Graph Pattern Matching – Do We Have To Reinvent The Wheel?

Andrey Gubichev   Manuel Then

Technische Universität München

June 22, 2014

Linked Data Benchmark Council

Motivation

Welc et al. Graph Analytics – Do We Have To Reinvent The Wheel? GRADES'13

- Shortest path algorithms on graphs
- Native Graph DB vs Relational Store
- Relational Store outperforms Graph DB

Motivation

Welc et al. Graph Analytics – Do We Have To Reinvent The Wheel? GRADES'13

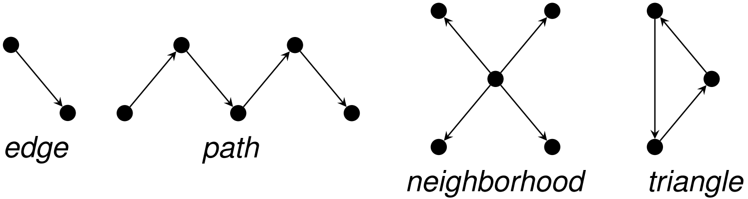
- Shortest path algorithms on graphs
- Native Graph DB vs Relational Store
- Relational Store outperforms Graph DB

This work: Graph Pattern Matching
Goals:

- Consider a Graph Pattern Matching workload
- Run it on systems from different domains (RDF, Property Graph, Relational)
- Get the best performance by modelling in a ”native” domain

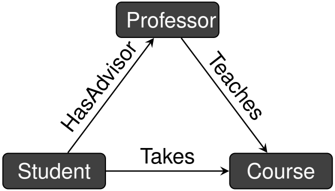
Graph Pattern Matching

- Graph $G = (V, E)$
- Query Pattern P – restrictions on nodes and edges
- Answer: subgraph of G that matches P (structural match, isomorphism)



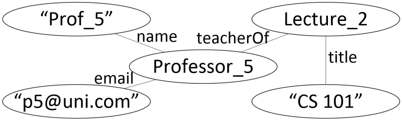
Testbed: LUBM benchmark

- Originally an RDF benchmark
- Data: universities, students, professors, lectures
- 14 SPARQL queries
- Queries: basic graph pattern matching



- Get rid of reasoning:
  - Re-write the queries
  - Add the inferred facts to the dataset

LUBM dataset in three different data models



(a) RDF, SPARQL

Professors

id	*name*	*email*
Professor_5	Prof_5	p5@uni.com

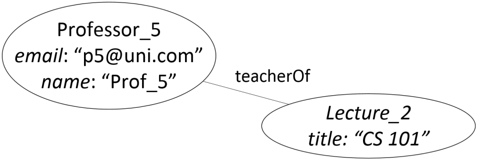
teacherOf

id_prof	*Id_lecture*
Professor_5	Lecture_2

Lectures

id	*title*
Lecture_2	CS 101

(b) Relational, SQL



(c) Property Graph, Cypher & native API

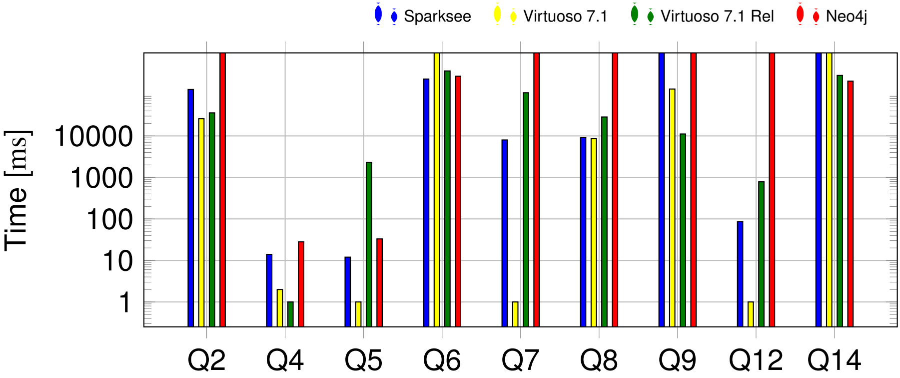
Systems & Datasets

- RDF: Virtuoso 6 (Row store), Virtuoso 7 (Column store), TripleRush
- Relational: Virtuoso 7 (Column store)
- Property Graph: Neo4j 2.0.1, Sparksee 5.0.0

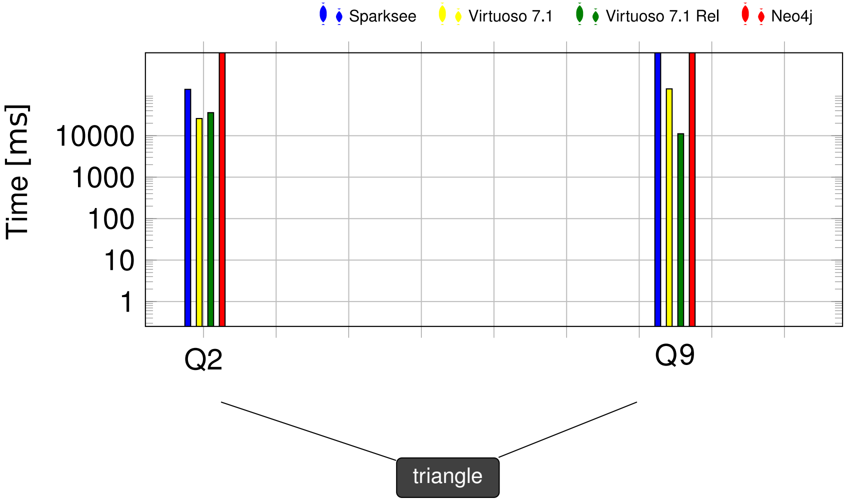
Datasets:

- LUBM-50: ca. 7 Million triples
- LUBM-8000: ca. 1 Billion triples

Results

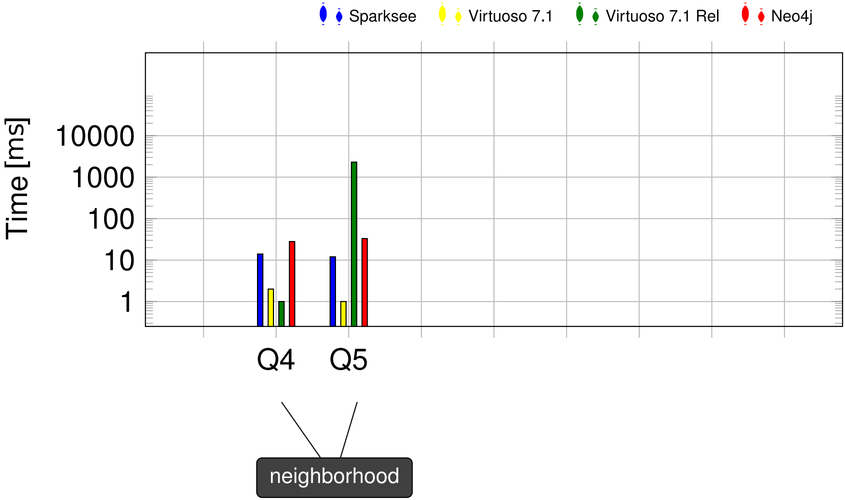

Sparksee
Virtuoso 7.1
Virtuoso 7.1 Rel
Neo4j
Time [ms]
10000
1000
100
10
1
Q2
Q4
Q5
Q6
Q7
Q8
Q9
Q12
Q14

Results



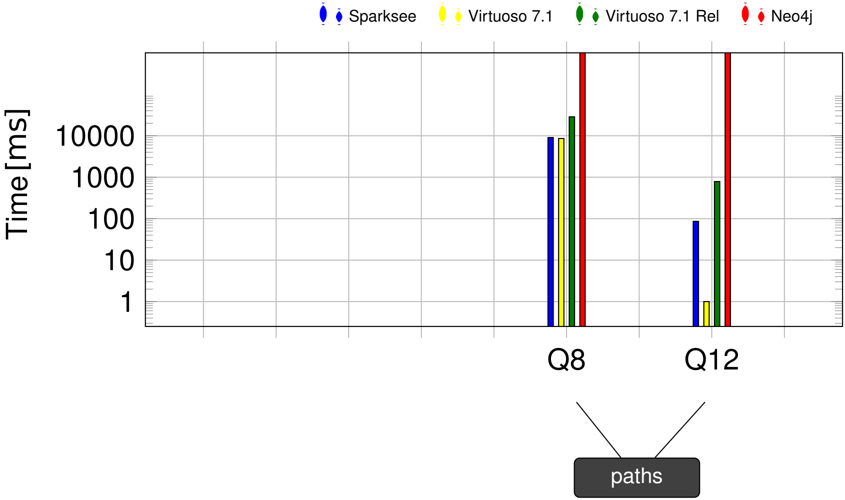
Sparksee
Virtuoso 7.1
Virtuoso 7.1 Rel
Neo4j
Time [ms]
10000
1000
100
10
1
Q2
Q9
triangle

Results

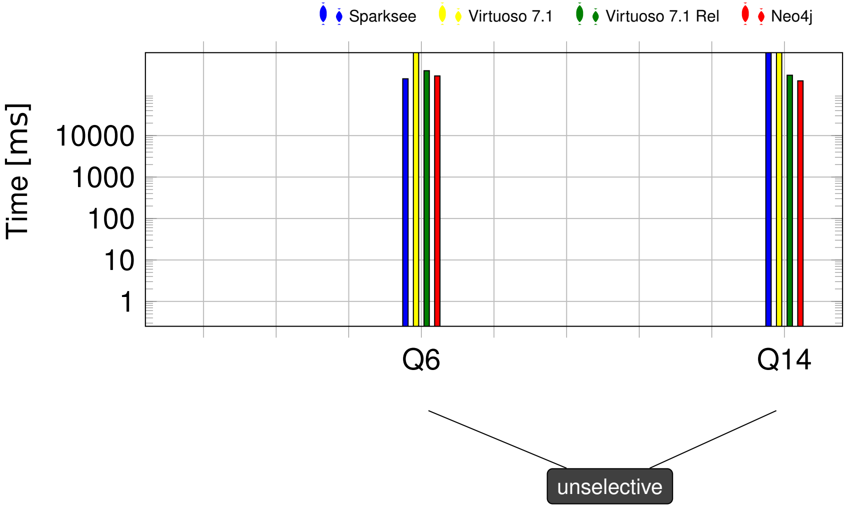


Sparksee
Virtuoso 7.1
Virtuoso 7.1 Rel
Neo4j
Time [ms]
10000
1000
100
10
1
Q4
Q5
neighborhood

Results


Sparksee
Virtuoso 7.1
Virtuoso 7.1 Rel
Neo4j
Time [ms]
10000
1000
100
10
1
Q8
Q12
paths

Results


Sparksee
Virtuoso 7.1
Virtuoso 7.1 Rel
Neo4j
Time [ms]
10000
1000
100
10
1
Q6
Q14
unselective

Lessons learnt (per system)

- **Sparksee**:
  - API-only system
  - Application developer has to figure out the execution plan
  - Even then the performance is far from optimal
- **Neo4j**:
  - Declarative query language, but no query optimizer (as of 2.0)
  - Times out when the graph pattern does not have a fixed starting point
  - Does not scale to large datasets

Lessons learnt (per system)

- **TripleRush**:
  - fast on small datasets
  - too high memory consumption for the larger dataset
- **Virtuoso**:
  - consistently good performance
  - column store outperforms row store
  - RDF version outperforms relational

Lessons learnt (per query type)

- Triangle matching challenging for all systems
- Fixed path queries efficient except for Neo4j
- Simple neighborhood matching is efficient
- Voluminous results problematic for all systems