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
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. 1: An example RDF graph.

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

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
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 {
  ?Z takesCourse ?Y .
}

Fig. 3: A SPARQL query (Q2) on sample RDF graph.

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-obligier 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.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Triples</th>
<th>#Subjects</th>
<th>#Objects</th>
<th>#Predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUBM-10240</td>
<td>1,410 M</td>
<td>222 M</td>
<td>165 M</td>
<td>17</td>
</tr>
<tr>
<td>WSDTS</td>
<td>109 M</td>
<td>5.2 M</td>
<td>9.8 M</td>
<td>86</td>
</tr>
<tr>
<td>DBPSB</td>
<td>15 M</td>
<td>0.3 M</td>
<td>5.2 M</td>
<td>14,128</td>
</tr>
<tr>
<td>YAGO2</td>
<td>190 M</td>
<td>10.5 M</td>
<td>54.0 M</td>
<td>99</td>
</tr>
</tbody>
</table>

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

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

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

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

<table>
<thead>
<tr>
<th></th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
<th>L4</th>
<th>L5</th>
<th>L6</th>
<th>L7</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUBM-10240</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wukong</td>
<td>587</td>
<td>87</td>
<td>222</td>
<td>0.43</td>
<td>0.18</td>
<td>0.95</td>
<td>516</td>
</tr>
<tr>
<td>Overhead (%)</td>
<td>12.0</td>
<td>10.3</td>
<td>8.6</td>
<td>4.7</td>
<td>5.6</td>
<td>6.3</td>
<td>10.1</td>
</tr>
</tbody>
</table>
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
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