Graph Database Systems

Iztok Savnik
University of Primorska, Slovenia

FAMNIT, 2022

Outline

- Introduction to GDM
- Storage level representations
- Data distribution
- Query procesing

Introduction to Graph Data Model

Graph Data Model

- Graph database
  - Database uses graphs for the representation of data and queries
- Vertexes
  - Represent things, books, events, persons, concepts, classes, types, ...
- Arcs
  - Represent properties, relationships, associations, ...
  - Arcs have labels !
- Various names of a graph database
  - Triplestore, RDF database, Linked data, Linked open data, Knowledge bases and Knowledge graphs

Position of graph databases

- Key-value model
- Baseline graph data model
- Relational data model
- Knowledge graphs

Simplicity
Complexity

Graph Data Model (GDM)

- Baseline: Graph representation !
  - More complex than KV data model
  - More simple and uniform than relational model
- Graph representation + KR dictionary
  - Turns into Knowledge Representation Language
  - Adding schema level to GDM
    - RDF Schema -> adding types to KR representation
    - Represents alternative to AI Frames (KR lang)
    - Triples are more popular recently (see Cyc system)
  - Adding logic level to GDM
    - OWL; description logic; fragments of predicate calculus.
  - More expressive than the relational model.
- Query models: KV, Graph and Relational models

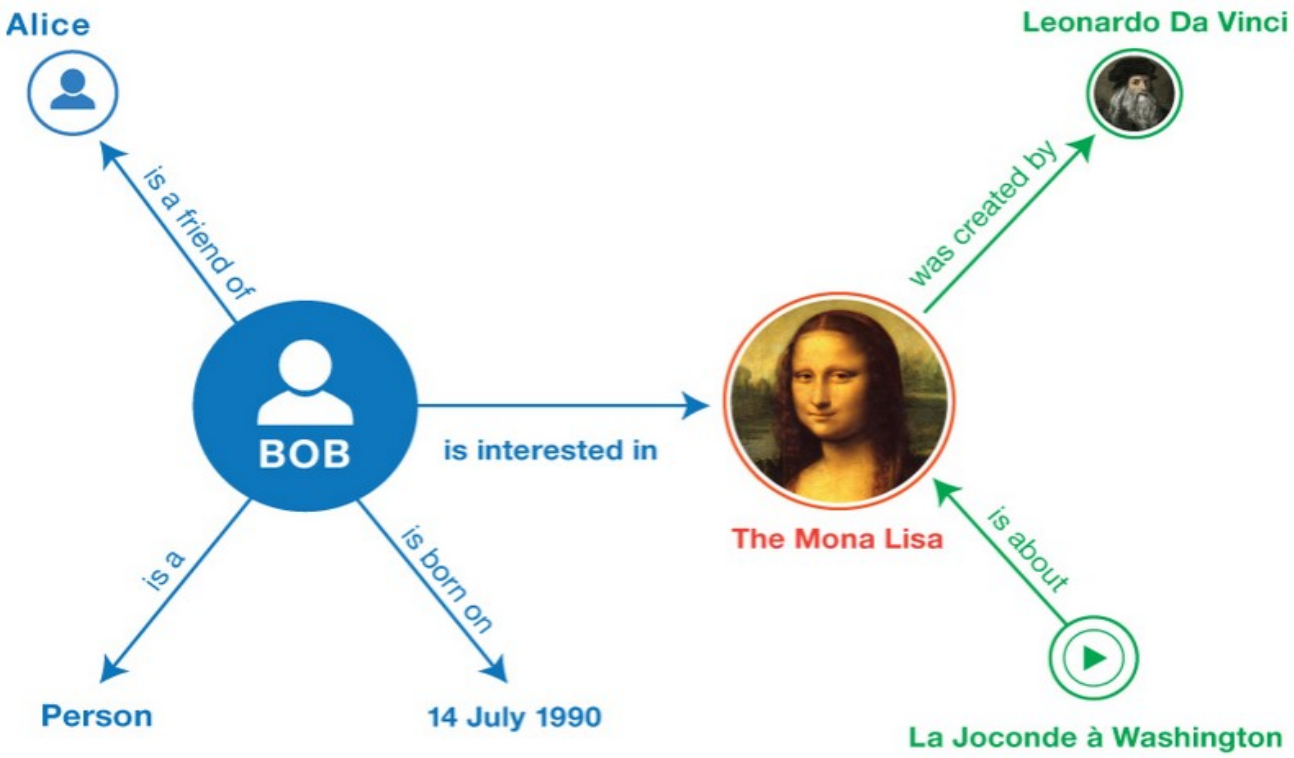
  1. Key-value access + MapReduce system
  2. Algebra of graphs + SPARQL
  3. Relational model + SQL

RDF

- Resource Description Framework
  - Tim Berners Lee, 1998, 2009 ...
- What is behind ?
  - Graphs are fundamental representation language.
  - Can represent data and knowledge!
  - Can be used for representation of data on the Internet
  - Can be used as medium for the exchange of scientific data
  - Can be extended with logic (see OWL, DL)
- Novel applications require some form of reasoning
  - Intelligent assistants, recommendation systems, system diagnostics, …
  - Data and knowledge will be integrated in novel applications
  - Many reasoners use triples (graphs) to represent knowledge and data

RDF

```
<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>
```



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#

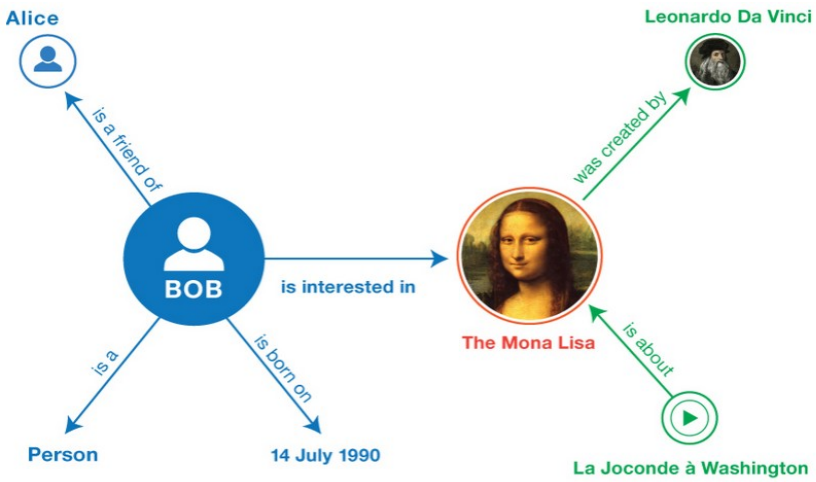
N3, TVS, Turtle, TriG, N-Triples RDF/XML, RDF/JSON

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> .
<http://example.org/bob#me> <http://schema.org/birthDate> "1990-07-04"^^<http://www.w3.org/2001/XMLSchema#date> .
<http://example.org/bob#me> <http://xmlns.com/foaf/0.1/topic_interest> <http://www.wikidata.org/entity/Q12418> .
<http://www.wikidata.org/entity/Q12418> <http://purl.org/dc/terms/title> "Mona Lisa" .
<http://www.wikidata.org/entity/Q12418> <http://purl.org/dc/terms/creator> <http://dbpedia.org/resource/Leonardo_da_Vinci> .
<http://data.europeana.eu/item/04802/243FA8618938F4117025F17A8B813C5F9AA4D619> <http://purl.org/dc/terms/subject>
```

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" ;
16      dcterms:creator <http://dbpedia.org/resource/Leonardo_da_Vinci> .
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 (KR 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

Construct	Syntactic form	Description
Class (a class)	**C** rdf:type rdfs:Class	**C** (a resource) is an RDF class
Property (a class)	**P** rdf:type rdf:Property	**P** (a resource) is an RDF property
type (a property)	**I** rdf:type **C**	**I** (a resource) is an instance of **C** (a class)
subClassOf (a property)	**C1** rdfs:subClassOf **C2**	**C1** (a class) is a subclass of **C2** (a class)
subPropertyOf (a property)	**P1** rdfs:subPropertyOf **P2**	**P1** (a property) is a sub-property of **P2** (a property)
domain (a property)	**P** rdfs:domain **C**	domain of **P** (a property) is **C** (a class)
range (a property)	**P** rdfs:range **C**	range of **P** (a property) is **C** (a class)

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

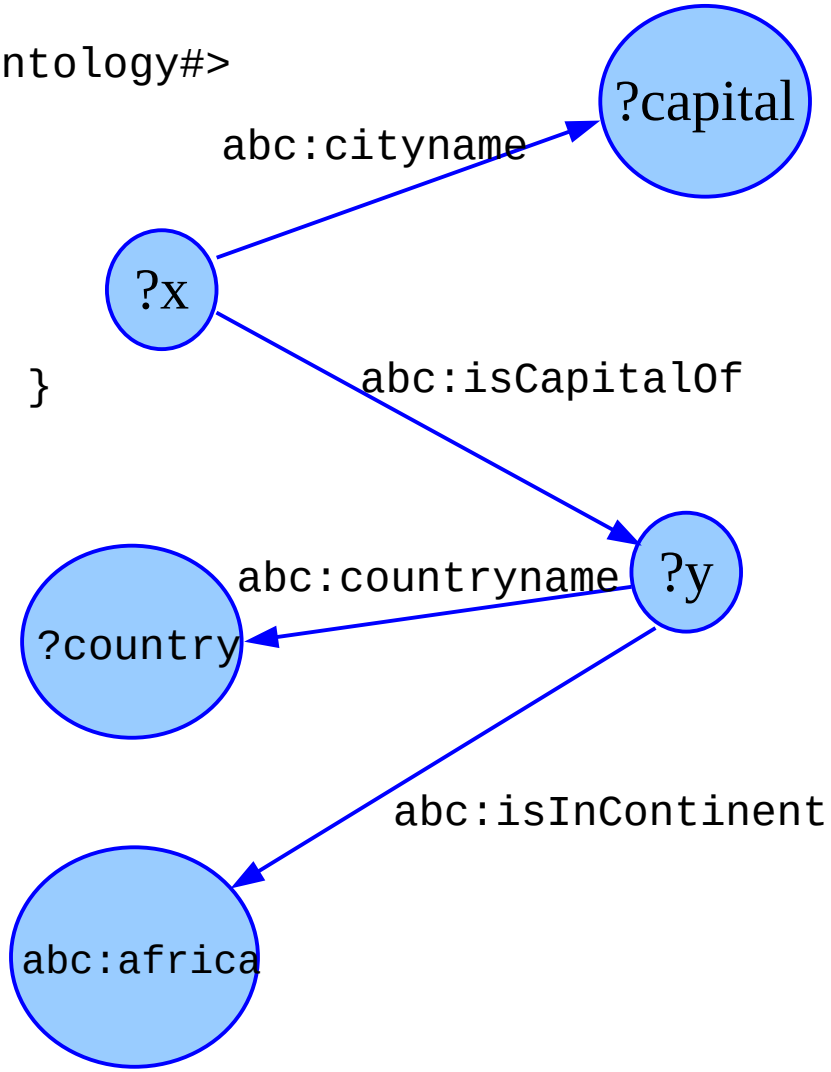
ex:MiniVan rdfs:subClassOf ex:Van .
ex:MiniVan rdfs:subClassOf ex:PassengerVehicle .

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

Example SPARQL query

```
PREFIX
   abc: <http://mynamespace.com/exampleOntology#>
SELECT ?capital ?country
WHERE { ?x abc:cityname ?capital.
        ?y abc:countryname ?country.
        ?x abc:isCapitalOf ?y.
        ?y abc:isInContinent abc:africa. }
```



Logic - OWL

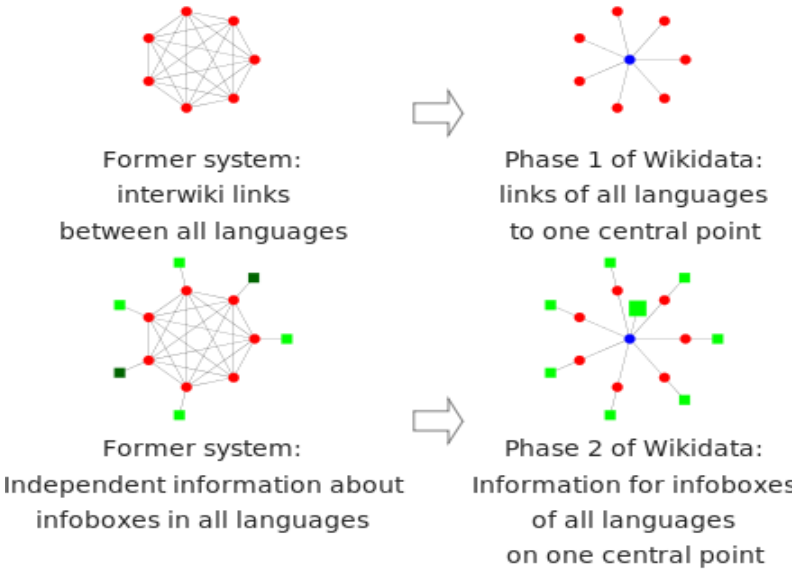
- Ontology language
  - Knowledge representation + Logic
- Based on description logic
  - Fragments of predicate calculus
  - Hierarchy of DL languages
- OWL reasoners
  - FaCT++, HermiT, RacerPro, Pellet, ...

Collective graph databases

- Datasets gathered in the form of a graph
- Wordnet
  - Princeton's large lexical database of English.
  - Cognitve synonims: 117,000 synsets
    - Synonymy, hyponymy (ISA), meronymy (part-whole), antonymy
- Linked Open Data
  - Wikipedia, Wikibooks, Geonames, MusicBrainz, WordNet, DBLP bibliography
  - Science community, Governments, Publishing, Media, ...
  - Active community
    - http://en.wikipedia.org/wiki/Open_Data
    - https://en.wikipedia.org/wiki/Linked_data

Collective graph databases

- Wiki Data
  - https://www.wikidata.org/
- Knowledge graphs
  - Freebase, Google KG
  - Microsoft, Yahoo KGs
  - Yago (MPI)
  - Semantic search engines



Storage level

Relational representation

- Extending relational DBMS
  - Virtuoso, Oracle, IBM, ...
- 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, ...
- Sets of objects with common properties (triples) stored in a relational tables
- Advantages
  - All properties read at once (star queries)
- Drawbacks
  - Property tables can have complex schemata
  - The values of some attibutes may be rare

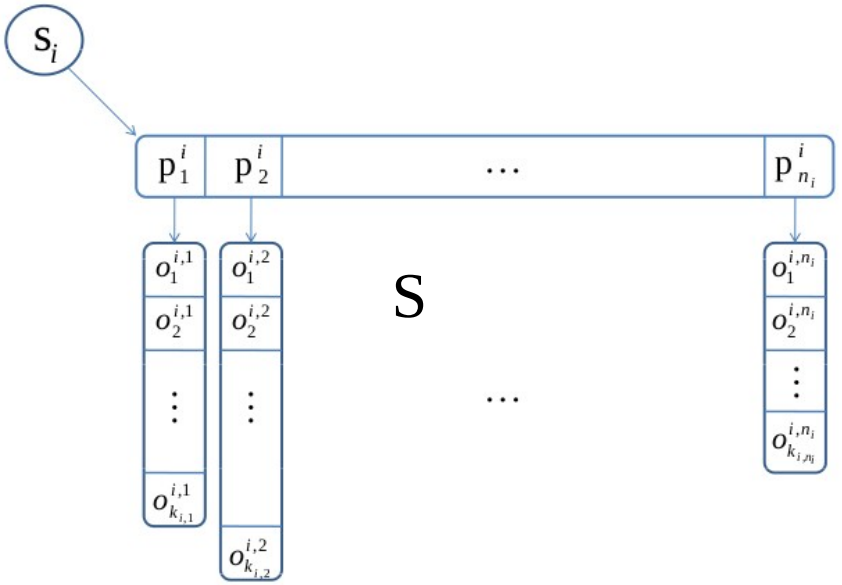
Index-based representation

- Covering indexes
  - RDF-3X, YAR2, 4store, Hexastore, ...
- RDF-3X (MPI, 2009)
  - Compressed clustered B+-tree
    - 6 indexes for each permutation of S, P and O
    - Triples are sorted in lexicographical order!
  - 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
- 3-level special index
  - Appropriate for some types of joins (sort-merge), set operations
  - 5-fold increase of DB size

SPO index entry



Columnar representation

- Vertical partitioning of RDF (Yale, 2009)
  - Daniel Abadi
  - Triple table is stored into n two-column tables

Type

ID1	BookType
ID2	CDType
ID3	BookType
ID4	DVDType
ID5	CDType
ID6	BookType

Author

ID1	"Fox, Joe"

Title

ID1	"XYZ"
ID2	"ABC"
ID3	"MNO"
ID4	"DEF"
ID5	"GHI"

Artist

ID2	"Orr, Tim"

Copyright

ID1	"2001"
ID2	"1985"
ID5	"1995"
ID6	"2004"

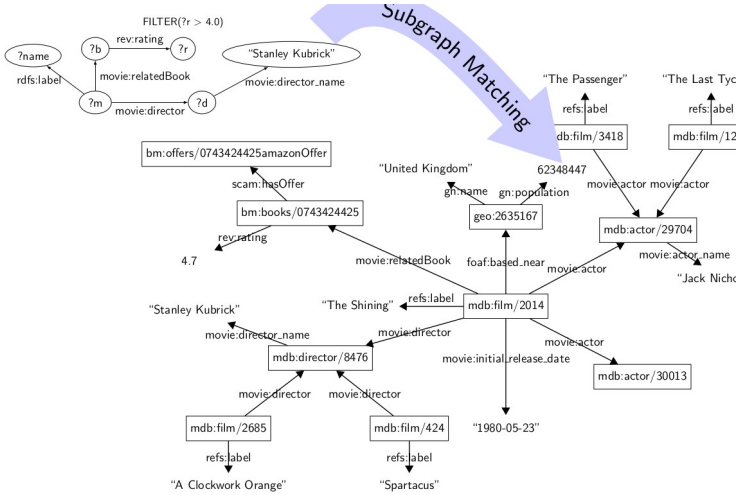
Language

ID2	"French"
ID3	"English"

- Advantages
  - Reduced I/O: reading only the needed properties
  - Optimizations: column compression, fixed-length tuples, direct access to sorted files.
  - Optimized column merge code (e.g. merge join)
  - Column-oriented query optimizer.
  - Materialized path expressions
- Disadvantages
  - Increased number of joins
  - Insertions incure higher overhead (multiple col.)

Graph-based representation

- Native graph representation
  - Nodes with associated adjacency lists
  - Subgraph matching (NP)
- Examples of systems
  - gStore, Neo4j, Trinity.RDF
- Example: 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 --> query graphs; 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



Data distribution

Outline

- Triple-store distribution
  - Hash horizontal partitioning
  - Locality-based horizontal partitioning
  - N-hop guarantee horizontal partitioning
  - Self-evolving partitioning
  - Semantic-aware partitioning

Horizontal partitioning in GDB

- Basic hash partitioning
  - Hash partition triples across multiple machines
    - Hash all triples using one of S, P, O, SP, SO, PO, SPO, ...
    - Some structure can be retained e.g. props of objects
  - Parallelize access to these machines
    - All servers return results at the same time
    - Synchronization and data transfer may be bottleneck
- Locality preserving partitioning
  - Triples are distributed in locality-based partitions
    - Queries are split into sub-queries
    - Sub-queries are executed on servers with 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
- The multilevel graph partitioning schemes
  - METIS, Karypis and Kumar (SIAM J. of Comp., 1998)

  1) Reduces the size of a graph by collapsing the vertices to obtain an abstract graph
  2) Graph is then min-cut partitioned and
  3) Finally un-coarsened to enumerate the members of the graph partitions.

  - Very efficient in practical applications, such as, finite element methods, linear programming, VLSI, and transportation.
-

Locality-based horizontal partitioning

- 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
- Originaly 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
    - METIS algorithm is used for graph partitioning
    - 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
  - 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 prunning of partitions

N-hop guarantee horizontal partitioning

- H-RDF-3X
  - Huang, Abadi, Ren: Scalable SPARQL Querying of Large RDF Graphs, VLDB, 2011
- Partitioning the data across nodes
  - Aim = accelerate query processing through locality optimizations
  - METIS used for min-cut vertex graph partitioning
    - rdf:type triples are removed before
  - Edge partitioning needed (not node partitioning)
- Triple placement
  - We have vertex graph partitioning
  - Simple conversion: use S part partition for complete triple
  - Triples on the borders are replicated

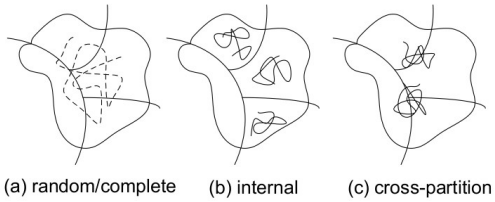
N-hop guarantee horizontal partitioning

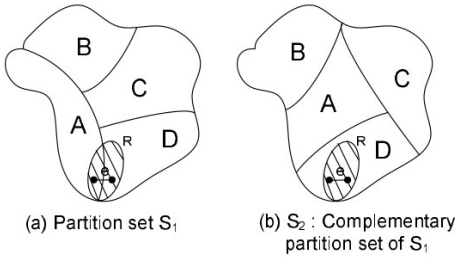
- More replication results less communication
- Controlled amount of replication
  - Directed n-hop guarantee
  - Start with 1-hop guarantee and then proceed to 2-hop guarantee, ...
  - Partitions are extended to conform n-hop guarantee

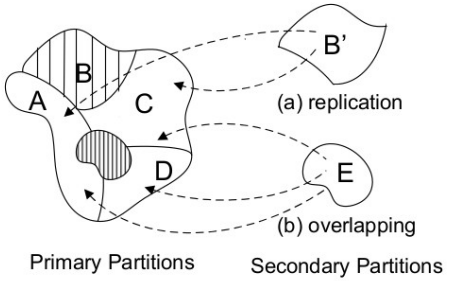
- Decomposing SPARQL queries into high performance fragments
  - Take advantage of how data is partitioned in a cluster.
    - Query fragments are parallelizable without communiction
  - How partitions are handled for querying is presented in the last part on Query processing

Self Evolving Partitioning

- SEDGE, Uni. California at Santa Barbara, 2012
  - S. Yang, X. Yan, B. Zong, and A. Khan. Towards Effective Partition Management for Large Graphs. SIGMOD, 2012.
  - Minimize inter-machine communication during graph query processing in multiple machines
  - Implemented on top of Pregel
- Primary partition set
  - Using normalized cut algorithm (METIS)
- Secondary partitions
  - Complementary primary partitions
  - Unbalanced query workload (focused)
  - Replicate partitions (load-balance) or
  - New overlapping partition (cover query)
- Dynamic partitioning
  - Cross-partition Hotspots







Semantic-Aware Partitioning

- big3store: distributed triple-store
  - Yahoo! Japan Research & University of Primorska, 2014-2019
  - Storing knowledge graphs (RDF+RDFS models)
  - Erlang programming environment
- Ordering the space of triple-patterns
  - Types of triple-patterns are ordered in a poset
  - Statistics of edge types (for a given KG)
  - Schema graph of a KG: types of graph edges

1) Cluster the data on the schema level
   - Compute skeleton graph (incl types of approp size)
     - Edge types that serve as fragments
   - Partition the skeleton graph! Use any partitioning method.
2) Distribute the extensions of the schema partitions

Query processing

Graph algebra

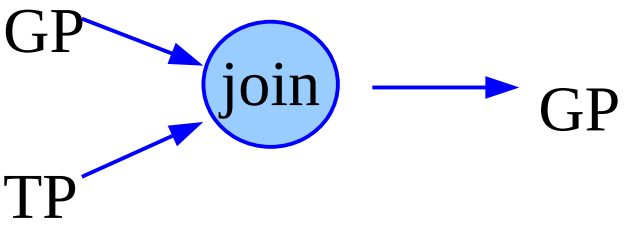
- Operations
  - Triple-pattern
  - Select
  - Project
  - Join
  - Union, Intersect, Difference
  - Leftjoin
- Algebra of sets of graphs
- Sets of graphs are input and output of operations
  - Triple is a very simple graph
  - Graph is a set of triples

Logical algebra

- Triple-pattern is an access method
  - tp_1 = (?x,p,o), tp_2 = (?x,p,?y), ...
  - tp_1 retrieves all triples with given P and O
- Triple pattern syntax
  - TP ::= (S | V,P | V,O | V)

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
  - GP ::= join(GP,GP)
  - Second argument is TP in left-deep trees



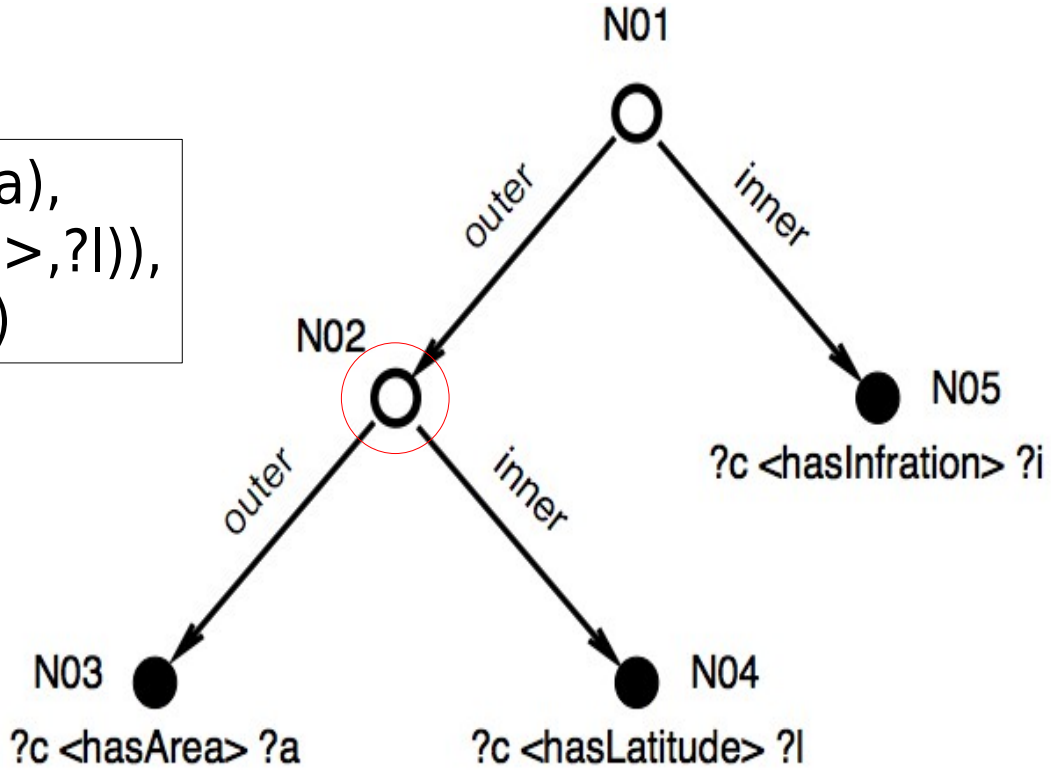
Logical algebra

Triple-pattern

```
tp(?c,<hasArea>,?a)
```

Operation join

```
join( join( tp(?c,<hasArea>,?a),
            tp(?c,<hasLatitude>,?l)),
      tp(?c,<hasInflation>,?i))
```

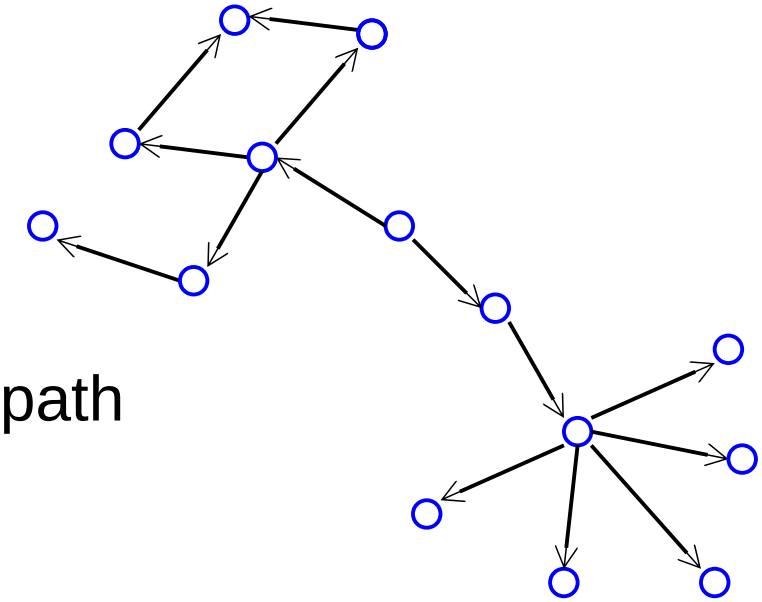


Physical operations

- Access method (AM)
  - Triple-pattern operation
  - Includes select and project operations
  - Methods: File scan (DFS), Key-value store, B+ index, Custom index
- Join, Leftjoin
  - Logical join operation
  - Includes select and project operations
  - Algorithms: Index NL join, merge-join, hash join, main memory joins
- Union, intersect and difference
  - Retain the schema of parameters

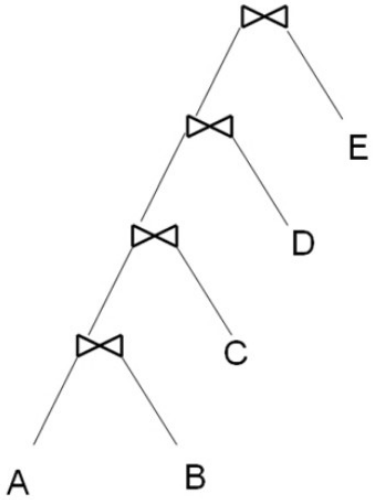
Query structures

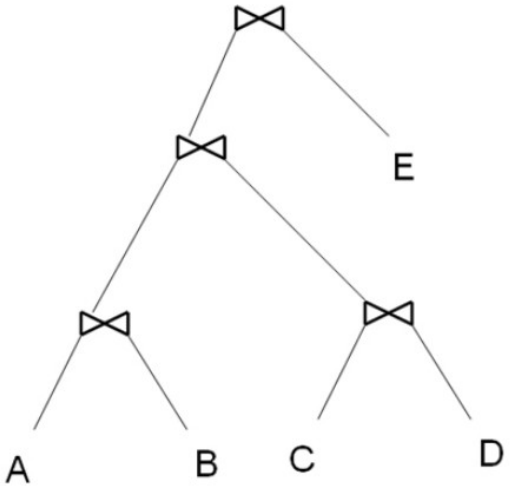
- Star queries
  - Objects and their properies
  - Similar to relations
  - Often units of optimization
  - Often a target to process locally
- Path queries
  - Sequence of joins that form the path
  - Often interconnect star queries
  - Shortest path queries
- Large search space
  - $O(n \times 2^n)$ star queries, $O(3^n)$ path queries
- Cost-based static optimization
  - For both cases



Query structures

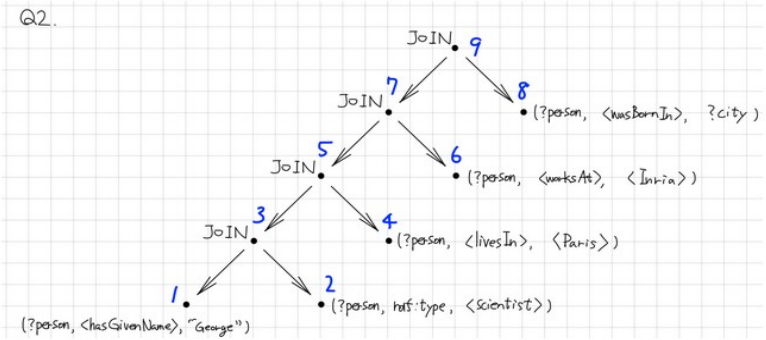
- Left-deep trees
  - Pipelined parallelism
  - Static query optimization (pipeline)
  - Dynamic (greedy) optimization possible
  - Star and path queries
- Bushy trees
  - More opportunities for parallel execution
  - Query fragments
    - Parts of QT are fragments
    - Composed of star queries?

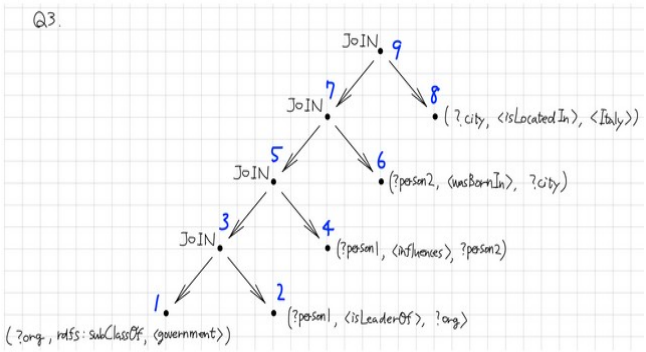




Graph patterns

- Basic graph patterns
  - Set of triple-patterns linked by joins
- Graph-patterns are units of optimization
  - Optimization can be based on dynamic programming
  - Bottom-up computation of execution plans
- Graph-patterns similar to SQL blocks
  - Operations select and project packed into joins and TPs
  - Operations select and project pushed-down to leafs of a query
  - Joins can now freely shift -> Join re-ordering





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

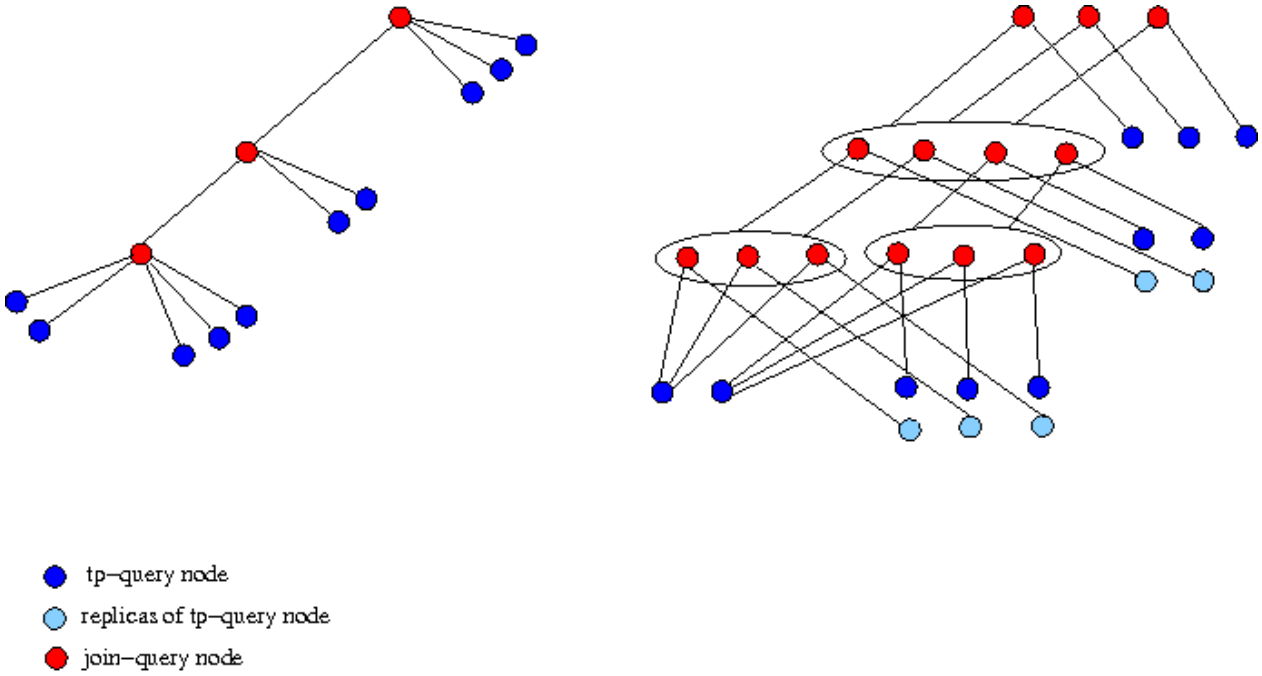
- Query optimization
  - Bottom-up dynamic programming algorithm
    - Classical approach! (similar to System R)
    - Keeps track of a set of the plans for interesting orders
  - Cost-based query optimization
    - Statistics (histograms) stored in aggregate indexes
    - Plan prunning based on cost estimation (heuristics)
  - Join re-ordering in bushy trees
    - Possible large number of joins
    - Star-shaped sub-queries are the primary focus
- Query evaluation
  - Extensive use of Merge join (all orderings are available)
  - Uses also a variant of hash join

Clusters of servers

- Usually shared-nothing servers
  - May also be shared disk or shared memory
- A federated database system
  - Transparently maps multiple autonomous database systems into a single federated database
- Parallel database systems
  - Custom implementation of all DBMS components
  - Storage manager: custom, KV store, ...
- Typically have coordinator nodes and data nodes
    - Not all nodes have the same functionality
- Examples:
  - H-RDF-3X, TriAD, WARP, EAGRE, Trinity.RDF

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

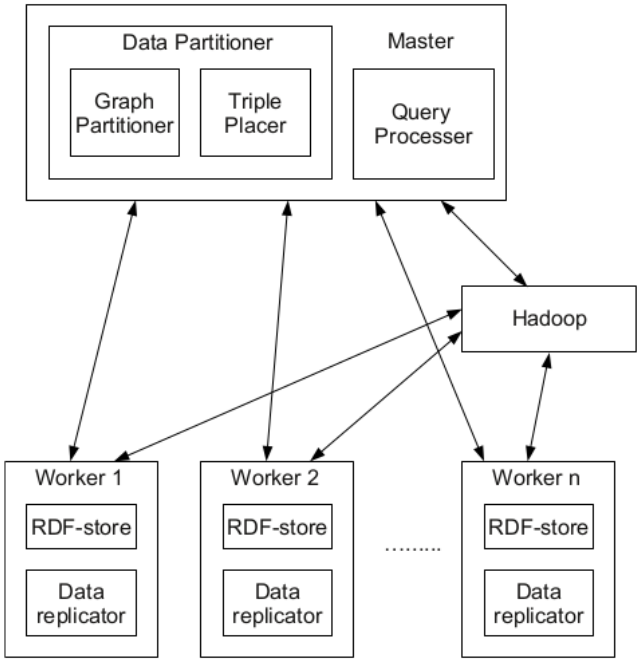
Partitioned parallelism

Pipelined parallelism

Independent parallelism

Example: H-RDF-3X, 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
- Close to distributed Ingres
  - See lecture on Distributed query processing



Example: H-RDF-3X, 2011

- Locality-based partitioning
  - METIS used for min-cut graph partitioning
  - Partitioning helps accelerate query processing
    - Through locality optimizations
  - Placement with n-hop replication
- Partitioning presented in section on graph DB partitioning

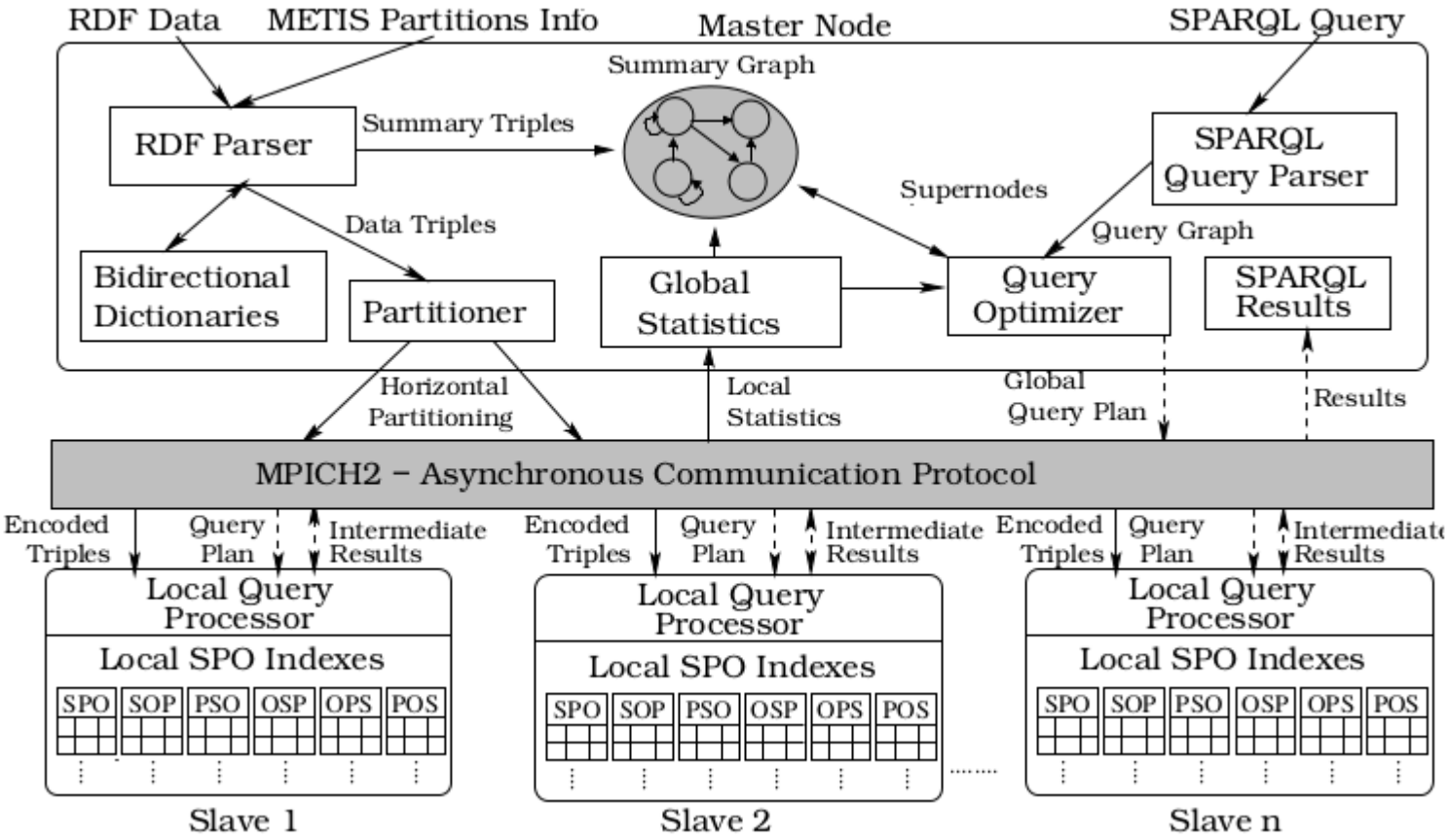
Example: H-RDF-3X, 2011

- Algorithm for automatically decomposing queries into parallelizable chunks
  - Concept of PWOC queries
    - PWOC=Parallelizable without communiction
    - 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: TriAD, 2014

- Federated centralized system
  - Extension of centralized RDF-3X to distributed environment
  - Based on asynchronous message passing
- System architecture
  - System R* style (see lecture on Distr.query proc.)
  - 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



Example: TriAD, 2014

- Construction of summary graph
  - Nodes are partitioned in disjunctive partitions (supernodes)
    - Graph partitioning with METIS
    - Edges with distinct labels are choosen 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
    - Bottom-up dynamic programming (determine join order)
      - Consider locality of the index structures at the slave nodes
      - Shipping cost of intermediate join results
      - Option to execute sibling paths of query plan in a multi-threaded fashion
    - Global query plan generated at the master is then communicated to all slaves
      - Multi-Threaded, asynchronous plan execution
      - Process the query against the data graph which is distributed
    - Determine locally best join order by using second DP optimizer
      - Precise statistics is used

Some research 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

Some research directions

- Abstracting the data graph
  - Construction of the abstract graph by
    - Data mining algorithms that group similarly structured sub-graphs
    - Employing graph partitioning for the construction of the abstract graphs
  - Abstract 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