LHD: Optimising Linked Data Query Processing Using Parallelisation

Xin Wang, Thanassis Tiropanis, Hugh C. Davis

Electronics and Computer Science

University of Southampton

Motivations

- High growth rate of Linked Data demands faster query engines.
- Parallelisation is a promising technique and has not been explored much in Linked Data query processing.
- The differences between DBMS and Linked Data leads to unique challenges and it's not straightforward to apply parallelization in Linked Data queries.

LHD: the parallel SPARQL engine

- LHD is a distributed SPARQL engine natively built on a parallel structure.
- Rather than the technical details described in our work, we'd be glad to see that LHD gives initial experiences for adopting parallelization in Linked Data queries, and **most importantly, reveals relevant open issues.**

Design issues

- **Responding time** estimation
- **Balance** between **effectiveness** and **efficiency** of query optimization
- Network connection is **dynamic** and has **limited capacity**

Components of LHD

Optimiser

- Responding time cost model
- Dynamic programming + Heuristics

Query plans

Query plan executor (logical execution)

- Adaptive and parallel infrastructure
- Data-driven model

Tasks

Traffic controller (physical execution)

- Traffic-jam proof

Responding time estimation

- Cardinality-based estimation

$$cost(q \bowtie p) = max(cost(q), cost(p))$$

$$cost(q \bowtie_B t) = cost(q) + cost(binding(q), t)$$

$$cost(t) = r_{tq} + card(t) \cdot r_{tt}$$

$$cost(binding(q), t) = card(q) \cdot r_{tq} + card(q \bowtie t) \cdot r_{tt}$$

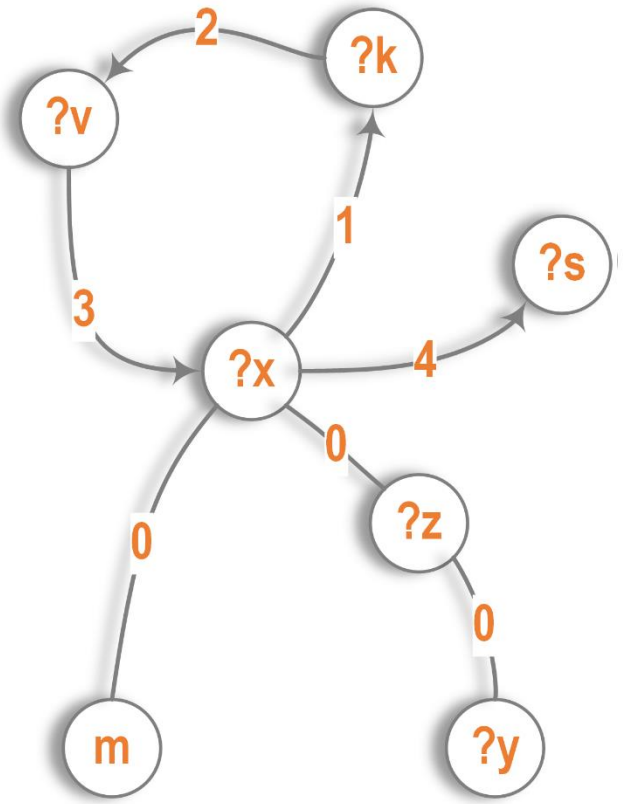
Optimisation algorithm

- To get a parallel query plan we firstly generate a sequential plan and parallelise it.
- Decouple generation of join relationship (or the join tree) and parallel execution order.

1. Generate a sequential query plan using dynamic programming
   a) Triple patterns that have a concrete node are always execute in parallel before others.
2. Decide the parallel execution order of a sequential plan.
   a) A triple pattern is executed as soon as its dependent bindings are ready.

Query execution (logical execution)

- Traverse a query plan and submits query tasks to traffic controller accordingly.



Traffic control (physical execution)

- For each data source separately maintain a certain number of query threads – traffic-jam proof.
- Query execution invokes query tasks rather than physical threads.
- Simplify traffic control.

A few open issues

1. Exhaustive search always give true optimal query plans, **if**, cost models are accurate to a certain extent. Are existing cost models (to be precise, cardinality estimation) meet the requirement?
2. To produce an accurate estimation requires certain detailed statistics, how hard is it to obtain detailed statistics from Linked Data cloud?
3. Static optimisation (producing query plans before execution) or dynamic optimization (producing query plans during execution)?
4. Co-reference (owl:sameAs)?