MMap: Fast Billion-Scale Graph Computation on a PC via Memory Mapping

Lead by
Zhiyuan (Jerry) Lin
Georgia Tech CS Undergrad
Now: Stanford 1st year PhD student


Graph Computation on Computer Cluster?

Steep learning curve

Cost

Overkill for smaller graphs

Best-of-breed Single-PC Approaches

- GraphChi – OSDI 2012
- TurboGraph – KDD 2013

What do they have in common?

- Sophisticated Data Structures
- Explicit Memory Management
Can We Do Less?
To get same or better performance?
e.g., auto memory management, faster, etc.
**Main Idea:** Memory-mapped the Graph

<table>
<thead>
<tr>
<th>source_id</th>
<th>target_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>44</td>
</tr>
<tr>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>999,998</td>
<td>875</td>
</tr>
<tr>
<td>999,999</td>
<td>255,750</td>
</tr>
</tbody>
</table>

- **Edge List file (e.g. tens of GB)**
- **Physical Memory (e.g. 8 GB)**
Main Idea: Memory-mapped the Graph

Un-Mapped

Mapped

That’s all!
How to compute PageRank for huge matrix?

Use the power iteration method

\[ p = c \ B \ p + \frac{(1-c)}{n} \ 1 \]

Can initialize this vector to any non-zero vector, e.g., all “1”s
Example: PageRank (implemented using MMap)


\( v_{\text{next}} = \frac{1-d}{N} + d \times E \times v_{\text{cur}} \)

(\text{where } E_{i,j} = \frac{I(i \rightarrow j)}{\text{deg}_i})

Fig. 3: Data structures used for computing PageRank. In our PageRank implementation, a binary edge list file and three node vectors are used.
1-step Neighbor Query Runtime on YahooWeb Graph (6.6 billion edges)

- TurboGraph: 154.7 ms
- MMap: 3.3 ms

- PageRank (3 iter.)
  - GraphChi: 2308 s
  - TurboGraph: 643 s
  - MMap: 579 s

- Conn. Comp.
  - GraphChi: 397 s
  - TurboGraph: 411 s
  - MMap: 274 s
Why Memory Mapping Works?

High-degree nodes’ info automatically cached/kept in memory for future frequent access

**Read-ahead paging** preemptively loads edges from disk.

Highly-optimized by the OS

No need to explicitly manage memory (less book-keeping)
Also works on tablets! (If you want.)

**Big Data on Small Devices (270M+ Edges)**

<table>
<thead>
<tr>
<th></th>
<th>Elapsed Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pokec</td>
<td>2.97</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>6.31</td>
</tr>
<tr>
<td>Orkut</td>
<td>14.7</td>
</tr>
<tr>
<td>Gplus</td>
<td>48.9*</td>
</tr>
</tbody>
</table>

- **Pokec** (31M edges)
- **LiveJournal** (69M edges)
- **Orkut** (117M edges)
- **Gplus** (272M edges)

- iPad mini
- Macbook Pro