Thundercat graphmap:

improving* locality in graph applications

c. beeson, l. cang, m. venkataswamy

dec 12, 2017

*conditions apply

1

the problem

“large graphs are large - small graphs are boring”

- c. beeson

MOTIVATION

- Everything is a graph problem*
- Lots of real world applications (social networks, internet, bioinformatics)
- Real graphs are big
- Large graphs have bad locality

* if we have time, challenge c beeson on this

IS EVERYTHING REALLY A GRAPH PROBLEM?

- Yes if you squint:
  - Network Flow is P-Complete
  - Subgraph Isomorphism is THE NP-Hard problem (besides SAT)
  - Really every problem is a traversal of a computational state space graph

$$\exists\ H \subseteq G \mid P(H)$$

APPLICATIONS WITH LARGE GRAPHS

- Facebook:
  - 1.71 B Vertices (2016)
  - If including pages people like
    > 1T edges

- YouTube:
  - 1 billion+ unique users watch 3.25 B hours of video monthly

https://code.facebook.com/posts/319004238457019/a-comparison-of-state-of-the-art-graph-processing-systems/

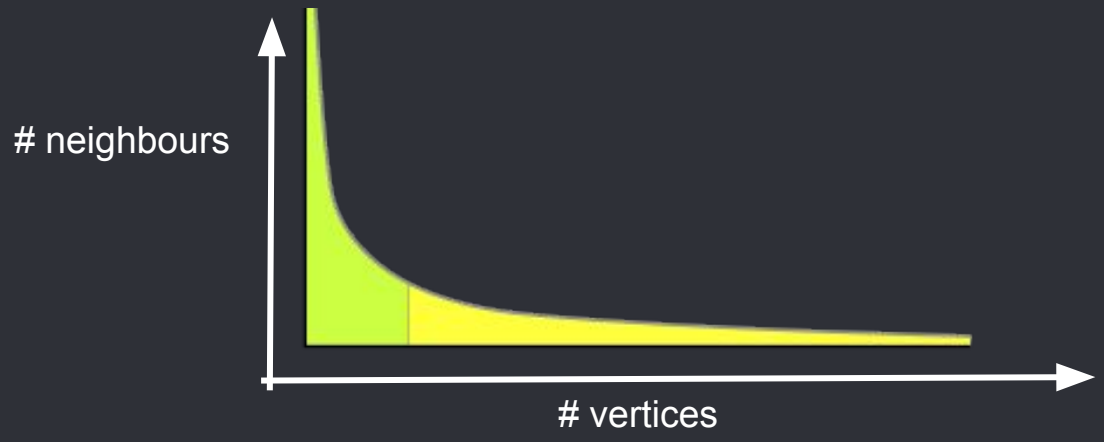
COMPENSATING FOR POOR LOCALITY

- Graph algorithms inefficient *(Nilakant, 2014)*
  - MapReduce and Hadoop not optimized for data propagation

- Distributed graph-parallel computations
  - vertex-centric models using message-passing *(Malewicz, 2010)*

- Real-world graphs are difficult to represent in a distributed environment

Ex. web,
social networks
(Gonzalez, 2012)



2

the investigation

“dun dun”

- c. beeson (law & order)

DATA SETS

Stanford Large Network Dataset Collection

	N	M	D	Before	After
Amazon	334,863	925,872	549	13M	1.4G
Friendster	65,608,366	1,806,067,135	???	20G	???
Youtube	1,134,890	2,987,624	28,754	38M	200G
Facebook	4,039	88,234	1,045	921K	65M
DBLP	317,080	1,049,866	343	14M	700M

IMPACT OF PARTITION ON GRAPH APPLICATIONS

Performance is affected by:

- Traversal order
- Fetch order

We planned:

3 *ordered-data partitions*

by

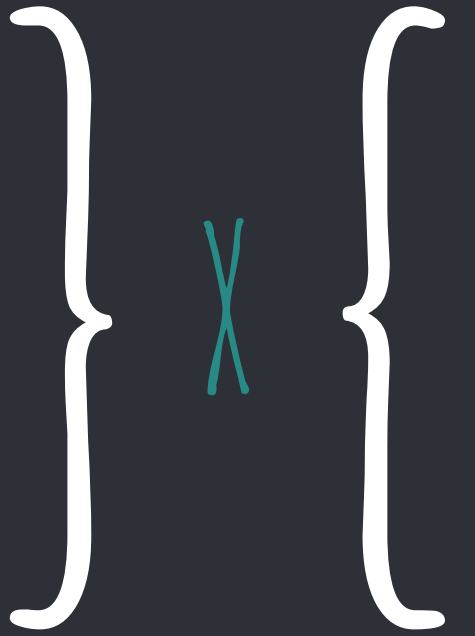
2 *traversal patterns*

to compare performance

COMPARING APPLICATION BY PARTITION

Applications

1. BFS
   Breadth First Search
2. DFS
   Depth First Search



Partitions

1. BFS
2. DFS
3. Random

OTHER PARTITION APPROACHES

Balanced Partition is NP-Hard

Fix parameter Optimally Solvable $O(n^{k\hat{}2})$

Approximable roughly within a log factor

Mostly Heuristics in practice

Approaches:

- Spectral
- Streaming
- Greedy
- Community Detection/Clustering
- Flows
- Local Search

METHODOLOGY

Building graphs

- Custom graph data structure
- Space O(N*D)

Static runs

- Application traverses on predefined partitions.
- ie the ordering of the graph in memory is different

Dynamic runs

- As the application processes nodes, the handler populates a page with nodes that are likely to be accessed.

3

the design

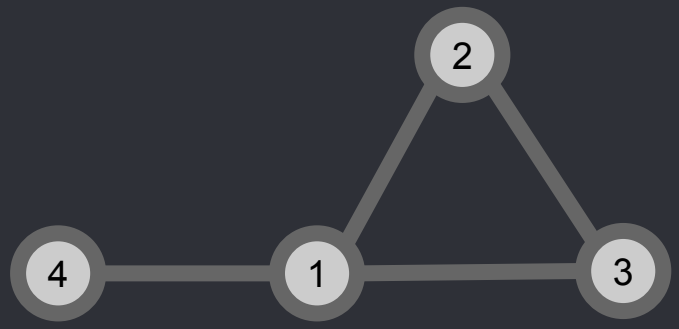
“aka poor life decisions...”

- c. beeson

DESIGN CONSIDERATIONS

0. Keep many nodes on a single page
1. Keep neighbouring nodes close
2. Each page must hold an integer number of nodes
3. Partition on the fly

GRAPH DATA STRUCTURE



HEADER

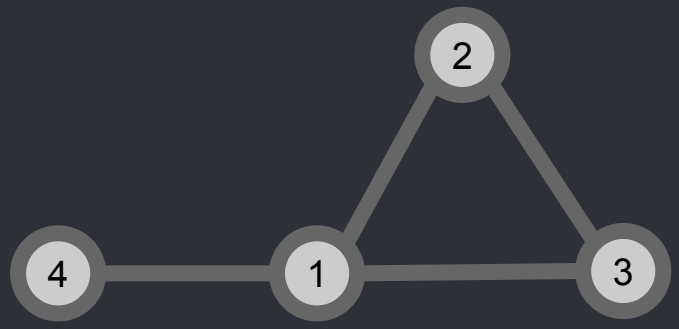
```
(4,4,4)
[(0,3)(1,2)(2,2)(3,1)]
[0:1,1:2,2:3,3:4]-----
----------------------
----------------------
----------------------
----------------------
----------------------
```

(N = # vertices
M = # edges
D = max_degree)
[(offset,degree)]
- - - - - - - - -
[offset:node]
- - - - - - - - -

Integer number of header pages

my_graph.g

GRAPH DATA STRUCTURE



NEIGHBOUR LIST

```
1:[2,3,4]
2:[1,3]
3:[1,2]
4:[1]
```

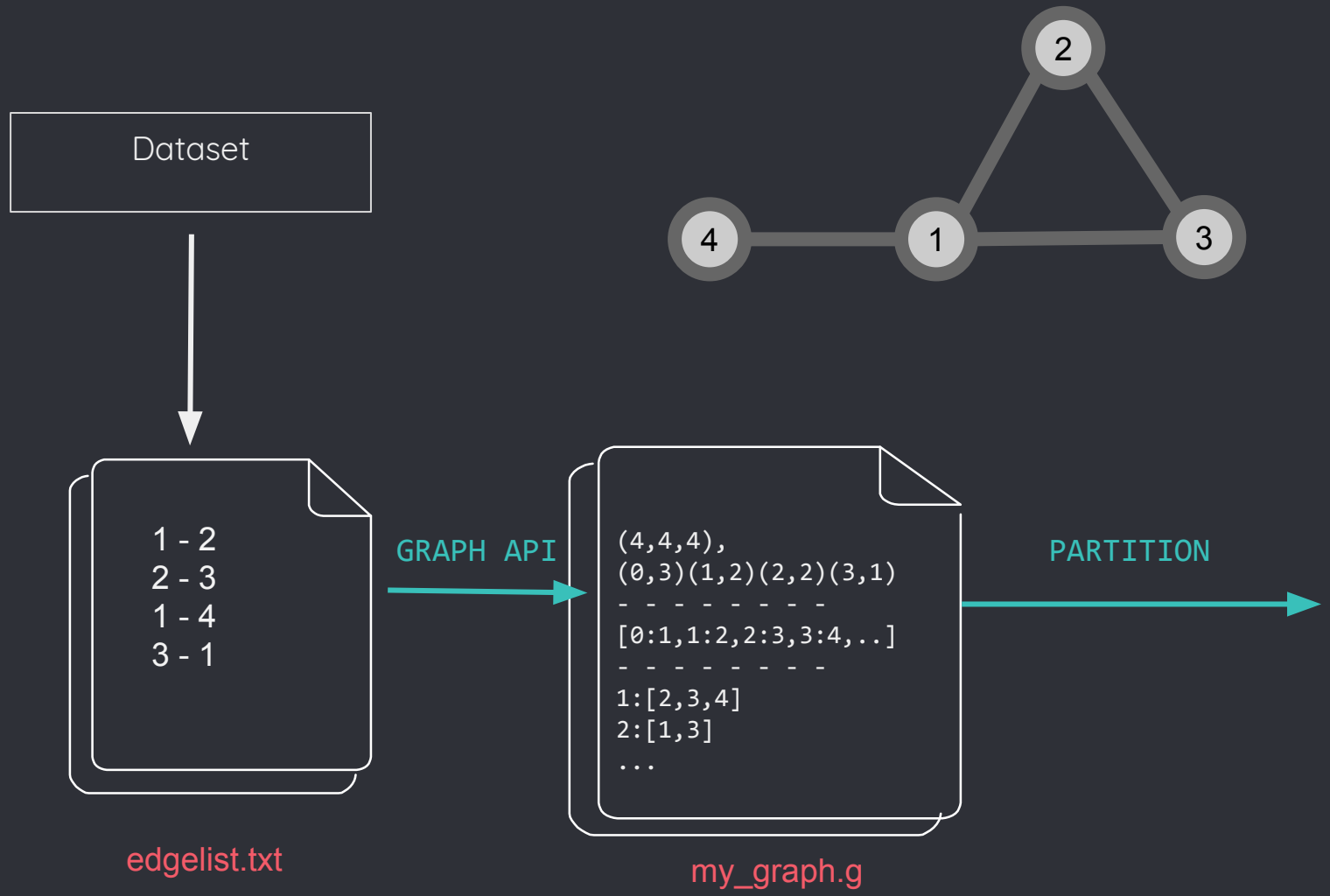
Neighbour-list

- Fixed width for easy partitioning
- Location based on offset as defined in the header

Nodes presented in offset order

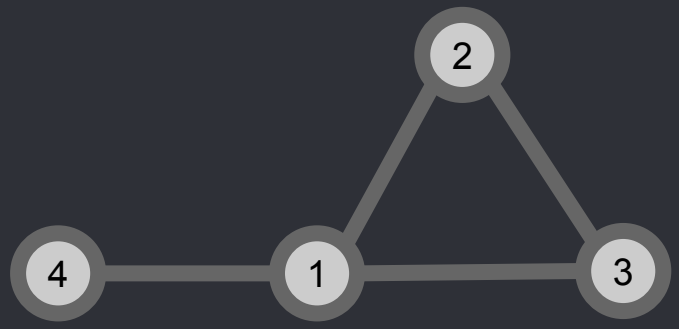
my_graph.g

PRE-PROCESSING



GRAPH DATA STRUCTURE

Random Partition



HEADER

```
(4,4,4)
[(0,3)(3,2)(2,2)(1,1)]
[0:1,1:4,2:3,3:2]-----
----------------------
----------------------
----------------------
----------------------
----------------------
```

(N = # vertices
M = # edges
D = max_degree)
[(offset,degree)]
- - - - - - - - -
[offset:node]
- - - - - - - - -

Integer number of header pages

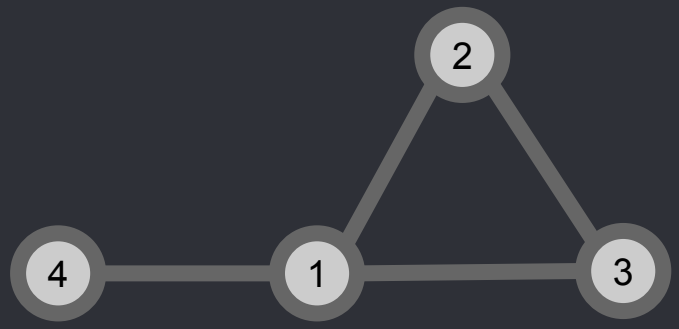
my_graph.g

GRAPH DATA STRUCTURE

Random Partition

NEIGHBOUR LIST

```
1:[2,3,4]
4:[1]
3:[1,2]
2:[1,3]
```



Neighbour-list

- Fixed width for easy partitioning
- Location based on offset as defined in the header

Nodes presented in offset order

my_graph.g

SYSTEM ARCHITECTURE: Dynamic Partitioning

Two Threads:

- One runs the application code with an anonymous region of memory

- One runs the page handler using userfaultfd to catch pagefaults from the anonymous region and has access to the graph file

ASSUMPTIONS/LIMITATIONS

- Requires Small Degree/Large Page
  - Fit multiple adjacency list per page
  - Fault across nodes not within a node
- Requires Linux kernel version > 2.4
  - (Because we use Userfaultfd)
- Userfaultfd Only Supports Anonymous, Huge, or Shared Memory Regions
- Applications Need to be Modified

4

the results

“plots not graphs”

- c. beeson

THE EXPERIMENT

Data

- Star graph (worst case space efficiency)
- K-complete graph (best case space efficiency)
- Facebook graph (real world dataset)

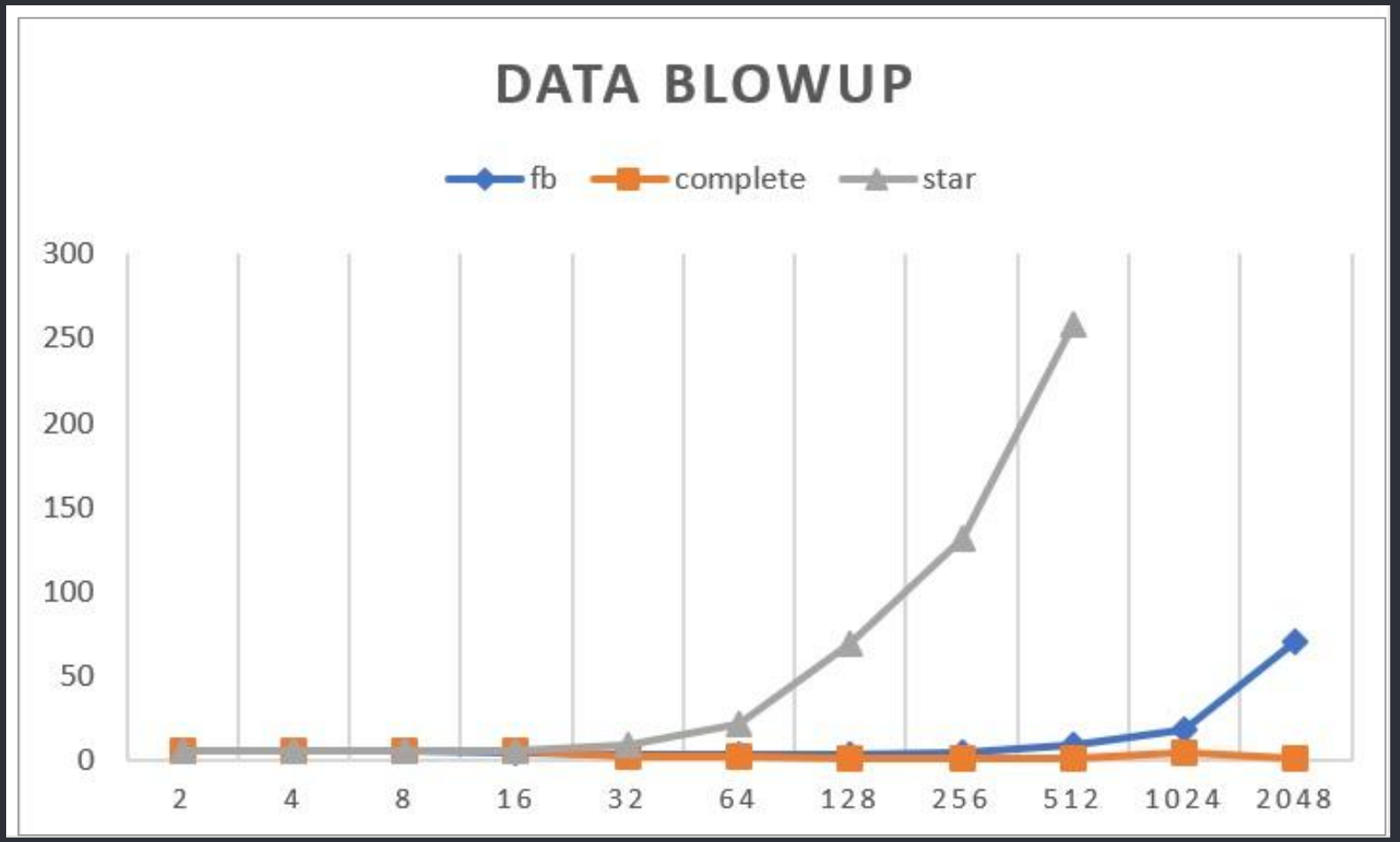
Partition

- Breadth/Depth first order
- Random order

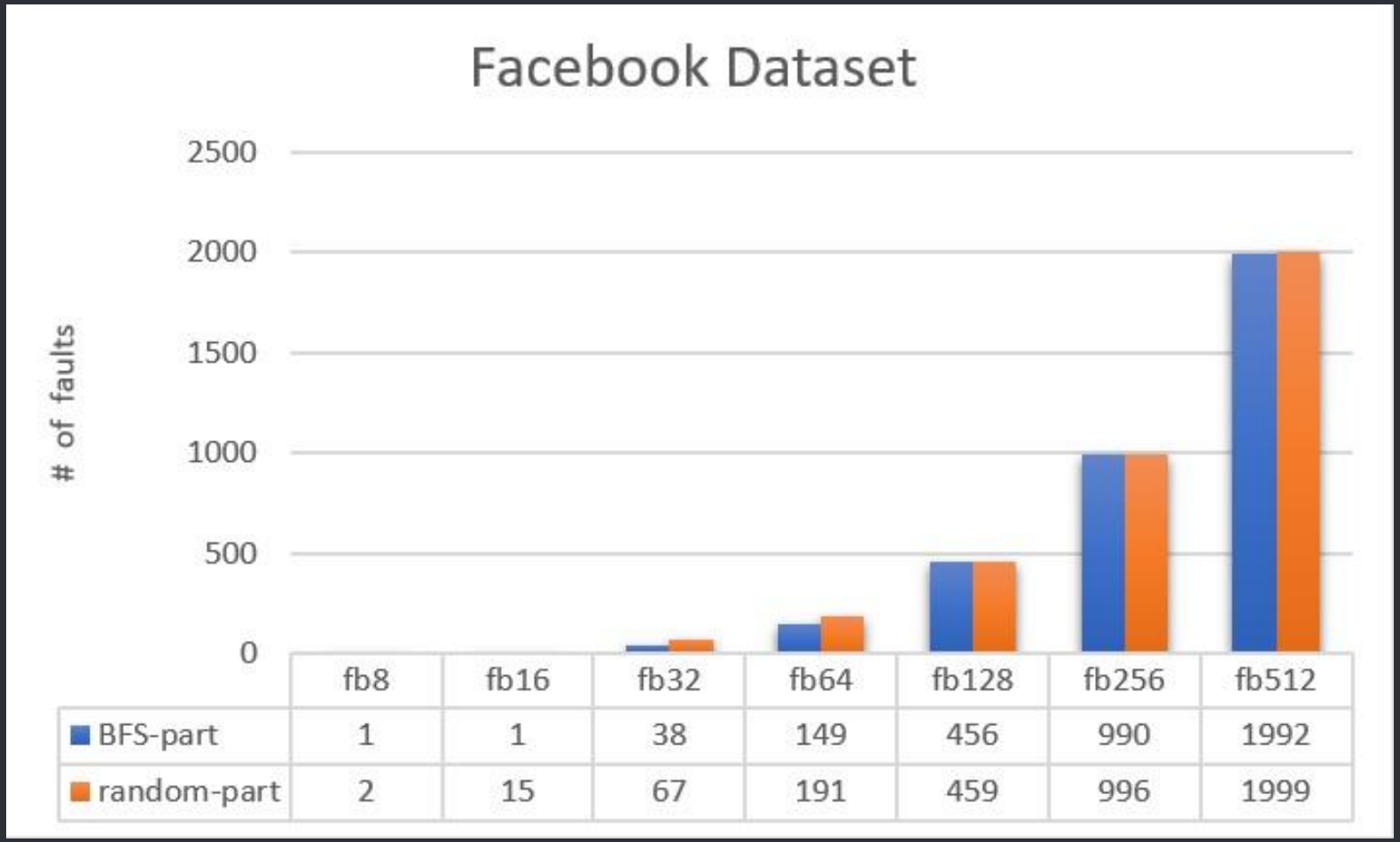
Application

- Breadth/Depth first traversal

DATA BLOW-UP

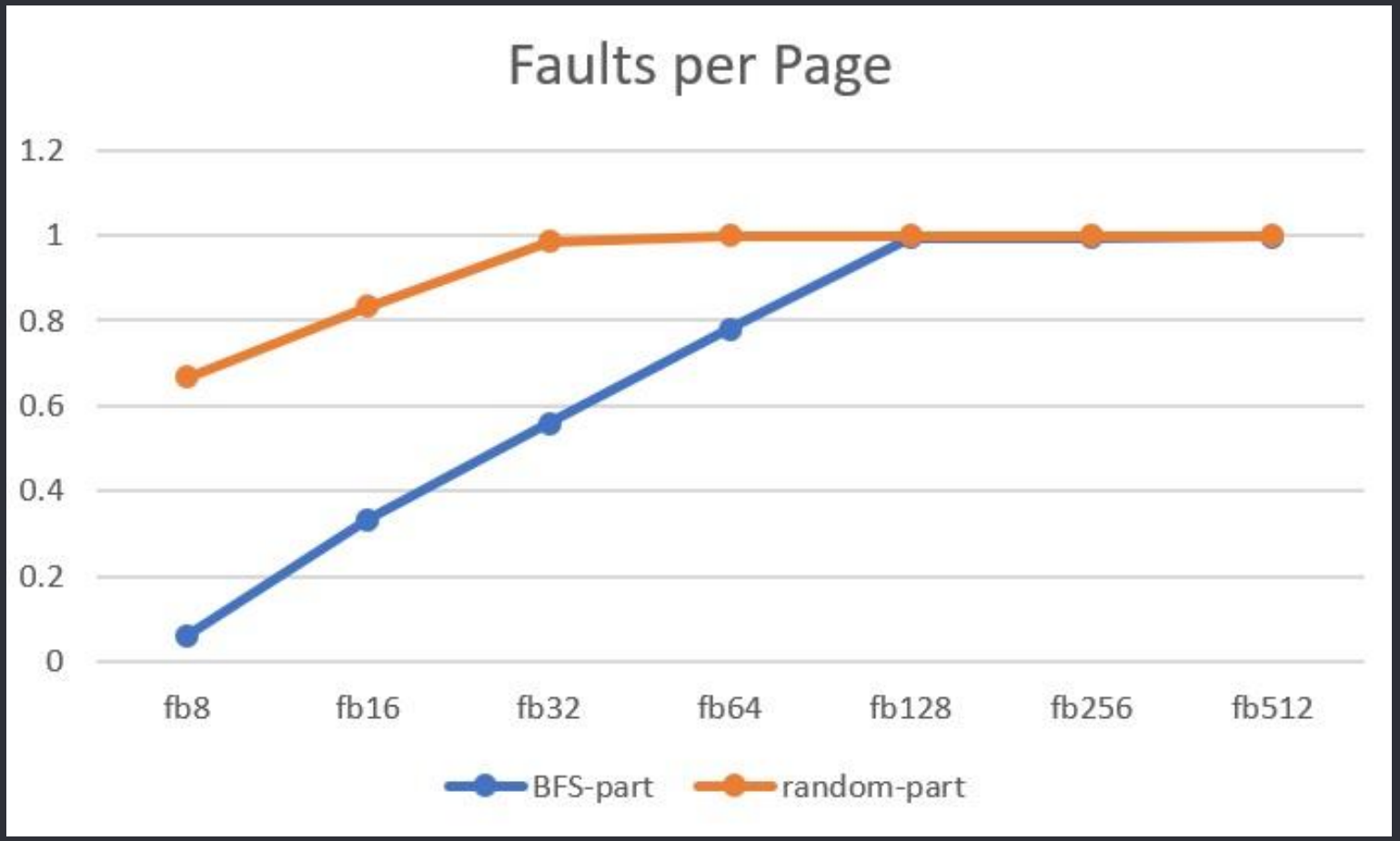

DATA BLOWUP
fb
complete
star
300
250
200
150
100
50
0
2
4
8
16
32
64
128
256
512
1024
2048

FACEBOOK PARTITION: RANDOM VS BFS/DFS



	fb8	fb16	fb32	fb64	fb128	fb256	fb512
BFS-part	1	1	38	149	456	990	1992
random-part	2	15	67	191	459	996	1999

FACEBOOK PARTITION: RANDOM VS BFS/DFS



THE SET UP

Oracle VM VirtualBox

Linux Ubuntu 16.04 i386 (64-bit)

1GB RAM, single-core CPU

5

the discussion

*“stop asking me for quotes for this sh*t”*

- c. beeson

IF YOU WANT TO BUILD ON THIS...

Looking Forward:

- Different Partitions
- Multipass Applications
- Different Graphs

Looking Back:

- Don’t continuously change indexes
- Know the scale of data before hand
- Know the tools* (mmap, userfaultfd, ioctl)

*Shout out to Amanda and Tony and Surbhi (and the NSS lab) for tolerating our questions

QUESTIONS

Quotations are commonly printed as a means of inspiration and to invoke philosophical thoughts from the reader.

DEMO