Version Traveler

Fast and Memory-Efficient Version Switching in Graph Processing Systems

Xiaoen Ju¹, Dan Williams², Hani Jamjoom², Kang G. Shin¹

¹University of Michigan
²IBM Research

USENIX ATC 2016
June 24, 2016, Denver, CO
Multi-version graph processing

- Understand dynamics of graph evolution
- Capture information beyond processing single version
- Examples:
  - Compute varying distance between users in social network
  - Analytics as graph evolves
Switching between versions

- **Arbitrary**: version switch not necessarily forward-only
- **Local**: next version often similar to current version

Detect change of distance between two users

Collaborative data analytics
A single-version system is inefficient

- Loading a full graph version each time ignores multi-version property
A single-version system is inefficient

- Loading a full graph version each time ignores multi-version property
- Deltas can improve switching speed
Challenge

- How should graph be represented?
- How should deltas be represented?

- Need to balance
  - Fast switching
  - Graph processing speed
    - Neighbor access
  - Compactness

Diagram:
- Neighbor access speed
- Version switching speed
- Compactness
Version Traveler (VT)

- Hybrid graph representation
  - Augmented compressed sparse row (CSR)
- Hybrid delta representation
  - Techniques to optimize for compactness
    - Sharing (trade switching ability for compactness)
    - Chaining (trade access efficiency for compactness)
- Implemented multi-version support in PowerGraph
  - 23x faster with 15% memory overhead
Background: CSR

- Compressed Sparse Row
- Used in PowerGraph and GraphX

- Compact
  - Number of neighbors are encoded by next src vid
- Fast neighbor access
Background: CSR

- Compressed Sparse Row
- Used in PowerGraph and GraphX

- Compact
  - Number of neighbors are encoded by next src vid
- Fast neighbor access
Background: CSR

- Compressed Sparse Row
- Used in PowerGraph and GraphX

- Compact
  - Number of neighbors are encoded by next src vid

- Fast neighbor access

- Expensive to modify
VT adds deltas to CSR

- Idea: use CSR for large unchanging subgraph
  - Add deltas for varying parts

Delta Cache
VT adds deltas to CSR

- Idea: use CSR for large unchanging subgraph
  - Add deltas for varying parts

![Diagram showing the process of adding deltas to CSR](image.png)
VT adds deltas to CSR

- Idea: use CSR for large unchanging subgraph
  - Add deltas for varying parts
VT adds deltas to CSR

- Idea: use CSR for large unchanging subgraph
  - Add deltas for varying parts
VT adds deltas to CSR

- Idea: use CSR for large unchanging subgraph
  - Add deltas for varying parts
VT adds deltas to CSR

- Idea: use CSR for large unchanging subgraph
  - Add deltas for varying parts
Hybrid deltas

- Storing entire neighbor array per delta entry is memory inefficient

- Sharing
  - Delta logs describe how shared neighborhood vector is modified

- Chaining
  - Represent neighborhood as chain of vectors
Evaluation setup

- Single machine
- Variety of workloads and algorithms
- Generate deltas for static graphs

<table>
<thead>
<tr>
<th>Dataset</th>
<th>V (M)</th>
<th>E (M)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon08</td>
<td>0.7</td>
<td>5.2</td>
<td>Similarity among books</td>
</tr>
<tr>
<td>Dblp11</td>
<td>1.0</td>
<td>6.7</td>
<td>Scientific collaboration</td>
</tr>
<tr>
<td>Wiki13</td>
<td>4.2</td>
<td>101.4</td>
<td>English Wikipedia</td>
</tr>
<tr>
<td>Livejournal</td>
<td>5.4</td>
<td>79.0</td>
<td>Friendship in LiveJournal social network</td>
</tr>
<tr>
<td>Twitter</td>
<td>41.7</td>
<td>1468.4</td>
<td>Twitter follower graph</td>
</tr>
<tr>
<td>Facebook</td>
<td>0.1</td>
<td>1.6</td>
<td>Friendship in regional Facebook network</td>
</tr>
<tr>
<td>Github</td>
<td>1.0</td>
<td>5.7</td>
<td>Collaboration in software development</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nop</td>
<td>Access neighbor and return</td>
</tr>
<tr>
<td>Bipart</td>
<td>Max matching in a bipartite graph</td>
</tr>
<tr>
<td>CC</td>
<td>Identify connected components</td>
</tr>
<tr>
<td>PageRank</td>
<td>Compute rank of each vertex</td>
</tr>
<tr>
<td>SSSP</td>
<td>Single-source shortest path</td>
</tr>
<tr>
<td>TC</td>
<td>Triangle count</td>
</tr>
</tbody>
</table>
VT: performance and compactness

- Amazon08 data set with 10.1% deltas, SSSP
- PowerGraph
  - Load every time
  - With full-version caching
- VT is 23x faster than non-caching, 15% of its memory
Other multi-version approaches

- Log-based
  - Apply a log on every version switch

- Bitmap-based
  - Consult bitmap on every access to determine neighborhood

Varying number of $\delta$ from 10 to 100

Varying size of $\delta$ from 0.01% to 1.0%
Realistic evolving graphs

- 10-version graphs
  - Deltas encode new friends/collaborators

- SSSP

**Regional Facebook and GitHub**

**Graphs**

- Dots represent days, and lines show edges in thousands over time.

**Bars**

- Three types: time-saving, mem-saving, util-improve

**Labels**

- X-axis: days
- Y-axis: edges (in thousands)
Realistic evolving graphs

- 10-version graphs
  - Deltas encode new friends/collaborators

- SSSP

- Much more evaluation in the paper!
Summary

- Multi-version graph processing needs new systems
- Tradeoffs involve
  - Compactness
  - Neighbor access efficiency
  - Switching time
- VT’s hybrid graph and delta balances tradeoffs
- Check out the paper and contact Xiaoen!
  - xiaoen.ju@gmail.com