# Fast and Concurrent RDF Queries with RDMA-Based Distributed Graph Exploration

JIAXIN SHI, YOUYANG YAO, RONG CHEN, HAIBO CHEN, FEIFEI LI

PRESENTED BY ANDREW XIA

APRIL 25, 2018

# Wukong

# Overview of Wukong

- Distributed in-memory RDF
- Low Latency, concurrent queries over large RDF datasets
- RDMA-friendly Graph Model and k/v Store
- RDMA-based full history pruning
- Combine data migration for low latency, high throughput via one-sided RDMA operations

# RDF (Resource Description Framework)

- (subject, predicate, object)
- Represent as a graph, subject & object are vertices, predicate is a directed edge
- SPARQL
- "select RD where GP"
  - GP is a set of triple patterns
  - RD is a result description

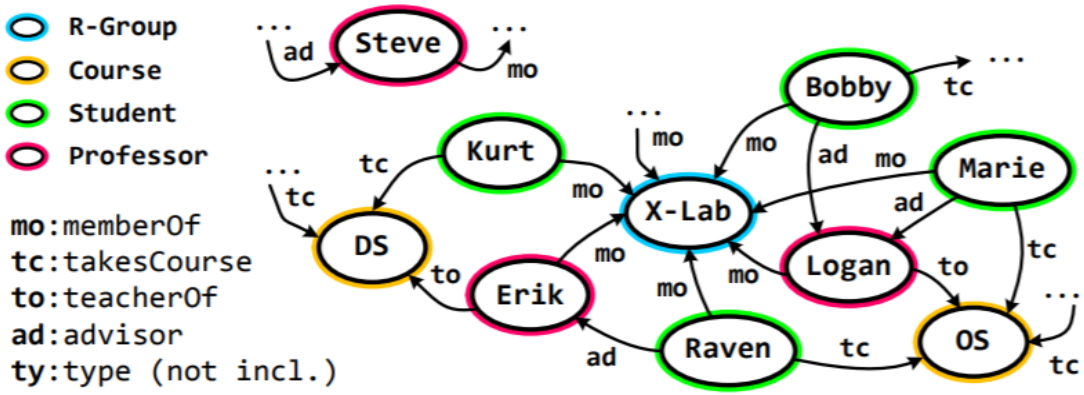



Fig. 1: An example RDF graph.

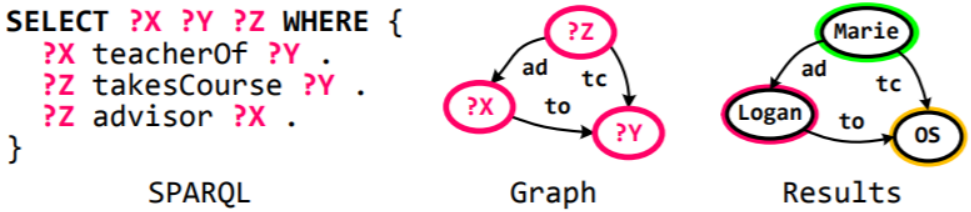



Fig. 3: A SPARQL query ($Q_2$) on sample RDF graph.

# Existing Solutions of RDF

- Triple Store & Triple Join
  - Store set of triples in relational database, leverage *join* operation
  - Costly join operations, redundant intermediate results
  - Can accelerate using redundant six SPO (subject, predicate, object) *permutation indexes* with more memory consumption
- Graph Store, Graph Exploration
  - Trinity.RDF stores RDF data in native graph model, with distributed in-memory k/v store, uses graph exploration
  - Final centralized join expensive

# RDMA (Remote Direct Memory Access)

- Bypass CPU when fetching data from other machines
- Cross-node memory access technique: low-latency, low CPU overhead
- Two-sided messages: SEND/RECV
- One-sided operations: READ, WRITE, fetch-and-add, compare-and-swap
- Latency of RDMA is relatively *insensitive* to payload sizes

# Wukong Architecture

- Cluster of servers, connected with RDMA features
- SPARQL Queries over RDF data
- Partition RDF graph into many shards across multiple machines
- Each Server has
  - Query engine
  - Graph store
- Query Processing
  - Partition query into chain of sub-queries across machines

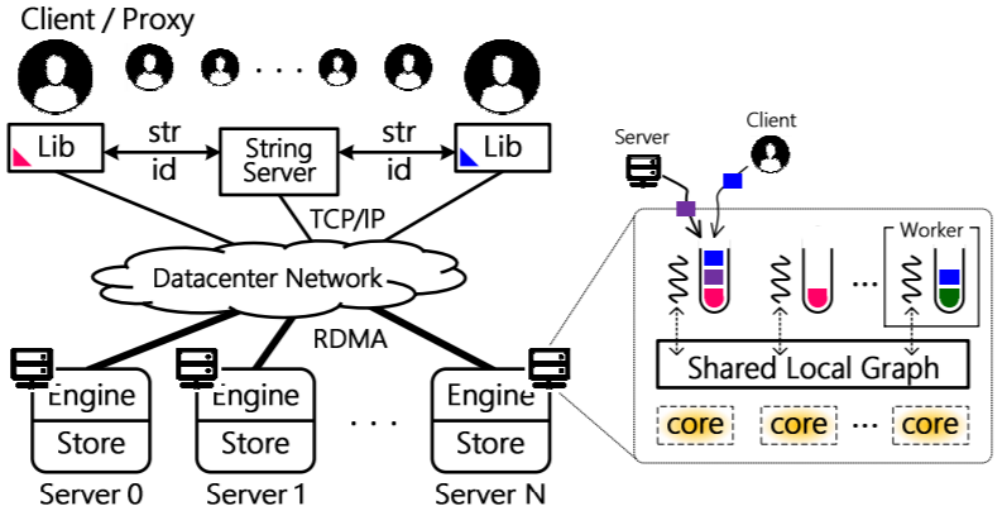



Fig. 5: The architecture overview of Wukong.

# Graph Model Index

- *normal vertex*: subject and objects
- *index vertex*
  - *predicate index*: maintain all subject and objects labeled with particular predicate
  - *type index*: group all subjects with same type

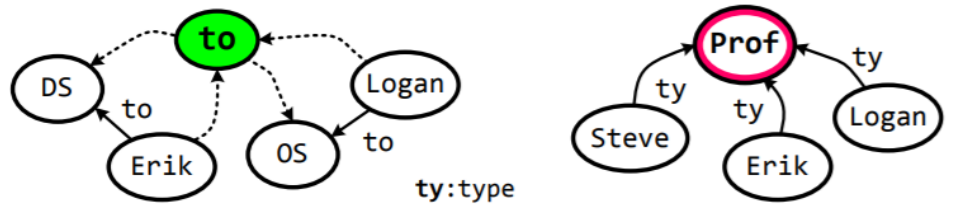



Fig. 6: Two types of index vertex of Wukong.

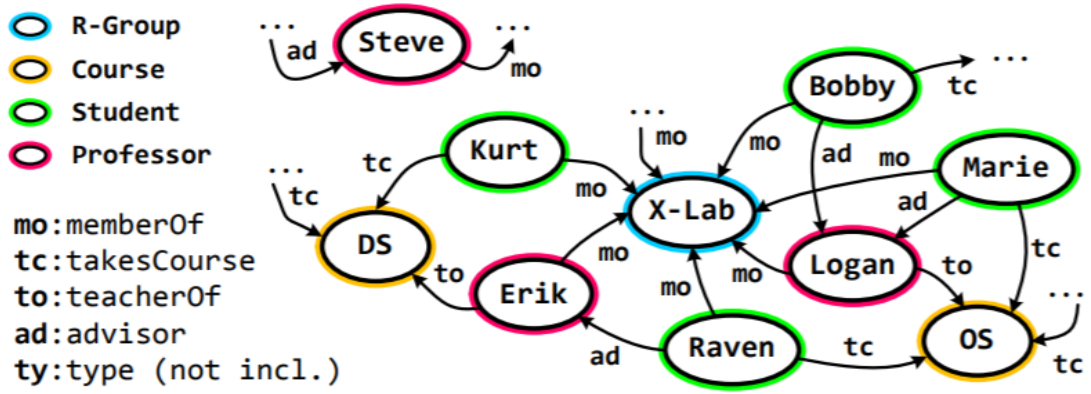



Fig. 1: An example RDF graph.

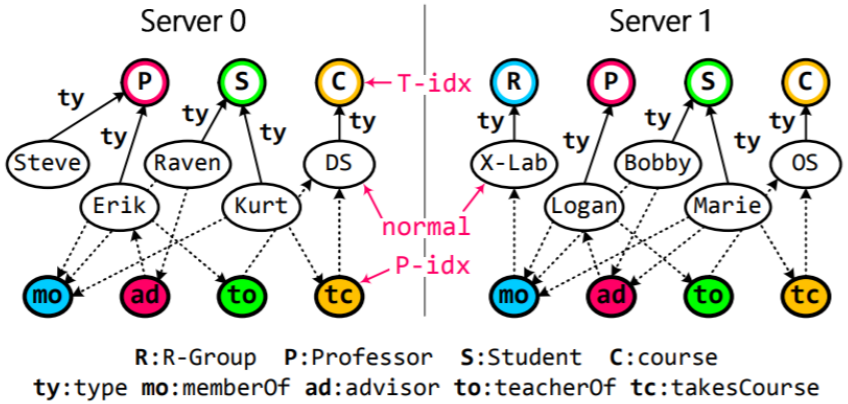



Fig. 7: A hybrid graph partitioning on two servers.

# Differentiated Graph Partitioning

- to support distributed queries, need to partition graph among multiple machines while maintaining access locality, parallelism
- each *normal* vertex randomly assigned (via hash) to machine with all edges
- each *index* vertex replicated among multiple machines, with edges linked to same machine

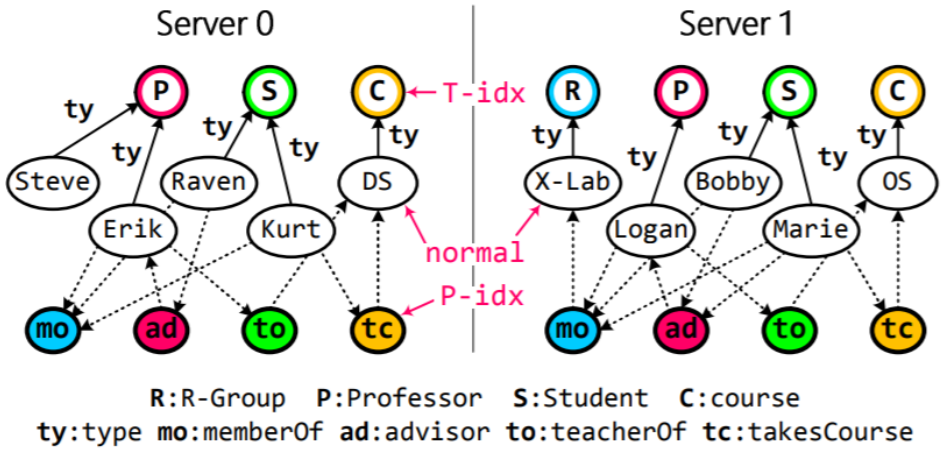



Fig. 7: A hybrid graph partitioning on two servers.

# RDMA-Friendly predicate based store

- Distributed k/v store
  - Key: (vertex ID, pred/type ID, edge direction)
  - Value: ( neighboring vertex ID or index ID)
- Observation: A SPARQL query will query neighboring vertices satisfying a predicate
- Special keywords
  - 0 vid: INDEX vertex
  - 0 p/tid: pred
  - 1 p/tid: type

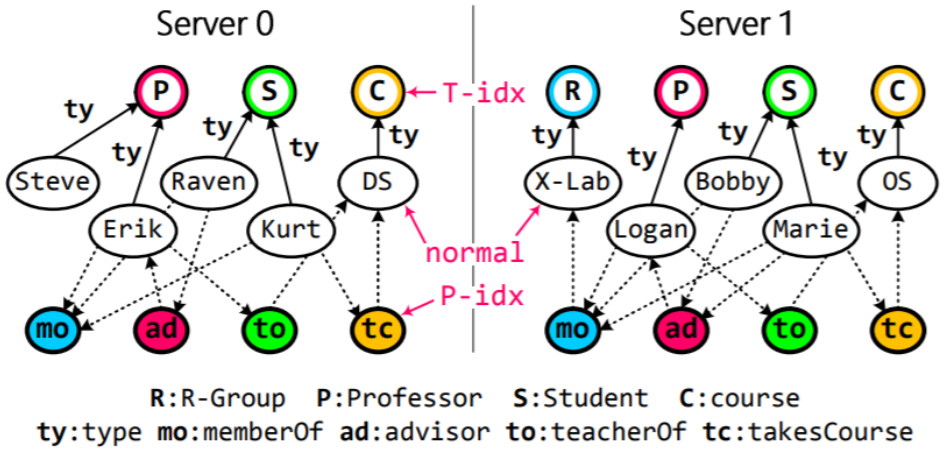



Fig. 7: A hybrid graph partitioning on two servers.

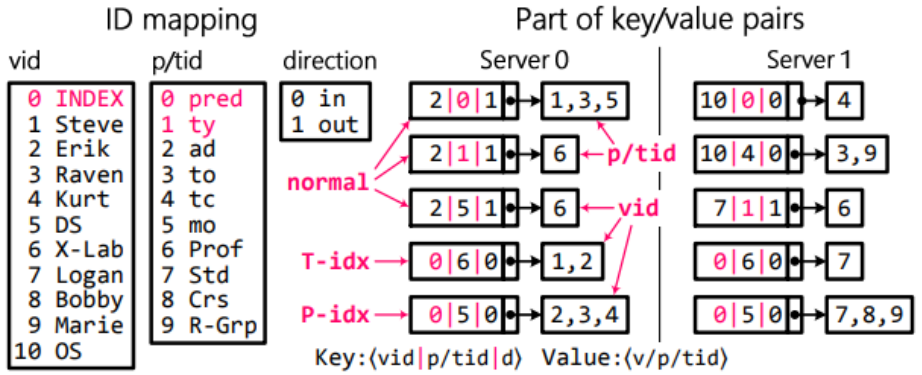
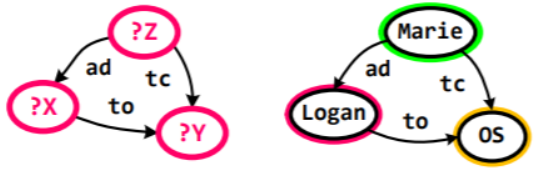



Fig. 8: The design of predicate-based key/value store.

# Query Processing

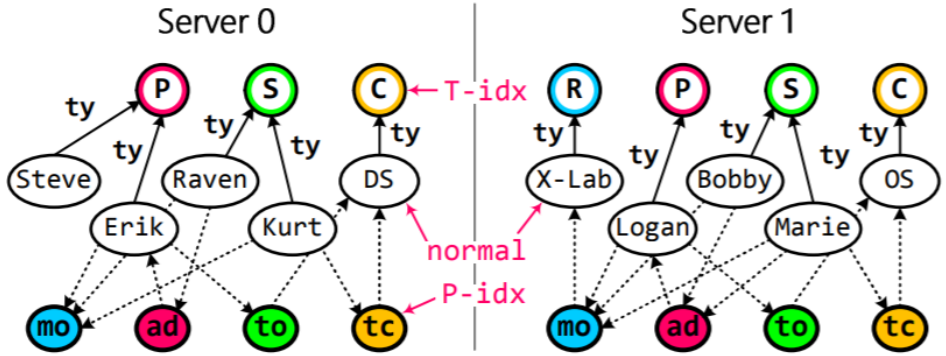
- Goal of query: find subject/object fitting subgraph pattern
- Wukong: Graph exploration in order defined by edge of subgraph
- If predicate known, subject / object are free variables, begin walk with predicate index
  - Continue searching graph satisfying triple patterns
- If predicate unknown, begin walk from constant vertex where p/tid=0

```
SELECT ?X ?Y ?Z WHERE {
  ?X teacherOf ?Y .
  ?Z takesCourse ?Y .
  ?Z advisor ?X .
}
```



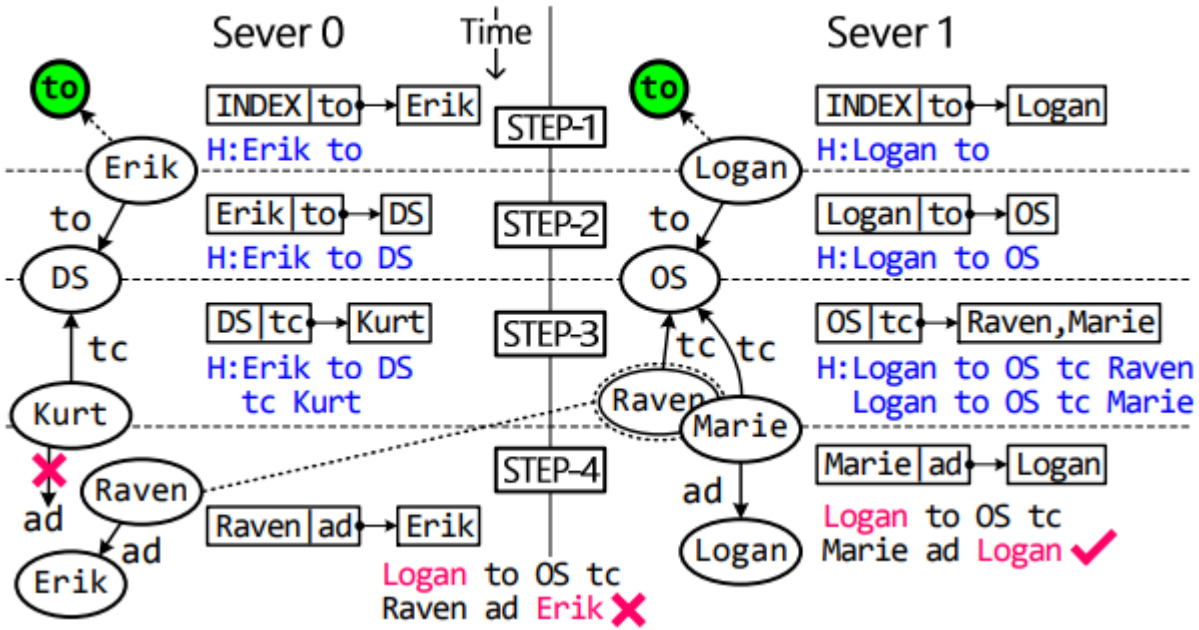

SPARQL Graph Results

Fig. 3: A SPARQL query ($Q_2$) on sample RDF graph.



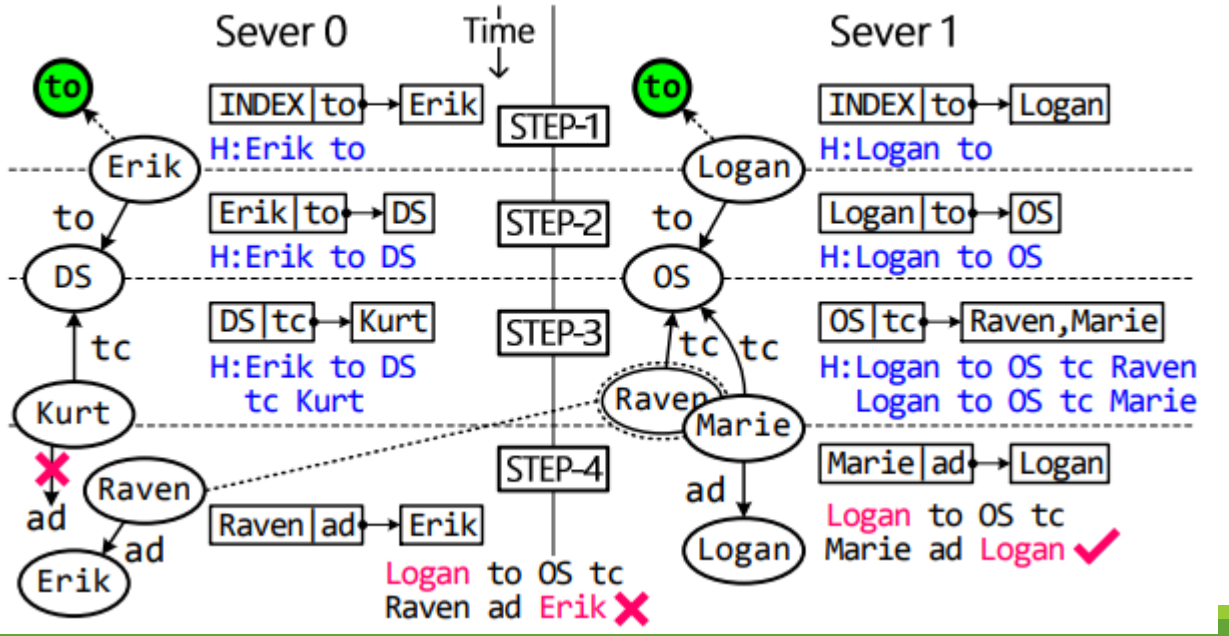

**R**:R-Group **P**:Professor **S**:Student **C**:course
**ty**:type **mo**:memberOf **ad**:advisor **to**:teacherOf **tc**:takesCourse

Fig. 7: A hybrid graph partitioning on two servers.



# Full History Pruning

- Certain tuples should be filtered out during graph exploration
- Wukong: pass full exploration history to next step across all machines
- Remove expensive cost of final join



# Migrating Execution or Data

- *In-place* execution
  - Bypass remote CPU via one-sided RDMA READ
- *Fork-Join* execution
  - Fetch many vertices
  - One-sided RDMA WRITE, push subquery with full history to remote machine
- Decide at runtime which execution mode
  - |N| RDMA operations
  - If |N| > 2*servers, fork-join
  - If |N| = # vertices, in-place

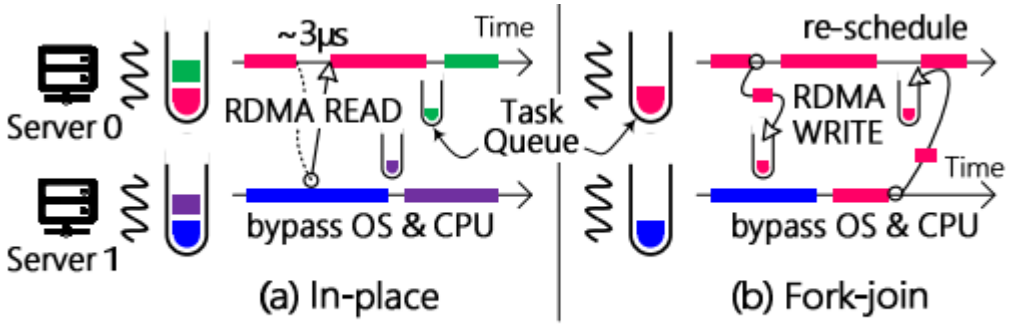
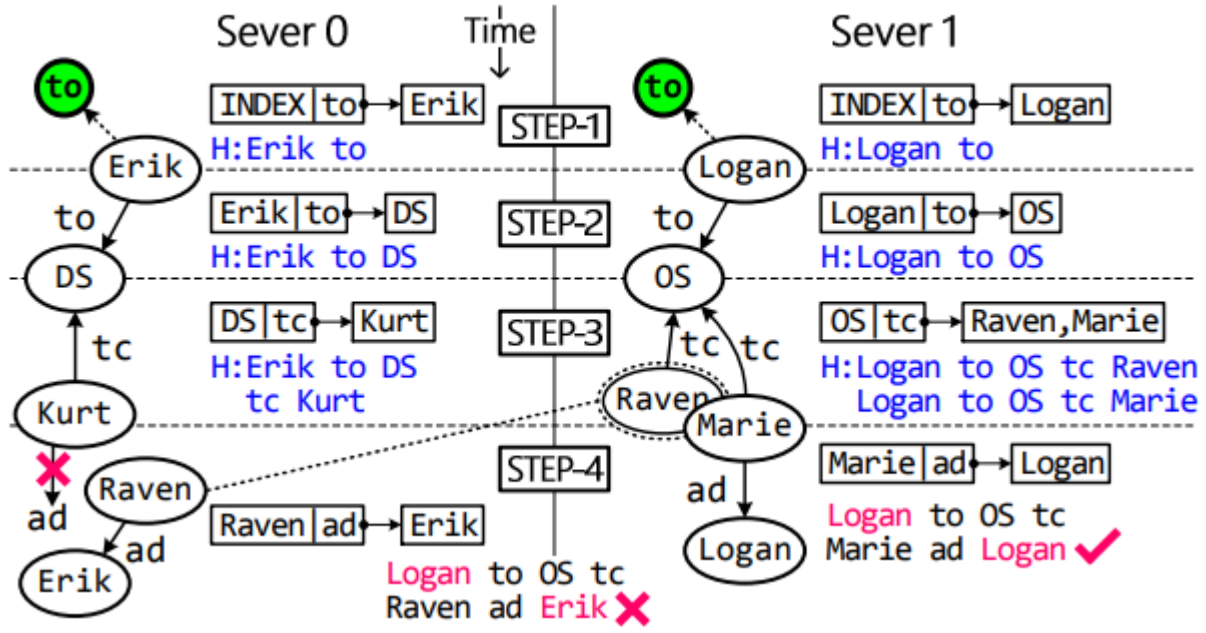



Fig. 10: A sample of (a) in-place and (b) fork-join execution.



# Concurrent Query Processing

- Motivation: what if some tasks for workers take much longer than other tasks?
- Work-stealing model
  - All tasks can be stolen from any worker thread queue
  - Isn't efficient for Wukong, because most tasks are small
- Worker-obliger work-stealing algorithm
  - Each worker keeps track of neighboring workers' task queue
  - If neighbors' tasks take too long, then worker will take some of neighbor's tasks

# Implementation

- Task Queue
  - Worker thread on each core, logical private task queue
  - One client queue, multiple server queues per server
- Launching Query
  - Start point of query either normal vertex or index vertex
  - Wukong will decide whether replicating index vertex queries is necessary depending on vertex degree
- Multi-Threading
  - Parallelize graph exploration via multiple threads processing parts of subgraph

# Evaluation

- Datasets
  - Two synthetic datasets, two real-life datasets
- Comparison
  - Centralized systems: RDF-3X, BitMat
  - Distributed systems: TriAD, Trinity.RDF, SHARD
- Large Queries: L1, L2, L3, L7
- Small Queries: L4, L5, L6

Table 1: A collection of real-life and synthetic datasets.

| Dataset | #Triples | #Subjects | #Objects | #Predicates |
|---|---|---|---|---|
| **LUBM**-10240 | 1,410 M | 222 M | 165 M | 17 |
| **WSDTS** | 109 M | 5.2 M | 9.8 M | 86 |
| **DBPSB** | 15 M | 0.3 M | 5.2 M | 14,128 |
| **YAGO2** | 190 M | 10.5 M | 54.0 M | 99 |

Table 2: The query performance (msec) on a single machine.

| **LUBM** 2560 | **Wukong** | **TriAD** | **TriAD-SG** (50K) | **RDF-3X** (mem) | **BitMat** (mem) |
|---|---|---|---|---|---|
| **L1** | 752 | 621 | 3,315 | 2.3E5 | abort |
| **L2** | 120 | 149 | 221 | 4,494 | 36,256 |
| **L3** | 306 | 316 | 3,101 | 3,675 | 752 |
| **L4** | 0.19 | 3.38 | 3.34 | 2.2 | 55,451 |
| **L5** | 0.11 | 2.34 | 1.36 | 1.0 | 52 |
| **L6** | 0.56 | 20.7 | 6.06 | 37.5 | 487 |
| **L7** | 671 | 2,176 | 2,753 | 9,927 | 19,323 |
| **Geo. M** | 15.7 | 72.3 | 108 | 441 | – |

Table 3: The query performance (msec) on a 6-node cluster.

| **LUBM** 10240 | **Wukong** | **TriAD** | **TriAD-SG** (200K) | **Trinity .RDF** | **SHARD** |
|---|---|---|---|---|---|
| **L1** | 516 | 2,110 | 1,422 | 12,648 | 19.7E6 |
| **L2** | 78 | 512 | 695 | 6,081 | 4.4E6 |
| **L3** | 203 | 1,252 | 1,225 | 8,735 | 12.9E6 |
| **L4** | 0.41 | 3.4 | 3.9 | 5 | 10.6E6 |
| **L5** | 0.17 | 3.1 | 4.5 | 4 | 4.2E6 |
| **L6** | 0.89 | 63 | 4.6 | 9 | 8.7E6 |
| **L7** | 464 | 10,055 | 11,572 | 31,214 | 12.0E6 |
| **Geo. M** | 16 | 190 | 141 | 450 | 9.1E6 |

# Evolving Graph Support

- Wukong can incrementally update graph with concurrent queries
- Low Overhead in latency, because of multi-threading

Table 4: The query latency (msec) of Wukong on evolving LUBM with 1 million triples/second ingestion rate.

| **LUBM**-10240 | **L1** | **L2** | **L3** | **L4** | **L5** | **L6** | **L7** |
|---|---|---|---|---|---|---|---|
| **Wukong** | 587 | 87 | 222 | 0.43 | 0.18 | 0.95 | 516 |
| **Overhead (%)** | 12.0 | 10.3 | 8.6 | 4.7 | 5.6 | 6.3 | 10.1 |

# Optimization Sources

- BASE: graph-exploration strategy with one-step pruning, via TCP/IP
- RDMA: one-sided RDMA operations
- FHP: full-history pruning
- IDX: add predicate/type index, differentiated graph partitioning
- PBS: predicate-based, finer-grained vertex decomposition
- DYN: in-place execution, switch between data migration and execution distribution

Table 5: The contribution of optimizations to query latency (msec) of Wukong. Optimizations are cumulative.

| LUBM 10240 | BASE | +RDMA | +FHP | +IDX | +PBS | +DYN |
|---|---|---|---|---|---|---|
| **L1** | 9,766 | 9,705 | 888 | 853 | 814 | 516 |
| **L2** | 2,272 | 2,161 | 1,559 | 84 | 79 | 78 |
| **L3** | 421 | 404 | 404 | 205 | 203 | 203 |
| **L4** | 1.49 | 0.79 | 0.78 | 0.78 | 0.56 | 0.41 |
| **L5** | 1.00 | 0.39 | 0.39 | 0.39 | 0.31 | 0.17 |
| **L6** | 3.84 | 1.40 | 1.37 | 1.37 | 1.17 | 0.89 |
| **L7** | 2,176 | 2,041 | 657 | 494 | 466 | 464 |
| **Geo. M** | 102.3 | 69.1 | 39.6 | 22.6 | 19.9 | 15.7 |

# Scalability

- Number of threads
- Number of machines
- Size of dataset
- Good practitioner of COST metric

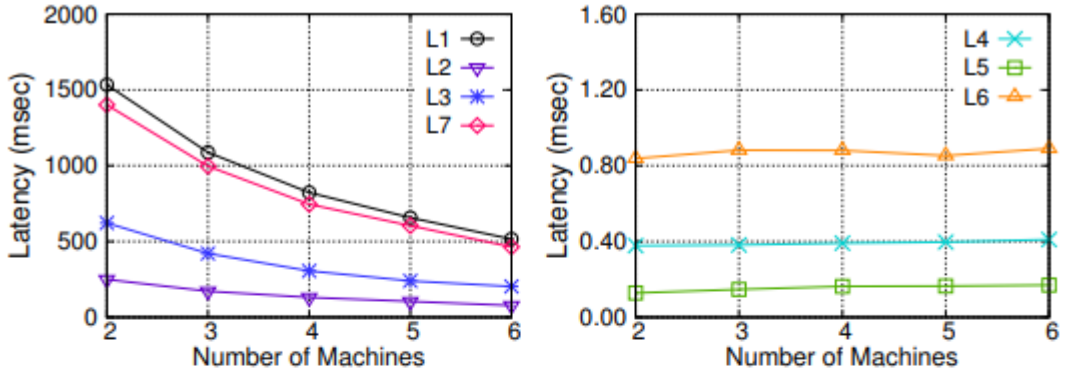



Fig. 15: The latency of queries in group (I) and (II) with the increase of machines on LUBM-10240.

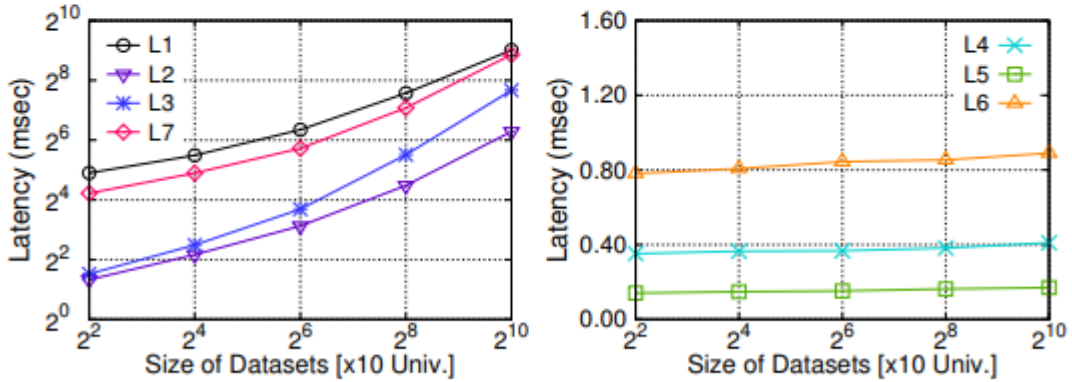



Fig. 16: The latency of queries in group (I) and (II) with the increase of LUBM datasets (40-10240).

# Memory Efficiency

- Triple stores (TriAD, RDF-3X, BitMAT) rely on six primary SPO permutation indexes for performance
  - However, high memory pressure
- Wukong: RDF data in graph form is more space efficient, only double triples for subjects and values
  - Currently k/v store hash table only has < 75% occupancy, can be improved

# Thanks!

# References

SHI, J., YAO, Y., CHEN, R., CHEN, H., AND LI, F. Fast and concurrent rdf queries with rdma-based distributed graph exploration. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16) (Savannah, GA, Nov. 2016), USENIX Association