GraphOne: A Data Store for Real-time Analytics on Evolving Graphs

Pradeep Kumar, H. Howie Huang
The George Washington University
Graph Applications: Big Data and Scientific Applications

- **Basic Building Block** for Modern Trade
  - Social Network, Navigation, E-Commerce, etc.
  - **Metadata Store** for Big Data

- **Scientific Applications**
  - Protein-Protein Interaction, Metabolic Network, etc.
Graph Applications: Big Data and Scientific Applications

- **Basic Building Block** for Modern Trade
  - Social Network, Navigation, E-Commerce, etc.
  - **Metadata Store** for Big Data

- **Scientific Applications**
  - Protein-Protein Interaction, Metabolic Network, etc.

- **And Many More**
  - Identification of Vulnerable code, Cyber Attack
  - Knowledge Graph
  - Key for Data Governance, e.g. GDPR

Li et al. Science'04
Definitions

- **Topological Graph**: Static (or slowly changing) Relationships
  - **Vertices**: Switches and Computers
  - **Edges**: Network Bandwidth between Vertices

- **Batch Analytics** on Topological Graph
  - Network Infrastructure Management
  - PageRank, Breadth First Traversal, etc.

- **Streaming Graphs**: Faster Arrival Rate
  - **Vertices**: Users, Processes, Computers + Ports, etc.
  - **Edges**: Network Data Flow, Authentication Logs, Process Events

- **Stream Analytics** on Streaming Graph
  - Identification of Security Risks*

---

* Grades’15 Paper from Pacific Northwest National Laboratory; LANL Net-flow dataset, 2015 and 2017; computers and Security’15, ...
Observation: Prior Works have Specialized/Private Data Stores

Research Question: How to perform diverse set of real-time analytics in presence of high arrival rate of graph data?

Findings:
- Specialized/Private Data Stores
- Integration of Many Systems
  - Excessive Data Duplication
  - Weakest Link Problem

<table>
<thead>
<tr>
<th>Solutions</th>
<th>Data Ingestion</th>
<th>Data Access Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Graph Databases</strong></td>
<td>Fine-grained (e.g., Neo4j)</td>
<td>Whole Data</td>
</tr>
<tr>
<td>(For Short Queries)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dynamic Graph Engines</strong></td>
<td>Fine-grained (e.g., Stinger)</td>
<td>Whole Data</td>
</tr>
<tr>
<td>(For Batch Analytics)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Stream Graph Engines</strong></td>
<td>Coarse-grained (e.g., Graphchi)</td>
<td>Whole Data</td>
</tr>
<tr>
<td>(For Stream Analytics)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>GraphOne</strong></td>
<td>Both</td>
<td>All</td>
</tr>
<tr>
<td><strong>Both</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>All</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Abstracting Data Store away from Graph Analytics Engine

Graph Analytics Framework

Data Sources

Graph Data Store

Stream Analytics

Batch Analytics

+ 

Anomaly

Community

Ranking

Actionable Knowledge
Idea: Abstracting Data Store away from Graph Analytics Engine

Graph Analytics Framework

Graph Data Store

Stream Analytics

Batch Analytics

Results:

- Over Neo4j and SQLite: 2 to 3 orders of higher ingestion rate
- Over Stinger: 5.36x more ingestion rate, over 3x speedup for analytics
- Over Static Graph Engine: No Pre-processing Cost
GraphOne: Background and Architecture

(a) Example graph

(b) Edge List
- **Log**: Captures Temporal Ordering
- Fine-grained Versioning is Free

(c) Adjacency List
- Indexed Data Structure
- Coarse-grained Versioning

Opportunity: **Hybrid Store**

Two New Abstractions: **Data Visibility**: Analytics choose their Ingestion type
GraphOne: Background and Architecture

(a) Example graph

(b) Edge List
- **Log**: Captures Temporal Ordering
- Fine-grained Versioning is Free

(c) Adjacency List
- Indexed Data Structure
- Coarse-grained Versioning

Opportunity: Hybrid Store

Two New Abstractions:

- **Data Visibility**: Analytics choose their Ingestion type

**Graph View:**
- **Static View**: Batch Analytics
- **Stream View**: Stream Analytics

Graph Data Updates → Stream Analytics → Batch Analytics

Stream View

Static View

Circular Edge Log

Adjacency Store

Hybrid Store

Data Management

Logging

Archiving

Coarse-grained

NVMe

Data Durability

New log → Old log

GraphView
(Data Visibility)
Hybrid Store Details

Edge Deletion at \( t_7 \)

Circular Edge Log

<table>
<thead>
<tr>
<th>t_0</th>
<th>t_1</th>
<th>t_2</th>
<th>t_3</th>
<th>t_4</th>
<th>t_5</th>
<th>t_6</th>
<th>t_7</th>
</tr>
</thead>
<tbody>
<tr>
<td>5,6</td>
<td>1,2</td>
<td>3,4</td>
<td>2,4</td>
<td>0,1</td>
<td>0,3</td>
<td>1,3</td>
<td>-2,4</td>
</tr>
</tbody>
</table>

Global Snapshot List

- \( S_0, 4 \)
- \( S_1, 8 \)

Multi-version Degree Array

<table>
<thead>
<tr>
<th>Vertex Array</th>
<th>Edge Arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-version Degree Array</td>
<td>Vertex Array</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>

Adjacency List
**Hybrid Store Details**

**GraphView**

- **Static View**
  - Whole Data Access

- **Stream View**
  - Data Access from a Time-Window

**Adjacency List**

- **Vertex Array**
- **Edge Arrays**

**Global Snapshot List**

- **Multi-version Degree Array**

**Circular Edge Log**

- Non-archived edges

**Edge Deletion at** $t_7$

**Circular Edge Log**

- 5,6, 1,2, 3,4, 2,4, 0,1, 0,3, 1,3, -2,4, 2,5, 3,6

**Non-archived edges**

**Global Snapshot List**

- S0, 4
- S1, 8

**Edge Arrays**

<table>
<thead>
<tr>
<th>Handle</th>
<th>Action</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>create-static-view</td>
<td>(flags)</td>
</tr>
<tr>
<td>1</td>
<td>delete-static-view</td>
<td>(handle)</td>
</tr>
<tr>
<td>2</td>
<td>get-nebr-degree</td>
<td>(handle, vertex-id)</td>
</tr>
<tr>
<td>3</td>
<td>get-nebrs</td>
<td>(handle, vertex-id, ptr)</td>
</tr>
<tr>
<td>4</td>
<td>update-view</td>
<td>(handle)</td>
</tr>
<tr>
<td>5</td>
<td>has-vertex-changed</td>
<td>(handle, vertex-id)</td>
</tr>
</tbody>
</table>
Hybrid Store: Optimizing the Archiving Phase

**Machine:** 2 Intel Xeon CPU socket
- 14 cores each,
- Hyperthreading enabled

**Dataset:** Kron-28 Graph
- 256 Million Vertex,
- 4 Billion Edges

---

**Archiving Phase**

- Local Buffers
  - Edges with source vertex range \([v_0, v_1]\)
  - Edges with source vertex range \([v_1, v_2]\)
- Non-atomic Archiving

---

**Archiving with Atomic Instruction**

Archiving with Edge Sharding

**Run Time (sec)**

<table>
<thead>
<tr>
<th>Thread Count</th>
<th>Archiving Rate</th>
<th>Millions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Adjacency list Data**
Hybrid Store: Optimizing Memory Usage

- Edge Log: Circular
- Degree Array: Older Versions Reused

- Edge Arrays: Many Memory Links
  - Cacheline Size Memory Allocation
  - Hub Vertex Handling

<table>
<thead>
<tr>
<th>Optimizations</th>
<th>Chain Count</th>
<th>Memory Needed (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average</td>
<td>Maximum</td>
</tr>
<tr>
<td>Baseline System</td>
<td>29.18</td>
<td>65536</td>
</tr>
<tr>
<td>Static GraphOne</td>
<td>0.45</td>
<td>1</td>
</tr>
<tr>
<td>+Cacheline Size allocation</td>
<td>2.96</td>
<td>65536</td>
</tr>
<tr>
<td>+Hub vertex Handling</td>
<td>2.47</td>
<td>3998</td>
</tr>
</tbody>
</table>

![Diagram showing chained edge arrays and vertex arrays with speedup analysis for Baseline, +Cacheline Size, +Hub Vertex.]
Results: Hybrid Store Composition and Ingestion Rate

- Max Non-archived Edge count:
  - \(2^{17}\) edges

- Archiving Threshold:
  - \(2^{16}\) edges

- Archive Rate:
  - 43.68 Million edges/s

- Faster Ingestion?
  - Logging rate:
    - \(~80\) Million edges/sec
  - Higher archiving threshold

<table>
<thead>
<tr>
<th>Graph Name</th>
<th>Vertex Count (in Millions)</th>
<th>Edge Count (in Millions)</th>
<th>Rates (Million Edges/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Logging</td>
</tr>
<tr>
<td>Twitter (D)</td>
<td>52.58</td>
<td>1963.26</td>
<td>82.62</td>
</tr>
<tr>
<td>Friendster (D)</td>
<td>68.35</td>
<td>2586.15</td>
<td>82.85</td>
</tr>
<tr>
<td>Subdomain (D)</td>
<td>101.72</td>
<td>2043.20</td>
<td>82.86</td>
</tr>
<tr>
<td>Kron-28 (U)</td>
<td>256.00</td>
<td>4096.00</td>
<td>79.23</td>
</tr>
<tr>
<td>LANL (D)</td>
<td>0.16</td>
<td>1521.19</td>
<td>35.98</td>
</tr>
</tbody>
</table>

D = Directed, U = Undirected
Results: Dynamic Graph Systems

Experimental Setup
- Two Intel Xeon CPU E5-2683 sockets with 14 Core each
- 500GB Memory
- Samsung 950 Pro NVMe SSD, 512GB

Dataset: RMAT Graph
- 4 Million Vertices, 64 Million Edges
- 8 bytes weight
- 40 Million updates, 2.5 Millions deletions

Against Stinger
- 5.36x Higher Ingestion Rate
- 12.76x and 3.18x Speedup for BFS and PageRank

Against SQLite and Neo4j:
- 2 to 3 Orders of Magnitude Higher Ingestion Rate
**Results: Cost of Data Management Over Static Graph Engine**

- **Static GraphOne**: For Static Graphs
  - Builds graph in entirety, Like Galois
- **GraphOne**: For Dynamic/Streaming Graphs
  - Builds graph one edge at a time

- 17% less performance for real-world graphs
  - Avoids pre-processing cost completely
  - 32.73 second pre-processing cost, 34.12x of BFS
GraphOne: Conclusion

- Batch and Stream Analytics on Evolving Graphs
- Hybrid Store: Two Abstractions
  - Data Visibility and GraphView
- Optimizations:
  - Edge Sharding and Memory Allocation
- Results:
  - Over Neo4j and SQLite: 2 to 3 orders of higher ingestion rate
  - Over Stinger: 5.36x more ingestion rate, over 3x speedup for analytics
  - Over Static Graph Engine: No Pre-processing Cost
- A Real-System you can use
  - Can convert text/logs/csv/json to graph
Thank You
pradeepk@email.gwu.edu