CSE6242: Data & Visual Analytics

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

Duen Horng (Polo) Chau

Associate Professor, College of Computing

Associate Director, MS Analytics

Georgia Tech

Mahdi Roozbahani

Lecturer, Computational Science & Engineering, Georgia Tech

Founder of Filio, a visual asset management platform

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

Image source: http://www.drupaltky.org/en/article/20
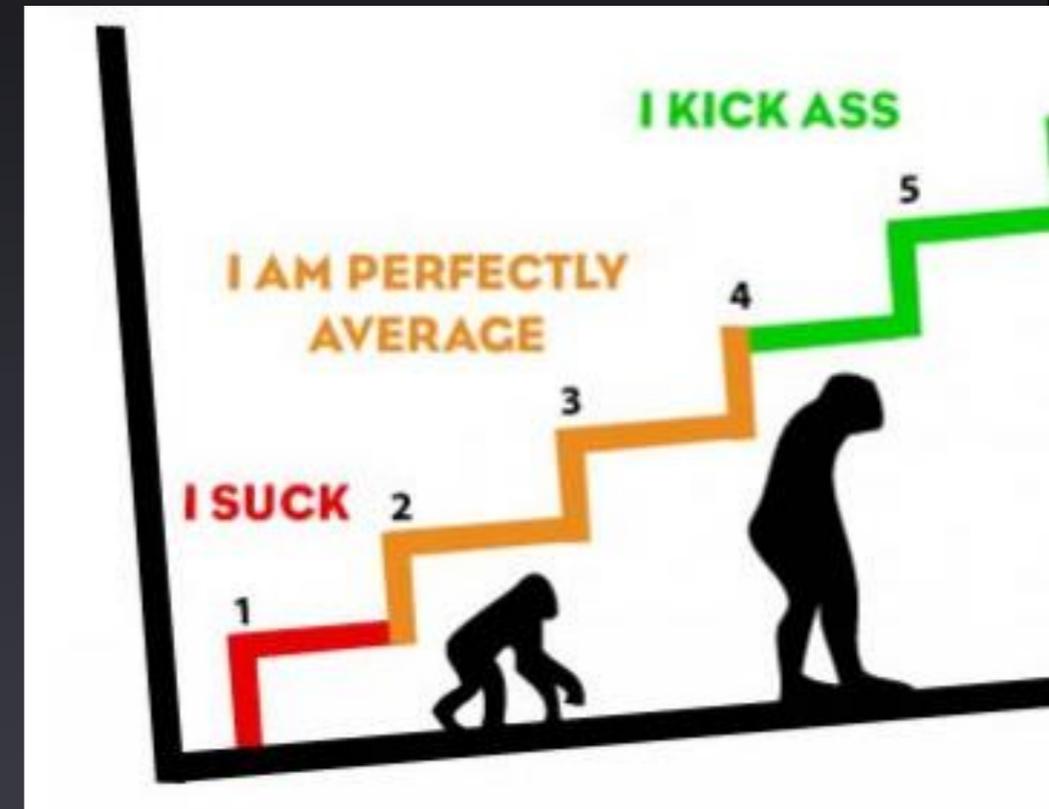
Best-of-breed Single-PC Approaches

- **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

source_id	target_id
0	1
0	4
0	5
0	8
1	1
1	44
1	50
2	3
3	3
3	10
999,998	
999,999	

Un-Mapped

Mapped
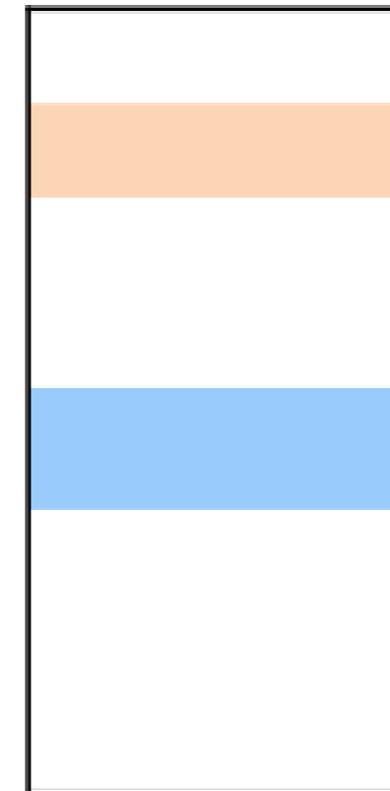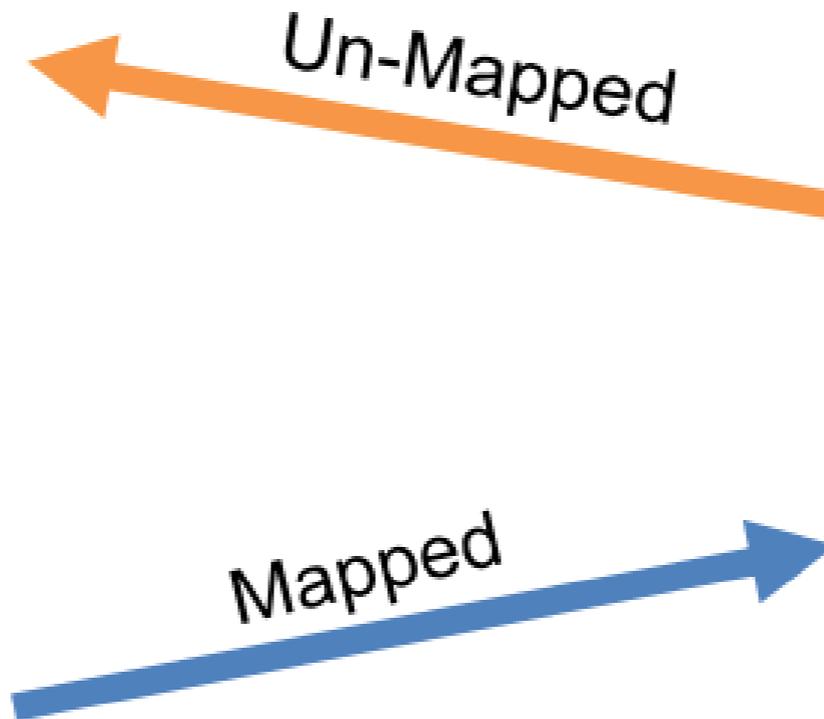
Edge
(e.g. tens of GB)

Physical Memory
(e.g. 8 GB)

That's all!

How to compute PageRank for huge matrix?

Use the power iteration method

http://en.wikipedia.org/wiki/Power_iteration
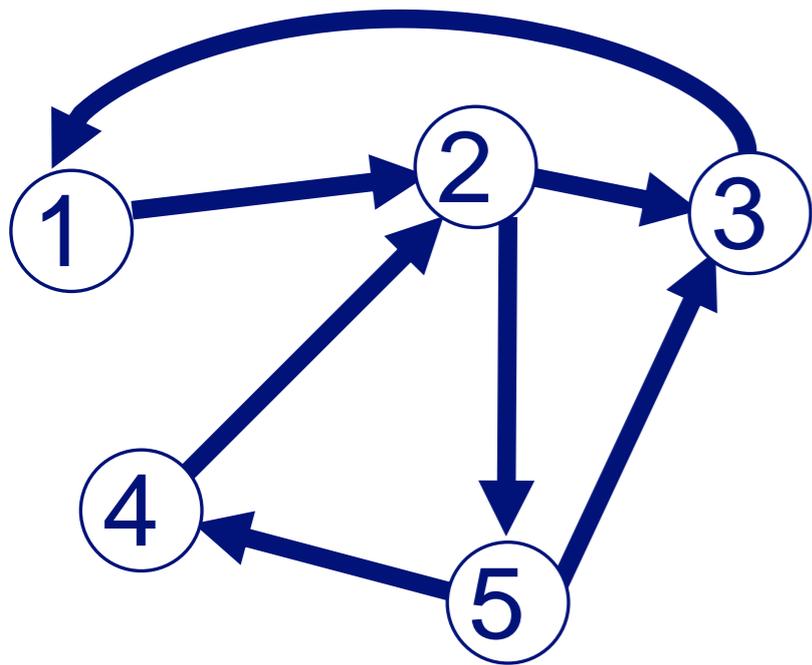
$$\mathbf{p} = c\,\mathbf{B}\,\mathbf{p} + \frac{(1-c)}{n}\,\mathbf{1}$$

$\mathbf{p'}$ $\qquad$ $\mathbf{B}$ $\qquad$ $\mathbf{p}$

$$\begin{bmatrix} p1 \\ p2 \\ p3 \\ p4 \\ p5 \end{bmatrix} = c \begin{bmatrix} & & & 1 & \\ 1 & & & 1 & \\ & 1/2 & & & 1/2 \\ & & & & 1/2 \\ & 1/2 & & & \end{bmatrix} \begin{bmatrix} p1 \\ p2 \\ p3 \\ p4 \\ p5 \end{bmatrix} + \frac{(1-c)}{n} \begin{bmatrix} 1 \\ 1 \\ \cdots \\ 1 \end{bmatrix}$$

Can initialize this vector to any non-zero vector, e.g., all "1"s

8

Example: PageRank (implemented using MMap)

http://www.cc.gatech.edu/~dchau/papers/14-bigdata-mmap.pdf

$$v^{next} = \frac{1-d}{N} + d \times E \times v^{cur}$$

$$\left(\text{where } E_{i,j} = \frac{I(i \to j)}{\deg_i}\right)$$

Node vector N1

	PRank-next
0	.000030
1	.000851
2	.000005
3	.002161
999,998	.000003
999,999	.000002

Edge list file E

source_id	target_id
0	1
0	4
0	5
0	8
1	1
1	44
1	50
2	3
3	3
3	10
999,998	175
999,998	875
999,999	255,750

Node vector N2

	out-deg
0	4
1	3
2	1
3	20
999,998	2
999,999	1

Node vector N3

	PRank-cur
0	.000032
1	.000850
2	.000005
3	.002154
999,998	.000003
999,999	.000002

Fig. 3: **Data structures used for computing PageRank.** In

8000 lines of code

PageRank (10 iter.)
- GraphChi: 54.0 s
- TurboGraph: 6.3 s
- MMap: 3.3 s

Conn. Comp.
- GraphChi: 6.4 s
- TurboGraph: 4.3 s
- MMap: 2.0 s
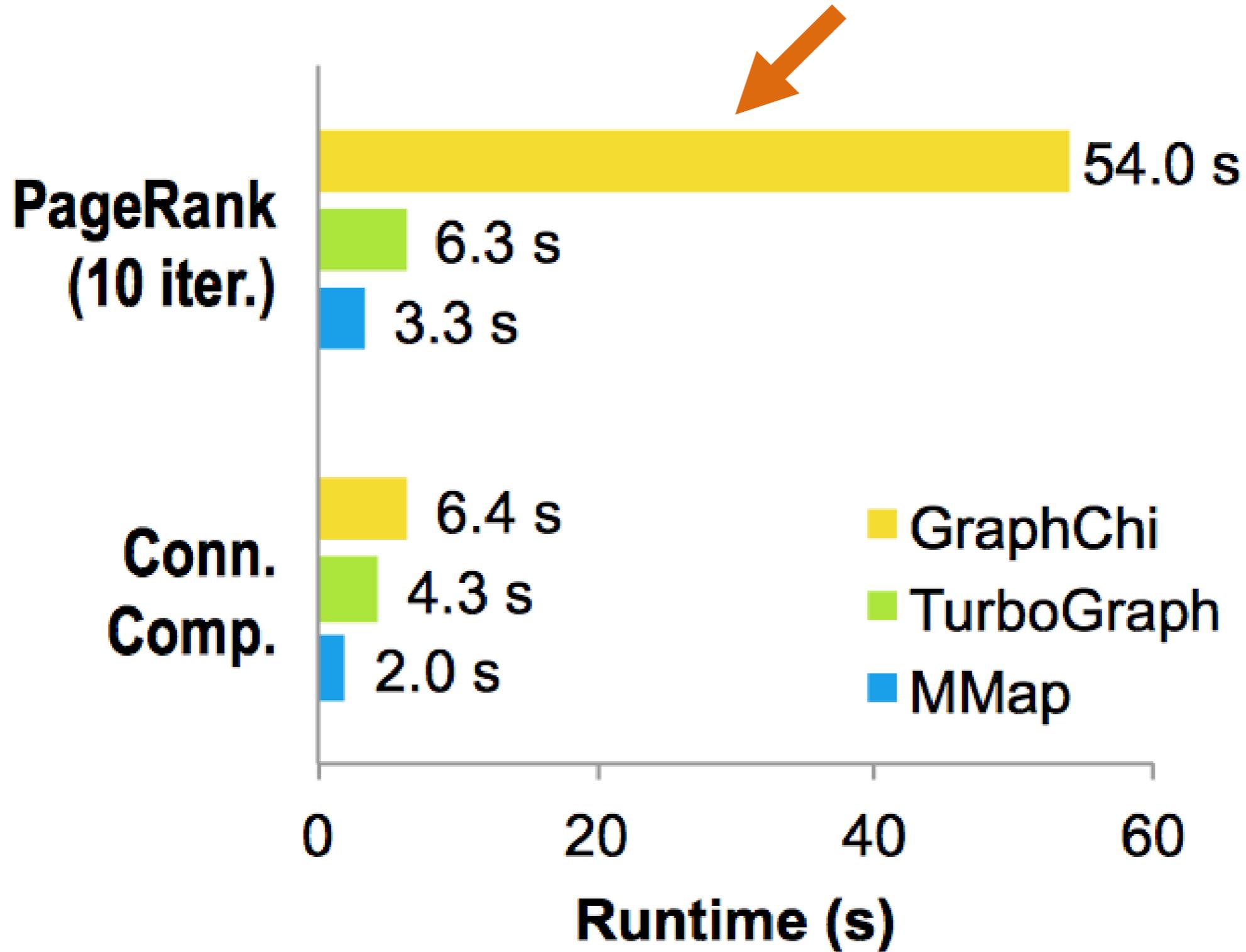
Runtime (s)

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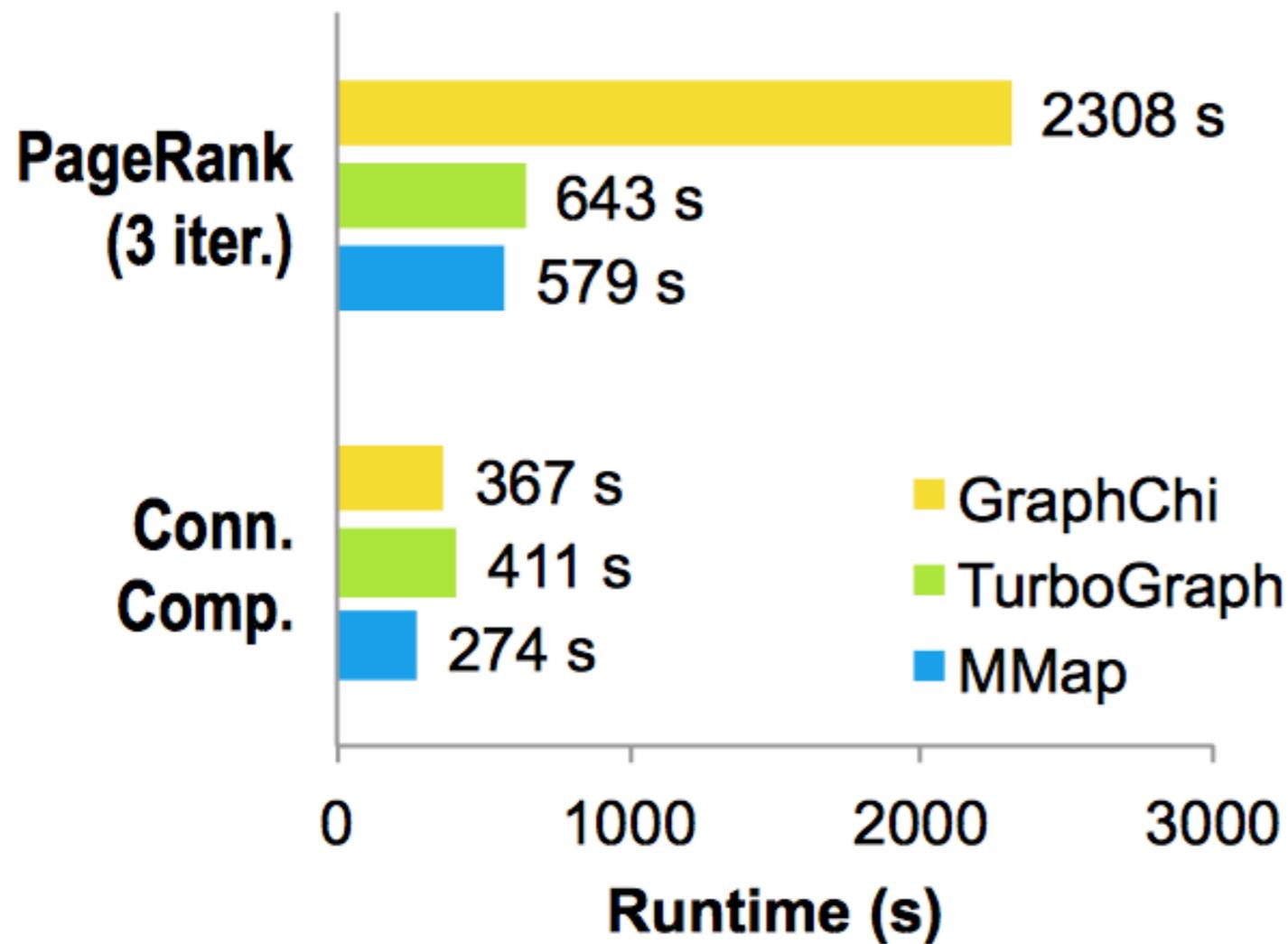
(c) YahooWeb graph (6.6B edges)

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

TurboGraph — 154.7 ms

MMap — 3.3 ms
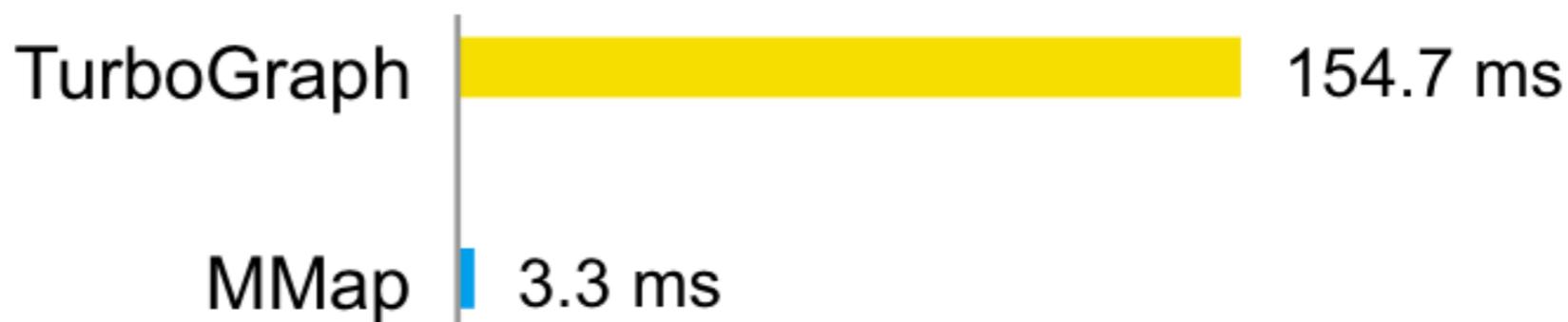
Why Memory Mapping Works?

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)

source_id	target_id
0	1
0	4
0	5
0	8
1	1
1	44
1	50
2	3
3	3
3	10
999,998	875
999,999	255,750

Un-Mapped

Mapped

Edge List file
(e.g. tens of GB)

Physical Memory
(e.g. 8 GB)
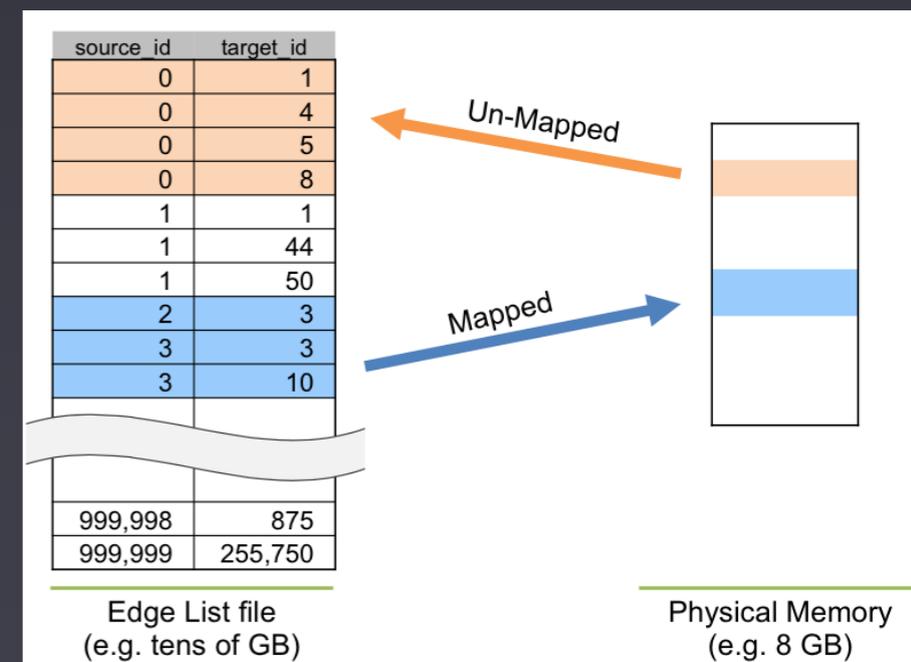
Also works on tablets! (If you want.)
Big Data on Small Devices (270M+ Edges)

Elapsed Time (s)

- Pokec (31M edges): iPad mini 2.97, Macbook Pro 0.7
- LiveJournal (69M edges): iPad mini 6.31, Macbook Pro 1.75
- Orkut (117M edges): iPad mini 14.7, Macbook Pro 2.73
- Gplus (272M edges): iPad mini 48.9*, Macbook Pro 9

Legend: ■ iPad mini ■ Macbook Pro
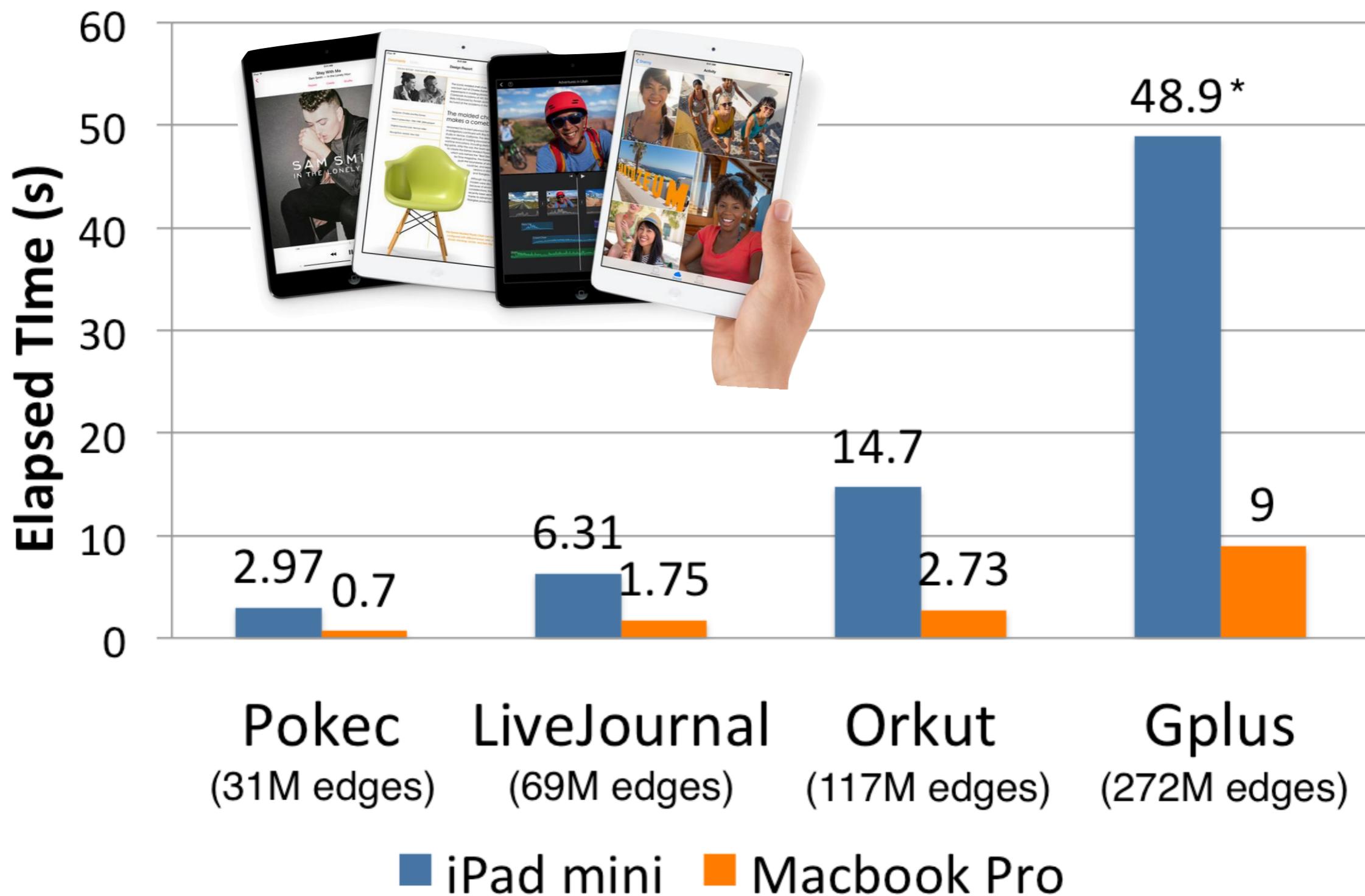
"Mobile" devices are now very powerful

Geekbench Results

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

MacRumors

https://www.macrumors.com/2018/11/01/2018-ipad-pro-benchmarks-geekbench/

15

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

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

Runtime (s)

RAM size = 32GB

Dataset Exceeds RAM

Dataset Size on Disk
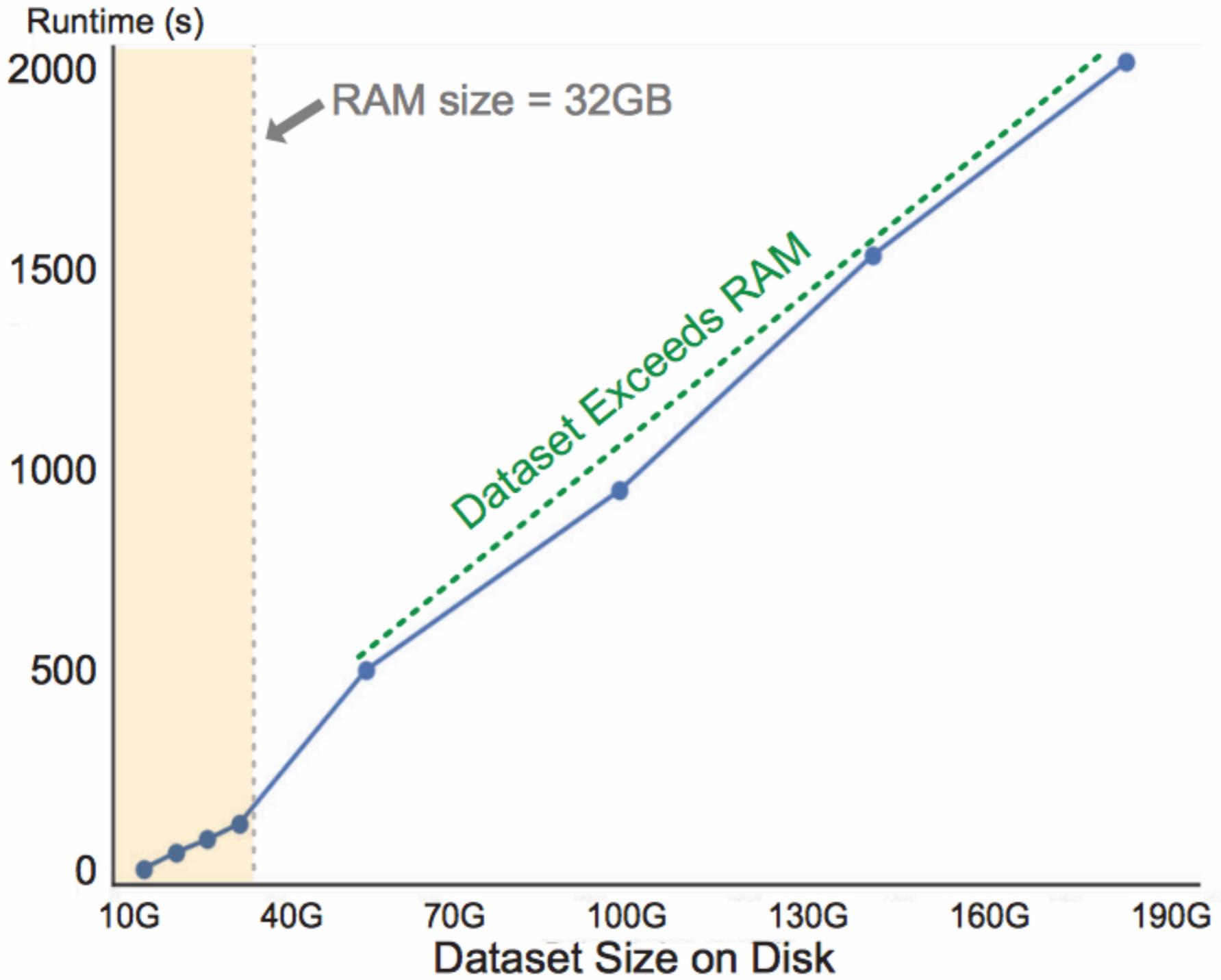
b M3 v.s. Spark
(4 & 8 Instances)

L-BFGS
1950s
2864s
8256s

K-Means
1604s
1164s
3491s

● M3
● 8x Spark
● 4x Spark

MMap project website http://poloclub.gatech.edu/mmap/
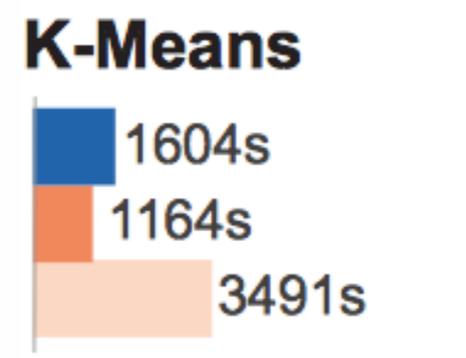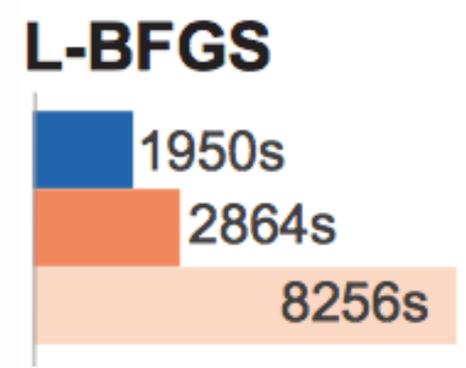
Scalable Machine Learning & Graph Mining via Virtual Memory

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