Property Graphs for Industry Solution at IBM

Yinglong Xia
IBM T.J. Watson Research Center
November 14th, 2014
IBM Research

Network Science Team
Glance of IBM Research's Efforts on Graph Technology

- **Industry solution drives graph research**
  - DARPA Anomaly Detection at Multiple Scales (ADAMS) project for insider threat detecting espionage, sabotage, and internal fraud
  - DARPA Social media monitoring (SMISC) for Enterprise live monitoring, thread detection, info flow analysis, and social media analysis

- **Enhancement of IM systems**
  - DB2 RDF as a triple store
  - DB2 Graph to support property graph query

- **Emerging Hardware / Accelerator**
  - CAPI for coherent interface
  - Active store and hardware accelerators
Graph Capability in IBM System G – Native Store

- Native Store
  - Performance oriented
  - Highly efficient graph access
  - Scale up & out

Graph primitives

Graph traversal

**User defined property class**

**Graph Key-Value Store Interface**

**Graph Visualization Toolkit**

**Graph Analytics Toolkit**

**Graph Processing Interface**

**Shared Memory Graph Library**

**Distr. Memory Graph Library**

**Graphs FPGA/HMC**

**Serial Graph Library**

**Graph RDMA/PAMI**

**Network Science Team**
Graph Capability in IBM System G – ScaleGraph

- IBM X10 Programming Language
  - Distributed functional language
  - Aim at productivity
  - Adopt APGAS programming model
  - Run as native code or on Java
  - Open sourced

- Graph computing model
  - XPIregel model for distribution execution
  - SpVM model for graph analytics

- Performance optimization
  - Native MPI for comm.
  - Avoid serialization
**DB2Graph**

- Building property graph store over relational DB
  - Try to translate a gremlin query into a single SQL
  - Optimize graph update operations
  - Support batch loading of property graphs

---

**Shredding table**
- Without index: 46 - 230 ms per property query
- With index: 3 – 71 ms per property query

**DB2XML**
- Without index: 100 - 2000 ms per property query
- With index: 4 – 1000 ms per property query

**DB2JSON**
- Without index: 247 - 262 ms per property query
- With index: 2 – 50 ms per property query

---

**DB2Graph UDFs**

**DB2Graph Stored Procedure**

**Hybrid Schema**

---

© 2013 IBM Corporation
DB2RDF

- In DB2 LUW 10.1 we support
  - Java API’s for RDF application consumers.
  - HTTP based SPARQL query
- DB2RDF – A set of user tables within a database schema that stores the RDF data
  - Direct Primary: stores the triples and the graph they belong to indexed by subject.
  - Direct Secondary: stores the RDF objects that share the same subject and predicate within a RDF graph.
  - Reverse Primary: stores the triples and the graph they belong to indexed by object.
  - Reverse Secondary: stores the RDF subjects that share the same object and predicate within a RDF graph.

<table>
<thead>
<tr>
<th>Direct Primary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject</td>
</tr>
<tr>
<td>s1</td>
</tr>
<tr>
<td>s2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Direct Secondary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph</td>
</tr>
<tr>
<td>lid:abc</td>
</tr>
<tr>
<td>lid:abc</td>
</tr>
<tr>
<td>lid:cdf</td>
</tr>
<tr>
<td>lid:cdf</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reverse Primary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
</tr>
<tr>
<td>vA</td>
</tr>
<tr>
<td>vF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reverse Secondary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Graph Benchmark for Innovations in Hardware

- Graph for benchmarking H/W
  - Traditional processor/memory design is not friendly to graph
  - Scale up & out limitation
  - Demanded by novel H/W design

<table>
<thead>
<tr>
<th>Experiment Data Set</th>
<th>Vertex #</th>
<th>Edge #</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDBC Synthetic Social Graph</td>
<td>1K</td>
<td>29.7K</td>
</tr>
<tr>
<td>Twitter Graph</td>
<td>13M</td>
<td>60M</td>
</tr>
<tr>
<td>IBM Knowledge Repo</td>
<td>154K</td>
<td>1.72M</td>
</tr>
<tr>
<td>Watson Gene Graph</td>
<td>2M</td>
<td>12.2M</td>
</tr>
<tr>
<td>CA Road Network</td>
<td>1.9M</td>
<td>2.8M</td>
</tr>
</tbody>
</table>

Performance profiling and breakdown wrt various graph algorithms.
Thank you