Distributed RDF Data Management and SPARQL Query Processing

M. Tamer Özsu
University of Waterloo
David R. Cheriton School of Computer Science
tamer.ozsu@uwaterloo.ca

April 8, 2021
This presentation draws upon collaborative research and discussions with Lei Zou (Peking University), Peng Peng (Hunan University), and Lei Chen (Hong Kong University of Science and Technology).
RDF Use Cases & Knowledge Graphs (KG)

- Yago and DBpedia extract facts from Wikipedia & represent as RDF → structural queries
- Communities build RDF data → Knowledge Graphs
  - E.g., biologists: Bio2RDF and Uniprot RDF
  - Specialized LOD → e.g., Life Science LOD (LSLOD)
- Web data integration
  - Linked Open Data (LOD) Cloud – think as a federation of KGs
RDF Data Volumes are Growing

- LOD consists of 1255 datasets with (estimated) >100B triples

- UniProt: >84B triples

- yago: 2B triples

- DBpedia: 9.5B triples

- WIKIDATA: 1.1 billion statements
RDF Data Volumes are Growing

- LOD consists of 1255 datasets with (estimated) >100B triples
- Growth is fast ...

As of March 2009

March ’09: 89 datasets

(http://lod-cloud.net/)
RDF Data Volumes are Growing

- LOD consists of 1255 datasets with (estimated) >100B triples
- Growth is fast ...

September '10: 203 datasets

(http://lod-cloud.net/)
RDF Data Volumes are Growing

- LOD consists of 1255 datasets with (estimated) >100B triples
- Growth is fast ...

April '14: 570 datasets

(http://lod-cloud.net/)
RDF Data Volumes are Growing

- LOD consists of 1255 datasets with (estimated) >100B triples
- Growth is fast ...

May ’20:
1255 datasets

(http://lod-cloud.net/)
Data Management Issues

- W3C defined data model and query language (SPARQL)
  - Declarative query processing
  - Obvious solutions not efficient
- Initial systems are single-machine (centralized)
- Growth is data set size → distributed solutions
- Growth of independent data sets → federated solutions

Scale-out

Federation

Partition 1
Partition 2
Partition 3
Partition 4
Partition 6

LinkedMDB
DBPedia
GeoNames
NYTunes

Jamendo
SWDogFood
ALTERNATIVES

Systems

Centralized
- Tentris
- Tunable-LSH
- MARVEL
- RDFox
- chameleon-db
- TripleBit
- gStore
- dipLODocusRDF
- BitMat
- SW-Store
- RDF-3X
- Hexastore
- 3Store
- WiseKG
- SPF
- SmartKG
- DaGe
- brTPF
- TFP
- DP2RPQ
- gStore-D2WORQ
- S2RDF
- WARP
- DiploCloud
- CliqueSquare
- Partout
- TriAD
- H2RDF
- SHAPE
- EAGRE
- Trinity-RDF
- H-RDF-3X
- HadoopRDF
- SHARD

Client/Server
- SPF
- SmartKG
- DaGe
- brTPF
- TFP
- WiseKG

Distributed/Parallel
- DP2RPQ
- gStore-D2WORQ
- S2RDF
- WARP
- DiploCloud
- CliqueSquare
- Partout
- TriAD
- H2RDF
- SHAPE
- EAGRE
- Trinity-RDF
- H-RDF-3X
- HadoopRDF
- SHARD
- MultiQuery
- Odyssey
- Lusail
- SemaGrow
- CoDA
- HiBiSCuS
- FedEx
- SPLENDID
- Anapsid
- DARQ

Federated
- MultiQuery
- Odyssey
- Lusail
- SemaGrow
- CoDA
- HiBiSCuS
- FedEx
- SPLENDID
- Anapsid
- DARQ
- ColChain
- PIQNIC
- Cyclades
- RDFpeers

P2P
- ColChain
- PIQNIC
- Cyclades
- RDFpeers

Streaming
- CQUEL
- C-SPARQL
- . . .
A More Fine-grained Look

Systems

Centralized
- Relational

Scale-out
- Graph-based
- Cloud-based
- Partial Evaluation

Federated
- SPARQL Endpoints
- No SPARQL Endpoints

Streaming
- P2P

Client/Server
- Query Partitioning
A More Fine-grained Look

Systems
- Centralized
  - Relational
  - Graph-based
- Scale-out
- Federated
  - SPARQL Endpoints
  - No SPARQL Endpoints
- Streaming

Client/Server
- Query Partitioning
- Cloud-based
- Partial Evaluation
- P2P
OUTLINE

RDF Introduction

Centralized Systems – Overview

Scale-out Systems

Federated Systems

Conclusions
OVERVIEW

RDF Introduction

Centralized Systems – Overview

Scale-out Systems

Federated Systems

Conclusions
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"
RDF Introduction

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

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

y:JN29704 hasName "Jack Nicholson"
y:JN29704 BornOrDate "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
- Resource descriptions can be contributed by different people/groups and can be located anywhere in the web
  - Integrated web “database”
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>

$U$: set of URIs
$B$: set of blank nodes
$L$: set of literals
## 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>mdb:film/2014</td>
<td>movie:director</td>
<td>mdb:director/8476</td>
</tr>
<tr>
<td>mdb:film/2014</td>
<td>movie:director</td>
<td>mdb:director/8476</td>
</tr>
<tr>
<td>mdb:film/2014</td>
<td>movie:music_contributor</td>
<td>mdb:music_contributor/4110</td>
</tr>
<tr>
<td>mdb:film/2014</td>
<td>foaf:based_near</td>
<td>geo:2635167</td>
</tr>
<tr>
<td>mdb:film/2685</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>
<tr>
<td>mdb:film/424</td>
<td>movie:director</td>
<td>mdb:director/8476</td>
</tr>
<tr>
<td>mdb:film/424</td>
<td>movie:actor</td>
<td>“Spartacus”</td>
</tr>
<tr>
<td>mdb:actor/29704</td>
<td>rdfs:label</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>movie:actor</td>
<td>“The Last Tycoon”</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>movie:music_contributor</td>
<td>mdb:director/8476</td>
</tr>
<tr>
<td>geo:2635167</td>
<td>gn:name</td>
<td>“United Kingdom”</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>“English”</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>
UniProt in RDF

- UniProt collects data from >150 biological resources
- Claim: “lack of a common standard to represent and link information makes data integration an expensive business” ⇒ RDF can help
UniProt in RDF – What does the data look like?

- UniProt accession for the human CYP51 protein – Q16850
- Encode it as RDF: http://purl.uniprot.org/uniprot/Q16850.rdf
UniProt in RDF – What does the data look like?

- UniProt accession for the human CYP51 protein – Q16850
- Encode it as RDF: http://purl.uniprot.org/uniprot/Q16850.rdf
- XML/RDF format

```xml
<rdf:Description rdf:about="http://purl.uniprot.org/citations/8619637">
  <rdf:type rdf:resource="http://purl.uniprot.org/core/Journal_Citation"/>
  <title>The ubiquitously expressed human CYP51 encodes lanosterol 14 alpha-demethylase, a cytochrome P450 whose expression is regulated by oxysterols.</title>
  <author>Stroemstedt M.</author>
  <author>Rozman D.</author>
  <author>Waterman M.R.</author>
  <skos:exactMatch rdf:resource="http://purl.uniprot.org/pubmed/8619637"/>
  <dcterms:identifier>doi:10.1006/abbi.1996.0193</dcterms:identifier>
  <date rdf:datatype="http://www.w3.org/2001/XMLSchema#gYear">1996</date>
  <name>Arch. Biochem. Biophys.</name>
  <volume>329</volume>
  <pages>73-81</pages>
</rdf:Description>
```
UniProt in RDF – What does the data look like?

- UniProt accession for the human CYP51 protein – Q16850
- Encode it as RDF: http://purl.uniprot.org/uniprot/Q16850.rdf
- XML/RDF format

```xml
<rdf:Description rdf:about="http://purl.uniprot.org/citations/8619637">
  <rdf:type rdf:resource="http://purl.uniprot.org/core/Journal_Citation"/>
  <title>The ubiquitously expressed human CYP51 encodes lanosterol 14 alpha-demethylase, a cytochrome P450 whose expression is regulated by oxysterols.</title>
  <author>Stroemstedt M.</author>
  <author>Rozman D.</author>
  <author>Waterman M.R.</author>
  <skos:exactMatch rdf:resource="http://purl.uniprot.org/pubmed/8619637"/>
  <dcterms:identifier>doi:10.1006/abbi.1996.0193</dcterms:identifier>
  <date rdf:datatype="http://www.w3.org/2001/XMLSchema#gYear">1996</date>
  <name>Arch. Biochem. Biophys.</name>
  <volume>329</volume>
  <pages>73-81</pages>
</rdf:Description>
```

- This can be shown as a table <Subject, Predicate, Object>
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
  
  $$(U \cup V) \times (U \cup V) \times (U \cup V \cup L)$$

  - $\exists x \ rdfs:label \ “The \ Shining”$
  - $P \ \text{FILTER} \ R$, where $P$ is a graph pattern expression and $R$ is a built-in SPARQL condition (i.e., analogous to a SQL predicate)
    - $\exists x \ rev:rating \ ?p \ \text{FILTER}(?p > 3.0)$
  - $P1 \ \text{AND/OPT/UNION} \ P2$, where $P1$ and $P2$ are graph pattern expressions

Example:

```
SELECT ?name
WHERE {
  ?m rdfs:label ?name.
  ?m movie:director ?d.
  ?d movie:director_name "Stanley Kubrick".
  ?b rev:rating ?r.
  FILTER (?r > 4.0)
}
```
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)
    - $P_1$ AND/OPT/UNION $P_2$, where $P_1$ and $P_2$ are graph pattern expressions

Example:

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

```sparql
SELECT ?name
WHERE {
    ?d movie:director_name "Stanley Kubrick".
    FILTER(?r > 4.0)
}
```
UniProt in RDF – The Data Can Be Queried

- RDF encoded UniProt data can be queried using SPARQL:
  http://sparql.uniprot.org/sparql

- Get the GO function for Q16850 (from UniProt SPARQL endpoint)
  PREFIX upc:<http://purl.uniprot.org/core/>
  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
  SELECT ?goid ?golabel
  WHERE {
    <http://purl.uniprot.org/uniprot/Q16850> a upc:Protein ;
    upc:classifiedWith ?keyword .
    ?keyword rdfs:seeAlso ?goid .
  }

- Find the differential expression of probes and the p value that map to Q16850 (from Expression Atlas SPARQL endpoint)
  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
  PREFIX atlasterms: <http://rdf.ebi.ac.uk/terms/atlas/>
  SELECT distinct ?valueLabel ?pvalue
  WHERE {
  }
  ORDER BY ASC (?pvalue)
UniProt in RDF – The Data Can Be Queried

- RDF encoded UniProt data can be queried using SPARQL:
  
  http://sparql.uniprot.org/sparql

- Get the GO function for Q16850 (from UniProt SPARQL endpoint)
  
  ```
  PREFIX upc:<http://purl.uniprot.org/core/>
  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
  SELECT ?goid ?golabel
  WHERE {
    <http://purl.uniprot.org/uniprot/Q16850> a upc:Protein ;
    upc:classifiedWith ?keyword .
    ?keyword rdfs:seeAlso ?goid .
  }
  ```

- Find the differential expression of probes and the p value that map to Q16850 (from Expression Atlas SPARQL endpoint)
  
  ```
  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
  PREFIX atlasterm: <http://rdf.ebi.ac.uk/terms/atlas/>
  SELECT distinct ?valueLabel ?pvalue
  WHERE {
  }
  ```
SPARQL Semantics

- Subgraph matching using homomorphism
- Operational semantics (for BGPs)
  - Find all triples that satisfy each triple pattern
  - Join them according to variables (s-s, s-o, o-o, p-p)

```
SELECT ?name
WHERE {
  ?m rdfs:label ?name.
  ?m movie:director ?d.
  ?d movie:director_name "Stanley Kubrick".
  ?m movie:actor ?c.
  ?c movie:actor_name "Jack Nicholson".
}
```
SPARQL Semantics

- Subgraph matching using homomorphism
- Operational semantics (for BGPs)
  - Find all triples that satisfy each *triple pattern*
  - Join them according to variables (s-s, s-o, o-o, p-p)

```sparql
SELECT ?name
WHERE {
  ?m rdfs:label ?name.
  ?m movie:director ?d.
  ?d movie:director_name "Stanley Kubrick".
  ?m movie:actor ?c.
  ?c movie:actor_name "Jack Nicholson".
}
```
SPARQL Semantics

- Subgraph matching using homomorphism
- Operational semantics (for BGPs)
  - Find all triples that satisfy each triple pattern
  - Join them according to variables (s-s, s-o, o-o, p-p)

```
SELECT ?name
WHERE {
  ?m rdfs:label ?name.
  ?m movie:director ?d.
  ?d movie:director_name "Stanley Kubrick".
  ?m movie:actor ?c.
  ?c movie:actor_name "Jack Nicholson".
}
```
SPARQL Query Shapes

Star query

Tree query

Chain (linear) query

Cycle query

Complex query
Overview

RDF Introduction

Centralized Systems – Overview
  Relational Approaches
  Graph-based Approaches

Scale-out Systems

Federated Systems

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

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

---

**Table:**

<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>moviedirector</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/29013</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:wikiPageArticle</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/074342425amazonOffer</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/Latin</td>
</tr>
</tbody>
</table>
Direct Relational Mapping

Bad Idea!...
Optimizations to Tabular Representation

1. Property table (Jena [Wilkinson, 2006], DB2-RDF [Bornea et al., 2013])
   - Group together the properties that tend to occur in the same (or similar) subjects
   - Eliminates some of the joins
   - Clustering is not trivial
   - Potentially a lot of NULLs
   - Multivalued properties are complicated

<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>movie:actor_name</td>
<td>”Jack Nicholson”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subject</th>
<th>Character</th>
<th>Name</th>
</tr>
</thead>
</table>
Optimizations to Tabular Representation

1. Property table (Jena [Wilkinson, 2006], DB2-RDF [Bornea et al., 2013])
2. Vertically partitioned tables (Binary tables) [Abadi et al., 2007, 2009]

▶ For each property, build a two-column table, containing both subject and object, ordered by subjects
▶ Grouping by properties
▶ Good for subject-subject joins but not with subject-object joins

<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>mdb:director/8476</td>
</tr>
<tr>
<td>mdb:film/2685</td>
<td>rdfs:label</td>
<td>“A Clockwork Orange”</td>
</tr>
<tr>
<td>mdb:actor/29704</td>
<td>movie:actor_name</td>
<td>“Jack Nicholson”</td>
</tr>
</tbody>
</table>

Subject | Object
---|---
mob:film/2685 | “The Clockwork Orange”
mdb:film/2014 | mdb:director/8476
mdb:film/2685 | mdb:director/8476
mdb:actor/29704 | “Jack Nicholson”
Optimizations to Tabular Representation

1. Property table (Jena [Wilkinson, 2006], DB2-RDF [Bornea et al., 2013])
2. Vertically partitioned tables (Binary tables) [Abadi et al., 2007, 2009]
3. Exhaustive indexing (RDF-3X [Neumann & Weikum, 2008, 2009], Hexastore [Weiss et al., 2008])
   - Create indexes for each permutation of the three columns: SPO, SOP, PSO, POS, OPS, OSP
   - Query components become range queries over individual relations with merge-join to combine
   - Index maintenance

<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>movie:actor_name</td>
<td></td>
</tr>
</tbody>
</table>
Graph-based Approach

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

- Answering SPARQL query \(\equiv\) 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
Overview

RDF Introduction

Centralized Systems – Overview

Scale-out Systems
   - Graph Partitioning
   - Query Partitioning Approaches
   - Partial Evaluation Approach
   - Cloud-based Approaches

Federated Systems

Conclusions
**What is the Objective**

Parallelize query processing with as little inter-site communication as possible.

▶ **Straightforward approach**
  ▶ Randomly (or by simple hashing) horizontally partition the triples
  ▶ Execute queries in parallel across partitions
  ▶ Examples: SHARD [Rohloff & Schantz, 2010], YARS2 [Harth et al., 2007], Virtuoso

▶ **Problem:** high intermediate results → high number of inter-partition joins

▶ **More intelligent partitioning** that takes into account graph connectivity
Graph Partitioning

More difficult because of edges → represent relationships

Diagram showing a graph with nodes and edges, partitioned into three partitions, each labeled as Partition 1, Partition 2, and Partition 3, with workers associated with each partition.
Graph Partitioning

- More difficult because of edges → represent relationships
- Two approaches
  - Edge-cut or vertex-disjoint
  - Vertex-cut or edge-disjoint
Graph Partitioning

- More difficult because of edges → represent relationships
- Two approaches
  - Edge-cut or vertex-disjoint
  - Vertex-cut or edge-disjoint
- Objectives
  - Allocate each vertex/edge to partitions such that partitions are mutually exclusive
  - Partitions are balanced
  - Minimize edge-/vertex-cuts to minimize communication
- Techniques are mostly workload-agnostic
**Vertex-Disjoint Partitioning**

- Put each vertex in one partition
- Edges will be cut – minimize these $\rightarrow$ causes communication
- Alternatives
  - Hashing $\rightarrow$ on the ids of the two vertices incident on edge
  - METIS family of algorithms
    - Considered the gold standard
Given an initial graph $G_0 = (V, E)$:

1. Produce a hierarchy of successively coarsened graphs $G_1, \ldots, G_n$ such that $|V(G_i)| > |V(G_j)|$ for $i < j$

2. Partition $G_n$ using some partitioning algorithm

3. Iteratively coarsen $G_n$ to $G_0$, and at each step
   a) Project the partitioning solution on graph $G_j$ to graph $G_{j-1}$
   b) Improve the partitioning of $G_0$
Vertex-Disjoint Partitioning Example
Vertex-Disjoint Partitioning Example

- Performs well for graphs with low-degree vertices
- Performs poorly on power-law graphs causing many edge-cuts
- METIS has computation overhead & cannot scale to large graphs → hashing on vertex labels

In RDF systems, minimize predicate-cuts rather than edge-cuts
Edge-Disjoint Partitioning

- Put each edge in one partition
- Vertices may need to be replicated – minimize these
- Alternatives
  - Hashing → on the ids of the two vertices incident on edge
  - Heuristics cognizant of graph characteristics → greedily decide on allocation of edge $i + 1$ based on the allocation of the previous $i$ edges to reduce vertex replication
Edge-Disjoint Partitioning Example
Edge-Disjoint Partitioning Example

+ Performs well on power-law graphs
+ Fast and highly parallelizable
- Not good for star queries
- Potentially high vertex replication
Edge-Disjoint Partitioning of our Example
Query Partitioning Approaches

- Works on partitioned RDF data
  \[ D = \{ D_1, \ldots, D_n \} \]

- SPARQL query \( Q \) is decomposed into a set of subqueries \( \{ Q_1, \ldots, Q_k \} \) \( \Rightarrow \) query graph is partitioned

- Distributed execution of \( \{ Q_1, \ldots, Q_k \} \) over \( \{ D_1, \ldots, D_n \} \)

- Examples: GraphPartition [Huang et al., 2011], WARP [Hose & Schenkel, 2013], Partout [Galarraga et al., 2014], Vertex-block [Lee & Liu, 2013]
Query Partitioning Approaches

- Works on partitioned RDF data
  \[ D = \{D_1, \ldots, D_n\} \]
- SPARQL query \( Q \) is decomposed into a set of subqueries \( \{Q_1, \ldots, Q_k\} \) ⇒ query graph is partitioned
- Distributed execution of \( \{Q_1, \ldots, Q_k\} \) over \( \{D_1, \ldots, D_n\} \)

Main Idea
Partition the data and the query in such a way that the interpartition joins are minimized
Two Examples

- **GraphPartition** [Huang et al., 2011]
  - Data is partitioned using METIS
  - Replicate vertices using *n-hop guarantee*
  - If the radius of $Q$ is not larger than $n$, $Q$ can be locally evaluated in each site
  - If the radius of $Q$ is larger than $n$, $Q$ is decomposed into several subqueries $Q_i$ that can be independently evaluated; then their results are joined
**Two Examples**

- **GraphPartition** [Huang et al., 2011]
  - Generate triple groups based on common subject or object
  - Semantic hashing triple groups
  - Allow some data replication between different partitions
- **Vertex-block** [Lee & Liu, 2013]
MPC – Minimum Predicate Cut

What is the main point?

- Vertex-disjoint partitioning objective is to minimize edge-cuts
- The main performance bottleneck in distributed SPARQL execution is inter-partition joins
  - If all predicates in a query are *internal* → execute independently over each partition without inter-partition join
- **Motivation:** Partition to maximize the number of *internal properties*
  - May increase edge-cuts, but minimizes number of unique predicate-cuts
How to Minimize Predicate-cuts

Desi_Magic

Ameesha_Patel

starring

birthplace

Maharashtra

starring

foundingDate

Kuunal_Goomer

1960-05-01

Zayeed_Khan

starring

birthplace

Saif_Ali_Khan

spouse

Kareena_Kapoor

1947-08-15

India

production

Mishti

residence

Kurbaan

foundingDate

birthplace
How to Minimize Predicate-cuts

Selecting Internal Pred.

Desi_Magic

Ameesha_Patel

Zayeed_Khan

India

Kuunal_Goomer

Saif_Ali_Khan

Kareena_Kapoor

Mishti

Maharashtra

Kurbaan

1947-08-15

1960-05-01

starring

birthplace

foundingDate

producer

birthplace

residence

foundingDate

residence

spouse

starring

starring

birthplace

birthplace

birthplace

birthplace
How to Minimize Predicate-cuts

Selecting Internal Pred.

Coarsening

Desi_Magic

starring

Zayeed_Khan

birthPlace

Saif_Ali_Khan

birthPlace

Kuunal_Goomer

producer

Kareena_Kapoor

1947-08-15

India

residence

Maharashtra

foundingDate

Mishti

Kurbaan

starring

Ameesha_Patel

starring

1960-05-01
How to Minimize Predicate-cuts

- Selecting Internal Pred.
- Coarsening
- Partitioning Coarse Graph

Desi_Magic

- starring: Ameesha_Patel
- producer: Kuunal_Goomer

Ameesha_Patel

- birthplace

Zayeed_Khan

- birthplace
- residence: Mishti

Maharashtra

- foundingDate: 1960-05-01

India

- birthplace

Kuunal_Goomer

- residence

Kareena_Kapoor

- birthplace
- spouse: Saif_Ali_Khan

Saif_Ali_Khan

- birthplace

Saif_Ali_Khan

- starring: Super-Vertex-2

Kurbaan

- birthplace
- starring: Super-Vertex-1

Mishti

- residence

Super-Vertex-1

- birthPlace

Super-Vertex-2

- birthPlace
How to Minimize Predicate-cuts

Selecting Internal Pred.  Coarsening  Partitioning Coarse Graph  Uncoarsening

Desi_Magic

Ameesha_Patel

starring

birthplace

Maharashtra

1960-05-01

foundingDate

Zayeed_Khan

birthplace

Saif_Ali_Khan

birthPlace

Kareena_Kapoor

Super-Vertex-1

Super-Vertex-2

Mishti

residence

Kuunal_Goomer

producer

1947-08-15

India

Saif_Ali_Khan

residence

spouse

starring

Kurbaan

birthPlace

birthPlace

birthPlace
MPC-based Query Processing

- Objective is to increase the *independently executable queries* (IEQ)
  - Star queries
  - Queries not containing any crossing predicates
  - Queries connected when crossing predicate edges are removed

![Diagram of query processing](image-url)
MPC-based Query Processing

- Objective is to increase the independently executable queries (IEQ)
  - Star queries
  - Queries not containing any crossing predicates
  - Queries connected when crossing predicate edges are removed

- Non-IEQ → decompose into a set of IEQs and join the results
  - Remove crossing predicate edges & edges with variables
  - Form subqueries
  - Add crossing predicate edges to subqueries
### Performance Advantage

<table>
<thead>
<tr>
<th></th>
<th>MPC</th>
<th>VC</th>
<th>METIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUBM</td>
<td>100</td>
<td>0</td>
<td>43</td>
</tr>
<tr>
<td>WatDiv</td>
<td>60</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>MusicBrains</td>
<td>86</td>
<td>14</td>
<td>57</td>
</tr>
<tr>
<td>YAGO2</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DBpedia</td>
<td>75</td>
<td>24</td>
<td>47</td>
</tr>
<tr>
<td>LinkedGeoData</td>
<td>100</td>
<td>84</td>
<td>97</td>
</tr>
</tbody>
</table>

Independently Executable Queries

**LUBM Query Performance**

Minimizing predicate-cuts can significantly improve the percentage of independently executable queries resulting in improved query performance.
Query Partitioning Approach – General Comments

+ High performance
+ Great for parallelizing centralized RDF data
  - Query decomposition may not be easy
  - Each proposed approach requires a specific graph partitioning strategy – no generic partitioning

It would be interesting to consider adapting the relational distributed query processing methodology
**Partitioned Query Evaluation Methodology**

**Adapting Relational Approach**

**Input:** Declarative query  
**Output:** Algebraic query  
Transforms query to algebraic form over $D$

**Input:** Algebraic query  
**Output:** Partitioned query  
Transforms query over $D$ to query over $D = \{D_1, \ldots, D_n\}$ determining $\{Q_1, \ldots, Q_k\}$

**Input:** $\{Q_1, \ldots, Q_k\}$  
**Output:** Distributed plan  
Decide how the results of $\{Q_1, \ldots, Q_k\}$ are combined

**Input:** $\{Q_1, \ldots, Q_k\}$  
**Output:** Local results  
Each $Q_i$ executed at one site over one $D_j$

**Input:** Local results  
**Output:** Final result  
Combine results of $\{Q_1, \ldots, Q_k\}$ according to distributed plan

There isn’t much work on algebraic SPARQL processing/optimization.
Partitioned Query Evaluation Methodology
Adapting Relational Approach

Input: Declarative query
Output: Algebraic query
Transforms query to algebraic form over $D$

Input: Algebraic query
Output: Partitioned query
Transforms query over $D$ to query over $D = \{D_1, \ldots, D_n\}$

Input: $\{Q_1, \ldots, Q_k\}$
Output: Distributed plan
Decide how the results of $\{Q_1, \ldots, Q_k\}$ are combined

Input: $\{Q_1, \ldots, Q_k\}$
Output: Local results
Each $Q_i$ executed at one site over one $D_i$

Input: Local results
Output: Final result
Combine results of $\{Q_1, \ldots, Q_k\}$ according to distributed plan

There isn’t much work on algebraic SPARQL processing/optimization
Partial Query Evaluation (PQE)

- RDF data $D$ is partitioned $\{D_1, \ldots, D_n\}$
- SPARQL query is not decomposed
- Partial query evaluation – Distributed gStore [Peng et al., 2016]

\[ f(x) \implies f'(s, d) \implies f''(f'(s), d)) \implies \text{Final Answer} \]

- Known inputs
- Unknown inputs
- Partial results

- Query is the function and each $D_i$ and its extended vertices form the known input
Distributed SPARQL Using PQE

Two steps:

1. Evaluate a query at each site to find local matches
   - These are local partial matches
Distributed SPARQL Using PQE

Two steps:

1. Evaluate a query at each site to find local matches
   ▶ These are local partial matches

2. Assemble the partial matches to get final result
   ▶ Crossing match
PQE Framework

SPARQL Query Q

Local Matches in site $S_1$

Local Matches in site $S_2$

Local Matches in site $S_3$

Assemble Local Partial Matches

Answer

Local Evaluation Assembly
PQE Framework

Local Evaluation

Local Matches in site $S_1$

Local Matches in site $S_2$

Local Matches in site $S_3$

Assemble Local Partial Matches

Answer

SPARQL Query $Q$
PQE Framework

SPARQL Query $Q$

Local Matches in site $S_1$

Local Matches in site $S_2$

Local Matches in site $S_3$

Assemble Local Partial Matches

Answer

Assembly

Centralized

Distributed
PQE Approach – General Comments

+ High performance due to parallelization
+ Do not have to deal with query decomposition
− RDF processing systems at each site need to be modified
Cloud-based Solutions

- RDF data $D$ is partitioned into $\{D_1, \ldots, D_n\}$ and placed on cloud platforms (such as HDFS, HBase)
- SPARQL query is executed as MapReduce jobs
- Data parallel execution
- Examples: HARD [Rohloff & Schantz, 2010], HadoopRDF [Husain et al., 2011], EAGRE [Zhang et al., 2013] and JenaHBase [Khadilkar et al., 2012]
Cloud-based Solutions

- RDF data $D$ is partitioned into $\{D_1, \ldots, D_n\}$ and placed on cloud platforms (such as HDFS, HBase)
- SPARQL query is executed as MapReduce jobs
- Data parallel execution

Main Idea

Query operations are MapReduce jobs $\rightarrow$ Operator implementations on MapReduce [Li et al., 2014]
An Example – HadoopRDF

- File organization (offline)
  - Predicate split: all triples sharing the same predicate go to one file.
  - Predicate object split: further dividing the files based on the objects

```
?m hasName ?name
"hasName" file

?m bornIn ?city
"bornIn" file

?m bornOnDate ?bd
"bornOnDate" file
```

```
hasName

foundingYear

BornOnDate

rdf:type

rdf:type+country

rdf:type+city
```

[Husain et al., 2011]
**An Example – HadoopRDF**

- File organization (offline)

- **Query Evaluation**
  - **Input file selection**: For each triple pattern, select the corresponding files
  - **MapReduce join**: Joining between two or more triple patterns on a variable
  - **Order selection**: Cost-based join order selection

```
?m bornIn ?city
?m hasName ?name
?m bornOnDate ?bd
```

```
“bornIn” file
“hasName” file
“bornOnDate” file
```

MR Join
Cloud-based Approaches – General Comments

Cloud-Based RDF Data Management

Zoi Kaoudi
Ioana Manolescu
Stamatis Zampetakis

Synthesis Lectures on Data Management

H.V. Jagadish, Series Editor
Cloud-based Approaches – General Comments

- High scalability
- Inherently fault-tolerant
  - Performance bounded by MapReduce
  - Intermediate results can be large
  - Optimizations need to be implemented outside the MapReduce platform
Overview

RDF Introduction

Centralized Systems – Overview

Scale-out Systems

Federated Systems
  SPARQL Endpoints
  No SPARQL Endpoints

Conclusions
SPARQL Endpoint Federation

- 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 $\{D_1, \ldots, D_n\}$
- E.g.: SPLENDID [G"orlitz & Staab, 2011], ANAPSID [Acosta et al., 2011]
SPARQL Endpoint Federation

- 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 $\{D_1, \ldots, D_n\}$
- E.g.: SPLENDID [Görlitz & Staab, 2011], ANAPSID [Acosta et al., 2011]

Main Idea

Data integration approach $\rightarrow$ Datasets exist at SPARQL endpoints and queries are pushed to them
SPARQL Endpoint Federation

- 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 $\{D_1, \ldots, D_n\}$
- E.g.: SPLENDID [Görlitz & Staab, 2011], ANAPSID [Acosta et al., 2011]
SPARQL Endpoint Federation

- 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 $\{D_1, \ldots, D_n\}$
- E.g.: SPLENDID [Görlitz & Staab, 2011], ANAPSID

Data included at the source
Supported access patterns
Statistical information

...
Federated Query Processing

SPARQL queries

Query Decomposition & Source Selection

Local Evaluation

Join Partial Matches

Control Site RDF Sources

LinkedMDB

Jamendo

SWDogFood

DBPedia

GeoNames

NYTimes
Query Decomposition

- Each triple pattern maps to a set of RDF sources based on the values of its subject, property, and object.

```
SELECT ?x ?n
WHERE {
    ?x g:parentFeature ?k.
    ?k g:name "Canada".
    ?y sameAs ?x.
}
```
Each triple pattern maps to a set of RDF sources based on the values of its subject, property, and object.
Each triple pattern maps to a set of RDF sources based on the values of its subject, property, and object.
Data Localization

- Metadata-based approaches
  - Use the information in the metadata repository to determine which sources are relevant
  - DARQ [Quilitz & Leser, 2008]
  - QTree [Harth et al., 2010; Prasser et al., 2012]
  - HiBISCUSt [Saleem & Ngomo, 2014]
  - ...

- ASK query-based approach
  - Asking whether or not a triple pattern has an answer at a source
  - FedEx [Schwarte et al., 2011a,b]
**Query Execution**

```sparql
SELECT ?x ?n
WHERE {
  ?x g:parentFeature ?k.
  ?k g:name "Canada".
  ?y sameAs ?x.
}
```

SELECT ?x
WHERE {
  ?x g:parentFeature ?k.
  ?k g:name "Canada".
}

SELECT ?y ?x
WHERE {
  ?y sameAs ?x.
}

SELECT ?y ?x
WHERE {
  ?y sameAs ?x.
}

SELECT ?y ?x
WHERE {
  ?y sameAs ?x.
}

SELECT ?y ?x
WHERE {
  ?y sameAs ?x.
}

SELECT ?y ?n
WHERE {
}

SELECT ?y ?x
WHERE {
  ?y sameAs ?x.
}

SELECT ?y ?x
WHERE {
  ?y sameAs ?x.
}

SELECT ?y ?x
WHERE {
  ?y sameAs ?x.
}
Not All RDF Storage Sites are SPARQL Endpoints

- Use the mediator-wrapper paradigm
- Wrappers provide SPARQL endpoint functionality
- Mediators may be introduced if wrappers are thin
- E.g.: DARQ [Quilitz & Leser, 2008], FedX [Schwarte et al., 2011b]
UniProt Federation – EBI RDF Platform

Curated computational models of biological processes

Sample information for reference samples and samples for which data exist in one of the EBI’s assay databases

Curated chemical database of bioactive molecules with drug-like properties

Genome databases for vertebrates and other eukaryotic species

Gene expression data from the Gene Expression Atlas

Curated and peer-reviewed pathways
Federated Access to UniProt Collection

Get the Reactome pathways where Q16850 is associated, then get all the other proteins in that pathway and pull out their expression from the atlas, along with the GO annotations from UniProt

```sparql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX biopax3: <http://www.biopax.org/release/biopax-level3.owl#>
PREFIX atlasterm: <http://rdf.ebi.ac.uk/terms/atlas/>
PREFIX upc:<http://purl.uniprot.org/core/>
SELECT DISTINCT ?pathwayname ?expressionValue ?golabel
WHERE {
    # Get the pathways that reference Q16850
    ?pathway biopax3:pathwayComponent
    [?rel [biopax3:entityReference <http://purl.uniprot.org/uniprot/Q16850>]].

    # Get the expression for those proteins
    SERVICE <http://www.ebi.ac.uk/rdf/services/atlas/sparql> {
        ?probe atlasterm:dbXref ?dbXref.
    }

    # Get the GO functions from Uniprot
    SERVICE <http://uniprot.org/sparql> {
        ?dbXref a upc:Protein ;
        upc:classifiedWith ?keyword .
        ?keyword rdfs:seeAlso ?goid .
    }
}
```
Federated Approach – General Comments

- Data integration approach – realistic for existing knowledge graphs
  - SPARQL endpoints are unreliable (up to 64% offline at any point)
  - Not all RDF data storage points are SPARQL endpoints – mediators are not easy
Overview

RDF Introduction

Centralized Systems – Overview

Scale-out Systems

Federated Systems

Conclusions
Take-aways

- Although a lot of work & systems, still immature
- Multiple communities
  - Semantic web → functionality
  - Data management → performance
- Most work only considers BGP, not full SPARQL 1.1
- Algebra definitions are starting → potential for relational style distributed execution
- Opportunities for views, multiquery optimization [Peng et al., 2018], cost-based optimization open
- Dynamism of the underlying RDF datasets needs to be considered
  - Harder to distribute due to potentially heavy data movement


References II


References III


Rohloff, K., & Schantz, R. E. (2010). High-performance, massively scalable distributed systems using the MapReduce software framework: the SHARD triple-store. In Proc. int. workshop on programming support innovations for emerging distributed applications. (Article No. 4)


