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

Acknowledgements

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 select knowledge

2B triples

DBpedia

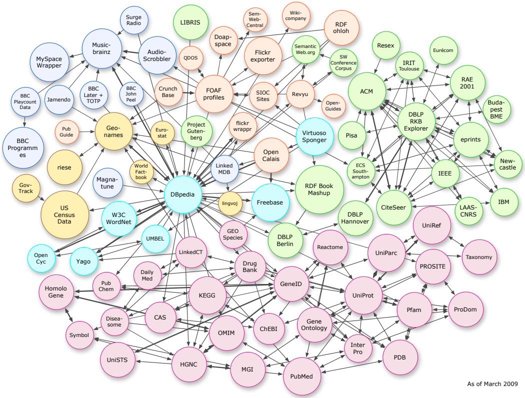
9.5B triples

WIKIDATA

1.1 billion statements

RDF Data Volumes are Growing

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

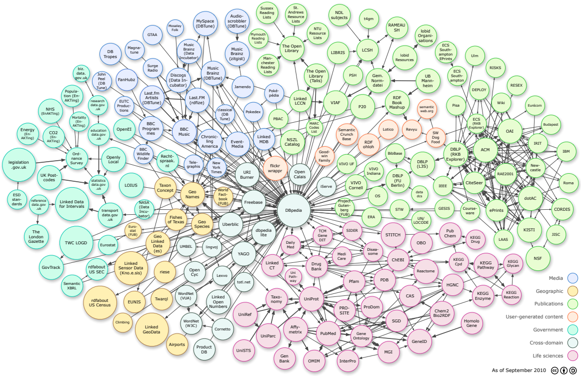


March '09:
89 datasets

(http://lod-cloud.net/)

RDF Data Volumes are Growing

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

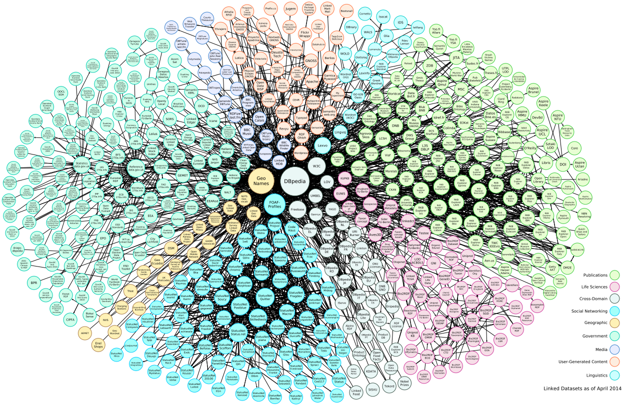


September '10:
203 datasets

(http://lod-cloud.net/)

RDF Data Volumes are Growing

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

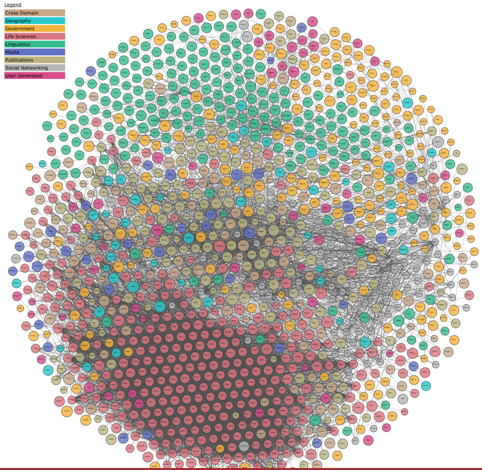


April ’14:
570 datasets

(http://lod-cloud.net/)

RDF Data Volumes are Growing

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

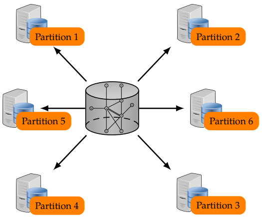


May '20:
1255 datasets

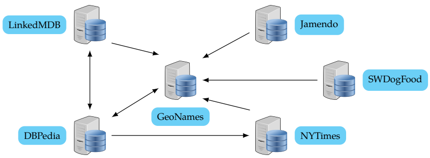
(http://lod-cloud.net/)

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

Alternatives

[Hose, 2021]

Systems

Centralized

Tentris
Tunable-LSH
MARVEL
RDFox
chameleon-db
TripleBit
gStore
dipLODocusRDF
BitMat
SW-Store
RDF-3X
Hexastore
3Store
. . .

Client/Server

WiseKG
SPF
SmartKG
DaGe
brTPF
TFP
. . .

Distributed/Parallel

DP2RPQ
gStore-D2WORQ
S2RDF
WARP
DiploCloud
CliqueSquare
Partout
TriAD
H2RDF
SHAPE
EAGRE
Trinity-RDF
H-RDF-3X
HadoopRDF
SHARD
. . .

Federated

MultiQuery
Odyssey
Lusail
SemaGrow
CoDA
HiBiSCuS
FedX
SPLENDID
Anapsid
DARQ
. . .

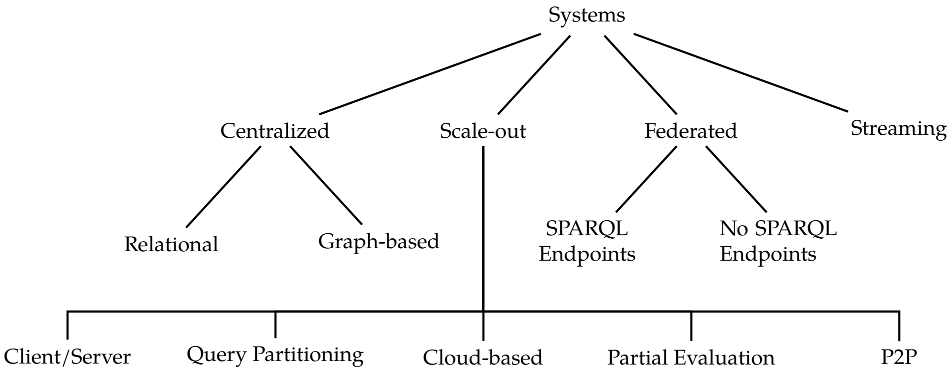
P2P

ColChain
PIQNIC
Cyclades
RDFpeers
. . .

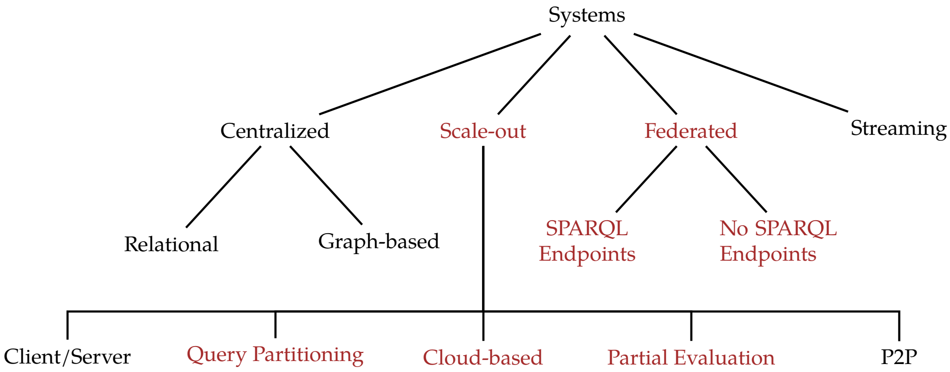
Streaming

CQUEL
C-SPARQL
. . .

A More Fine-grained Look



A More Fine-grained Look



Outline

- RDF Introduction
- Centralized Systems – Overview
- Scale-out Systems
- Federated Systems
- Conclusions

SPARQL

Scale-out

Overview

RDF Introduction

Centralized Systems – Overview

Scale-out Systems

Federated Systems

Conclusions

RDF Introduction

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

- Everything is an uniquely named resource

RDF Introduction

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

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

RDF Introduction

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

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

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

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"

movieActor

y:TS2014

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"

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

y:JN29704

y:JN29704 hasName "Jack Nicholson"
y:JN29704 BornOnDate "1937-04-22"

movieActor

y:TS2014

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

RDF Data Model

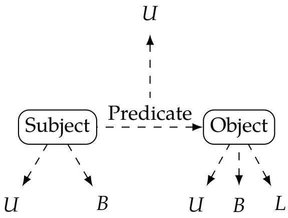
- Triple: Subject, Predicate (Property), Object (s, p, o)

  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



U: set of URIs
B: set of blank nodes
L: set of literals

Subject	Predicate	Object
http://...imdb.../film/2014	rdfs:label	"The Shining"
http://...imdb.../film/2014	movie:releaseDate	"1980-05-23"
http://...imdb.../29704	movie:actor_name	"Jack Nicholson"
...	...	...

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/

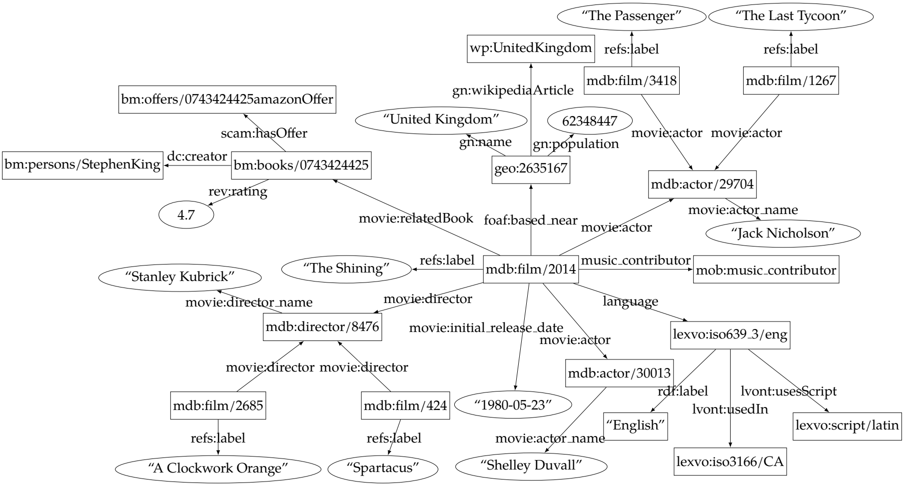
URI

Literal

URI

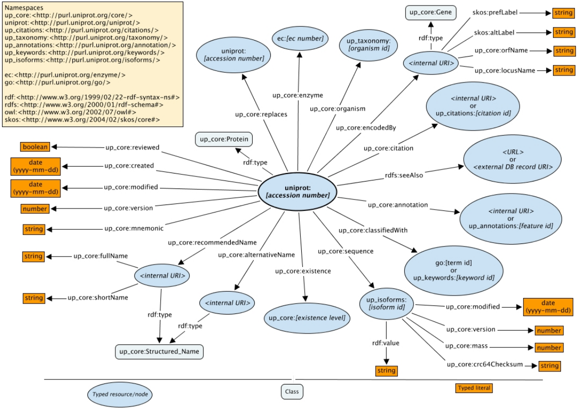
Subject	Predicate	Object
mdb: film/2014	rdfs:label	"The Shining"
mdb:film/2014	movie:initial_release_date	"1980-05-23"'
mdb:film/2014	movie:director	mdb:director/8476
mdb:film/2014	movie:actor	mdb:actor/29704
mdb:film/2014	movie:actor	mdb: actor/30013
mdb:film/2014	movie:music_contributor	mdb: music_contributor/4110
mdb:film/2014	foaf:based_near	geo:2635167
mdb:film/2014	movie:relatedBook	bm:0743424425
mdb:film/2014	movie:language	lexvo:iso639-3/eng
mdb:director/8476	movie:director_name	"Stanley Kubrick"
mdb:film/2685	movie:director	mdb:director/8476
mdb:film/2685	rdfs:label	"A Clockwork Orange"
mdb:film/424	movie:director	mdb:director/8476
mdb:film/424	rdfs:label	"Spartacus"
mdb:actor/29704	movie:actor_name	"Jack Nicholson"
mdb:film/1267	movie:actor	mdb:actor/29704
mdb:film/1267	rdfs:label	"The Last Tycoon"
mdb:film/3418	movie:actor	mdb:actor/29704
mdb:film/3418	rdfs:label	"The Passenger"
geo:2635167	gn:name	"United Kingdom"
geo:2635167	gn:population	62348447
geo:2635167	gn:wikipediaArticle	wp:United_Kingdom
bm:books/0743424425	dc:creator	bm:persons/Stephen+King
bm:books/0743424425	rev:rating	4.7
bm:books/0743424425	scom:hasOffer	bm:offers/0743424425amazonOffer
lexvo:iso639-3/eng	rdfs:label	"English"
lexvo:iso639-3/eng	lvont:usedIn	lexvo:iso3166/CA
lexvo:iso639-3/eng	lvont:usesScript	lexvo:script/Latn

RDF Graph


"The Passenger"
"The Last Tycoon"
wp:UnitedKingdom
refs:label
refs:label
mdb:film/3418
mdb:film/1267
gn:wikipediaArticle
bm:offers/0743424425amazonOffer
"United Kingdom"
62348447
scam:hasOffer
movie:actor
movie:actor
gn:name
gn:population
dc:creator
bm:persons/StephenKing
bm:books/0743424425
geo:2635167
mdb:actor/29704
rev:rating
4.7
movie:relatedBook
foaf:based_near
movie:actor
movie:actor_name
"Jack Nicholson"
"The Shining"
refs:label
mdb:film/2014
music_contributor
mob:music_contributor
"Stanley Kubrick"
movie:director_name
movie:director
language
mdb:director/8476
movie:initial_release_date
movie:actor
lexvo:iso639_3/eng
movie:director
movie:director
mdb:actor/30013
mdb:film/2685
mdb:film/424
"1980-05-23"
rdf:label
lvont:usesScript
lvont:usedIn
refs:label
refs:label
movie:actor_name
"English"
lexvo:script/latin
lexvo:iso3166/CA
"A Clockwork Orange"
"Spartacus"
"Shelley Duvall"

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

Subject

```
<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"/>
<foaf:primaryTopicOf rdf:resource="https://www.ncbi.nlm.nih.gov/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>
```

Predicate

Object

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

Subject

```
<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"/>
<foaf:primaryTopicOf rdf:resource="https://www.ncbi.nlm.nih.gov/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>
```

Predicate

Object

- This can be shown as a table <Subject, Predicate, Object >

RDF Query Model – SPARQL

- Query Model - **S**PARQL **P**rotocol **a**nd **R**DF **Q**uery **L**anguage
- 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

RDF Query Model – SPARQL

- Query Model - **S**PARQL **P**rotocol **a**nd **R**DF **Q**uery **L**anguage
- 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

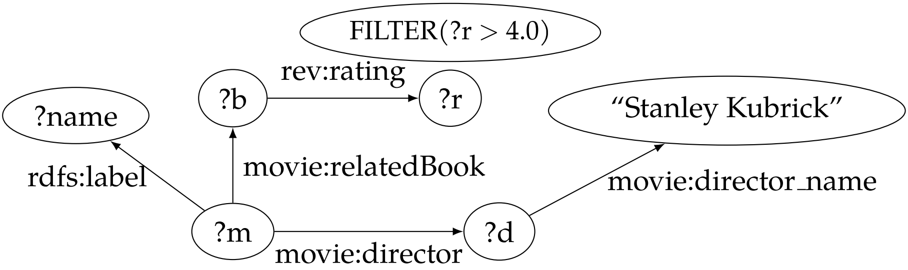
$$(U \cup V) \times (U \cup V) \times (U \cup V \cup L)$$

    - ?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:

```
SELECT ?name
WHERE {
  ?m rdfs:label ?name. ?m movie:director ?d.
  ?d movie:director_name "Stanley Kubrick".
  ?m movie:relatedBook ?b. ?b rev:rating ?r.
  FILTER(?r > 4.0)
}
```

SPARQL Queries

```
SELECT ?name
WHERE {
  ?m rdfs:label ?name. ?m movie:director ?d.
  ?d movie:director_name "Stanley Kubrick".
  ?m movie:relatedBook ?b. ?b rev:rating ?r.
  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

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 .
  ?goid rdfs:label ?golabel .
}
```

- 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 {
  ?value rdfs:label ?valueLabel .
  ?value atlasterms:pValue ?pvalue .
  ?value atlasterms:isMeasurementOf ?probe .
  ?probe atlasterms:dbXref <http://purl.uniprot.org/uniprot/Q16850> .
```

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)

?m	rdfs:label	?name
...	...	...
...	...	...

?m	movie:director	?d
...	...	...
...	...	...

?d	movie:director_name	"Stanley Kubrick"
...	...	"Stanley Kubrick"
...	...	"Stanley Kubrick"

?m	movie:actor	?c
...	...	...
...	...	...

?c	movie:actor	"Jack Nicholson"
...	...	"Jack Nicholson"

```
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)

?m	rdfs:label	?name
. . .	. . .	. . .
. . .	. . .	. . .

o-o join

?m	movie:director	?d
. . .	. . .	. . .
. . .	. . .	. . .

o-o join

?d	movie:director_name	“Stanley Kubrick”
. . .	. . .	“Stanley Kubrick”
. . .	. . .	“Stanley Kubrick”

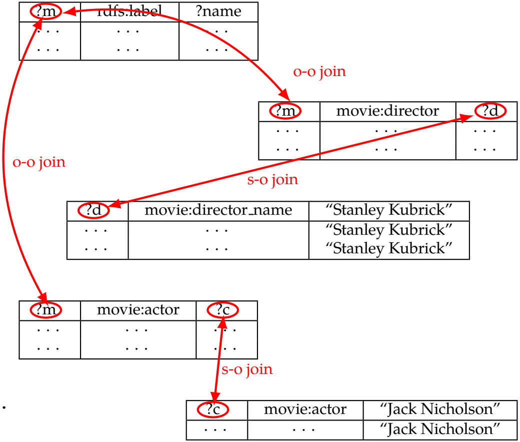
?m	movie:actor	?c
. . .	. . .	. . .
. . .	. . .	. . .

?c	movie:actor	“Jack Nicholson”
. . .	. . .	“Jack Nicholson”

```
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

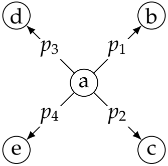
- Subgraph matching using homomorphism
- Operational semantics (for BGPs)
  - Find all triples that satisfy each *triple pattern*
  - Join them according to variables (s-s, s-o, o-o, p-p)



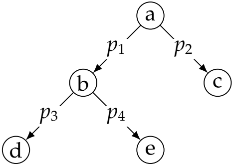

```
SELECT ?name
WHERE {
  ?m rdfs:label ?name.
  ?m movie:director ?d.
  ?d movie:director_name "Stanley Kubrick".
  ?m movie:actor ?c.
  ?c movie:actor_name "Jack Nicholson".
}
```

SPARQL Query Shapes

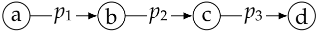
Star query



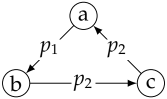
Tree query



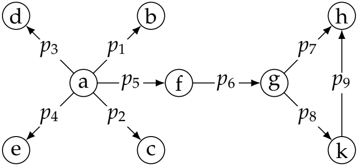
Chain (linear) query



Cycle query



Complex query



▸ Scale-out

Overview

RDF Introduction

Centralized Systems – Overview
- Relational Approaches
- Graph-based Approaches

Scale-out Systems

Federated Systems

Conclusions

Direct Relational Mapping

```
SELECT ?name
WHERE {
  ?m rdfs:label ?name. ?m movie:director ?d.
  ?d movie:director_name "Stanley Kubrick".
  ?m movie:relatedBook ?b. ?b rev:rating ?r.
  FILTER(?r > 4.0)
}
```

Subject	Property	Object
mdb:film/2014	rdfs:label	"The Shining"
mdb:film/2014	movie:initial_release_date	"1980-05-23"
mdb:film/2014	movie:director	mdb:director/8476
mdb:film/2014	movie:actor	mdb:actor/29704
mdb:film/2014	movie:actor	mdb: actor/30013
mdb:film/2014	movie:music_contributor	mdb: music_contributor/4110
mdb:film/2014	foaf:based_near	geo:2635167
mdb:film/2014	movie:relatedBook	bm:0743424425
mdb:film/2014	movie:language	lexvo:iso639-3/eng
mdb:director/8476	movie:director_name	"Stanley Kubrick"
mdb:film/2685	movie:director	mdb:director/8476
mdb:film/2685	rdfs:label	"A Clockwork Orange"
mdb:film/424	movie:director	mdb:director/8476
mdb:film/424	rdfs:label	"Spartacus"
mdb:actor/29704	movie:actor_name	"Jack Nicholson"
mdb:film/1267	movie:actor	mdb:actor/29704
mdb:film/1267	rdfs:label	"The Last Tycoon"
mdb:film/3418	movie:actor	mdb:actor/29704
mdb:film/3418	rdfs:label	"The Passenger"
geo:2635167	gn:name	"United Kingdom"
geo:2635167	gn:population	62348447
geo:2635167	gn:wikipediaArticle	wp:United_Kingdom
bm:books/0743424425	dc:creator	bm:persons/Stephen+King
bm:books/0743424425	rev:rating	4.7
bm:books/0743424425	scom:hasOffer	bm:offers/0743424425amazonOffer
lexvo:iso639-3/eng	rdfs:label	"English"
lexvo:iso639-3/eng	lvont:usedIn	lexvo:iso3166/CA
lexvo:iso639-3/eng	lvont:usesScript	lexvo:script/Latn

Direct Relational Mapping

```
SELECT ?name
WHERE {
  ?m rdfs:label ?name. ?m movie:director ?d.
  ?d movie:director_name "Stanley Kubrick".
  ?m movie:relatedBook ?b. ?b rev:rating ?r.
  FILTER(?r > 4.0)
}
```

Subject	Property	Object
mdb:film/2014	rdfs:label	"The Shining"
mdb:film/2014	movie:initial_release_date	"1980-05-23"
mdb:film/2014	movie:director	mdb:director/8476
mdb:film/2014	movie:actor	mdb:actor/29704
mdb:film/2014	movie:actor	mdb: actor/30013
mdb:film/2014	movie:music_contributor	mdb: music_contributor/4110
mdb:film/2014	foaf:based_near	geo:2635167
mdb:film/2014	movie:relatedBook	bm:0743424425
mdb:film/2014	movie:language	lexvo:iso639-3/eng
mdb:director/8476	movie:director_name	"Stanley Kubrick"
mdb:film/2685	movie:director	mdb:director/8476
mdb:film/2685	rdfs:label	"A Clockwork Orange"
mdb:film/424	movie:director	mdb:director/8476
mdb:film/424	rdfs:label	"Spartacus"
mdb:actor/29704	movie:actor_name	"Jack Nicholson"
mdb:film/1267	movie:actor	mdb:actor/29704
mdb:film/1267	rdfs:label	"The Last Tycoon"
mdb:film/3418	movie:actor	mdb:actor/29704
mdb:film/3418	rdfs:label	"The Passenger"
geo:2635167	gn:name	"United Kingdom"
geo:2635167	gn:population	62348447
geo:2635167	gn:wikipediaArticle	wp:United_Kingdom
bm:books/0743424425	dc:creator	bm:persons/Stephen+King
bm:books/0743424425	rev:rating	4.7
bm:books/0743424425	scom:hasOffer	bm:offers/0743424425amazonOffer
lexvo:iso639-3/eng	rdfs:label	"English"
lexvo:iso639-3/eng	lvont:usedIn	lexvo:iso3166/CA
lexvo:iso639-3/eng	lvont:usesScript	lexvo:script/Latn

```
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

```
SELECT ?name
WHERE {
  ?m rdfs:label ?name. ?m movie:directo
  ?d movie:director_name "Stanley Kubr
  ?m movie:relatedBook ?b. ?b rev:r
  FILTER(?r > 4.0)
}
```

Subject	Property	Object
mdb:film/2014	rdfs:label	"The Shining"
mdb:film/2014	movie:initial_release_date	"1980-05-23"
mdb:film/2014	movie:director	mdb:director/8476
mdb:film/2014	movie:actor	mdb:actor/29704
mdb:film/2014	movie:actor	mdb: actor/30013
mdb:film/2014	movie:music_contributor	mdb: music_contributor/4110
mdb:film/2014	foaf:based_near	geo:2635167
mdb:film/2014	movie:relatedBook	bm:0743424425
mdb:film/2014	movie:language	lexvo:iso639-3/eng
mdb:director/8476	movie:director_name	"Stanley Kubrick"
mdb:film/2685	movie:director	mdb:director/8476
mdb:film/2685	rdfs:label	"A Clockwork Orange"
mdb:film/424	movie:director	mdb:director/8476
mdb:film/424	rdfs:label	"Spartacus"
mdb:actor/29704	movie:actor_name	"Jack Nicholson"
mdb:film/1267	movie:actor	mdb:actor/29704
mdb:film/1267	rdfs:label	"The Last Tycoon"
mdb:film/3418	movie:actor	mdb:actor/29704
mdb:film/3418	rdfs:label	"The Passenger"
geo:2635167	gn:name	"United Kingdom"
geo:2635167	gn:population	62348447
geo:2635167	gn:wikipediaArticle	wp:United_Kingdom
bm:books/0743424425	dc:creator	bm:persons/Stephen+King
bm:books/0743424425	rev:rating	4.7
bm:books/0743424425	scom:hasOffer	bm:offers/0743424425amazonOffer
lexvo:iso639-3/eng	rdfs:label	"English"
lexvo:iso639-3/eng	lvont:usedIn	lexvo:iso3166/CA
lexvo:iso639-3/eng	lvont:usesScript	lexvo:script/Latn


```
T3,
WHERE T1.p="rdfs:l
AND T2.p="movie:rela    ook"
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

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

Subject	Property	Object
mdb:film/2014	rdfs:label	"The Shining"
mdb:film/2014	movie:director	mdb:director/8476
mdb:film/2685	movie:director	mdb:director/8476
mdb:film/2685	rdfs:label	"A Clockwork Orange"
mdb:actor/29704	movie:actor_name	"Jack Nicholson"
...	...	...

Subject	refs:label	movie:director
mob:film/2014	"The Shining"	mob:director/8476
mob:film/2685	"The Clockwork Orange"	mob:director/8476

Subject	movie:actor_name
mdb:actor	"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]
   - 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

Subject	**Property**	**Object**
mdb:film/2014	rdfs:label	"The Shining"
mdb:film/2014	movie:director	mdb:director/8476
mdb:film/2685	movie:director	mdb:director/8476
mdb:film/2685	rdfs:label	"A Clockwork Orange"
mdb:actor/29704	movie:actor_name	"Jack Nicholson"
. . .	. . .	. . .

refs:label

Subject	**Object**
mob:film/2014	"The Shining"
mob:film/2685	"The Clockwork Orange"

movie:director

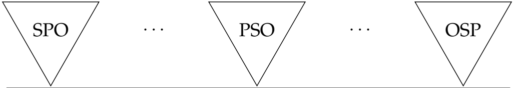
Subject	**Object**
mdb:film/2014	mdb:director/8476
mdb:film/2685	mdb:director/8476

movie:actor_name

Subject	**Object**
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

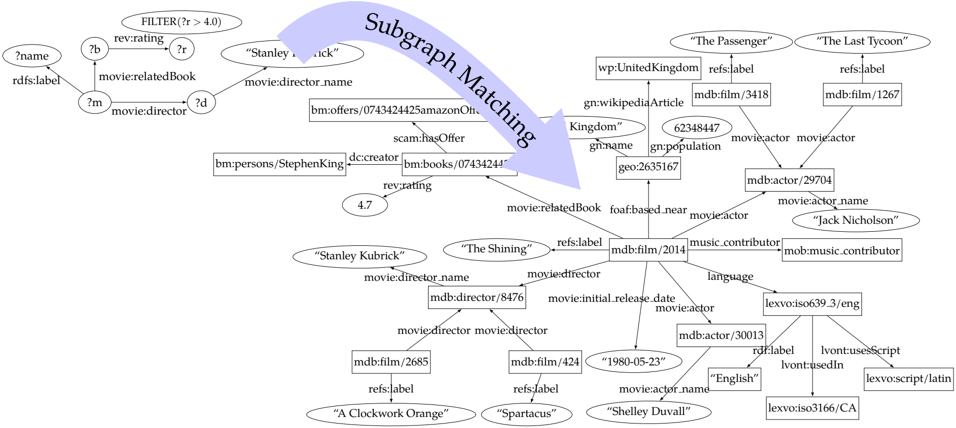


Subject	Property	Object
mdb:film/2014	rdfs:label	"The Shining"
mdb:film/2014	movie:director	mdb:director/8476
mdb:film/2685	movie:director	mdb:director/8476
mdb:film/2685	rdfs:label	"A Clockwork Orange"
mdb:actor/29704	movie:actor_name	"Jack Nicholson"
. . .	. . .	. . .

Graph-based Approach

[Zou & Özsu, 2017]

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



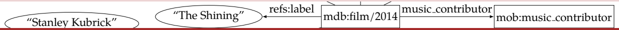
Graph-based Approach

[Zou & Özsu, 2017]

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

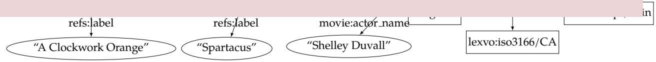
Advantages

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



Disadvantages

- Graph pattern matching is expensive



Overview

RDF Introduction

Centralized Systems – Overview

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

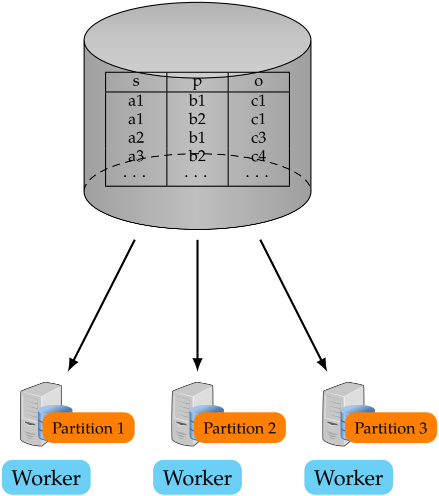
Federated Systems

Conclusions

What is the Objective

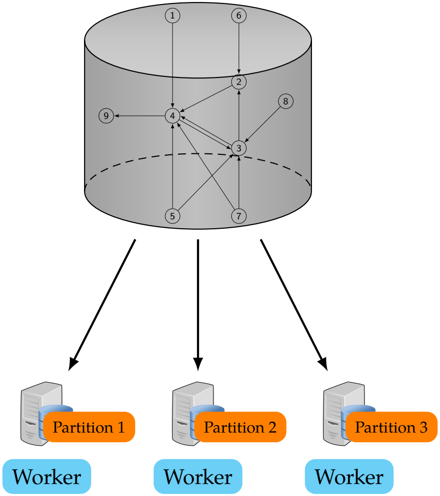
Parallelize query processing with as little inter-site communication as possible.

- 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



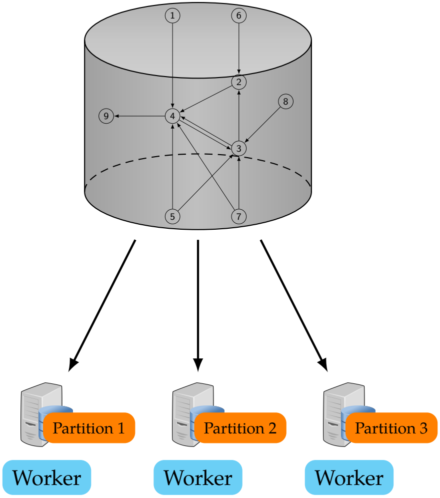
Graph Partitioning

- More difficult because of edges → represent relationships



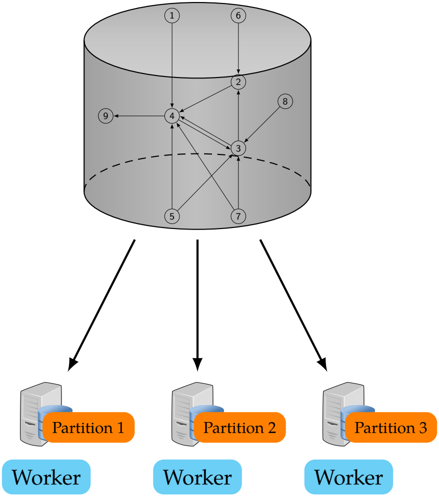
Graph Partitioning

- 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

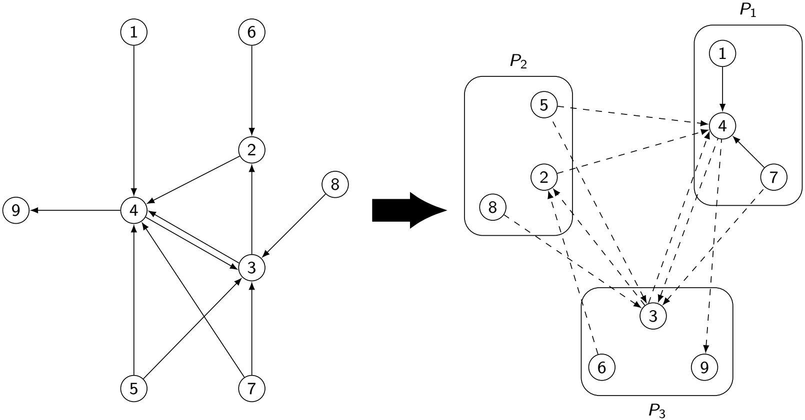
METIS Vertex-Disjoint Partitioning

[Karypis & Kumar, 1998]

Given an initial graph $G_0 = (V, E)$:

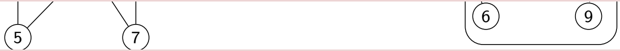
1. Produce a hierarchy of successively coarsened graphs $G_1, \ldots, G_n$ such that $|V(G_i)| > |V(G_j)|$ for $i < j$
2. Partition G_n using some partitioning algorithm
3. Iteratively coarsen G_n to G_0, and at each step
   a) Project the partitioning solution on graph G_j to graph G_{j-1}
   b) Improve the partitioning of G_0

Vertex-Disjoint Partitioning Example



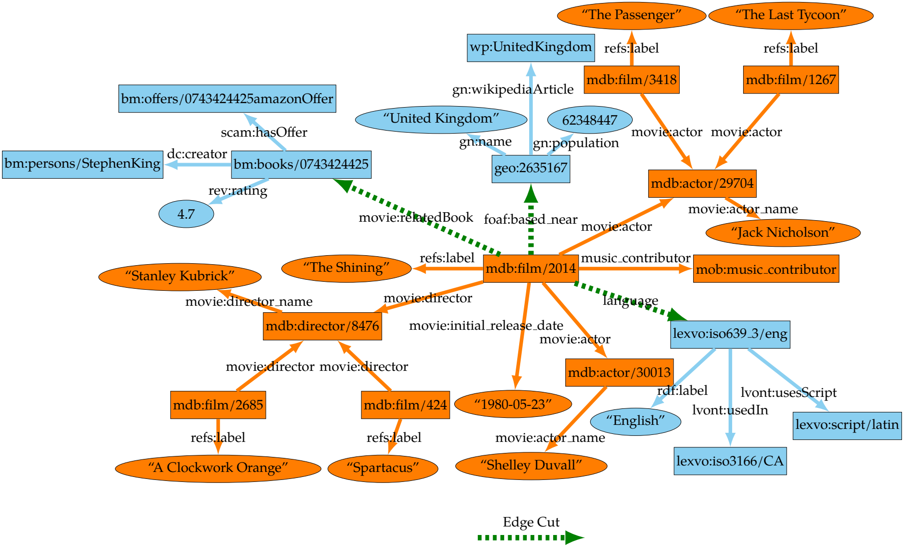
Vertex-Disjoint Partitioning Example

- + 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

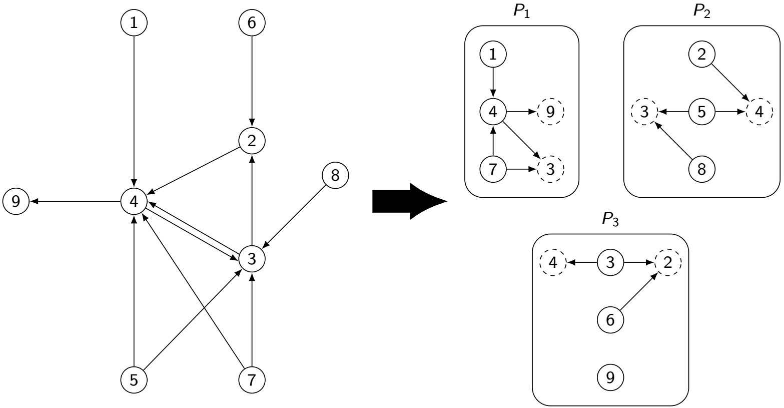
METIS on our Example



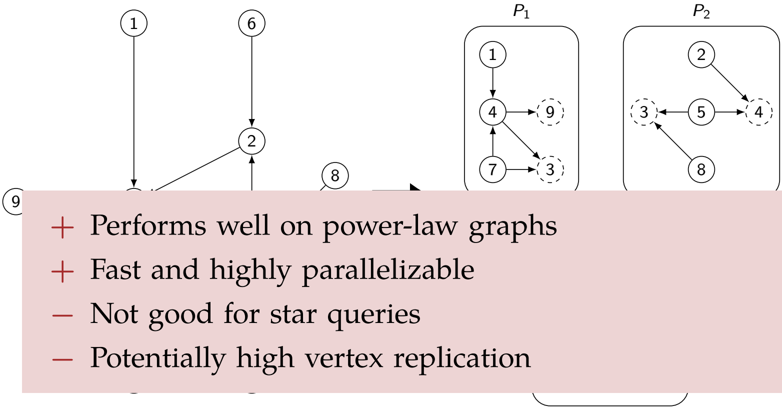
Edge-Disjoint Partitioning

- Put each edge in one partition
- Vertices may need to be replicated – minimize these
- Alternatives
  - Hashing $\rightarrow$ on the ids of the two vertices incident on edge
  - Heuristics cognizant of graph characteristics $\rightarrow$ 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

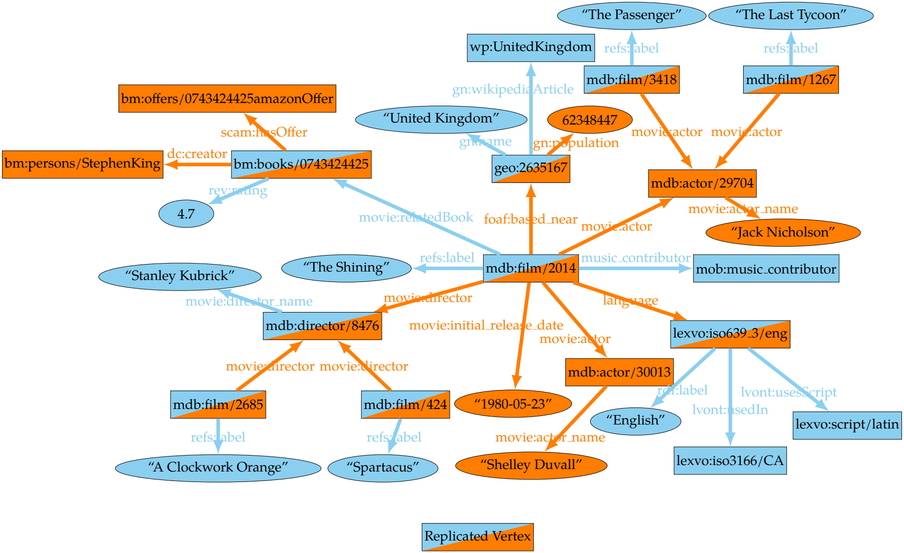


Edge-Disjoint Partitioning Example



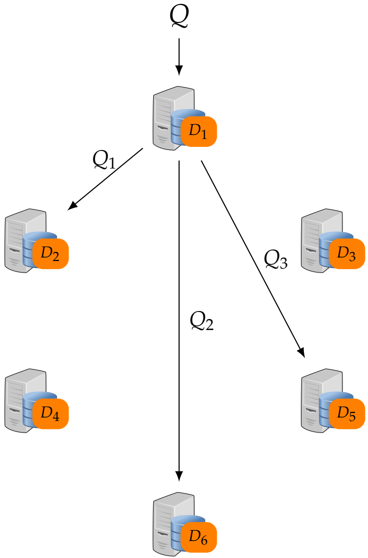
- + Performs well on power-law graphs
- + Fast and highly parallelizable
- – Not good for star queries
- – Potentially high vertex replication

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]

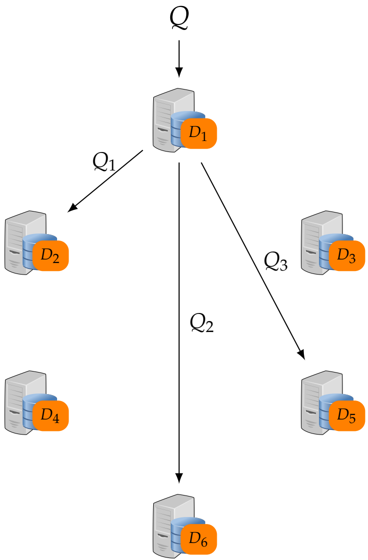


Query Partitioning Approaches

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

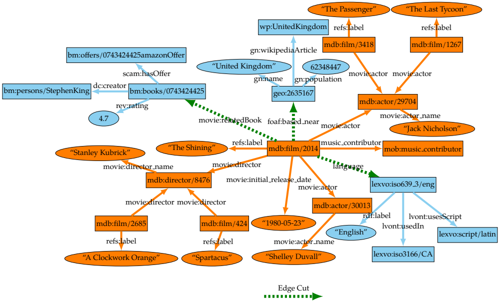
Main Idea

Partition the data and the query in such a way that the interpartition joins are minimized



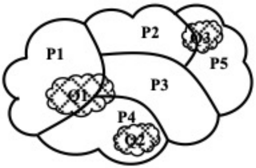
Two Examples

- 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

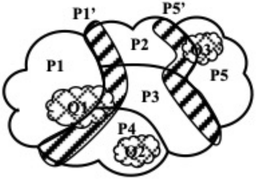


Two Examples

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



Baseline Partitions



Expanded Partitions

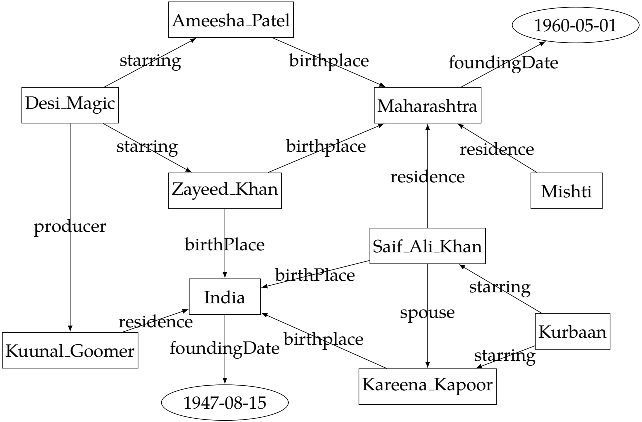
MPC – Minimum Predicate Cut

[Peng et al., 2021]

What is the main point?

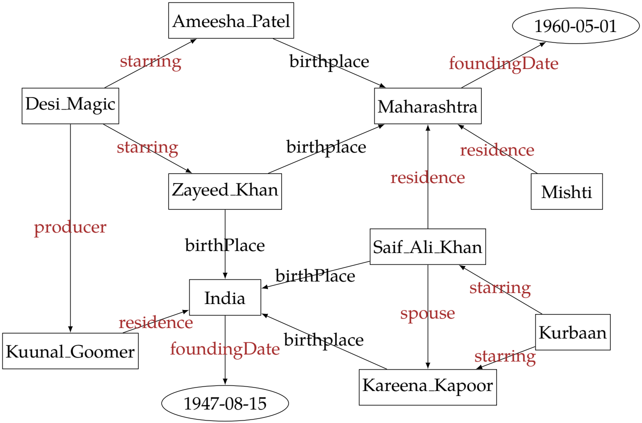
- Vertex-disjoint partitioning objective is to minimize edge-cuts
- The main performance bottleneck in distributed SPARQL execution is inter-partition joins
  - If all predicates in a query are *internal* $\rightarrow$ execute independently over each partition without inter-partition join
- Motivation: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

How to Minimize Predicate-cuts



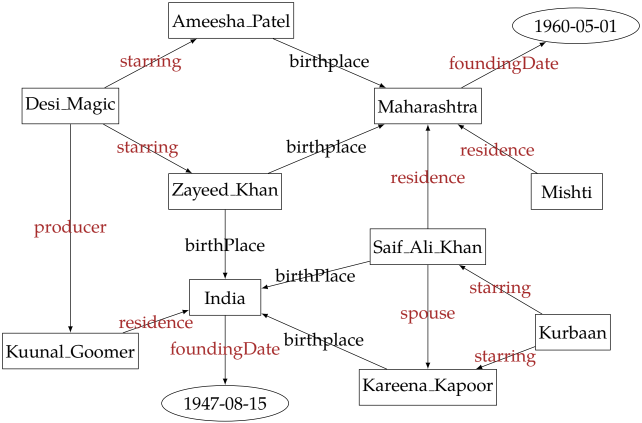
How to Minimize Predicate-cuts

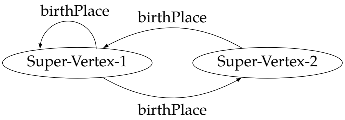
Selecting Internal Pred.



How to Minimize Predicate-cuts

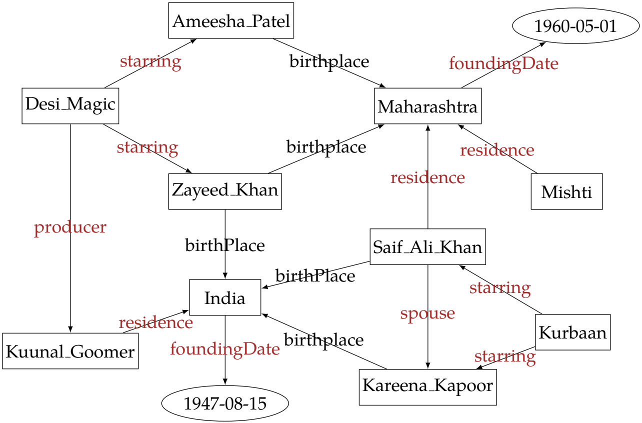
Selecting Internal Pred. → Coarsening

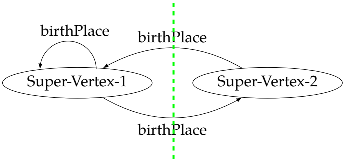




How to Minimize Predicate-cuts

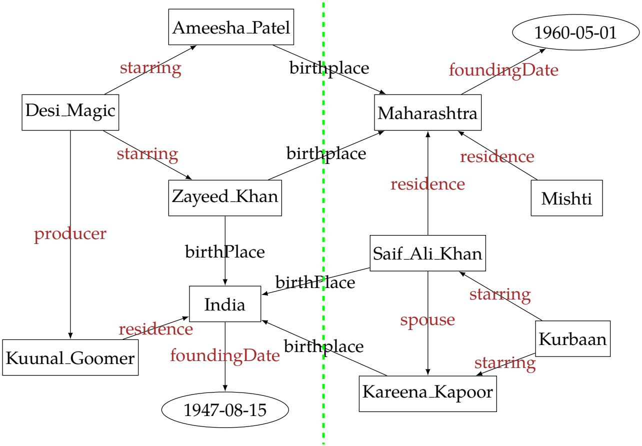
Selecting Internal Pred. → Coarsening → Partitioning Coarse Graph

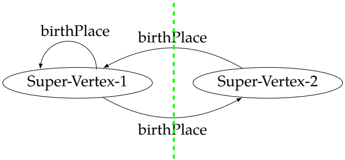




How to Minimize Predicate-cuts

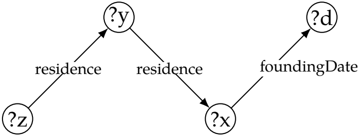
Selecting Internal Pred. → Coarsening → Partitioning Coarse Graph → Uncoarsening

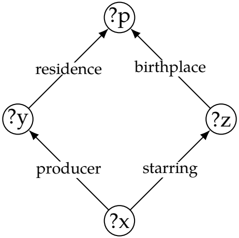




MPC-based Query Processing

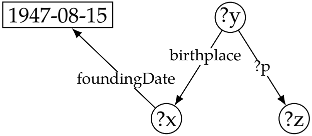
- Objective is to increase the *independently executable queries* (IEQ)
  - Star queries
  - Queries not containing any crossing predicates
  - Queries connected when crossing predicate edges are removed

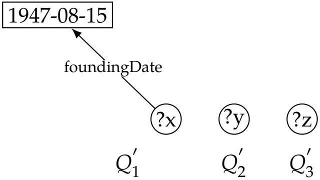


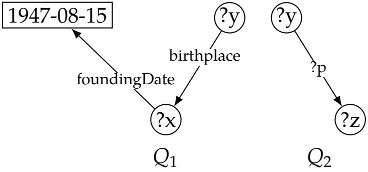


MPC-based Query Processing

- 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



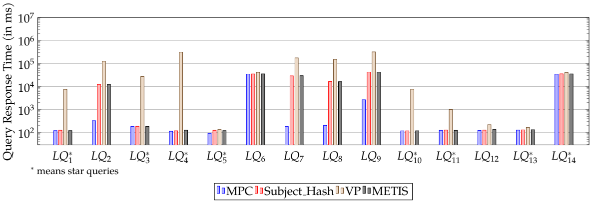




Performance Advantage

	MPC	VC	METIS
LUBM	100	0	43
WatDiv	60	0	50
MusicBrains	86	14	57
YAGO2	100	0	0
DBpedia	75	24	47
LinkedGeoData	100	84	97

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

Query Decomposition → Data Localization → Distributed Query Plan → Local Execution → Distributed Execution

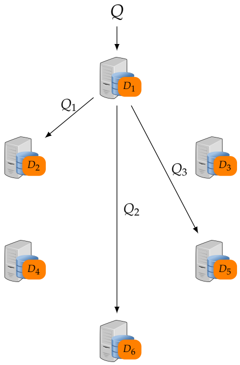
Query Decomposition
Input: Declarative query
Output: Algebraic query
Transforms query to algebraic form over D

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

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

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

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

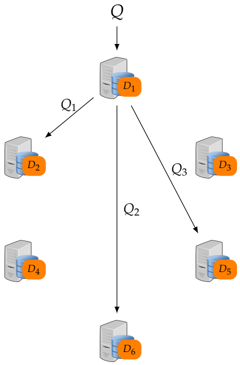


Partitioned Query Evaluation Methodology

Adapting Relational Approach

Query Decomposition → Data Localization → Distributed Query Plan → Local Execution → Distributed Execution

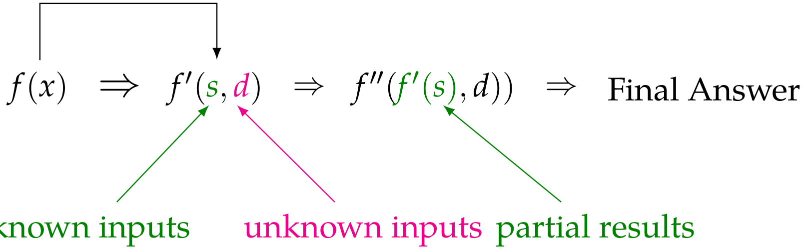
Query Decomposition	Data Localization	Distributed Query Plan	Local Execution	Distributed Execution
Input: Declarative query Output: Algebraic query Transforms query to algebraic form over D	Input: Algebraic query Output: Partitioned query Transforms query over D to query over $D = \{D_1, \dots, D_n\}$ determining $\{Q_1, \dots, Q_k\}$	Input: $\{Q_1, \dots, Q_k\}$ Output: Distributed plan Decide how the results of $\{Q_1, \dots, Q_k\}$ are combined	Input: $\{Q_1, \dots, Q_k\}$ Output: Local results Each Q_i executed at one site over one D_j	Input: Local results Output: Final result Combine results of $\{Q_1, \dots, Q_k\}$ 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]



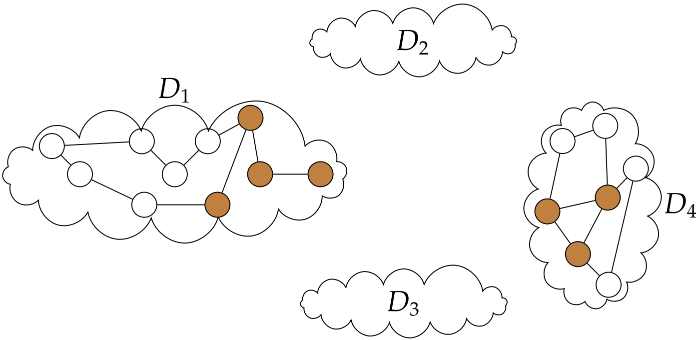
- Query is the function and each D_i+ its extended vertices form the known input

Distributed SPARQL Using PQE

[Peng et al., 2016]

Two steps:

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

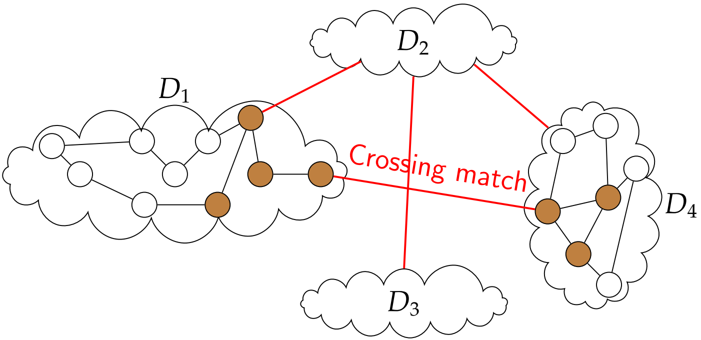


Distributed SPARQL Using PQE

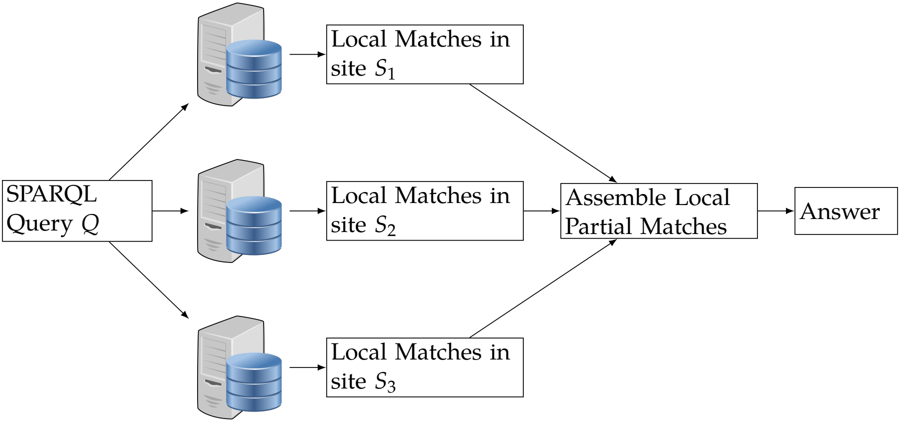
[Peng et al., 2016]

Two steps:

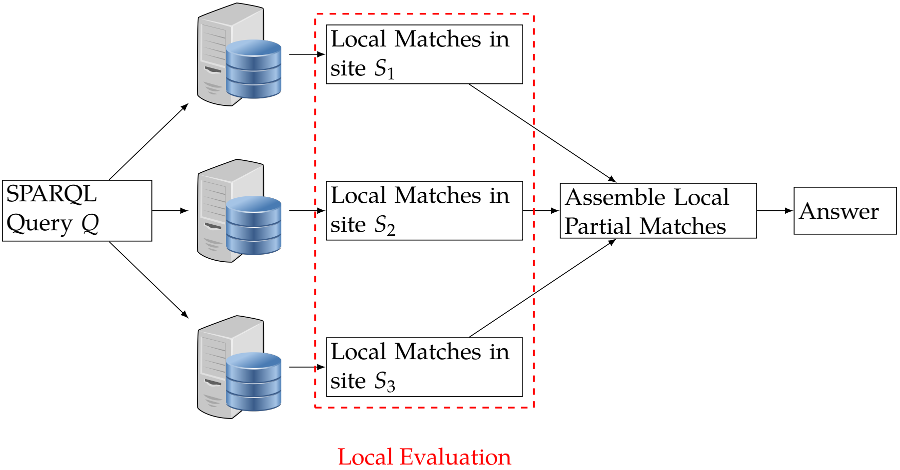
1. Evaluate a query at each site to find local matches
   - These are local partial matches
2. Assemble the partial matches to get final result
   - Crossing match



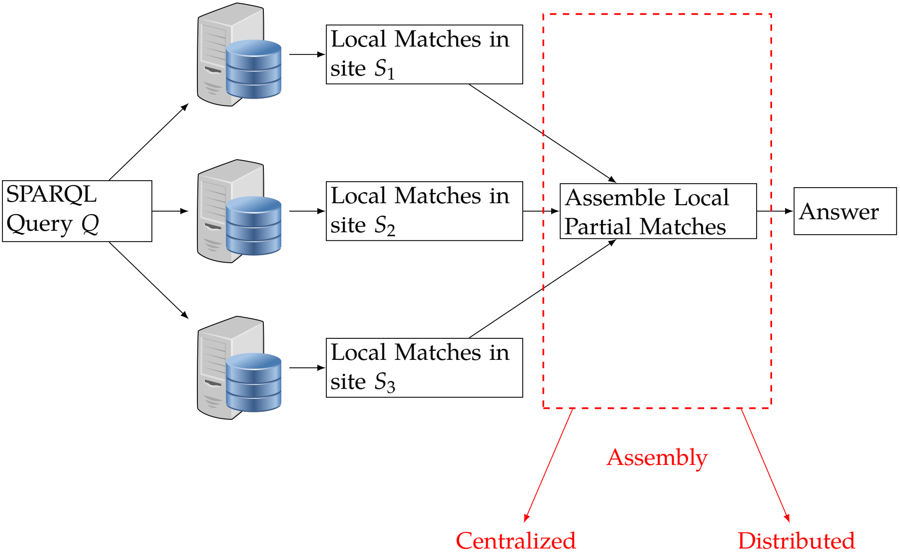
PQE Framework



PQE Framework



PQE Framework



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

[Kaoudi & Manolescu, 2015]

- 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

[Kaoudi & Manolescu, 2015]

- 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

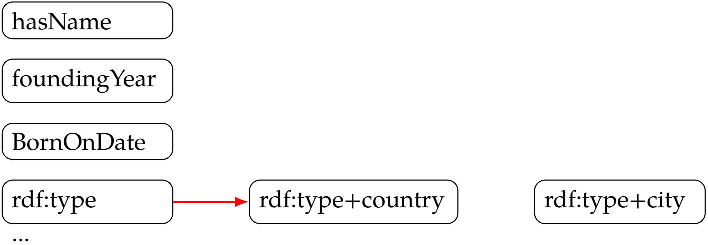
Main Idea

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

An Example – HadoopRDF

[Husain et al., 2011]

- 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

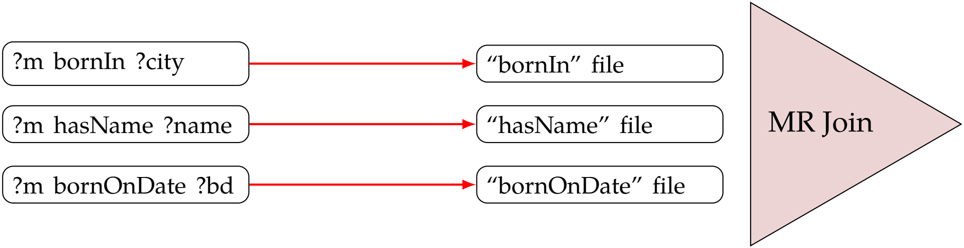


An Example – HadoopRDF

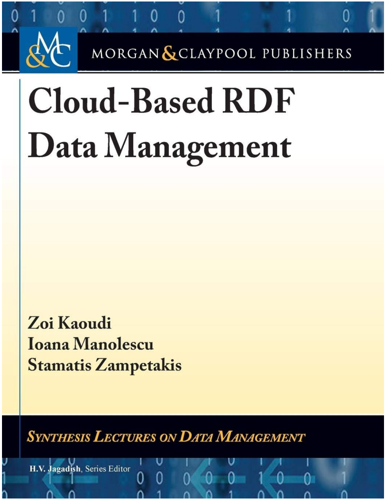
[Husain et al., 2011]

- File organization (offline)

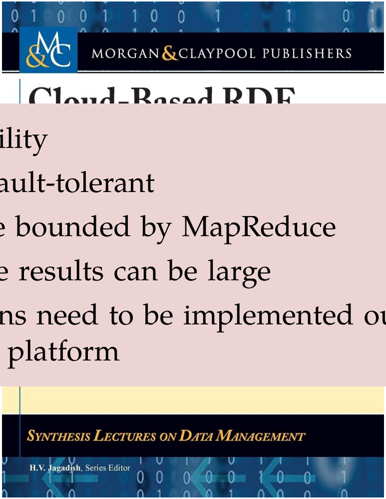
- Query Evaluation
  - Input file selection: For each triple pattern, select the corresponding files
  - MapReduce join: Joining between two or more triple patterns on a variable
  - Order selection: Cost-based join order selection



Cloud-based Approaches – General Comments



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

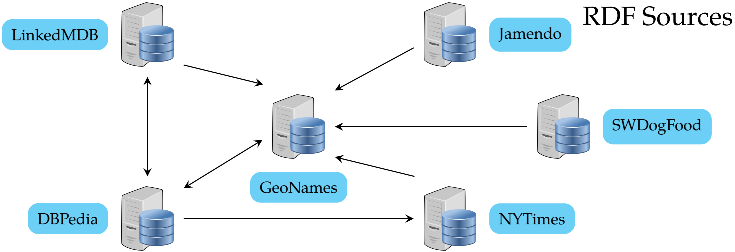
Scale-out Systems

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 [Görlitz & Staab, 2011], ANAPSID [Acosta et al., 2011]

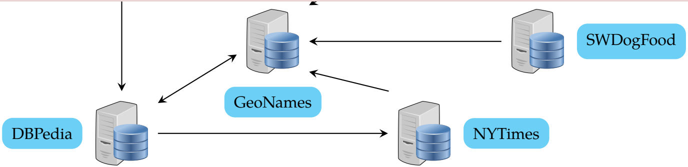


SPARQL Endpoint Federation

- 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 [Acosta et al., 2011]

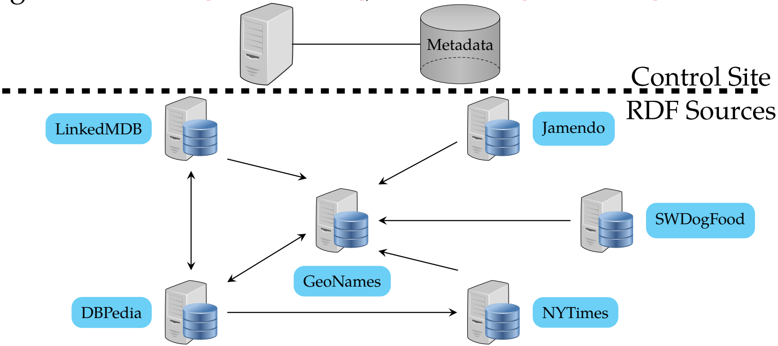
Main Idea

Data integration approach → Datasets exist at SPARQL endpoints and queries are pushed to them



SPARQL Endpoint Federation

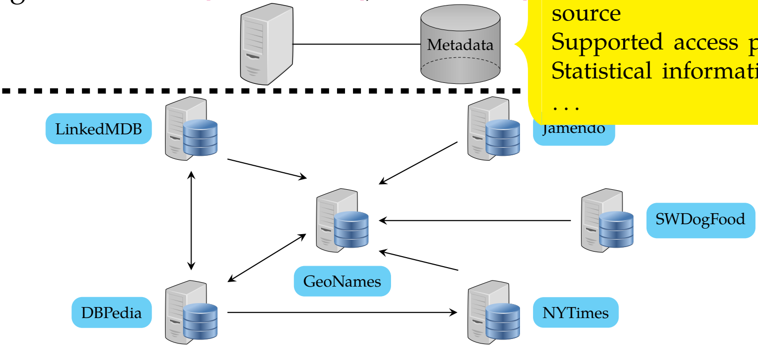
- No data re-partitioning/re-distribution
- Consider $D = D_1 \cup D_2 \cup \ldots \cup D_n$; D_i: SPARQL endpoint
- SPARQL query decomposed $Q = \{Q_1, \ldots, Q_k\}$
- Distributed execution of $\{Q_1, \ldots, Q_k\}$ over $\{D_1, \ldots, D_n\}$
- E.g.: SPLENDID [Görlitz & Staab, 2011], ANAPSID [Acosta et al., 2011]



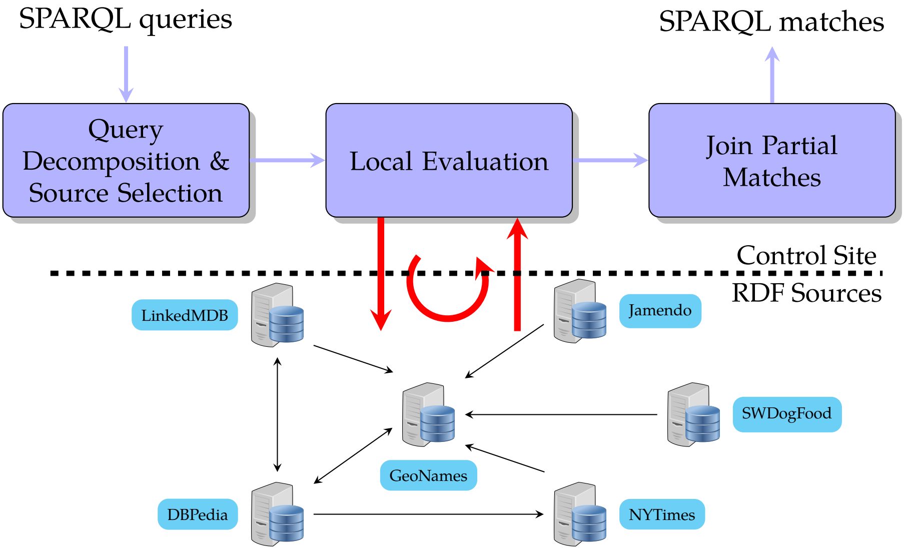
SPARQL Endpoint Federation

- 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



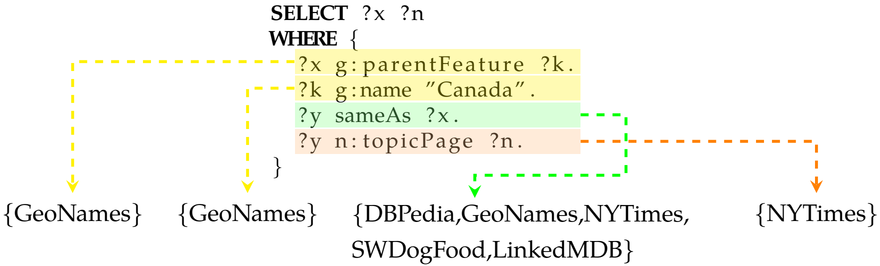
Query Decomposition

- 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.
   ?y n:topicPage ?n.
}
```

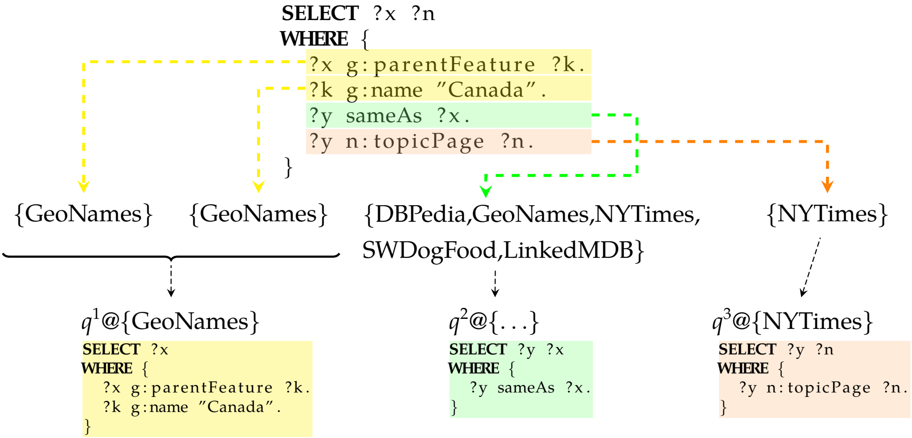
Query Decomposition

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



Query Decomposition

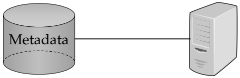
- Each triple pattern maps to a set of RDF sources based on the values of its subject, property, and object.



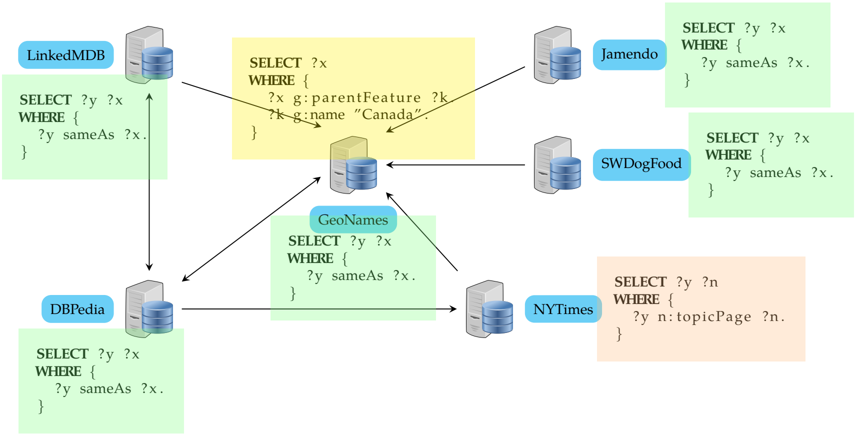
Data Localization

- 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]
  - HiBISCus [Saleem & Ngomo, 2014]
  - ...
- ASK query-based approach
  - Asking whether or not a triple pattern has an answer at a source
  - FedX [Schwarte et al., 2011a,b]

Query Execution

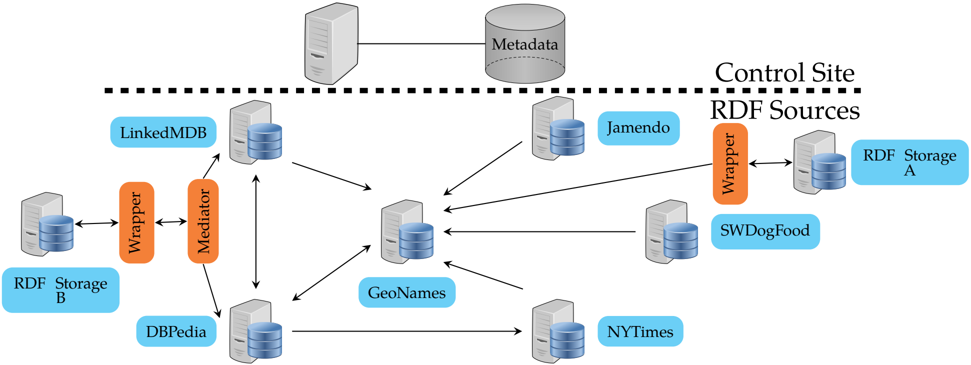



```
SELECT ?x ?n
WHERE {
  ?x g:parentFeature ?k.
  ?k g:name "Canada".
  ?y sameAs ?x.
  ?y n:topicPage ?n.
}
```



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

BioModels	Curated computational models of biological processes
BioSamples	Sample information for reference samples and samples for which data exist in one of the EBI's assay databases
ChEMBL	Curated chemical database of bioactive molecules with drug-like properties
e!Ensembl	Genome databases for vertebrates and other eukaryotic species
Expression Atlas	Gene expression data from the Gene Expression Atlas
Reactome	Curated and peer-reviewed pathways
UniProt	

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

```
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 atlasterms: <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 rdf:type biopax3:Pathway .
 ?pathway biopax3:displayName ?pathwayname .
 ?pathway biopax3:pathwayComponent
 [?rel [biopax3:entityReference ?dbXref]] .
 ?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> {
  ?value rdfs:label ?expressionValue .
  ?value atlasterms:pValue ?pvalue .
  ?value atlasterms:isMeasurementOf ?probe .
  ?probe atlasterms:dbXref ?dbXref .
 }

 # get the GO functions from Uniprot
 SERVICE <http://uniprot.org/sparql> {
  ?dbXref a upc:Protein ;
  upc:classifiedWith ?keyword .
  ?keyword rdfs:seeAlso ?goid .
  ?goid rdfs:label ?golabel .
 }
```

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 I

Abadi, D. J., Marcus, A., Madden, S., & Hollenbach, K. (2009). SW-Store: a vertically partitioned DBMS for semantic web data management. *VLDB J.*, *18*(2), 385–406. Retrieved from `https://doi.org/10.1007/s00778-008-0125-y`

Abadi, D. J., Marcus, A., Madden, S. R., & Hollenbach, K. (2007). Scalable semantic web data management using vertical partitioning. In *Proc. 33rd int. conf. on very large data bases* (pp. 411–422).

Acosta, M., Vidal, M.-E., Lampo, T., Castillo, J., & Ruckhaus, E. (2011). ANAPSID: an adaptive query processing engine for SPARQL endpoints. In *Proc. 10th int. semantic web conf.* (pp. 18–34).

Aluç, G., Özsu, M. T., Daudjee, K., & Hartig, O. (2013). *chameleon-db: a workload-aware robust RDF data management system* (Tech. Rep. No. CS-2013-10). University of Waterloo. (Available at `https://cs.uwaterloo.ca/sites/ca.computer-science/files/uploads/files/CS-2013-10.pdf`)

Bornea, M. A., Dolby, J., Kementsietsidis, A., Srinivas, K., Dantressangle, P., Udrea, O., et al. (2013). Building an efficient RDF store over a relational database. In *Proc. acm sigmod int. conf. on management of data* (p. 121-132). Retrieved from `http://doi.acm.org/10.1145/2463676.2463718`

Galarraga, L., Hose, K., & Schenkel, R. (2014). Partout: a distributed engine for efficient RDF processing. In *Proc. 26th int. world wide web conf. (companion volume)* (pp. 267–268). Retrieved from `http://doi.acm.org/10.1145/2567948.2577302`

Görlitz, O., & Staab, S. (2011). SPLENDID: SPARQL endpoint federation exploiting VOID descriptions. In *Proc. 2nd int. workshop on consuming linked data.*

Harth, A., Hose, K., Karnstedt, M., Polleres, A., Sattler, K., & Umbrich, J. (2010). Data summaries for on-demand queries over linked data. In *Proc. 19th int. world wide web conf.* (pp. 411–420). Retrieved from `http://doi.acm.org/10.1145/1772690.1772733`

Harth, A., Umbrich, J., Hogan, A., & Decker, S. (2007). YARS2: A federated repository for querying graph structured data from the web. In *Proc. 6th int. semantic web conf.* (pp. 211–224).

Hose, K. (2021). The quest for knowledge. In *Proc. 24th int. conf. on extending database technology.* (Keynote presentation)

References II

Hose, K., & Schenkel, R. (2013). WARP: Workload-aware replication and partitioning for RDF. In *Proc. workshops of 29th int. conf. on data engineering* (pp. 1–6). Retrieved from http://dx.doi.org/10.1109/ICDEW.2013.6547414

Huang, J., Abadi, D. J., & Ren, K. (2011). Scalable SPARQL querying of large RDF graphs. *Proc. VLDB Endowment*, *4*(11), 1123–1134.

Husain, M. F., McGlothlin, J., Masud, M. M., Khan, L. R., & Thuraisingham, B. (2011). Heuristics-based query processing for large RDF graphs using cloud computing. *IEEE Trans. Knowl. and Data Eng.*, *23*(9), 1312–1327.

Kaoudi, Z., & Manolescu, I. (2015). RDF in the clouds: A survey. *VLDB J.*, *24*, 67–91.

Karypis, G., & Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. on Scientific Comput.*, *20*(1), 359–392.

Khadilkar, V., Kantarcioglu, M., Thuraisingham, B. M., & Castagna, P. (2012). Jena-HBase: A distributed, scalable and efficient RDF triple store. In *Proc. international semantic web conference posters & demos track.*

Lee, K., & Liu, L. (2013). Scaling queries over big RDF graphs with semantic hash partitioning. *Proc. VLDB Endowment*, *6*(14), 1894–1905. Retrieved from http://www.vldb.org/pvldb/vol6/p1894-lee.pdf

Li, F., Ooi, B. C., Özsu, M. T., & Wu, S. (2014). Distributed data management using MapReduce. *ACM Comput. Surv.*, *46*(3), Article No. 31.

Neumann, T., & Weikum, G. (2008). RDF-3X: a RISC-style engine for RDF. *Proc. VLDB Endowment*, *1*(1), 647–659.

Neumann, T., & Weikum, G. (2009, September). The RDF-3X engine for scalable management of RDF data. *VLDB J.*, *19*(1), 91–113.

Peng, P., Özsu, M. T., Zou, L., & Yan, C. (2021). *MPC: minimum property-cut RDF graph partitioning.* Submitted for publication.

Peng, P., Zou, L., Özsu, M. T., Chen, L., & Zhao, D. (2016). Processing SPARQL queries over distributed RDF graphs. *VLDB J.*, *25*(2), 243–268.

References III

Peng, P., Zou, L., Özsu, M. T., & Zhao, D. (2018). Multi-query optimization in federated RDF systems. In *Proc. 23rd int. conf. on database systems for advanced applications* (pp. 745–765).

Prasser, F., Kemper, A., & Kuhn, K. A. (2012). Efficient distributed query processing for autonomous rdf databases. In *Proc. 15th int. conf. on extending database technology* (pp. 372–383).

Quilitz, B., & Leser, U. (2008). Querying distributed RDF data sources with SPARQL. In *Proc. 5th european semantic web conf.* (pp. 524–538).

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)

Saleem, M., & Ngomo, A. N. (2014). HiBISCuS: Hypergraph-based source selection for SPARQL endpoint federation. In *Proc. 11th extended semantic web conf.* (pp. 176–191). Retrieved from http://dx.doi.org/10.1007/978-3-319-07443-6_13

Schwarte, A., Haase, P., Hose, K., Schenkel, R., & Schmidt, M. (2011a). FedX: A federation layer for distributed query processing on linked open data. In *Proc. 8th extended semantic web conf.* (pp. 481–486).

Schwarte, A., Haase, P., Hose, K., Schenkel, R., & Schmidt, M. (2011b). Fedx: Optimization techniques for federated query processing on linked data. In *Proc. 10th int. semantic web conf.* (pp. 601–616). Retrieved from https://doi.org/10.1007/978-3-642-25073-6_38

Weiss, C., Karras, P., & Bernstein, A. (2008). Hexastore: sextuple indexing for semantic web data management. *Proc. VLDB Endowment*, *1*(1), 1008–1019.

Wilkinson, K. (2006, October). *Jena property table implementation* (Tech. Rep. No. HPL-2006-140). HP Laboratories Palo Alto.

Zhang, X., Chen, L., Tong, Y., & Wang, M. (2013). EAGRE: towards scalable I/O efficient SPARQL query evaluation on the cloud. In *Proc. 29th int. conf. on data engineering* (pp. 565–576).

References IV

Zou, L., Mo, J., Chen, L., Özsu, M. T., & Zhao, D. (2011). gStore: answering SPARQL queries via subgraph matching. *Proc. VLDB Endowment*, *4*(8), 482–493.

Zou, L., & Özsu, M. T. (2017, 12). Graph-based RDF data management. *Data Science and Engineering*, *2*(1), 56–70. Retrieved from `https://dx.doi.org/10.1007/s41019-016-0029-6`

Zou, L., Özsu, M. T., Chen, L., Shen, X., Huang, R., & Zhao, D. (2014). gStore: A graph-based SPARQL query engine. *VLDB J.*, *23*(4), 565–590.