:population</td>
<td>62348447</td>
</tr>
<tr>
<td>geo:2635167</td>
<td>gn:wikipediaArticle</td>
<td>wp:United_Kingdom</td>
</tr>
<tr>
<td>bm:books/0743424425</td>
<td>dc:creator</td>
<td>bm:persons/Stephen+King</td>
</tr>
<tr>
<td>bm:books/0743424425</td>
<td>rev:rating</td>
<td>4.7</td>
</tr>
<tr>
<td>bm:books/0743424425</td>
<td>scom:hasOffer</td>
<td>bm:offers/0743424425amazonOffer</td>
</tr>
<tr>
<td>lexvo:iso639-3/eng</td>
<td>rdfs:label</td>
<td>&quot;English&quot;</td>
</tr>
<tr>
<td>lexvo:iso639-3/eng</td>
<td>lvont:usedIn</td>
<td>lexvo:iso3166/CA</td>
</tr>
<tr>
<td>lexvo:iso639-3/eng</td>
<td>lvont:usesScript</td>
<td>lexvo:script/Latn</td>
</tr>
</tbody>
</table>

Easy to implement but too many self-joins!
Naïve Triple Store Design

SELECT ?name
WHERE {
  ?d movie:director_name "Stanley Kubrick".
  FILTER(?r > 4.0)
}

## SELECT T1.object FROM T as T1, T as T2, T as T3, T as T4, T as T5
WHERE T1.p="rdfs:label"
AND T2.p="movie:relatedBook"
AND T3.p="movie:director"
AND T4.p="rev:rating"
AND T5.p="movie:director_name"
AND T1.s=T2.s
AND T1.s=T3.s
AND T2.o=T4.s
AND T3.o=T5.s
AND T4.o > 4.0
AND T5.o="Stanley Kubrick"
SELECT ?name
WHERE {
  ?d movie:director_name "Stanley Kubrick" .
  FILTER(?r > 4.0)
}

Easy to implement but too many self-joins!

WHERE T1.p="rdfs:label"
AND T2.p="movie:relatedBook"
AND T3.p="movie:director"
AND T4.p="rev:rating"
AND T5.p="movie:director_name"
AND T1.s=T2.s
AND T1.s=T3.s
AND T2.o=T4.s
AND T3.o=T5.s
AND T4.o > 4.0
AND T5.o="Stanley Kubrick"
Exhaustive Indexing

- RDF-3X [Neumann and Weikum, 2008, 2009], Hexastore [Weiss et al., 2008]
- Strings are mapped to ids using a mapping table

Original triple table

<table>
<thead>
<tr>
<th>Subject</th>
<th>Property</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>mdb:director/8476</td>
<td>movie:director_name</td>
<td>“Stanley Kubrick”</td>
</tr>
<tr>
<td>mdb:film/2685</td>
<td>movie:director</td>
<td>mdb:director/8476</td>
</tr>
</tbody>
</table>

Encoded triple table

<table>
<thead>
<tr>
<th>Subject</th>
<th>Property</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>5</td>
</tr>
</tbody>
</table>

Mapping table

<table>
<thead>
<tr>
<th>ID</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>mdb: film/2014</td>
</tr>
<tr>
<td>1</td>
<td>rdfs:label</td>
</tr>
<tr>
<td>2</td>
<td>“The Shining”</td>
</tr>
<tr>
<td>3</td>
<td>movie:initial_release_date</td>
</tr>
<tr>
<td>4</td>
<td>“1980-05-23”</td>
</tr>
<tr>
<td>5</td>
<td>mdb:director/8476</td>
</tr>
<tr>
<td>6</td>
<td>movie:director_name</td>
</tr>
<tr>
<td>7</td>
<td>“Stanley Kubrick”</td>
</tr>
<tr>
<td>8</td>
<td>mdb:film/2685</td>
</tr>
<tr>
<td>9</td>
<td>movie:director</td>
</tr>
</tbody>
</table>
Exhaustive Indexing

- RDF-3X [Neumann and Weikum, 2008, 2009], Hexastore [Weiss et al., 2008]
- Strings are mapped to ids using a mapping table
- Triples are indexed in a clustered B+ tree in lexicographic order

<table>
<thead>
<tr>
<th>Subject</th>
<th>Property</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

B+ tree

Easy querying through mapping table
Exhaustive Indexing

- RDF-3X [Neumann and Weikum, 2008, 2009], Hexastore [Weiss et al., 2008]
- Strings are mapped to ids using a mapping table
- Triples are indexed in a clustered B+ tree in lexicographic order
- Create indexes for permutations of the three columns: SPO, SOP, PSO, POS, OPS, OSP

<table>
<thead>
<tr>
<th>Subject</th>
<th>Property</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

B+ tree

Easy querying through mapping table
Exhaustive Indexing–Query Execution

- Each triple pattern can be answered by a range query
- Joins between triple patterns computed using merge join
- Join order is easy due to extensive indexing

<table>
<thead>
<tr>
<th>Subject</th>
<th>Property</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>mdb: film/2014</td>
</tr>
<tr>
<td>1</td>
<td>rdfs:label</td>
</tr>
<tr>
<td>2</td>
<td>“The Shining”</td>
</tr>
<tr>
<td>3</td>
<td>movie:initial_release_date</td>
</tr>
<tr>
<td>4</td>
<td>“1980-05-23”</td>
</tr>
<tr>
<td>5</td>
<td>mdb:director/8476</td>
</tr>
<tr>
<td>6</td>
<td>movie:director_name</td>
</tr>
<tr>
<td>7</td>
<td>“Stanley Kubrick”</td>
</tr>
<tr>
<td>8</td>
<td>mdb:film/2685</td>
</tr>
<tr>
<td>9</td>
<td>movie:director</td>
</tr>
</tbody>
</table>
Exhaustive Indexing–Query Execution

- Each triple pattern can be answered by a range query
- Joins between triple patterns computed using merge join
- Join order is easy due to extensive indexing

Advantages

- Eliminates some of the joins – they become range queries
- Merge join is easy and fast

<table>
<thead>
<tr>
<th>ID</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>movie:initial_release_date</td>
</tr>
<tr>
<td>4</td>
<td>“1980-05-23”</td>
</tr>
<tr>
<td>5</td>
<td>mdb:director/8476</td>
</tr>
<tr>
<td>6</td>
<td>movie:director_name</td>
</tr>
<tr>
<td>7</td>
<td>“Stanley Kubrick”</td>
</tr>
<tr>
<td>8</td>
<td>mdb:film/2685</td>
</tr>
<tr>
<td>9</td>
<td>movie:director</td>
</tr>
</tbody>
</table>
Exhaustive Indexing–Query Execution

- Each triple pattern can be answered by a range query
- Joins between triple patterns computed using merge join
- Join order is easy due to extensive indexing

Advantages

- Eliminates some of the joins – they become range queries
- Merge join is easy and fast

Disadvantages

- Space usage
- Expensive updates
Property Tables

- Grouping by entities; Jena [Wilkinson, 2006], DB2-RDF [Bornea et al., 2013]

- **Clustered property table:** group together the properties that tend to occur in the same (or similar) subjects

- **Property-class table:** cluster the subjects with the same type of property into one property table

<table>
<thead>
<tr>
<th>Subject</th>
<th>Property</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>mdb:film/2014</td>
<td>movie:director</td>
<td>mdb:director/8476</td>
</tr>
<tr>
<td>mdb:film/2685</td>
<td>movie:director</td>
<td>“A Clockwork Orange”</td>
</tr>
<tr>
<td>mdb:actor/29704</td>
<td>rdfs:label</td>
<td>“Jack Nicholson”</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subject</th>
<th>refs:label</th>
<th>movie:director</th>
</tr>
</thead>
<tbody>
<tr>
<td>mob:film/2685</td>
<td>“The Clockwork Orange”</td>
<td>mob:director/8476</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subject</th>
<th>movie:actor_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>mdb:actor/29704</td>
<td>“Jack Nicholson”</td>
</tr>
</tbody>
</table>

NCSU/2016-04-15
Property Tables

- Grouping by entities; Jena [Wilkinson, 2006], DB2-RDF [Bornea et al., 2013]

- Clustered property table: group together the properties that tend to occur in the same (or similar) subjects

Property-class table: cluster the subjects with the same type of property into one property table

Advantages

- Fewer joins
- If the data is structured, we have a relational system – similar to normalized relations

<table>
<thead>
<tr>
<th>Subject</th>
<th>refs:label</th>
<th>movie:director</th>
</tr>
</thead>
<tbody>
<tr>
<td>mob:film/2685</td>
<td>“The Clockwork Orange”</td>
<td>mob:director/8476</td>
</tr>
<tr>
<td>mob:actor/29704</td>
<td>name “Jack Nicholson”</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subject</th>
<th>movie:actor_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>mdb:actor</td>
<td>“Jack Nicholson”</td>
</tr>
</tbody>
</table>
Property Tables

- Grouping by entities; Jena [Wilkinson, 2006], DB2-RDF [Bornea et al., 2013]

- **Clustered property table**: group together the properties that tend to occur in the same (or similar) subjects

<table>
<thead>
<tr>
<th>Subject</th>
<th>Property</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>mdb:film/2014</td>
<td>rdfs:label</td>
<td>&quot;The Shining&quot;</td>
</tr>
<tr>
<td>mdb:film/2014</td>
<td>movie:director</td>
<td>mdb:director/8476</td>
</tr>
<tr>
<td>mdb:film/2685</td>
<td>movie:director</td>
<td>mdb:director/8476</td>
</tr>
<tr>
<td>mdb:actor/29704</td>
<td>name</td>
<td>&quot;Jack Nicholson&quot;</td>
</tr>
</tbody>
</table>

---

**Advantages**

- Fewer joins
- If the data is structured, we have a relational system – similar to normalized relations

<table>
<thead>
<tr>
<th>mdb:film/2685</th>
<th>movie:director</th>
<th>mdb:director/8476</th>
</tr>
</thead>
<tbody>
<tr>
<td>mob:film/2014</td>
<td>&quot;The Shining&quot;</td>
<td>mob:director/8476</td>
</tr>
</tbody>
</table>

**Disadvantages**

- Potentially a lot of NULLs
- Clustering is not trivial
- Multi-valued properties are complicated
Binary Tables

- Grouping by properties: For each property, build a two-column table, containing both subject and object, ordered by subjects [Abadi et al., 2007, 2009]
- Also called vertical partitioned tables
- $n$ two column tables ($n$ is the number of unique properties in the data)

<table>
<thead>
<tr>
<th>Subject</th>
<th>Property</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>mdb:film/2014</td>
<td>rdfs:label</td>
<td>&quot;The Shining&quot;</td>
</tr>
<tr>
<td>mdb:film/2014</td>
<td>movie:director</td>
<td>mdb:director/8476</td>
</tr>
<tr>
<td>mdb:film/2685</td>
<td>movie:director</td>
<td>&quot;A Clockwork Orange&quot;</td>
</tr>
<tr>
<td>mdb:film/2685</td>
<td>rdfs:label</td>
<td>&quot;Jack Nicholson&quot;</td>
</tr>
<tr>
<td>mdb:actor/29704</td>
<td>movie:actor_name</td>
<td>&quot;Jack Nicholson&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subject</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>mob:film/2014</td>
<td>&quot;The Shining&quot;</td>
</tr>
<tr>
<td>mob:film/2685</td>
<td>&quot;The Clockwork Orange&quot;</td>
</tr>
</tbody>
</table>
Binary Tables

- Grouping by properties: For each property, build a two-column table, containing both subject and object, ordered by subjects.

  [Abadi et al., 2007, 2009]

  Also called vertical partitioned tables. $n$ two column tables ($n$ is the number of unique properties in the data).

<table>
<thead>
<tr>
<th>Subject</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>mob:film/2014</td>
<td>&quot;The Shining&quot;</td>
</tr>
<tr>
<td>mob:film/2685</td>
<td>&quot;The Clockwork Orange&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subject</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>mdb:film/2014</td>
<td>&quot;The Shining&quot;</td>
</tr>
<tr>
<td>mdb:film/2685</td>
<td>&quot;The Clockwork Orange&quot;</td>
</tr>
<tr>
<td>mdb:director/8476</td>
<td></td>
</tr>
<tr>
<td>mdb:actor/29704</td>
<td>&quot;Jack Nicholson&quot;</td>
</tr>
</tbody>
</table>

Advantages

- Supports multi-valued properties
- No NULLs
- No clustering
- Read only needed attributes (i.e. less I/O)
- Good performance for subject-subject joins
Binary Tables

- Grouping by properties: For each property, build a two-column table, containing both subject and object, ordered by subjects

<table>
<thead>
<tr>
<th>Subject</th>
<th>Property</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>mdb:film/2014</td>
<td>rdfs:label</td>
<td>&quot;The Shining&quot;</td>
</tr>
<tr>
<td>mdb:film/2014</td>
<td>movie:director</td>
<td>mdb:director/8476</td>
</tr>
<tr>
<td>mdb:film/2685</td>
<td>movie:director</td>
<td>mdb:director/8476</td>
</tr>
<tr>
<td>mdb:film/2685</td>
<td>rdfs:label</td>
<td>&quot;A Clockwork Orange&quot;</td>
</tr>
<tr>
<td>mdb:actor/29704</td>
<td>movie:actor</td>
<td>name</td>
</tr>
<tr>
<td>mob:film/2014</td>
<td>mob:film/2685</td>
<td>refs:label</td>
</tr>
</tbody>
</table>

Advantages

- Supports multi-valued properties
- No NULLs
- No clustering
- Read only needed attributes (i.e. less I/O)
- Good performance for subject-subject joins

Disadvantages

- Not useful for subject-object joins
- Expensive inserts
Graph-based Approach

- Answering SPARQL query $\equiv$ subgraph matching using homomorphism
- gStore [Zou et al., 2011, 2014], chameleon-db [Aluç et al., 2013]
Graph-based Approach

- Answering SPARQL query ≡ subgraph matching using homomorphism
- gStore [Zou et al., 2011, 2014], chameleon-db [Aluç et al., 2013]

Advantages

- Maintains the graph structure
- Full set of queries can be handled
Graph-based Approach

- Answering SPARQL query ≡ subgraph matching using homomorphism
- gStore [Zou et al., 2011, 2014], chameleon-db [Aluç et al., 2013]

Advantages

- Maintains the graph structure
- Full set of queries can be handled

Disadvantages

- Graph pattern matching is expensive
Let \( e_{81} \) be a variable. Using an appropriate hash function, we set \( m \) out of \( M \) bits in \( e_{81} \) to be '1'.

In this section, we briefly discuss the techniques used in the gStore system; full details are given in elsewhere [5], [6].

For each query, we perform the multi-way join over these candidate filtering and joining. First, we generate the candidates for each filter module, and then we select the best one from them.

Fig. 3. SPARQL and Query Graph

A. Encoding Techniques

In RDF graphs, each node represents a concept and each edge represents a relationship between concepts. To efficiently store and query these graphs, we need to encode them into a compact data structure. A common approach is to use a graph representation like the VS*-tree.

The VS*-tree is a compressed representation of an RDF graph. It stores the graph in a way that allows for efficient querying. The VS*-tree is constructed by selecting a set of vertices that cover all the edges in the graph. These vertices are then stored in a compact data structure.

The VS*-tree can be used to efficiently answer SPARQL queries. To answer a query, we first build a query graph from the query. Then, we match this query graph against the VS*-tree. The matching process involves traversing the VS*-tree and checking for matches against the edges and vertices in the query graph.

The VS*-tree is a powerful tool for efficient querying of RDF graphs. It allows for fast retrieval of data and can be used in a variety of applications such as web services and linked data platforms.
Two Systems

- 12,000 lines of C++ code under Linux (plus code for SPARQL parser)
- Encode each vertex of RDF graph as a bit array capturing the neighbourhood relationship ($G^*$)
- Build a multilevel summary tree index ($VS^*$-tree) to capture “connections”

### gStore

<table>
<thead>
<tr>
<th>Offline</th>
<th>Online</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: RDF data</td>
<td>Input: SPARQL Query</td>
</tr>
<tr>
<td>RDF Parser</td>
<td>SPARQL Parser</td>
</tr>
<tr>
<td>RDF Graph Builder</td>
<td>Query Graph</td>
</tr>
<tr>
<td>Encoding Module</td>
<td>Signature Graph</td>
</tr>
<tr>
<td>VS*-tree builder</td>
<td>Filter Module</td>
</tr>
<tr>
<td>VS*-tree</td>
<td>Node Candidate</td>
</tr>
<tr>
<td>VS*-tree Store</td>
<td>Join Module</td>
</tr>
<tr>
<td>Key-Value Store</td>
<td>Results</td>
</tr>
</tbody>
</table>

**Encoding Techniques**

- Caching strategy and multicore-based query optimization in pruning methods in Section III-A. Another technical issue is query processing by subgraph matching over the signature filtering and joining. First, we generate the candidates for each

**Build a multilevel summary tree index ($VS^*$-tree) to capture “connections”**
Two Systems

gStore

- Encode the query graph similarly (Q*)
- Find candidate matching nodes of Q* in G* using VS*-tree
- Multiway join of the candidates

- 12,000 lines of C++ code under Linux (plus code for SPARQL parser)
- Encode each vertex of RDF graph as a bit array capturing the neighbourhood relationship (G*)
- Build a multilevel summary tree index (VS*-tree) to capture “connections”
Two Systems

- 35,000 lines of C++ code under Linux (plus code for SPARQL 1.0 parser)
- Adaptivity to workload due to variability of Web workloads and the variability of composition of SPARQL triple patterns
- An experiment [Aluç et al., 2014a]
  - No single system is a sole winner across all queries
  - No single system is the sole loser across all queries, either
  - 2–5 orders of magnitude difference in the performance between the best and the worst system for a given query
  - The winner in one query may timeout in another
  - Performance difference widens as dataset size increases

- Group-by-query approach [Aluç et al., 2014b]
Outline

1. LOD and RDF Introduction
2. Data Warehousing Approach
   - Relational Approaches
   - Graph-Based Approaches
3. SPARQL Federation Approach
   - Distributed RDF Processing
   - SPARQL Endpoint Federation
4. Live Querying Approach
   - Traversal-based approaches
   - Index-based approaches
   - Hybrid approaches
5. Conclusions
Remember the Environment

- Distributed environment
- Some of the data sites can process SPARQL queries – SPARQL endpoints
- Not all data sites can process queries
Remember the Environment

- Distributed environment
- Some of the data sites can process SPARQL queries – SPARQL endpoints
- Not all data sites can process queries
- Alternatives
Remember the Environment

- Distributed environment
- Some of the data sites can process SPARQL queries – SPARQL endpoints
- Not all data sites can process queries
- Alternatives
  - Data re-distribution + query decomposition
Remember the Environment

- Distributed environment
- Some of the data sites can process SPARQL queries – SPARQL endpoints
- Not all data sites can process queries
- Alternatives
  - Data re-distribution + query decomposition
  - Data re-distribution + partial evaluation
Remember the Environment

- Distributed environment
- Some of the data sites can process SPARQL queries – SPARQL endpoints
- Not all data sites can process queries
- Alternatives
  - Data re-distribution + query decomposition
  - Data re-distribution + partial evaluation
  - SPARQL federation: just process at SPARQL endpoints
Remember the Environment

- Distributed environment
- Some of the data sites can process SPARQL queries – SPARQL endpoints
- Not all data sites can process queries
- Alternatives
  - Data re-distribution + query decomposition
  - Data re-distribution + partial evaluation
  - SPARQL federation: just process at SPARQL endpoints
  - Live querying (next section)
Distributed RDF Processing

- RDF data warehouse is partitioned and distributed
  - RDF data $D = \{D_1, \ldots, D_n\}$
  - Allocate each $D_i$ to a site
Distributed RDF Processing

- RDF data warehouse is partitioned and distributed
  - RDF data \( D = \{ D_1, \ldots, D_n \} \)
  - Allocate each \( D_i \) to a site
- Partitioning alternatives
  - Table-based (e.g., [Husain et al., 2011])
  - Graph-based (e.g., [Huang et al., 2011; Zhang et al., 2013])
  - Unit-based (e.g., [Gurajada et al., 2014; Lee and Liu, 2013])
Distributed RDF Processing [Kaoudi and Manolescu, 2015]

- RDF data warehouse is partitioned and distributed
  - RDF data $D = \{D_1, \ldots, D_n\}$
  - Allocate each $D_i$ to a site
- Partitioning alternatives
  - Table-based (e.g., [Husain et al., 2011])
  - Graph-based (e.g., [Huang et al., 2011; Zhang et al., 2013])
  - Unit-based (e.g., [Gurajada et al., 2014; Lee and Liu, 2013])
- SPARQL query decomposed $Q = \{Q_1, \ldots, Q_k\}$
- Distributed execution of $\{Q_1, \ldots, Q_k\}$ over $\{D_1, \ldots, D_n\}$
Distributed RDF Processing

- RDF data warehouse is partitioned and distributed
  - RDF data $D = \{D_1, \ldots, D_n\}$
  - Allocate each $D_i$ to a site

Partitioning alternatives
- Table-based (e.g., [Husain et al., 2011])
- Graph-based (e.g., [Huang et al., 2011; Zhang et al., 2013])
- Unit-based (e.g., [Gurajada et al., 2014; Lee and Liu, 2013])

- SPARQL query decomposed $Q = \{Q_1, \ldots, Q_k\}$
- Distributed execution of $\{Q_1, \ldots, Q_k\}$ over $\{D_1, \ldots, D_n\}$

- High performance
- Great for parallelizing centralized RDF data
- May not be possible to re-partition and re-allocate Web data (i.e., LOD)
- Query decomposition may not be easy
Partial Query Evaluation (PQE)

- RDF data warehouse is partitioned and distributed as before
  - RDF data $D = \{D_1, \ldots, D_n\}$
  - Allocate each $D_i$ to a site
Partial Query Evaluation (PQE)

- RDF data warehouse is partitioned and distributed as before
  - RDF data $D = \{D_1, \ldots, D_n\}$
  - Allocate each $D_i$ to a site
- SPARQL query is not decomposed
- Partial query evaluation – Distributed gStore [Peng et al., 2016]
  - Given function $f(s, d)$ and part of its input $s$, perform $f$'s computation that only depends on $s$ to get $f'(d)$
  - Compute $f'(d)$ when $d$ becomes available
  - Applied to, e.g., XML [Buneman et al., 2006]
Partial Query Evaluation (PQE)

- RDF data warehouse is partitioned and distributed as before
  - RDF data $D = \{D_1, \ldots, D_n\}$
  - Allocate each $D_i$ to a site
- SPARQL query is not decomposed
- Partial query evaluation – Distributed gStore [Peng et al., 2016]
  - Given function $f(s, d)$ and part of its input $s$, perform $f$’s computation that only depends on $s$ to get $f'(d)$
  - Compute $f'(d)$ when $d$ becomes available
  - Applied to, e.g., XML [Buneman et al., 2006]

- High performance due to parallelization
- Do not have to deal with query decomposition
- May not be possible to re-partition and re-allocate Web data (i.e., LOD)
- RDF storage sites need to be modified to handle partial query processing
Distributed SPARQL Using PQE

Two steps:
1. Evaluate a query at each site to find local matches
   - Query is the function and each $D_i$ is the known input
   - Inner match or local partial match

[Peng et al., 2016]
Distributed SPARQL Using PQE

Two steps:
1. Evaluate a query at each site to find local matches
   - Query is the function and each \( D_i \) is the known input
   - Inner match or local partial match
2. Assemble the partial matches to get final result
   - Crossing match
   - Centralized assembly
   - Distributed assembly
SPARQL Endpoint Federation

- Consider only the SPARQL endpoints for query execution
- No data re-partitioning/re-distribution
- Consider $D = D_1 \cup D_2 \cup \ldots \cup D_n$; $D_i$: SPARQL endpoint
SPARQL Endpoint Federation

- Consider only the SPARQL endpoints for query execution
- No data re-partitioning/re-distribution
- Consider $D = D_1 \cup D_2 \cup \ldots \cup D_n$; $D_i$: SPARQL endpoint
- SPARQL query decomposed $Q = \{Q_1, \ldots, Q_k\}$
- Distributed execution of $\{Q_1, \ldots, Q_k\}$ over $\{D_1, \ldots, D_n\}$
SPARQL Endpoint Federation

- Consider only the SPARQL endpoints for query execution
- No data re-partitioning/re-distribution
- Consider $D = D_1 \cup D_2 \cup \ldots \cup D_n$; $D_i$: SPARQL endpoint
- SPARQL query decomposed $Q = \{Q_1, \ldots, Q_k\}$
- Distributed execution of $\{Q_1, \ldots, Q_k\}$ over $\{D_1, \ldots, D_n\}$
- Systems
  - DARQ, FedEx [Schwarte et al., 2011], SPLENDID [Görlitz and Staab, 2011], ANAPSID [Acosta et al., 2011]
SPARQL Endpoint Federation

- Consider only the SPARQL endpoints for query execution
- No data re-partitioning/re-distribution
- Consider $D = D_1 \cup D_2 \cup \ldots \cup D_n$; $D_i$: SPARQL endpoint
- SPARQL query decomposed $Q = \{Q_1, \ldots, Q_k\}$
- Distributed execution of $\{Q_1, \ldots, Q_k\}$ over $\{D_1, \ldots, D_n\}$
- Systems
  - DARQ, FedX [Schwarte et al., 2011], SPLENDID [Görlitz and Staab, 2011], ANAPSID [Acosta et al., 2011]

- Data integration approach
- May be the only way to proceed if data is distributed
- Not all RDF data storage points are SPARQL endpoints
Outline

1. LOD and RDF Introduction
2. Data Warehousing Approach
   - Relational Approaches
   - Graph-Based Approaches
3. SPARQL Federation Approach
   - Distributed RDF Processing
   - SPARQL Endpoint Federation
4. Live Querying Approach
   - Traversal-based approaches
   - Index-based approaches
   - Hybrid approaches
5. Conclusions
Live Query Processing

- Not all data resides at SPARQL endpoints
- Freshness of access to data important
- Potentially countably infinite data sources
- Live querying
  - On-line execution
  - Only rely on linked data principles
- Alternatives
  - Traversal-based approaches
  - Index-based approaches
  - Hybrid approaches
SPARQL Query Semantics in Live Querying

- **Full-web semantics**
  - Scope of evaluating a SPARQL expression is **all** Linked Data
  - Query result completeness cannot be guaranteed by any (terminating) execution
SPARQL Query Semantics in Live Querying

- **Full-web semantics**
  - Scope of evaluating a SPARQL expression is **all** Linked Data
  - Query result completeness cannot be guaranteed by any (terminating) execution

- **Reachability-based query semantics**
  - Query consists of a SPARQL expression, a set of seed URIs $S$, and a reachability condition $c$
  - Scope: all data along paths of data links that satisfy the condition
  - Computationally feasible
Traversing Approaches

- Discover relevant URIs recursively by traversing (specific) data links at query execution runtime [Hartig, 2013; Ladwig and Tran, 2011]
- Implements reachability-based query semantics
  - Start from a set of seed URIs
  - Recursively follow and discover new URIs
- Important issue is selection of seed URIs
- Retrieved data serves to discover new URIs and to construct result
Traversing Approaches

- Discover relevant URIs recursively by traversing (specific) data links at query execution runtime [Hartig, 2013; Ladwig and Tran, 2011]
- Implements reachability-based query semantics

Advantages

- Easy to implement.
- No data structure to maintain.
- Important issue is selection of seed URIs
- Retrieved data serves to discover new URIs and to construct result
Traversing Approaches

- Discover relevant URIs recursively by traversing (specific) data links at query execution runtime \[\text{[Hartig, 2013; Ladwig and Tran, 2011]}\]
- Implements reachability-based query semantics

**Advantages**

- Easy to implement.
- No data structure to maintain.
- Important issue is selection of seed

**Disadvantages**

- Possibilities for parallelized data retrieval are limited
- Repeated data retrieval introduces significant query latency.
Outline

1. LOD and RDF Introduction
2. Data Warehousing Approach
   - Relational Approaches
   - Graph-Based Approaches
3. SPARQL Federation Approach
   - Distributed RDF Processing
   - SPARQL Endpoint Federation
4. Live Querying Approach
   - Traversal-based approaches
   - Index-based approaches
   - Hybrid approaches
5. Conclusions
Conclusions

- RDF and Linked Object Data seem to have considerable promise for Web data management
Conclusions

- RDF and Linked Object Data seem to have considerable promise for Web data management
- More work needs to be done
  - Query semantics
  - Adaptive system design
  - Optimizations – both in data warehousing and distributed environments
- Live querying requires significant thought to reduce latency
Conclusions

What I did not talk about:

- Not much on general distributed/parallel processing
- Not much on SPARQL semantics
- Nothing about RDFS – no schema stuff
- Nothing about entailment regimes $> 0 \Rightarrow$ no reasoning
Thank you!

Research supported by

NSERC

INNOVATION.CA

Ontario

CAS Research Technology Incubation Lab

IBM Canada Lab


References III


References V


