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

Duen Horng (Polo) Chau
Associate Professor
Associate Director, MS Analytics
Machine Learning Area Leader, College of Computing
Georgia Tech

Partly based on materials by
Professors Guy Lebanon, Jeffrey Heer, John Stasko, Christos Faloutsos, Parishit Ram (GT PhD alum; SkyTree), Alex Gray
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


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

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)


\[ v_{next} = \frac{1 - d}{N} + d \times E \times v_{cur} \]

(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.
8000 lines of code

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

- **Elapsed Time (s)**
  - Pokesc *(31M edges)*: 2.97
  - LiveJournal *(69M edges)*: 6.31
  - Orkut *(117M edges)*: 14.7
  - Gplus *(272M edges)*: 48.9*

* iPad mini  
* Macbook Pro
“Mobile” devices are now very powerful

**Geekbench Results**

<table>
<thead>
<tr>
<th></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: Blue dot
  - 40G: Orange dot
  - 70G: Yellow dot
  - 100G: Green dot
  - 130G: Red dot
  - 160G: Purple dot
  - 190G: Green dot

- **Runtime (s)**
  - 0 to 2000 s

- **RAM size = 32GB**

**M3 vs. Spark**

- **L-BFGS**
  - 10G: 1950s
  - 40G: 2864s
  - 70G: 8256s

- **K-Means**
  - 10G: 1604s
  - 40G: 1164s
  - 70G: 3491s

Legend:
- Blue: M3
- Orange: 8x Spark
- Light Orange: 4x Spark
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

- Faster I/O Operations
- Less Overhead