MaiterStore: A Hot-aware, High-Performance Key-Value Store for Graph Processing

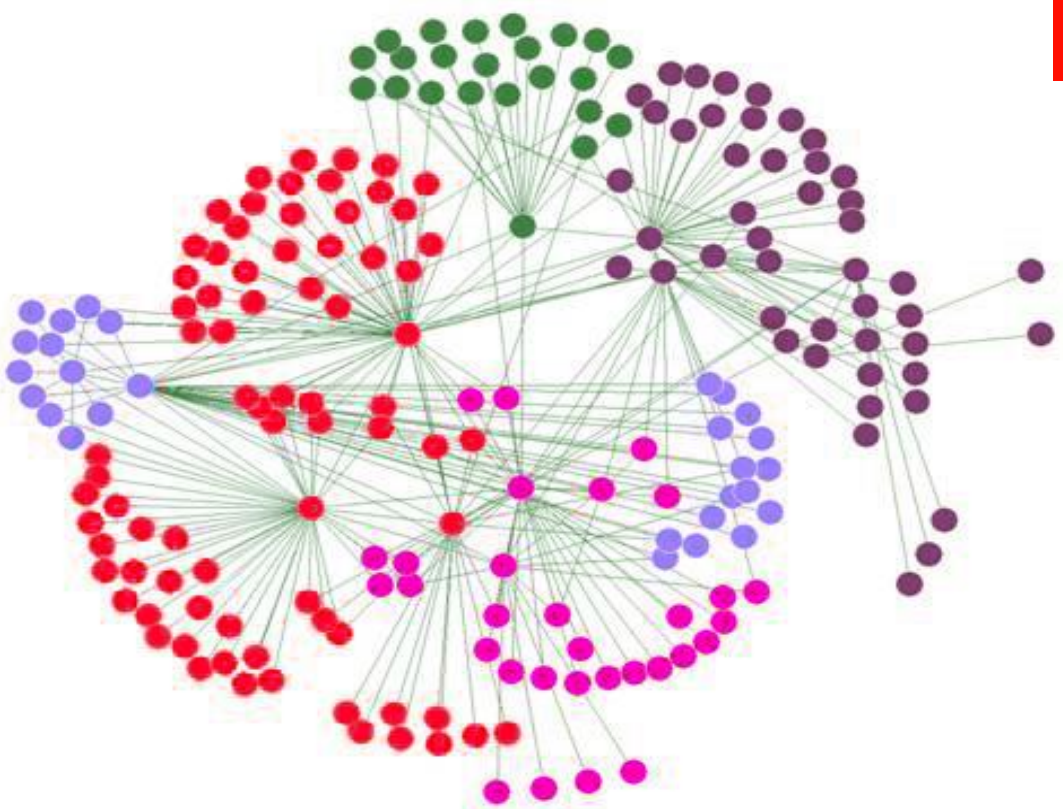
Dong Chang, Yanfeng Zhang, **Ge Yu**
Northeastern University, China

Outline

- Introduction
- Preliminaries
- System Design & Implementation
- Experiments
- Conclusion & Future work

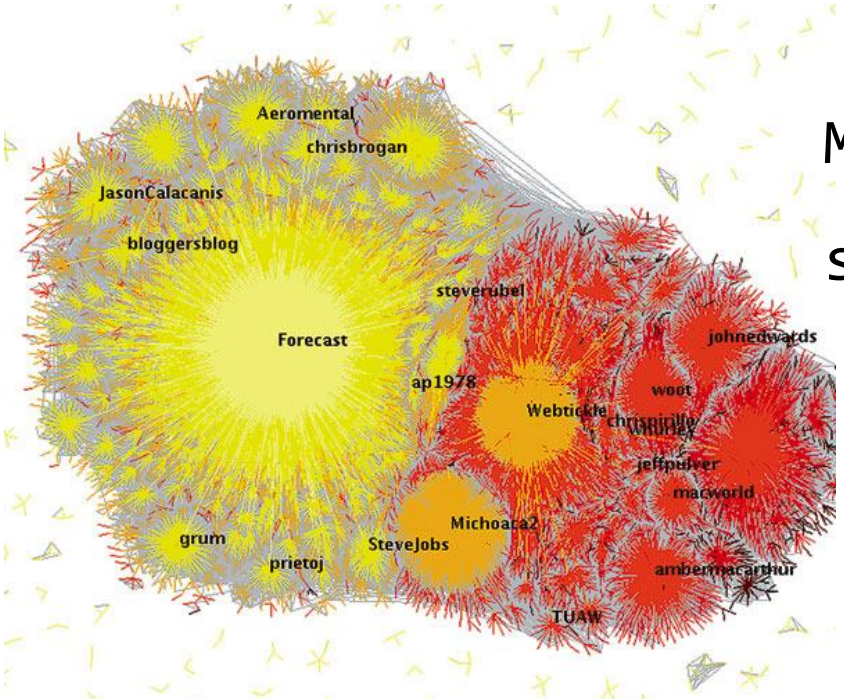
Introduction

- Big graphs are everywhere



Introduction

- Natural Graphs



Twitter network visualization,
by Akshay Java, 2009

Data size is really **big**

Many graph algorithms involves substantial **random** access.

Distributed in-memory graph computation system

Introduction

- Distributed in-memory graph processing systems

GraphLab

Spark

Maiter

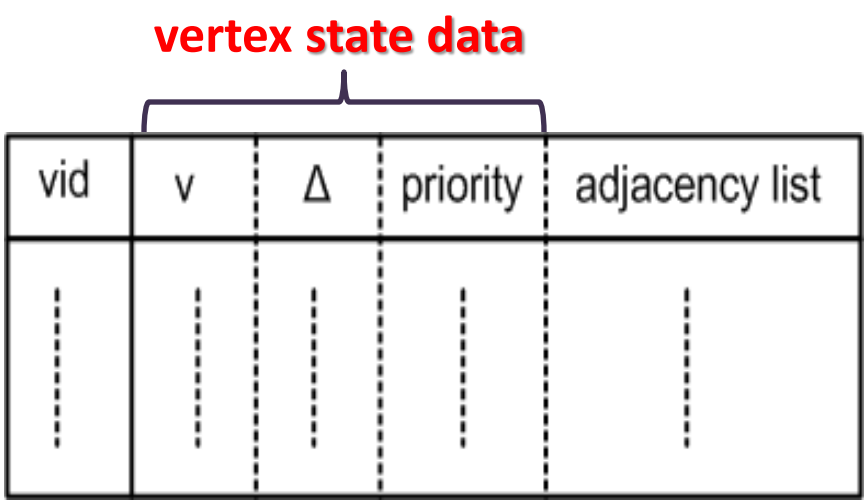
real world graphs: too large

DRAM: steep price & relatively small capacity

hot-spot property & **skewed power-law degree distribution:** not memory-efficient

Preliminaries

Maiter –A distributed in-memory graph computation system based on delta-based accumulative iteration computation.



Maiter state table on each worker

Vertex state data:
volatile but smaller

Vertex adjacency list:
immutable but much larger

Vertices with **higher priority** are prone to be the hot-spot of the graph.

Preliminaries

- In Maiter, gigantic graph data **exhausts** the limited memory of each worker.

How to deal with such situation?

Consideration:

1. Random access for vertices
2. Vertices may not fit into memory

SSD/Flash

Preliminaries

SSD(Solid Disk Driver)

- Faster than disk(esp. ***fast random read access speed***), cheaper than DRAM
- Reads and writes happen at the granularity of a ***flash page***
- **Random writes** are slow and bad for flash life (erase-before-write)

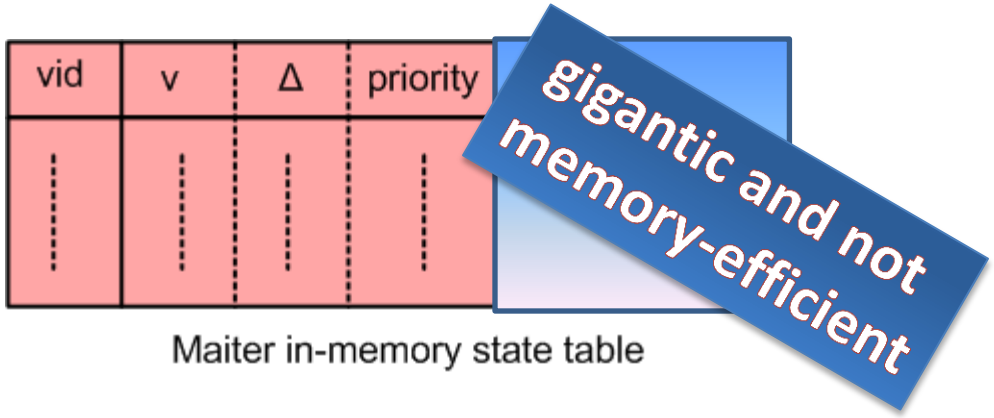
System Design & Implementation

- In Maiter, gigantic graph data **exhausts** the limited memory of each worker.

System Design & Implementation

MaiterStore

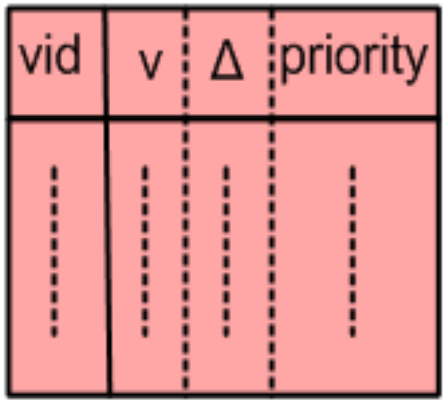
MaiterStore goal: destage the huge **static graph-structured data** to SSD while retaining Maiter's high performance.



System Design & Implementation

MaiterStore

MaiterStore goal: destage the huge **static graph-structured data** to SSD while retaining Maiter's high performance.

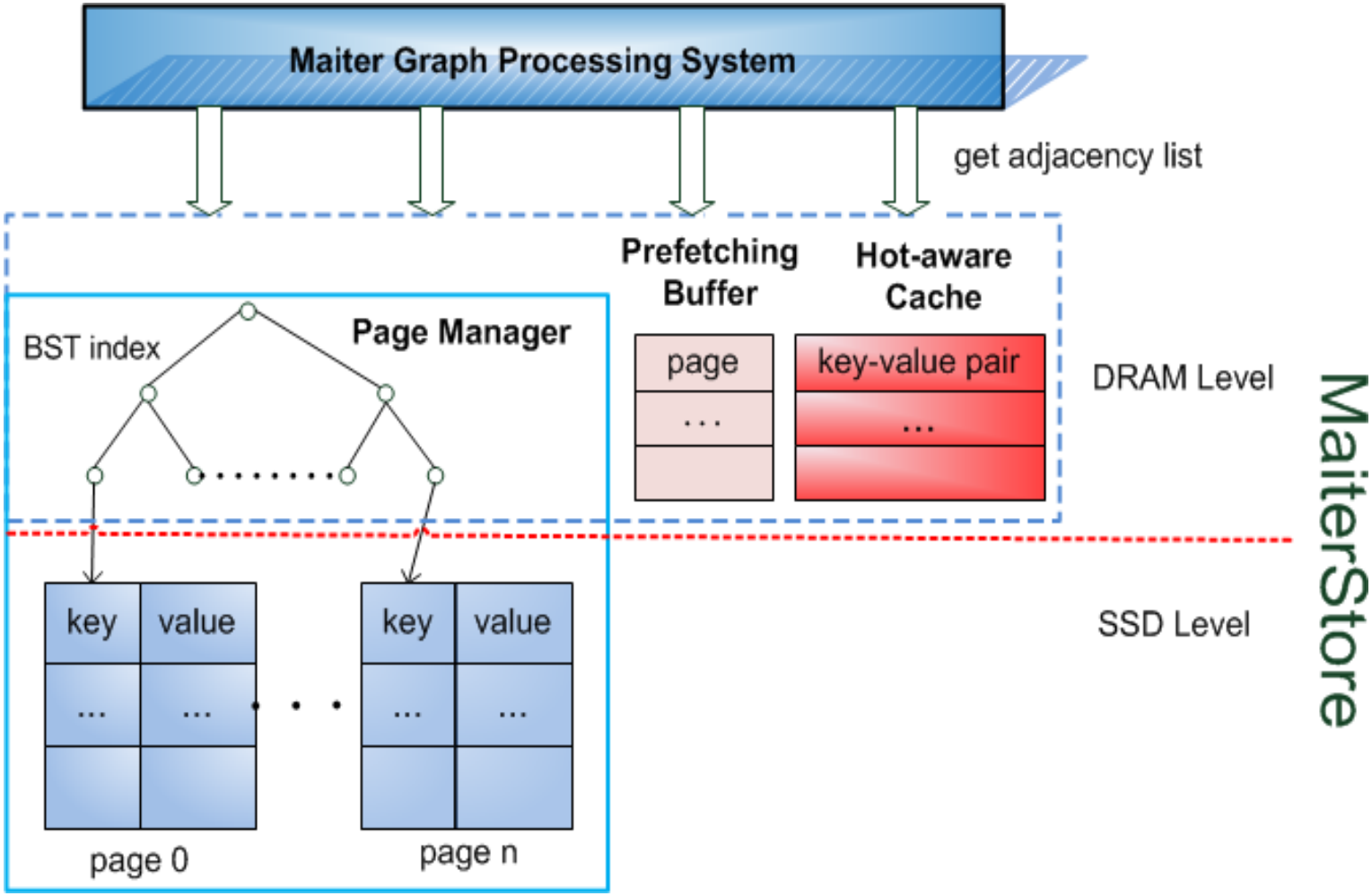


graph state data in **memory**

graph structure data on **SSD**

System Design & Implementation

MaiterStore Architecture



MaiterStore Components

vid	priority	others
1	5.0	. . .
2	1.5	. . .
3	5.0	. . .
4	2.0	. . .
5	1.0	. . .
6	1.5	. . .

Maiter State Table

DRAM

Get adjacency list of vertex **1**

Hot-aware Cache

missing

MaiterStore

Prefetching Buffer

Page

Page

missing

Read pages from SSD when missing

SSD

Page Manager

1	2 3 4
2	3

Page 0

3	1 6
4	1 3

Page 1

5	3
6	1

Page 2

MaiterStore Components

vid	priority	others
1	5.0	. . .
2	1.5	. . .
3	5.0	. . .
4	2.0	. . .
5	1.0	. . .
6	1.5	. . .

Maiter State Table

DRAM

Get adjacency

list of vertex **1**

Hot-aware Cache

MaiterStore

Prefetching Buffer

1	2 3 4	Page 0
2	3	
3	1 6	Page 1
4	1 3	

Read pages from SSD when missing

SSD

Page Manager

1	2 3 4
2	3

Page 0

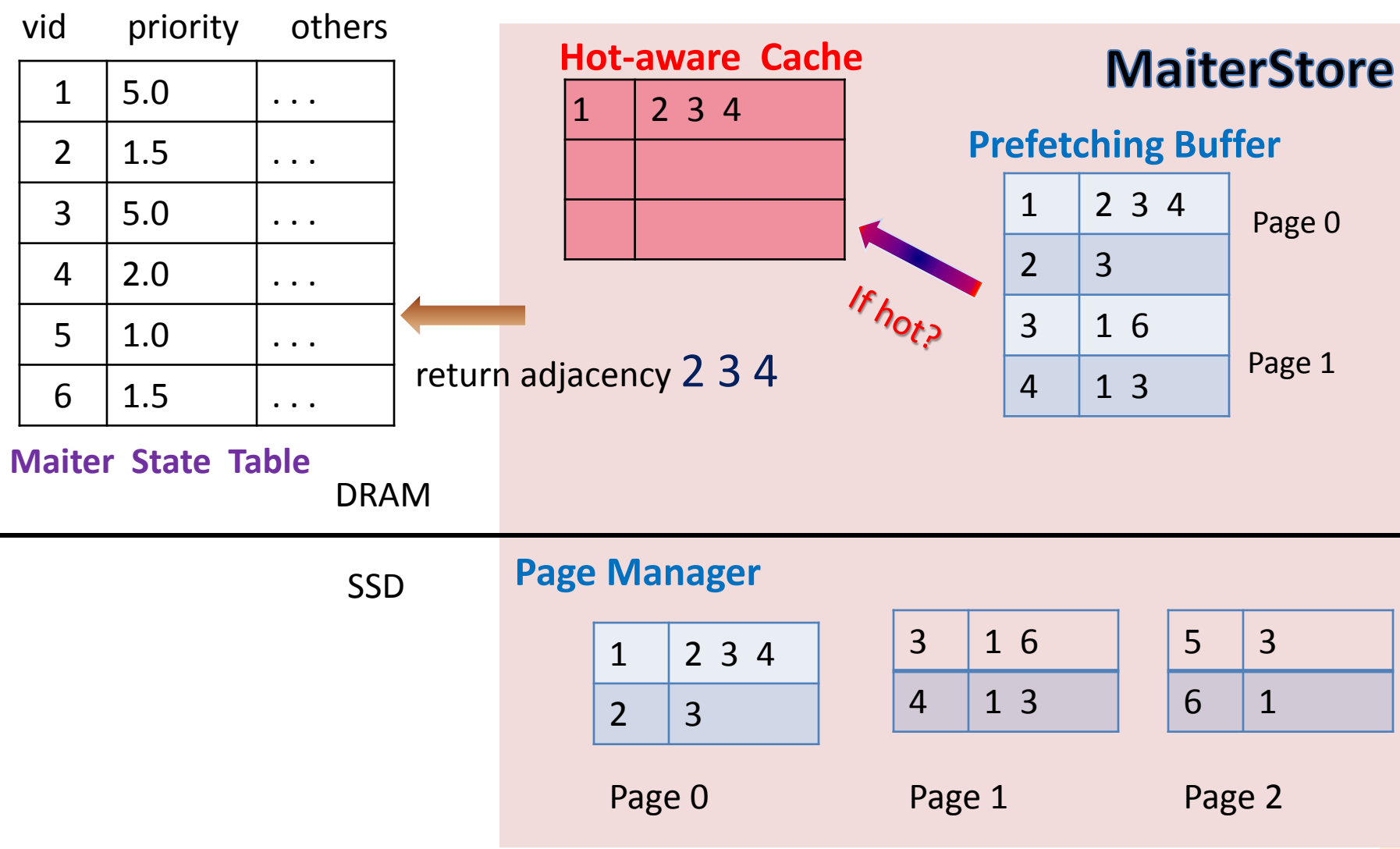
3	1 6
4	1 3

Page 1

5	3
6	1

Page 2

MaiterStore Components



MaiterStore

- **Page Manager**

  equip the *static graph-structured data* as key-value pairs and store them on SSD in units of page.

- **Prefetching Buffer**

  page-based storage structure for fetching the to-be-accessed pages from SSD

MaiterStore

- **Hot-aware Cache (HAC)**

keep those hot and performance-critical vertices in memory.

The hotness of the vertex:

1. priority field in Maiter state table
2. according to graph algorithm, e.g. PageRank value

(hotness, vid, adjlist)

50	5	adjlist5
40	4	adjlist4
30	3	adjlist3
20	2	adjlist2
10	1	adjlist1

request (hotness, vid) (35, 6)

50	5	adjlist5
40	4	adjlist4
35	6	adjlist6
30	3	adjlist3
20	2	adjlist2

request (80, 4)

80	4	adjlist4
50	5	adjlist5
35	6	adjlist6
30	3	adjlist3
20	2	adjlist2

MaiterStore API

- User defined function

```
virtual void parseKV(string line, K*vid, V*adjList) = 0;
```

- MaiterStore system API

```
V get(const K vid);
void put(const K vid, const V adjList);
```

Experiments

- Environment Setting
  - H/W:

    A local cluster of 4 commodity PCs.

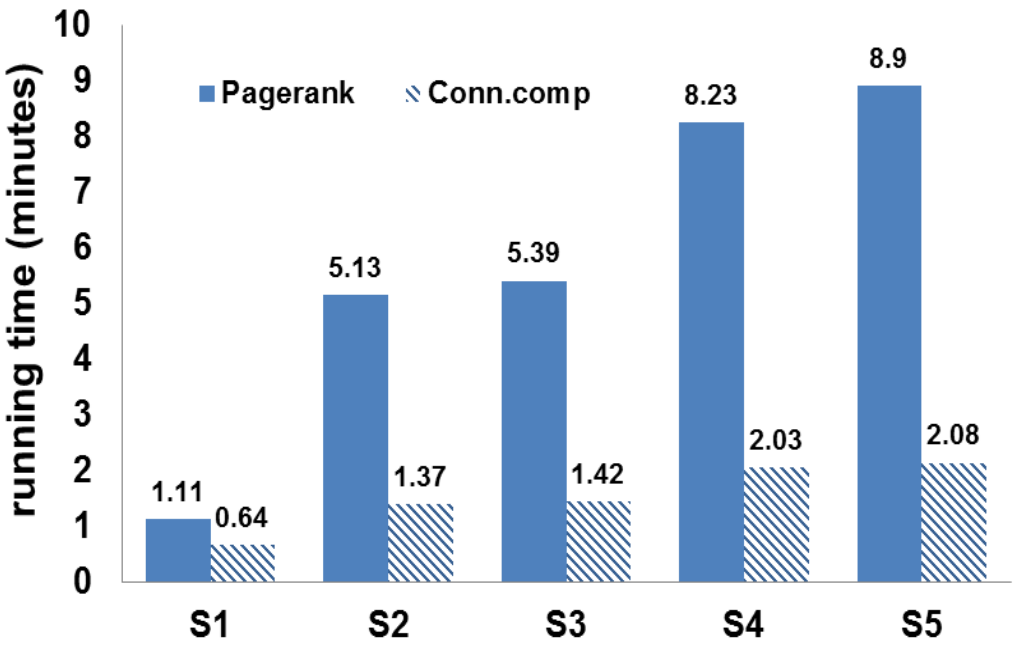
    Amazon EC2 Cloud of 30 machines.
  - Data: general graphs

Dataset	Nodes	Edges	Source
Web-Google(55MB)	916,428	6,078,254	*Stanford snap dataset*
Web-BerkStan(62MB)	685,230	7,600,595	*Stanford snap dataset*
Web Graph(11GB)	50,000,000	686,231,717	clueweb09

- PageRank & Connected Components

Experiments

- Performance of MaiterStore

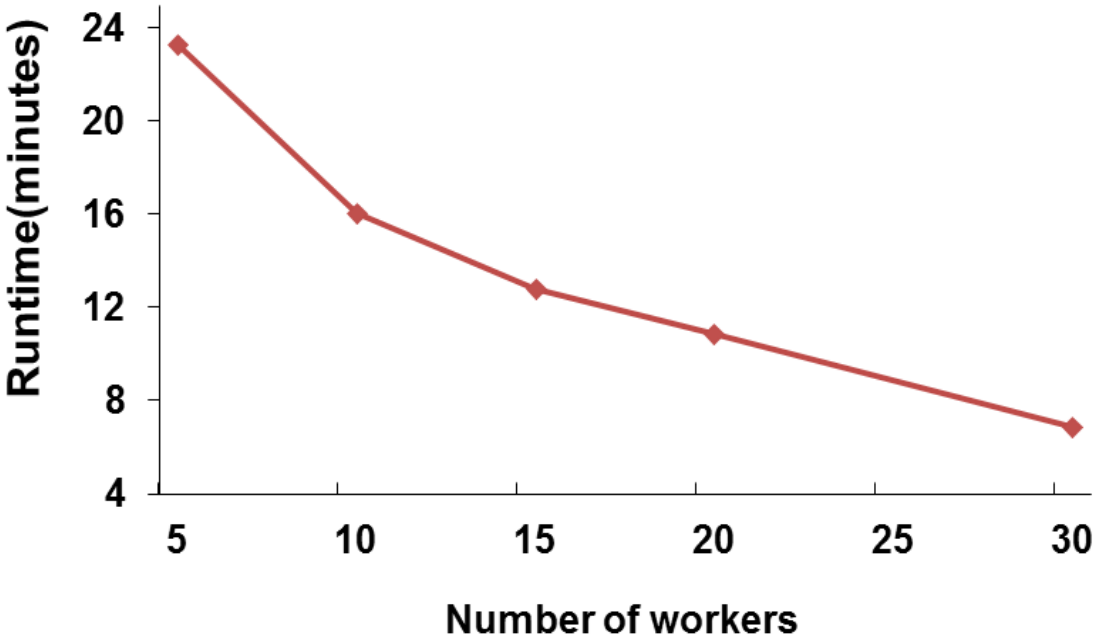


Comparison of running time under different settings

S1: in memory only
S2: in SSD + Prefetching Buffer + HAC, i.e., MaiterStore
S3: in HDD + Prefetching Buffer + HAC
S4: in SSD + Prefetching Buffer
S5: in HDD + Prefetching Buffer
S6: in SSD only (67.05mins Pagerank)
S7: in HDD only (89.86mins Pagerank)

Experiments

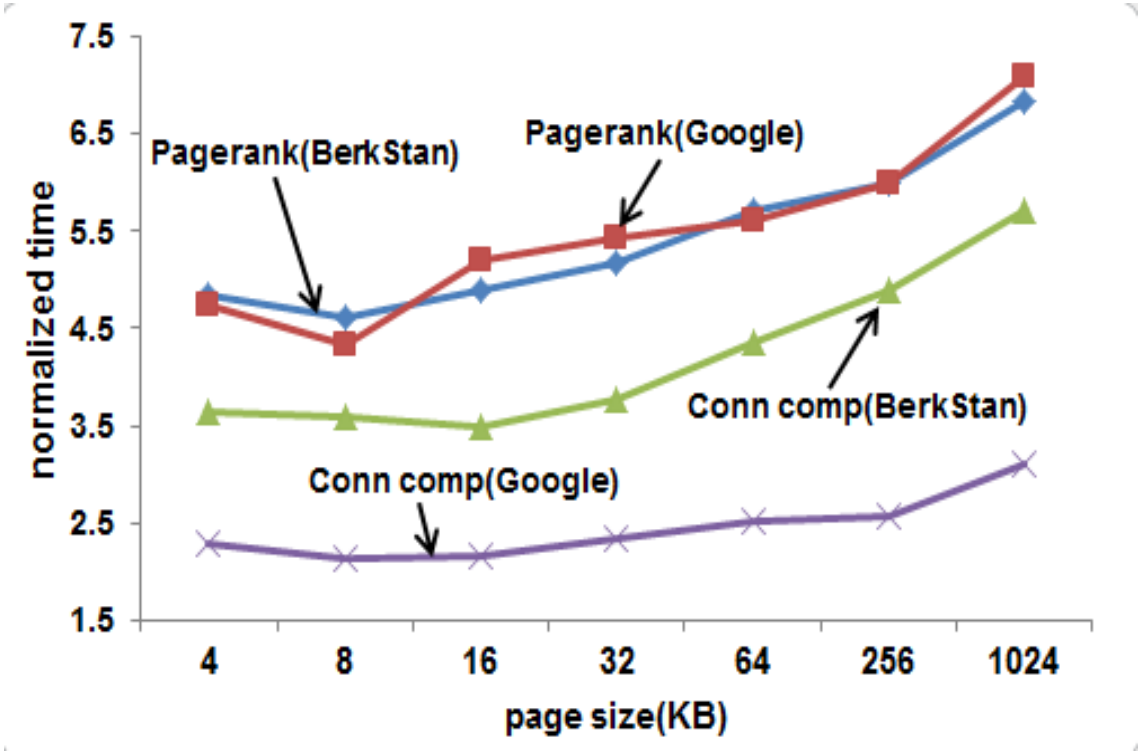
- Scalability of MaiterStore



Connected Components: varying number of workers on EC2 cluster

Experiments

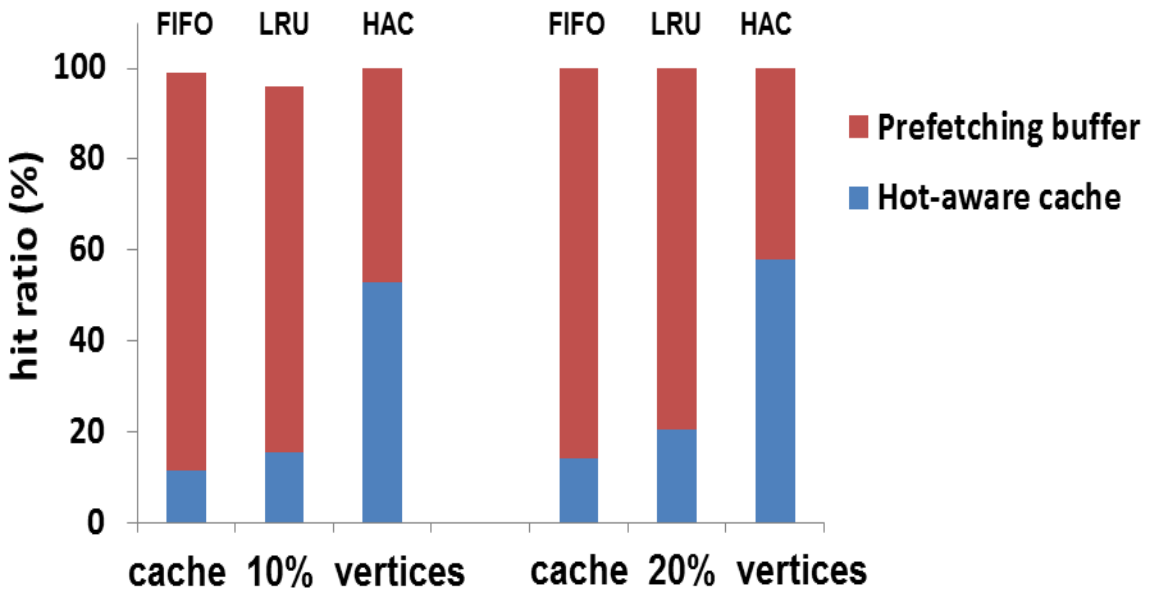
- Effect of the Page Size



Effect of page size on performance

Experiments

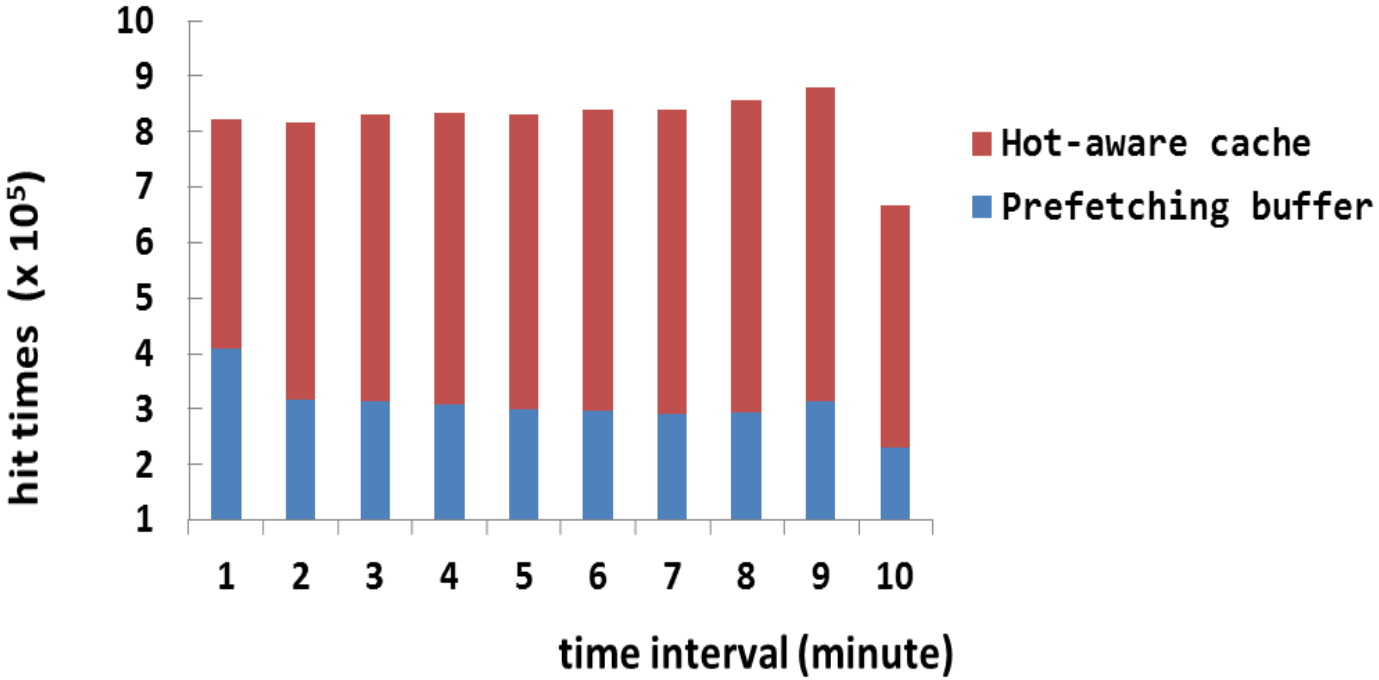
- Comparison with the HAC VS. FIFO VS. LRU



Effect of cache size under HAC, FIFO and LRU

Experiments

- Effect of the HAC and Prefetching Buffer



Impact of prefetching buffer and HAC

Conclusion

- Conclusion

  MaiterStore is specialized for large-scale graph storage framework with efficient **prefetching buffer** and **hot-aware cache**.

- Future work

  Apply MaiterStore to other in-memory graph processing frameworks based on key-value table beyond Maiter.

Thank You!