The architectures of triple-stores

Iztok Savnik
University of Primorska, Slovenia

Tutorial, DBKDA 2018, Nice.
Outline

• Introduction
• Triple data model
• Storage level representations
• Data distribution
• Query processing
Introduction
Terminology

• Triple-store
• RDF database
• Graph database
• Linked data
• Linked open data
• Knowledge bases
• Knowledge graphs
Position of triple stores

• Key-value model
• Relational data model
• Triple data model
Key-value data model

- Simple data and query model
  - BASE (Basically Available, Soft-state, Eventual consistency), CAP theorem
  - CRUD (Create, Read, Update, Delete)

- Automatic data distribution
  - Consistent hashing, Sharding

- Eventual consistency
  - Time-stamps and vector clocks
  - Distributed protocols: Gossip, Quorum 2PC
Relational data model

• Mathematical model of relations
  – Intuitive tabular representation
• Query model
  – Relational algebra and calculus, SQL
• Scalability
  – Round-Robin, hash, range partitioning, sharding
• Consistency
  – TPC, distributed 2PC
• Availability, tolerance for network partitions
Triple data model

- Graph data model
  - Baseline: graph representation
  - RDFS: knowledge representation language
    - Predicate calculus, description logic

- Query model
  1. Relational model + SQL
  2. Key-value access + MapReduce system
  3. Algebra of triples + SPARQL
Triple data model

• Data model
  – Baseline triple model
    • More complex than KV data model
    • More simple and uniform than relational model
  – Triple model + RDFS
    • more expressive than relational model

• Scalability
  – Automatic partitioning is possible
    • Hash partitioning, graph partitioning, sharding
    • Some ideas from KV model and some from relational model
Triple data model

• Consistency, availability, tolerance to network partitions, ...
  – Most of the above properties are hard to achieve in relational model
    • Consistency clashes with updates and high replication
    • Availability clashes with the weak tolerance to faults
    • Tolerance to network partitions would need and upgrade of RDBMS
  – Many ideas from KV model are applicable to TDM
    • Hash partitioning, eventual consistency, new storage systems, ...
Triple data model
Graph data model

- **Graph database**
  - Database that uses graphs for the representation of data and queries
- **Vertexes**
  - Represent things, persons, concepts, classes, ...
- **Arcs**
  - Represent properties, relationships, associations, ...
  - Arcs have **labels**!
RDF

• **Resource Description Framework**
  – Tim Berners Lee, 1998, 2009 ...
  – This is movement!

• **What is behind?**
  – Graphs are fundamental representation?
    • Can express any other data model
    • Many times serve as the theoretical basis
  – Graphs can represent data and knowledge?
    • Data and knowledge will be integrated in novel applications
    • Many reasoners use triple-representation of knowledge and data, e.g., Cyc
RDF

– Novel applications require some form of reasoning
  • Intelligent assistants, system diagnostics, ...
<Bob> <is a> <person>.
<Bob> <is a friend of> <Alice>.
<Bob> <is born on> <the 4th of July 1990>.
<Bob> <is interested in> <the Mona Lisa>.
<the Mona Lisa> <was created by> <Leonardo da Vinci>.
<the video 'La Joconde à Washington'> <is about> <the Mona Lisa>
RDF syntax

• N3, TVS
• Turtle
• TriG
• N-Triples
• RDF/XML
• RDF/JSON
Name spaces

• Using **short names for URL-s**
  – Long names are tedious
• Simple but strong concept
• **Defining name space:**

prefix rdf:, namespace URI: http://www.w3.org/1999/02/22-rdf-syntax-ns#
prefix rdfs:, namespace URI: http://www.w3.org/2000/01/rdf-schema#
prefix dc:, namespace URI: http://purl.org/dc/elements/1.1/
prefix owl:, namespace URI: http://www.w3.org/2002/07/owl#
prefix ex:, namespace URI: http://www.example.org/ (or http://www.example.com/)
prefix xsd:, namespace URI: http://www.w3.org/2001/XMLSchema#
N-Triples

<http://example.org/bob#me> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> .
<http://example.org/bob#me> <http://xmlns.com/foaf/0.1/knows> <http://example.org/alice#me> .

Turtle

01  BASE <http://example.org/>
02  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
03  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
04  PREFIX schema: <http://schema.org/>
05  PREFIX dcterms: <http://purl.org/dc/terms/>
06  PREFIX wd: <http://www.wikidata.org/entity/>
07
08  <bob#me>  
09      a foaf:Person ;
10      foaf:knows <alice#me> ;
11      schema:birthDate "1990-07-04"^^xsd:date ;
12      foaf:topic_interest wd:Q12418 .
13
14  wd:Q12418
15      dcterms:title "Mona Lisa" ;
17
18  <http://data.europeana.eu/item/04802/243FA8618938F4117025F17A8B813C5F9AA4D619> 
19      dcterms:subject wd:Q12418 .
Additional RDF Constructs

- Complex values
  - Bags, lists, trees, graphs
- Empty nodes
- Types of atomic values
- Types of nodes
- Reification
RDF Schema

- **RDFS**
- **Knowledge representation language**
  - Not just graph any more!
  - AI Frames, Object Model
- **Small dictionary for RDFS**
  - rdfs:class, rdfs:subClassOf, rdfs:type
  - rdfs:property, rdfs:subPropertyOf
  - rdfs:domain, rdfs:range
## RDFS Concepts

<table>
<thead>
<tr>
<th>Construct</th>
<th>Syntactic form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class (a class)</td>
<td>C rdf:type rdfs:Class</td>
<td>C (a resource) is an RDF class</td>
</tr>
<tr>
<td>Property (a class)</td>
<td>P rdf:type rdf:Property</td>
<td>P (a resource) is an RDF property</td>
</tr>
<tr>
<td>type (a property)</td>
<td>I rdf:type C</td>
<td>I (a resource) is an instance of C (a class)</td>
</tr>
<tr>
<td>subClassOf (a property)</td>
<td>C1 rdfs:subClassOf C2</td>
<td>C1 (a class) is a subclass of C2 (a class)</td>
</tr>
<tr>
<td>subPropertyOf (a property)</td>
<td>P1 rdfs:subPropertyOf P2</td>
<td>P1 (a property) is a sub-property of P2 (a property)</td>
</tr>
<tr>
<td>domain (a property)</td>
<td>P rdfs:domain C</td>
<td>domain of P (a property) is C (a class)</td>
</tr>
<tr>
<td>range (a property)</td>
<td>P rdfs:range C</td>
<td>range of P (a property) is C (a class)</td>
</tr>
</tbody>
</table>
Classes

ex:MotorVehicle rdf:type rdfs:Class .
ex:PassengerVehicle rdf:type rdfs:Class .
ex:Van rdf:type rdfs:Class .
ex:Truck rdf:type rdfs:Class .
ex:MiniVan rdf:type rdfs:Class .

ex:Van rdfs:subClassOf ex:MotorVehicle .
ex:Truck rdfs:subClassOf ex:MotorVehicle .

ex:MiniVan rdfs:subClassOf ex:Van .
SPARQL

- **SPARQL Protocol and RDF Query Language**
- **SPARQL query**
  - Graph can include variables in place of constants
- **Operations**
  - JOIN (natural, left-join)
  - AND, FILTER, UNION, OPTIONAL
- **Commercial DBMS-s**
  - Implement RDF and SPARQL
PREFIX abc: <http://mynamespace.com/exampleOntology#>

SELECT ?capital ?country
WHERE { ?x abc:cityname ?capital.
   ?x abc:isCapitalOf ?y.
   ?y abc:isInContinent abc:africa. }
Logic - OWL

- **Ontology language**
  - Knowledge representation + Logic
- **Based on description logic**
  - Fragments of predicate calculus
  - Hierarchy of DL languages
- **OWL reasoners**
  - FaCT++, Hermit, RacerPro, Pellet, ...
Wordnet

- Princeton's large lexical database of English.
  - Cognitive synonyms: **synsets**
    - 117,000 synsets
  - Synsets are linked by:
    - conceptual-semantic relationships, and
    - lexical relationships.
    - Include **definitions** of synsets.
  - Main relationships:
    - Synonymy, hyponymy (ISA), meronymy (part-whole), antonymy
Linked Open Data

• Datasets are represented in RDF
  – Wikipedia, Wikibooks, Geonames, MusicBrainz, WordNet, DBLP bibliography
• Number of triples: 33 Giga \( (10^9) \) (2011)
• Governments:
  – USA, UK, Japan, Austria, Belgium, France, Germany, ...
• Active community
  – http://www.w3.org/LOD
### Basic Statistics

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Average</th>
<th>Min</th>
<th>Max</th>
<th>Median</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triples</td>
<td>67,544.15</td>
<td>0</td>
<td>47,054,407</td>
<td>337.0</td>
<td>192,230,648</td>
</tr>
<tr>
<td>Entities</td>
<td>18,105.28</td>
<td>0</td>
<td>9,319,918</td>
<td>80.0</td>
<td>54,225,309</td>
</tr>
<tr>
<td>Literals</td>
<td>30,137.45</td>
<td>0</td>
<td>31,476,008</td>
<td>166.0</td>
<td>90,261,655</td>
</tr>
<tr>
<td>Blanks</td>
<td>3,554.83</td>
<td>0</td>
<td>3,565,513</td>
<td>0.0</td>
<td>10,646,711</td>
</tr>
<tr>
<td>Blanks as subject</td>
<td>1,742.85</td>
<td>0</td>
<td>1,910,532</td>
<td>0.0</td>
<td>5,219,831</td>
</tr>
<tr>
<td>Blanks as object</td>
<td>1,812.01</td>
<td>0</td>
<td>3,564,789</td>
<td>0.0</td>
<td>5,426,969</td>
</tr>
<tr>
<td>Subclasses</td>
<td>1.6</td>
<td>0</td>
<td>2,000</td>
<td>0.0</td>
<td>4,779</td>
</tr>
<tr>
<td>Typed subjects</td>
<td>7,387.12</td>
<td>0</td>
<td>6,990,722</td>
<td>39.0</td>
<td>22,124,421</td>
</tr>
<tr>
<td>Labeled subjects</td>
<td>1,219.97</td>
<td>0</td>
<td>1,440,595</td>
<td>0.0</td>
<td>3,653,811</td>
</tr>
<tr>
<td>Average properties per entity</td>
<td>4.98</td>
<td>0.0</td>
<td>91.16</td>
<td>3.71</td>
<td></td>
</tr>
<tr>
<td>Average string length typed</td>
<td>13.28</td>
<td>0.0</td>
<td>436.0</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>Average string length untyped</td>
<td>391.77</td>
<td>0.0</td>
<td>181,576.0</td>
<td>10.0</td>
<td></td>
</tr>
<tr>
<td>Average class hierarchy depth</td>
<td>3.24</td>
<td>1</td>
<td>9</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Links</td>
<td>15,379.59</td>
<td>0</td>
<td>13,252,430</td>
<td>57.0</td>
<td>46,061,873</td>
</tr>
<tr>
<td>Average property hierarchy depth</td>
<td>1.5</td>
<td>1</td>
<td>3</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Vocabularys</td>
<td>4.27</td>
<td>1</td>
<td>18</td>
<td>3.0</td>
<td>12,110</td>
</tr>
<tr>
<td>Classes</td>
<td>4.36</td>
<td>1</td>
<td>330</td>
<td>3.0</td>
<td>10,384</td>
</tr>
<tr>
<td>Properties</td>
<td>17.58</td>
<td>1</td>
<td>254</td>
<td>16.0</td>
<td>49,916</td>
</tr>
</tbody>
</table>

9960 datasets

149,423,660,620 triples from **2973 datasets** (192,230,648 triples from **2838 dumps**, 149,231,429,972 from **151 datasets via SPARQL**)

Problems with **6971 datasets** (70.1%): **6578 dumps having errors**, **393 SPARQL endpoints with errors**
Open Data

The home of the U.S. Government’s open data
Here you will find data, tools, and resources to conduct research, develop web and mobile applications, design data visualizations, and more.

GET STARTED
SEARCH OVER 194,804 DATASETS

Federal Student Loan Program Data

BROWSE TOPICS

Agriculture  Climate  Consumer  Ecosystems  Education  Energy  Finance

Health  Local Government  Manufacturing  Maritime  Ocean  Public Safety  Science & Research
Freebase

- Free, knowledge graph:
  - people, places and things,
  - 3,041,722,635 facts, 49,947,845 topics
- Semantic search engines are here!
Freebase

• Based on graphs:
  – nodes, links, types, properties, namespaces

• Google use of Freebase
  – Knowledge graph
  – Words become concepts
  – Semantic questions
  – Semantic associations
  – Browsing knowledge
  – Knowledge engine

• Available in RDF
Knowledge graph

• Google’s Knowledge Graph
  – 70 billion facts, oct 2016
  – Box to the right of search results, since 2012
  – Google Assistant and Google Home voice queries

• Knowledge Vault, Google, 2014
  – Initiative to succeed the capabilities of the Knowledge Graph
    • ... to deal with facts, automatically gathering and merging information from across the Internet into a knowledge base capable of answering direct questions, such as "Where was Madonna born"
YAGO

- 10 Mega ($10^6$) concepts
  - 120M facts about these entities
  - Max Planc Institute, Informatik
  - Accuracy of 95%

- Includes:
  - Wikipedia, WordNet, GeoNames
  - Links Wordnet to Wikipedia taxonomy (350K concepts)
  - Anchored in time and space
Wikidata

- Free knowledge base with 46,769,977 items
  - 14,913,910 - 2015
- Collecting structured data
- Properties of
  - person, organization,
Cyc - knowledge base

- **Knowledge base**
  - Doug Lenat
  - Conceptual networks (ontologies)
  - Higher ontology, basic theories, specific theories
  - Predefined semantic relationships
  - 500,000 terms, including about 17,000 types of relations, and about 7,000,000 assertions relating these terms

- **Common sense reasoner**
  - Based on predicate calculus
  - Rule-based reasoning
Cyc

Knowledge Base Layers

Upper Ontology

Core Theories: Space, Time, Causality, ...

Upper Ontology: Abstract Concepts

Domain-Specific Theories

Facts: Instances

Domain-Specific Theories
Storage level
Outline

• Triple-store representation
  – Relational representation
  – Property table
  – Index-based representation
  – Columnar representation
  – Graph-based representation
Relational representation

- Extending relational DBMS
  - Virtuoso, Oracle ...
- Statistics does not work
  - Structure of triple-store is more complex than bare 3-column table
- Extensions of relational technologies
  - Adding RDF data type in SQL
  - Virtuoso indexes store statistics
  - Quad table is represented by two covering indexes
    - GSPO and OGPS
Property table

- **Property table** in relational DBMS
  - Jena, DB2RDF, Oracle, ...
- Triples are grouped by properties
  - Property table is defined for groups
- **Advantages**
  - All properties read at once (star queries)
- **Drawbacks**
  - Property tables can have complex schemata
  - The values of some attributes may be rare
  - Sorting and clustering by S part of triples not possible
Index-based representation

- **Covering indexes**
  - RDF-3X, YAR2, 4store, Hexastore, ...

- **RDF-3X (MPI, 2009)**
  - Compressed clustered B+-tree
  - Sorted lexicographically for range scans
  - Compression based on order of triples
  - Aggregate indexes
    - Two keys + counter
    - One key + counter
Index-based representation

- **Hexastore (Uni Zuerich, 2008)**
  - Treats subjects, properties and objects equally
  - Every possible ordering of 3 elements is materialized
    - SPO, SOP, PSO, POS, OSP, and OPS
  - The result is a **sextuple indexing scheme**
    1. All three, S|P|O-headed divisions of data
    2. Each division has appropriate S|P|O vector pairs
    3. Each vector pair has associated S|P|O values
Index-based representation

- Hexastore
  - 3-level special index
  - Appropriate for some types of joins
    - Merge-joins
  - Reduction of unions and joins
  - 5-fold increase of DB size
Columnar representation

- **Vertical partitioning** of RDF (Yale, 2009)
  - Daniel Abadi
  - Triples table is stored into \( n \) two-column tables
    - \( n \) is the number of unique properties in the data

- **Advantages**
  - reduced I/O: **reading only the needed properties**
  - Column-oriented data **compression**
Columnar representation

- Optimizations for fixed-length tuples.
- Optimized column merge code
- Direct access to sorted files
- Column-oriented query optimizer.

- Materialized path expressions
  - Direct mapping is stored instead of paths
  - Can speed-up queries enormously (... is critics)

- Disadvantages
  - Increased number of joins.
Graph-based representation

- **Native graph representation**
  - Nodes have associated adjacency lists
    - Links to nodes connected to a given node
  - Subgraph matching using homomorphism

- **Examples of systems**
  - gStore, Neo4j, Trinity.RDF

- **Graph homomorphism are NP-complete**
  - Scalability of the approach is questionable
Graph-based representation

• gStore
  – Works directly on the RDF graph and the SPARQL query graph
  – Use a signature-based encoding of each entity and class vertex to speed up matching
    • Get all class instances, all subjects with a given property, …
    • Speeding up some basic operations
  – Filter-and-evaluate
    • Queries are transformed into query graphs
    • Use a false positive algorithm to prune nodes and obtain a set of candidates;
    • Evaluation of joins between candidate sets
  – Use an index (VS*-tree) over the data signature graph (has light maintenance load) for efficient pruning
Graph-based representation
Data distribution
Outline

• Triple-store distribution
  – Hash horizontal partitioning
  – Locality-based horizontal partitioning
  – N-hop guarantee horizontal partitioning
  – Semantic hash partitioning
  – Semantic-aware partitioning
Horizontal hash partitioning

• Hash partitioning on a given key
  – A key can be any value
    • Subset of tuple|triple components
    • Component of object
  – Triples, tuples, objects are distributed
    • Round-robin, hash or range partitioning
    • Partitioned parallelism
    • Key-based and range access are directed to a single or to a subset of systems
  – Sometimes complete partitions (fragments) are hashed
Horizontal hash partitioning

• **Data partitioning in relational systems**
  – Hash-partitioning
    • Scales up to few hundreds of servers
    • All results go to the coordinator
    • **Network bandwidth** may be a bottleneck
  – Range-based partitioning
    • Attribute range is divided into subsets
    • Problems with skew
  – Predicate-based partitioning
    • Minterms, selectivity, access frequencies
    • The art of the design, complex, skewed
Horizontal hash partitioning

• Hash partitioning in NoSQL systems
  – Fundamental method of key-value databases
  – Very efficient for a simple key-value data model
    • Simple data access by means of nicely defined keys
  – **Consistent hashing** method gives very good results
    • Keys are uniformly distributed to servers
    • Allows adding/removing servers in run-time
  – Dynamo, Cassandra, Bigtable, ...

• **Triple-stores are based on both**
  – Relational and Key-Value models
Horizontal hash partitioning

• **Basic hash partitioning**
  – Hash partition triples across multiple machines, and parallelize access to these machines as much as possible at query time
  – All servers return results at the same time

• **Locality preserving hash partitioning**
  – Triples are distributed in locality-based partitions
  – Queries are split into sub-queries
  – Sub-queries are executed on servers that store the data
Horizontal hash partitioning

- Hash partitioning on S part of triples
  - **Object oriented view**
    - Objects are represented by groups of triples having the same S part
    - Triples representing objects are hashed into the same node numbers
  - **This is random partitioning**
    - There are no correlations among objects mapped to a given node number
- Systems
  - SHARD, 4store, YARS2, Virtuoso, TDB, ...
Locality-based horizontal partitioning

• **Use of min-cut graph partitioning**
  – METIS algorithms are often used
  – Nodes are partitioned into k partitions

• **Placement of triples into partitions follows the partitioning of nodes**
  – Therefore, *subject-based partitioning*
  – Partitions are replicated as in key-value systems to obtain better availability
  – Query is decomposed; query fragments posed to partitions

• **Originally proposed by**
  – Scalable SPARQL Querying of Large RDF Graphs, Huang, Abadi, VLDB, 2011.
Locality-based horizontal partitioning

- TriAD (MPI, 2014)
  - **Summary graph** is computed first
    - Supernodes are constructed from the data graph
      - Link between supernodes if there exists a strong connectivity between them
    - Intuition: processing query on summary graph eliminates partitions that are not addressed
    - METIS algorithm is used for graph partitioning
  - Locality information provided by the summary graph leads to **sharding**
    - Entire partitions are hashed to nodes
    - Triples on the edge between two partitions are placed in both partitions
    - Join-ahead pruning of partitions
N-hop guarantee horizontal partitioning

- Huang, Abadi, Ren: Scalable SPARQL Querying of Large RDF Graphs, VLDB, 2011

- Leveraging state-of-the-art single node RDF-store technology
  - Columnar representation is used
  - Careful fix-sized record implementation
  - Merge-joins are optimized

- Partitioning the data across nodes
  - Accelerate query processing through locality optimizations
  - Edge partitioning is used (not node partitioning)
  - METIS used for min-cut vertex graph partitioning
    - rdf:type triples are removed before
N-hop guarantee horizontal partitioning

- **Triple placement**
  - We have vertex based partitioning
  - Simple way: use S part partition for complete triple
  - Triples on the borders are replicated
  - More replication results less communication
  - Controlled amount of replication
    - Directed n-hop guarantee
    - Start with 1-hop guarantee and then proceed to 2-hop guarantee, ...
    - Partitions are extended to conform n-hop guarantee
- Decomposing SPARQL queries into **high performance fragments** that take advantage of how data is partitioned in a cluster.
Semantic hash partitioning

• **Minimizing the amount of interpartition coordination and data transfer**
  – None of the existing data partitioning techniques takes this into account
  – Kisung Lee, Ling Liu, Scaling Queries over Big RDF Graphs with Semantic Hash Partitioning, VLDB, 2013

• **Semantic hash partitioning algorithm** performs data partitioning in three main steps:
  1. Building a set of triple groups which are baseline building blocks for semantic hash partitioning.
     • S, O and S+O triple groups
     • Star queries can be answered fast in parallel
Semantic hash partitioning

2. **Grouping the baseline building blocks** to generate baseline hash partitions
   - S, O, S+O -based grouping
   - Hashing groups to partitions based on S|O|S+O
   - Technique to bundle different triple groups into one partition

3. **Generating Semantic Hash Partitions**
   - Mapping triple groups to baseline is simple and generates well balanced partitions
   - Poor performance for complex non-star queries.
   - The **hop-based triple replication** was proposed for this reason.
   - Semantic hash partitions are defined to maximize intra-partition query processing.
Self Evolving Distributed Graph Management Environment


• 2-level partition management architecture
  – Complimentary primary partitions and dynamic secondary partitions
  – Minimize inter-machine communication during graph query processing in multiple machines

• Implemented on top of Pregel
Entity-class partitioning

• EAGRE (HKUST, 2013)
  – Semantic-aware partitioning
  – Goal is to reduce the I/O cost incurred during query processing
    • Speed-up queries with range filter expressions
    • A distributed I/O scheduling solution
      – Finding the data blocks most likely to contain the answers to a query.
    • Entity-based compression scheme for RDF
Entity-class partitioning

– Procedure
  • RDF graph is transformed into an **entity graph** where only nodes that have out-going edges are kept
  • Entities with similar properties are grouped together into an **entity class**
  • The **compressed RDF graph** contains only entity classes and the connections between them (properties)
  • The global compressed entity graph is then **partitioned using METIS**
Semantic-aware partitioning

- big3store: distributed triple-store
  - In development from 2014
  - Yahoo! Japan Research & University of Primorska
  - Erlang programming environment

- The **main idea** of the method
  1. Cluster the data on the schema level
     - Use statistics for the estimation
  2. Distribute the extensions of the schema partitions
big3store: partitioning method

1. Choose a skeleton graph from the hierarchy of edge types
   – Edge types are ordered into partially ordered set
   – Start from the top most general edge type
   – Specialize edge types until they are of appropriate size

2. Cluster a skeleton graph to obtain k partitions
   – Cluster strongly connected edges together
   – Connectivity is defined by means of the statistics of edge types
big3store: Computing skeleton graph

Top of schema triple hierarchy

(owl:Thing, rdf:Property, owl:Thing)

↑ = “is more specific triple”

○ = schema triple

● = schema triples that have the interpretation of appropriate size

= edges of the skeleton graph

Schema graph = selected schema triples

(employee, worksAt, organization) (person, worksAt, company)

(engineer, worksAt, organization) (person, worksAt, company)

...
Given:
- statistics of TS
- skeleton graph $G_s$

Schema graph
- selected schema triples
- represented as graph!

Distance function:
- distance between edges $e_1$ and $e_2$
  - based on shortest path $p$ starting with $e_1$ and ending with $e_2$
  - estimate the number of path $p$ instances
  - estimate the cardinality of each join in a path $p$ by using the statistics of TS
big3store: Clustering skeleton graph

Clustering algorithm:
- any clustering algorithm
  - strongly connected edge types are clustered together
  - maximize average strength of the paths among all different pairs of nodes from a partition (see problem definition, page 7)

Statistics:
- For each schema triple ts:
  # instances of edge type ts
  # distinct values of edge type ts
  estimation of the size of joins

Result:
dbkda18
- partitions of $G_s$ (sets of edges)
Query processing
Outline

• Query processing
  – Algebra of graphs
    • Logical algebra
    • Physical algebra
  – Parallel execution of operations
  – Centralized triple-store systems
  – Federated centralized database systems
  – State-of-the-art directions
RDF algebra

- select
- project
- join
- union, intersect, difference
- leftjoin

- **Algebra of sets of graphs**
- Sets of graphs are input and output of operations
  - Triple is a very simple graph
  - Graph is a set of triples
RDF algebra

Triple-patterns

Graph-patterns

\[
GP ::= TP | select(GP, C) | join(GP, GP) | union(GP, GP) | intsc(GP, GP) | diff(GP, GP) | leftjoin(GP, GP)
\]

\[
TP ::= (S | V, P | V, O | V)
\]

\[
C ::= V OP V | V OP O | C \land C | C \lor C | \neg C
\]

\[
OP ::= = | \neq | > | \geq | < | \leq
\]

\[
S ::= URI | Blank-Node
\]

\[
P ::= URI
\]

\[
O ::= URI | Blank-Node | Literal
\]

\[
V ::= ?a .. ?z
\]

Variables

Conditions
Logical algebra

• **Triple-pattern is access method**
  - $tp_1 = (?, p, o)$, $tp_2 = (?, p, ?y)$, ...
  - $tp_1$ retrieves all triples with given $P$ and $O$

• **Triple pattern syntax**
  - $TP ::= (S | V, P | V, O | V)$

• **Triple-pattern semantics**

\[
\llbracket (t_1, t_2, t_3) \rrbracket_{db} = \{ (s, p, o) \mid (s, p, o) \preceq db \wedge ground((s, p, o)) \wedge (s, p, o) \sim (t_1, t_2, t_3) \} 
\]
Logical algebra

• Join operation
  – Joins all graphs from outer sub-tree with graphs from inner triple-pattern
  – Common variables from outer and inner graphs must match

• Syntax
  – \( \text{GP ::= ... | join(GP,GP) | ...} \)
  – Second argument is TP in left-deep trees

• Semantics

\[
\left[ \text{join}(gp_1, gp_2) \right]_{db} = \{ g_1 \cup g_2 \mid g_1 \in \left[ gp_1 \right]_{db} \land g_2 \in \left[ gp_2 \right]_{db} \land \forall v \in vs : \text{val}(v, gp_1, g_1) = \text{val}(v, gp_2, g_2) \}
\]
Logical algebra

Triple-pattern

\[ \text{tp}(?c, \text{<hasArea>}, ?a) \]

Operation join

\[
\text{join}( \text{join}( \text{tp}(?c, \text{<hasArea>}, ?a), \\
\text{tp}(?c, \text{<hasLatitude>}, ?l)), \\
\text{tp}(?c, \text{<hasInfration>}, ?i))
\]

SPARQL query language

```
SELECT * WHERE {
  ?c \text{<hasArea>} ?a .
  ?c \text{<hasLatitude>} ?l .
  ?c \text{<hasInfration>} ?i
}
```
Physical operations

- **Access method (AM)**
  - Triple-pattern operation
  - Includes select and project operations
- **Join**
  - Logical join operation
  - Includes select and project operations
- **Union, intersect and difference**
  - Retain the schema of parameters
Physical operations

- **Implementation of TP access method**
  - Distributed file system AM
    - Read and filter appropriate file
    - Vertical partitioning: predicate files are searched
  - Index-based triple-store
    - Key-value store:
      - Direct lookup, prefix lookup and scan over table T
    - Covering B+ index for the keys given in TP
      - Access with ALL possible subsets of \{ S, P, O \}
- **Federated centralized systems**
  - Query processing pushed to data nodes
    - Data nodes are centralized RDF stores (e.g., RDF-3X)
  - Query is represented by a tree of processes
Physical operations

• Join implementation
  – Index nested-loop join
    • Rya (Inria, 2012)
    • $H_2$ RDF (Uni Athens, 2012)
  – Merge-join
    • RDF-3X (extensively uses merge-join)
    • TriAD (distributed merge-join on sharded data)
    • Hexastore (merge-joins as first-step pairwise joins)
  – Hash-join
    • Virtuoso (almost never preferred for RDF)
    • TriAD (distributed hash-join on sharded data)

• Main-memory join
  • AMADA main-memory hash join (Inria, 2012)
Physical algebra

- **Left-deep trees**
  - Pipelined parallelism
  - Dynamic (greedy) optimization possible
- **Bushy trees**
  - More opportunities for parallel execution
- **Large search space**
  - $O(n \times 2^n)$ star queries, $O(3^n)$ path queries
- **Cost-based static optimization**
  - For both cases
Graph patterns

• **Set of triple-patterns linked by joins**
  – select and project packed into joins and TPs

• **Graph-patterns similar to SQL blocks**
  – select and project pushed-down to leafs of query
  – Joins can now freely shift -> **Join re-ordering**

• **Graph-patterns are units of optimization**
  – Optimization can be based on dynamic programming
  – Bottom-up computation of execution plans
Centralized systems

• Single server system
• Based on the relational database technology
• Best of breed example:
  – RDF-3X (MPI)
  – Classical query optimization
  – Multiple index approach
Example: RDF-3X

- **6 B+ tree indexes**
  - All interesting orders can be materialized

- **Query optimization**
  - **Join re-ordering in bushy trees**
    - Possible large number of joins
    - Star-shaped sub-queries are the primary focus
  - **Cost-based query optimization**
    - Statistics (histograms) stored in aggregate indexes
    - Plan pruning based on cost estimation (heuristics)
  - **Bottom-up dynamic programming algorithm**
    - Keeps track of a set of the plans for interesting orders
    - Exhaustive use of merge-join algorithm
    - Uses also a variant of hash join
Federated centralized database systems

- A federated database system transparently maps multiple autonomous database systems into a single federated database
  - Stand alone shared-nothing servers
  - Typically have coordinator nodes and data nodes
    - Not all nodes have the same functionality

- **Examples:**
  - TriAD
  - Huang et al.
  - WARP
Query parallelism

- Partitioned parallelism
- Pipelined parallelism
- Independent parallelism
Query parallelism

• TP processing is distributed
  – Data addressed by a TP is distributed
  – Processing TP in parallel

• Left-deep trees form pipelines
  – Each join on separate server?
    • Join runs on the same machine as its inner TP
  – Faster query evaluation

• Bushy trees
  – Parallel execution of sub-trees and operations

• Split joins to more smaller parallel joins
  – Exploiting multiple processors and cores
  – Parallel execution of joins
Example: Huang et al., 2011

- Huang, Abadi, Ren: Scalable SPARQL Querying of Large RDF Graphs, VLDB, 2011
- Architecture
  - RDF-3X used as centralized local triple-store
  - Hadoop is linking distributed data stores
  - Master server and slave data stores
- Locality-based partitioning
  - METIS used for min-cut graph partitioning
  - Partitioning helps accelerate query processing
    - Through locality optimizations
  - Placement with n-hop replication
Example: Huang et al., 2011

• Algorithm for automatically decomposing queries into parallelizable chunks
  – Concept of PWOC queries
    • PWOC=Parallelizable without communication
    • Concept of central vertex in query graph
      – Minimal “distance of farthest edge” (DoFE)
    • Central vertex is native in a partition with n-hop guarantee
      – DoFE < n => PWOC query
  – Non-PWOC queries
    • Decompose into PWOC subqueries
    • Minimal edge partitioning of a graph into subgraphs of bounded diameter (well studied problem in theory)
      – Heuristics: Choose decomposition with minimal number of PWOC components
      – More PWOC components more work for Hadoop
Example: WARP, 2013


- **Architecture**
  - Improved design of Huang, et.al., 2011
    - Graph-based RDF partitioning
    - Distributed parallel query processing in combination with MapReduce
  - RDF-3X is used as the local database system
Example: WARP, 2013

• **Query evaluation**
  – Query is posed to all servers and only those with the data respond
  – One-Pass Queries (MPQ)
    • Triple-patterns and star queries
    • More complex queries are analyzed to see if they can be evaluated in extended n-hop partition
      – Concept of center node is used
      – Only one partition gives the results; no need to handle duplicates
  – Multi-Pass Queries (MPQ)
    • Optimizator creates all possible splits of MPQ
      – Left-deep plan is considered among sub-queries
      – Heuristic to choose the split with the smallest number of subqueries
      – No statistics, no cost-based optimization
    • Merge-joins implemented instead of MapReduce joins
Example: WARP, 2013

- **Workload-aware replication of triples across partitions**
  - METIS-based partitioning as in Huang, et.al, 2011
    - N-hop replication is also used
  - Relational systems define the partitions on the basis of the predicates that are used in the queries
    - This method has been extended to triple-stores
    - It is possible only because of the simplicity of triple-stores
    - Representative Workload is computed ...
  - To increase the fraction of OPQ queries
    - Increase the n-hop replication horizon
    - Systematic replication for frequently issued queries
  - Replication for MPQs
    - Optimization phase takes into account the structure of partitions
    - Sub-queries are attuned to existing partitions
Example: TriAD, 2014

• Federated centralized system
  – Extension of centralized RDF-3X to distributed environment
  – Based on asynchronous message passing

• System architecture
  – Master-slave, shared-nothing model
  – Master node
    • Metadata about indexed RDF facts stored in local indexes
    • Summary graph, bidirectional dictionaries, global statistics, query optimizer
  – Slave nodes
    • Include local indexes, local query processor
    • Exchange intermediate results with asynchronous messages
Example: TriAD, 2014

- Construction of **summary graph**
  - Nodes are partitioned in disjunctive partitions (supernodes)
    - Graph partitioning with METIS
    - Edges with distinct labels are chosen among supernodes
    - Optimal number of partitions is determined
      - Cost model optimization of summary and data graph querying
  - Summary graph is indexed at the master node
- Horizontal partitioning of data triples
  - Locality defined by summary graph is preserved
    - Hashing summary graph partitions into the grid-like distribution scheme
    - Hashing based on S and O together with supernodes
    - Triples belonging to the same supernode are placed on the same horizontal partition
Example: TriAD, 2014

• Query processing
  – “pruning stage”, is performed entirely at master node
    • Executing queries on summary graph (at master)
      – bindings of supernode identifiers to query variables (exploratory-based)
      – determine the best exploration order using a first DP-based optimizer over the summary graph statistics
    • Eliminates unneeded partitions – partition prunning
  – Distribution aware query optimizer
    • Process the query against the data graph which is distributed
    • Determine the best join order by using a second DP optimizer
      – Precise statistics is used
    • Global query plan generated at the master is then communicated to all slaves
      – Multi-Threaded, asynchronous plan execution
Example: big3store, 2016

• Yahoo! Japan and University of Primorska
  – Implementation in Erlang environment

• Architecture
  – Master and slave nodes
  – Master node (front server) compiles the query
    • Creating tree of processes on slave nodes
    • No optimization at the moment (queries are programmed)
    • Dynamic load-balancing through replicated slave nodes
  – Slave nodes store graph partitions
    • Store partitions of the complete graph
    • Slave nodes are replicated to achieve high throughput
    • Pipelined execution of joins on slave nodes (data servers)
Example: big3store, 2016

• Data distribution
  – Semantic-based partitioning
    • Triple-stores include very expressive schema definition
    • Knowledge representation language (RDF+RDFS)
  – Partitioning based on the structure of query space
  – Construction of the skeleton graph
    • Schema graph is constructed from the stored schema
  – Skeleton graph is partitioned
    • Some clustering algorithm used maximizing the strength of the partitions
  – Data triples partitioning follows the skeleton graph
    • Some triples are in more than one partitions
Example: big3store, 2016

• Query evaluation
  – Programmer defines
    • the query structure and
    • chooses implementation of algebra operations
  – Front servers compiles the query
    • Creates query tree, determines the data servers
    • Spawns query tree
      – Creating and linking the processes at the data nodes
    • Controls the evaluation of the queries
  – Data servers
    • Store SPO permutation indexes
    • Various implementations of joins and access methods
Example: Trinity.RDF

- **Main-memory** distributed triple-store
  - Native graph representation in memory
  - Efficient in-memory **graph exploration** instead of join operations
  - Zeng, Et.Al., A Distributed Graph Engine for Web Scale RDF Data, VLDB 2013

- **Exploration-based SPARQL query processing**
  - Decomposes a SPARQL query into a set of triple patterns
  - Graph explorations to generate bindings for each of the triple pattern
  - Using binding information of the explored subgraphs to prune candidate matches in a greedy manner
State-of-the-art directions

• Data manipulation in main memory
  – Huge main memory is available currently
  – Most queries are executed much faster in main memory

• Careful construction of localized partitions
  – Data that is frequently queried together is stored in one partition
  – Network communication is significantly reduced

• Utilization of the schema in triple-stores
  – All novel triple-stores have rich schemata provided as RDFS triples
  – Schemata can be used for speeding up queries and for semantic-aware partitioning
State-of-the-art directions

• Abstracting the data graph
  – Construction of the summary graph by
    • Data mining algorithms that group similarly structured sub-graphs
    • Employing graph partitioning for the construction of the summary graphs
  – Summary graph can be exploited for
    • Construction of well-localized partitions
    • Directing the evaluation query

• Workload-aware partitioning
  – Exploiting workload for the definition of partitions
  – Dynamical run-time adjustment of the partitions
Thank you !