Large Scale Data Management

Graph Stores, Neo4J

Silviu **Maniu**, LIG, Univ. Grenoble Alpes

Graph Data

Graphs

Graphs correspond to a natural organization of knowledge

They generalize

- Relations
- Trees (documents)
- Key-value pairs

Graph stores simplify / facilitate data representation

They do not simplify query evaluation (and may make it more complex)

Graphs

Data model: *G(V,E)* edges *E* between vertices *V*

Edges can be: directed vs. undirected, weighted vs. unweighted

Nodes/vertices can be:

- Unlabelled
- Having a single label or type
- Set of attribute-value pairs (property graphs)

Graph can have semantics (RDF, knowledge graphs)

Example Graphs

Social Networks

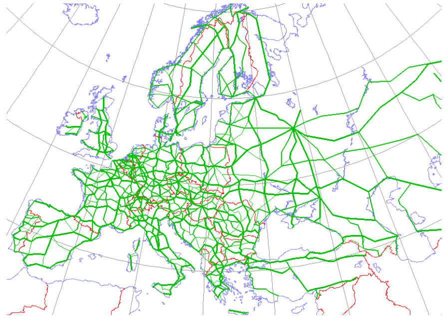
Edges: persons**, nodes:** relationships

by Michael Coghlan, CC BY-SA 2.0 via Flickr

Example Graphs

Road Networks

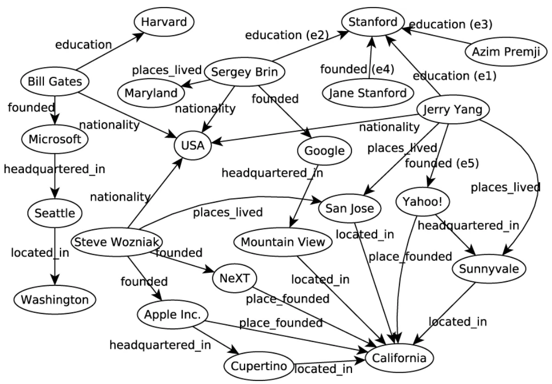
Edges: roads/connections**, nodes:** intersections/cities/etc.



Example Graphs

Knowledge Graphs / Semantic Data

Edges: subjects/objects**, nodes:** predicates



Graph Data Models

RDF

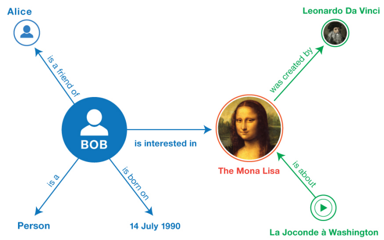
RDF: Data model for (originally) data interchange on the Web

Structured in triples: (subject, predicate, object)

Graph: triples form a directed, labeled/typed, graph

Nodes: are generally URIs allowing to address everything

Schemas: RDF schema (RDFS), OWL



Graph Data Models

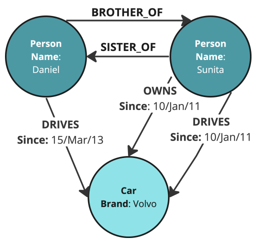
Property Graphs

Nodes: entities / objects / concepts

Edges: directed relationships between nodes

Properties: key-values pairs to nodes/edges (name: "Alice")

Labels: edge/node types



Graphs and Algorithms

Graphs and algorithms on graphs are important:

- **classic algorithms**: shortest paths, community detection, counting triangles, PageRank
- **data mining/machine learning on graphs**: collaborative filtering, belief propagation, mean-field algorithms, graph neural networks
- **querying:** matching patterns / paths on graphs

Graphs and Algorithms

Distributing Graph Computation

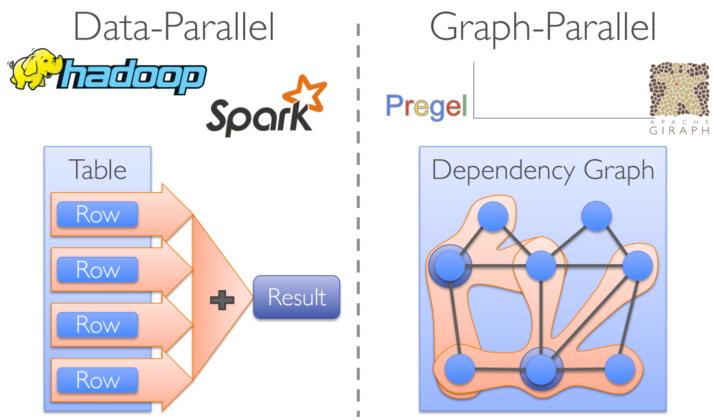
Challenges in efficiently processing graph algorithms:

- locality issues: since different vertices are needed
- little work per vertex: some vertices may not be used at all
- parallelism is not the same everywhere

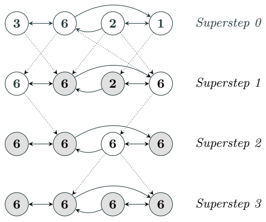
MapReduce, Spark are data parallel: very efficient for tabular data, aggregating data – not the case for graphs

Graph Parallel vs. Data Parallel

Graph Parallel: computation in each vertex



Graph Processing with Pregel



Implements a variant of the Bulk Synchronous Processing model

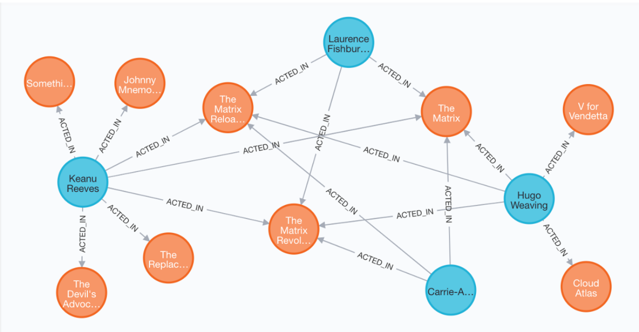
Computation Model

1. input: each vertex in the graph receives initial data
2. supersteps with global synchronization points
   - each vertex receives the same user-defined function to execute in a superstep
   - between supersteps, messages are sent between vertices (either via links, or globally)
   - in a superstep, each vertex can modify its state, and set-up a message to be sent to other vertices
   - vertices can vote to halt the computation; if they receive new messages they must re-activate
3. computation stops when all vertices vote to halt and are not reactivated

Graph Stores: Neo4J

Neo4J Model
Property Graphs

Neo4J uses a property graph data model (directed, labeled, key-value properties)



Neo4J Model
Data Manipulation Language

Data manipulation language (CRUD): **Cypher**, used to describe data and patterns to be matched

Node descriptions in Cypher:

`()` - empty anonymous node

`(matrix)` - node whose identifier is matrix.

`(:Movie)` - node of type Movie

`(matrix:Movie)` - node whose ID is matrix and type Movie

`(matrix:Movie {title: "The Matrix"})`
- node with an attribute

`(matrix:Movie {title: "The Matrix", released: 1997})`
- node with two attributes

Neo4J Model

Data Manipulation Language

Relationship descriptions in Cypher:

`--` (undirected) vs. `-->` or `<--` (directed)

Sample relationship descriptions:

```
-->
-[role]->        - relationship ID
-[:ACTED_IN]->     - relationship type
-[role:ACTED_IN]->
-[role:ACTED_IN {roles: ["Neo"]}]->
```

–relationship with attributes

Neo4J Architecture

Database Organisation

On-disk database in files:

- `nodestore` – node related data
- `relationship` – relationship related data
- `property` – properties (key/value)
- `label` - (for indexes) index related label data

Schema-less: fixed record size for quick random access (via offsets), stored on disk but cached in memory

Optimized for graph traversal = linked list, every node and relationship has pointers to the first in the chain

Neo4J Architecture

Transaction Control, Clustering

Transaction control is ACID when one server

- Write-ahead logs (WAL) for recovery - records the transaction log

Distribution/clustering, servers dealing with one or more databases:

- Primary mode: for safety and transaction control
- Secondary mode: for scalability and read performance

When clustering

- **Causal consistency**: each client writing to a primary server is guaranteed to read the same data from a secondary server

Data Manipulation with Cypher

Patterns combine node and relationship descriptors:

```
(keanu:Person:Actor {name: "Keanu Reeves"} )
-[role:ACTED_IN {roles: ["Neo"] } ]->
(matrix:Movie {title: "The Matrix"} )
```

Data creation:

```
CREATE (a:Person { name:"Tom Hanks", born:1956 })
-[r:ACTED_IN { roles: ["Forrest"]}]-> (m:Movie
{ title:”Forrest Gump",released:1994 })

CREATE (d:Person { name:"Robert Zemeckis",
born:1951 }) -[:DIRECTED]->(m)
```

Querying with Cypher

MATCH pattern RETURN matched variables

```
MATCH (p:Person { name:"Tom Hanks" })
              -[r:ACTED_IN]->(m:Movie)
RETURN m.title, r.roles
```

Successive match-create-return steps can be used to update the data:

```
MATCH (p:Person { name:"Tom Hanks" })
CREATE (m:Movie { title:"Cloud
Atlas",released:2012 })
CREATE (p)-[r:ACTED_IN { roles: ['Zachry']}]->(m)
RETURN p,r,m
```

Querying with Cypher

Inserting data only if it didn't exist:

```
MERGE (m:Movie { title:"Cloud Atlas" })
  // create or check the existence of movie node m
ON CREATE SET m.released = 2012
  // if we had to create it, set the release year

RETURN m
```

Insert relationship only if it did not exist:

```
MATCH (m:Movie { title:"Cloud Atlas" })
MATCH (p:Person { name:"Tom Hanks" })
MERGE (p)-[r:ACTED_IN]->(m)
ON CREATE SET r.roles =['Zachry']

RETURN p,r,m
```

Querying with Cypher

Returning Results

```
MATCH (a { name: "A" })-[r]->(b)

RETURN * //all results, all properties of a,r,b


MATCH (n)
RETURN n.age // returns null if no age


MATCH (a { name: "A" })

RETURN a.age > 30, "I'm a literal”,(a)—>() //edge
creation
```

Other Cypher Functionalities

- Boolean conditions:

```
MATCH (n)

WHERE n.name = 'Peter' XOR (n.age < 30 AND n.name = "Tobias") OR
NOT (n.name = "Tobias" OR n.name="Peter")

RETURN n
```

- Optional matching: somewhat like outer joins, replace missing with `NULL`
- Returned data can be: ordered, truncated, aggregated
- Unwind: unfolds a collection into a set `UNWIND[1,2,3] AS x RETURN x // three results`
- Indexes: `CREATE INDEX ON :Person(name)`
- `EXPLAIN` to get the query plan, `PROFILE` to measure the effort

Neo4J Architecture

Query Perfomance via Indexes

Search performance (automatically used if present):

- Range: default, B+ trees
- Text and Point: for string and coordinate operations
- Lookup: only for node label and relationship type (activated by default)

Semantic indexes (**not** automatically used):

- Full-text: search for content, via Apache Lucene
- Vector: similarity search via embeddings

Summary

Query Perfomance via Indexes

- Very convenient data model, natural representation
- Typically no strict schema
- No standard query language (effort to extend Cypher to standard, GQL)
- Knowledge graphs are a particular case (RDF and SPARQL are standards)
- Other powerful tools around: distributed graph stores (Pregel, Spark GraphX)
    - Extra dimension: graph partitioning
    - Less effort on query language; in progress

To Read Further

Articles

- Malewicz et al., *Pregel: a system for large-scale graph processing*, SIGMOD 2010
- Rodriguez, Neubauer, *Constructions from Dots and Lines*, Bul. Am. Soc. Info. Sci. Tech. 36(6), 2010
- Francis et al., *Cypher: An Evolving Query Language for Property Graphs*, SIGMOD 2018

Documentation

- Neo4J: https://neo4j.com/docs/, Cypher: https://neo4j.com/docs/cypher/