GAIA: A System for Interactive Analysis on Distributed Graphs Using a High-Level Language

Zhengping Qian, Chenqiang Min, Longbin Lai, Yong Fang, Gaofeng Li, Youyang Yao, Bingqing Lyu, Xiaoli Zhou, Zhimin Chen, Jingren Zhou

Alibaba Group
Graph Data are Prevalent

Social networks
Electricity grid
(Blackout of August 14, 2003)

The Internet graph

Traffic graph
(Visualization of global flights)

On-line behavior targeting

Gene analysis
A property graph is composed of vertices (nodes), edges (links), and properties (key/value pairs).
Challenges for Large Graph Traversal at Alibaba

Although many distributed and parallel graph processing frameworks exist, they are too hard to program (for non-technical users)

A humongous graph
- 1-10B vertices (e.g., products, buyers, etc.)
- 10B-1T edges (e.g., clicks, payments, etc.)
Current State of the Art

• Gremlin: the de-facto standard graph traversal query language

Example. Find top k cycles of length 1, starting from the account with id “2”.

```
g.V().has('id','2').as('a')
  .repeat(out()).simplePath().times(1-1)
  .where(out().eq('a'))
  .path()
  .limit(k)
```

• A lot of good (graph traversal) semantics

→ Existing solutions suffer from a lack of scalability

GAIA is designed to faithfully preserve the programming model of Gremlin, and extend it to large, distributed graphs
Data-Parallel Execution of Gremlin

An example query Q1

```
// g is the input graph
// V is the vertex collection
// out is the out collection
// select is the select collection
// has is the has label collection
// by is the by property collection

g.V().has('id', '2').out().select().by('name')
```
SCOPE Abstraction

An example query Q2

\[ g.V().has('id', '2').out().select().by(out().count()) \]

Allows a subgraph of dataflow to be treated as a single operator

Separation of contexts can be specified by SCOPE policies, e.g.,

Input traversers

Output traversers
Compilation of Control-Flow Constructs

An example query Q3

g.V().has('firstname','Tom').as('a')
  .repeat(out()).simplePath().times(k)
  .where(out().eq('a'))
  .path()
Dynamic Dependency Tracking

An example query Q2

g.V().has('id', '2').out().select().by(out().count())
Distributed Execution and Optimizations

- Advanced optimizations (see paper for more details), e.g.,
  - Hybrid BFS/DFS traversal to bound memory usage
  - Early stop to avoid wasted computation

Dynamic scheduling

Compute node 1

DOP=2[1]

Local executor 1

Memory

CPU

CPU

CPU

CPU

Compute node 2

DOP=2[2]

Local executor 2

Memory

CPU

CPU

CPU

CPU
Implementation and Evaluation

Implemented a full-fledged graph system, integrated with TinkerPop
Extensive evaluation using the LDBC Social Network Benchmark (Interactive Workloads)

Single-threaded performance (vs JanusGraph)
• On average 260×, up to 1430×
• CR-3,5,9 timed out in JanusGraph

Compare with optimized, hand-coded implementation
• 16× and 14× speedup, with 21× and 10× less memory than Timely Dataflow and Plato (Gemini)

Scalability
• 6× from 8 to 64 workers
Concluding Remarks

- Interactive graph exploration plays a crucial role in the life cycle of modern Web-scale graph analytics

- GAIA extends previously graph systems in two ways
  - Exploits Gremlin to provide a high-level language for traversal
  - Supports automatic parallel execution with optimizations

- Open source release is available at:
  https://github.com/alibaba/GraphScope/tree/main/research/gaia
THANK YOU!
QUESTIONS?

zhengping.qzp@alibaba-inc.com