TriAD: A Distributed Shared-Nothing RDF Engine based on Asynchronous Message Passing

Sairam Gurajada (MPII), Stephan Seufert (MPII), Iris Miliaraki (Yahoo), Martin Theobald (UAntwerp)

**Motivation**

RDF is ubiquitous...
- Many organizations now support and publish RDF data
  Eg. DBpedia (400M), YAGO2 (130M), Freebase,...

**How to scale and yet be efficient?**
- in managing and querying RDF data
  **Our Approach: TriAD RDF store**
- Scalability – main-memory backed distributed setting
- Efficiency – 1. Asynchronous query processing
  2. Join-ahead pruning

**Problems with Current Approaches**

**Problem 1: Synchronous processing of joins**
Hadoop-based systems process joins using iterative & synchronous MapReduce jobs

**Problem 2: Pruning dangling triples**
- Dangling triples are the triples that appear in intermediate relations but not part of final joins
- Approaches: Graph exploration, Sideways Information Passing (SIP)
- Graph exploration – works well for selective queries
  SIP – needs synchronization among join operators

**Our Approach (in a nut shell)**

- Summary index (via summary bindings) is used for pruning dangling triples (Join-ahead pruning)
- Asynchronous distributed query execution over RDF index

**TriAD Architecture**

**Graph Summarization & Query Processing Workflow**

**Stage 1: Join-ahead pruning (of dangling triples)**

**Stage 2: Distributed & asynchronous query execution**

**Evaluation**

- **Query Performance**
- **Scalability**
- **Summary sizes**

Contact: gurajada@mpi-inf.mpg.de

**Graph summarization**

\[
C_{Q,n} := c_s + c_p,n \quad \frac{d |V_s|}{E_D} + \lambda \frac{c_D}{d}
\]

Optimal summary size

\[
|V_s| := \sqrt{\frac{\lambda E_D}{d}}
\]

\(\lambda\) - tuning parameter

**Binding: Person: P1, P2, P4**
**City: P1, P2, P4**
**Prize: P2, P4**