A Highly Efficient Runtime and Graph Library for Large Scale Graph Analytics

Ilie Gabriel Tanase – Research Staff Member, IBM TJ Watson

Yinglong Xia, Yanbin Liu, Wei Tan, Jason Crawford, Ching-Yung Lin – IBM TJ Watson

Lifeng Nai – Georgia Tech
System G v1.0 Architecture

Visualization
- I2 3D Network
- Network Propagation
- Huge Network
- Geo Network Visualization
- Graphical Model

Analytics
- Communities
- Graph Search
- Network Info
- Bayesian
- Centralalities
- Graph Query
- Shortest Paths
- Latent Net
- Ego Net Features
- Graph Matching
- Graph Sampling
- Markov Networks

Middleware
- BigInsights
- Infosphere Streams (ISS)
- Shared Memory Graph Library
- Distr. Memory Graph Library
- Graph Accelerator
- Generic Graph Library
- Graph Communication

Graph Processing Interface

Database
- GBase (update, scan, operators, indexing)
- HBase
- DB2 RDF
- TinkerPop Compliant DBs
- DB2
- Native Store
- HDFS

Graph Data Interface (PAMI/RDMA)
The Spectrum of Open Source Graph Technology

- In memory:
  - Single machine:
    - Jung
    - Fulgora
    - Aurelius
  - Clusters:
    - Neo4J
      - Neo4j.org
    - OrientDB
      - NuvolaBase
    - RedisGraph
      - MIT
    - Titan
      - Aurelius
  - DEX
    - Sparsity Tech.
  - GraphLab
    - CMU
  - NetworX
    - NetworX.org

- Disk:
  - Single machine:
    - Giraph
      - Apache
      - Yahoo, FB, …
  - Clusters:
    - Faunus
      - Aurelius
Motivation and Requirements

- Flexible Graph Datastructure
  - In memory only, persistent, both
  - Directed, undirected, directed with predecessors
  - ACID properties

- Run well on IBM machines including X86, Power, Bluegene
  - Large memory, large number of cores
  - Clusters with Infiniband or specialized networks (RDMA)

- Commercial solution
IBM Parallel Programming Library

- C++
  - Object oriented design - inheritance
  - Generic using templates
- Datastructures – Graphs, Hash Tables, Arrays
- Large shared memory
  - Concurrency
- Distributed memory clusters
  - Messaging API based on active messages and RDMA
IBM PPL Graph class hierarchy

Graph<VertexProperty, EdgeProperty, Directness>
  - in memory only
  - custom vertex and edge property

InDiskGraph
  - in memory and persistent storage

MultipropertyGraph
  - StorageType: InMemory, Hybrid
Multiproperty Graph

Key

Value

Label

Vertex

Person

Name: Gabriel
SSN: xxx-xx

Friend

since: 2010

Edge

Name: John

Company
Name: IBM

Works at
Since: 2010
Persistent Storage

- Write through policy for now
- Separate structure from properties: benefits computations based on structure only
- Efficient graph loading: on demand
- Versioning
Programming Model/ Runtime

- A graph is a collection of vertices
- Each vertex maintains its in and out edges

Parallel processing on IBM PPL graph
- Task based model of parallelism
  - `execute_tasks(wf, num_tasks)`
  - `for_each(graph, wf);`
  - `schedule_task_graph(tg)`
- Work stealing
- Two level nested parallelism
- Within shared memory for now
Performance Add Vertex

- Add vertices and for each vertex add a property
- Indexed
  - \( v = \text{add\_vertex}(); \ v.\text{set\_property}(\text{"name"}, \text{"vertex0"}); \)
- Titan with Berkeley DB backend

Intel Haswell 24 core 2.7GHz and 256 GB, SSD
Performance Add Edge

- Add edges randomly
- The source and target are specified as vertex properties
- `add_edge("vertex0", "vertex7")`
  - Index lookup
Performance - Query

- For a given vertex collect all its neighbors up to depth=3
  - ~1000 edges traversed per query
Query 2 - find the newest 20 posts from your friends

MATCH (:Person {id:{person_id}})-[:KNOWS]-(friend:Person)<-[::HAS_CREATOR]-(post:Post)
WHERE post.creationDate<=max_date
RETURN friend.id AS personId, friend.firstName AS personFirstName, friend.lastName AS personLastName,
    post.id AS postId, post.content AS postContent, post.creationDate AS postDate
ORDER BY postDate DESC
LIMIT 20

Query 4 - new topics

Find the top 10 most popular topics/tags (by the number of comments and posts) that your friends have been talking about in the last x hours.

MATCH (:Person {id:{person_id}})-[:KNOWS]-(friend:Person)
MATCH (friend)<-[::HAS_CREATOR]-(post:Post)
WHERE post.creationDate>={min_date} AND post.creationDate<=max_date
MATCH (post)-[::HAS_TAG]-(tag:Tag)
WITH DISTINCT tag, collect(tag) AS tags
RETURN tag.name AS tagName, length(tags) AS tagCount
ORDER BY tagCount DESC
LIMIT 10
RDF Graph Construction

- Load the .csv files for vertices
- Load the .csv files for edges
- Construct property graph in memory only

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Size</th>
<th>Properties</th>
<th>Load Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>100,000</td>
<td>8</td>
<td>0.45</td>
</tr>
<tr>
<td>Post</td>
<td>54,784,723</td>
<td>7</td>
<td>188.8</td>
</tr>
<tr>
<td>Forum</td>
<td>3,676,271</td>
<td>3</td>
<td>10.6</td>
</tr>
<tr>
<td>Place</td>
<td>5,130</td>
<td>3</td>
<td>0.01</td>
</tr>
<tr>
<td>Tag</td>
<td>12,144</td>
<td>3</td>
<td>0.03</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Edges</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Person-knows-person</td>
<td>2,887,797</td>
<td>0</td>
<td>3.21</td>
</tr>
<tr>
<td>Person-likes-post</td>
<td>208,241,439</td>
<td>0</td>
<td>311</td>
</tr>
<tr>
<td>Post-hasCreator-person</td>
<td>54,784,723</td>
<td>0</td>
<td>54</td>
</tr>
<tr>
<td>Post-hasTag-tag</td>
<td>42,797,703</td>
<td>0</td>
<td>34</td>
</tr>
<tr>
<td>Forum-contains-post</td>
<td>54,784,723</td>
<td>0</td>
<td>52</td>
</tr>
</tbody>
</table>
Query 2

![Graph showing query performance metrics over time. The x-axis represents time in seconds, ranging from 0 to 0.204. The y-axes represent the number of queries and the number of edges per query, with values ranging from 0 to 250 for queries and from 0 to 4000000 for edges per query. The graph includes a line chart and a bar chart, illustrating the trend of query execution and edge processing.]
Impact of Parallelism on Throughput

Using Tcmalloc

<table>
<thead>
<tr>
<th>Queries are processed by a single thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q2</td>
</tr>
<tr>
<td>Q2 P</td>
</tr>
<tr>
<td>Q4</td>
</tr>
<tr>
<td>Q4 P</td>
</tr>
<tr>
<td>Q6</td>
</tr>
<tr>
<td>Q6 P</td>
</tr>
</tbody>
</table>

Using standard malloc

<table>
<thead>
<tr>
<th>Queries are assigned to threads evenly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q2</td>
</tr>
<tr>
<td>Q2 P</td>
</tr>
<tr>
<td>Q4</td>
</tr>
<tr>
<td>Q4 P</td>
</tr>
<tr>
<td>Q6</td>
</tr>
<tr>
<td>Q6 P</td>
</tr>
</tbody>
</table>

#concurrent component

Impact of Parallelism on Throughput

Queries are assigned to threads evenly

Queries are processed by a single thread

#concurrent component

Impact of Parallelism on Throughput

Queries are assigned to threads evenly

Queries are processed by a single thread

#concurrent component
Graph databases are gaining in popularity

- Google, Facebook, Twitter, Paypal, BAML

<table>
<thead>
<tr>
<th>Feature</th>
<th>System G Native Store</th>
<th>Neo4j</th>
<th>Titan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Back-end</td>
<td>Graph</td>
<td>Graph</td>
<td>Non-graph</td>
</tr>
<tr>
<td>Scaling</td>
<td>Yes</td>
<td>Moderate</td>
<td>Yes</td>
</tr>
<tr>
<td>Traversal efficiency</td>
<td>Perfect</td>
<td>Good</td>
<td>Poor</td>
</tr>
<tr>
<td>Schemaless</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>User defined function</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Performance-critical App.</td>
<td>Perfect</td>
<td>Good</td>
<td>Poor</td>
</tr>
<tr>
<td>Multi-language APIs</td>
<td>C++, Java, Python, Shell</td>
<td>Java, Cypher</td>
<td>Java, Gremlin</td>
</tr>
</tbody>
</table>