MMap (Memory Mapping)
Simple, minimalist approach to scale up computation

Mahdi Roozbahani
Lecturer, Computational Science and Engineering, Georgia Tech
When should you use Spark/Hadoop, AWS, Azure?

And when should you not?
MMap
Fast Billion-Scale Graph Computation on a PC via Memory Mapping

Lead by
Zhiyuan (Jerry) Lin
Georgia Tech CS Undergrad
Now: Stanford 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>
</tbody>
</table>

Un-Mapped

[Table Diagram]

Mapped

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

Use the power iteration method

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

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


Fig. 3: Data structures used for computing PageRank. In our PageRank implementation, a binary edge list file and three node vectors are used.
8000 lines of code

(a) LiveJournal graph (69M edges)
(c) YahooWeb graph (6.6B edges)

1-step Neighbor Query Runtime on YahooWeb Graph (6.6 billion edges)

- TurboGraph: 154.7 ms
- MMap: 3.3 ms
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)

![Graph showing elapsed time for different datasets on different devices.](image)

- **PoKeC** (31M edges): 2.97 s on iPad mini, 0.7 s on Macbook Pro
- **LiveJournal** (69M edges): 6.31 s on iPad mini, 1.75 s on Macbook Pro
- **Orkut** (117M edges): 14.7 s on iPad mini, 2.73 s on Macbook Pro
- **Gplus** (272M edges): 48.9 s on iPad mini, 9 s on Macbook Pro
“Mobile” devices are now very powerful

**Geekbench Results**

<table>
<thead>
<tr>
<th>Device</th>
<th>Chip</th>
<th>Single-Core Score</th>
<th>Multi-Core Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018 iPad Pro</td>
<td>A12X</td>
<td>5,025</td>
<td>18,106</td>
</tr>
<tr>
<td>iPhone XS / XS Max / XR</td>
<td>A12</td>
<td>4,795</td>
<td>11,167</td>
</tr>
<tr>
<td>2017 iPad Pro</td>
<td>A10X</td>
<td>3,913</td>
<td>9,327</td>
</tr>
<tr>
<td>2018 iPad</td>
<td>A10</td>
<td>3,474</td>
<td>5,914</td>
</tr>
<tr>
<td>2018 15” MacBook Pro</td>
<td>2.2GHz six-core Core i7</td>
<td>4,928</td>
<td>21,165</td>
</tr>
<tr>
<td></td>
<td>2.6GHz six-core Core i7</td>
<td>5,053</td>
<td>21,351</td>
</tr>
<tr>
<td></td>
<td>2.9GHz six-core Core i9</td>
<td>5,344</td>
<td>22,552</td>
</tr>
</tbody>
</table>

Lead by Dezhi (Andy) Fang, Georgia Tech CS Undergrad. Now: Airbnb software engineer

**M3 Runtimes (10 Iterations) for Logistic Regression (L-BFGS)**

- **Dataset Size on Disk**
  - 10G
  - 40G
  - 70G
  - 100G
  - 130G
  - 160G
  - 190G

- **Runtime (s)**
  - 0
  - 500
  - 1000
  - 1500
  - 2000

- **RAM size = 32GB**

- **Dataset Exceeds RAM**

**M3 v.s. Spark**

- **L-BFGS**
  - 4 Instances:
    - 1950s
    - 2864s
    - 8256s

- **8x Spark**
  - 1604s
  - 1164s
  - 3491s

- **4x Spark**

**Diagrams**

- Graph showing M3 vs. Spark with different runtimes for dataset sizes.
Scalable Machine Learning & Graph Mining via Virtual Memory

Memory Mapping based computation is a minimalist approach that forgoes sophisticated data structures, explicit memory management, and optimization techniques but still achieve high speed and scalability, by leveraging the fundamental memory mapping (MMap) capability found on operating systems.

Broader Impacts of this Project

Large datasets in terabytes or petabytes are increasingly common, calling for new kinds of scalable machine learning approaches. While state-of-the-art techniques often use complex designs, specialized methods to store and work with large datasets, this project proposes a minimalist approach that forgoes such complexities, by leveraging the fundamental virtual memory capability found on all modern operating systems, to load into the virtual memory space the large datasets.