XPGraph: XPline-Friendly Persistent Memory Graph stores for Large-Scale Evolving Graphs

Rui Wang[†], Shuibing He[†], Weixu Zong[†], Yongkun Li[‡], Yinlong Xu[‡]

[†] Zhejiang University

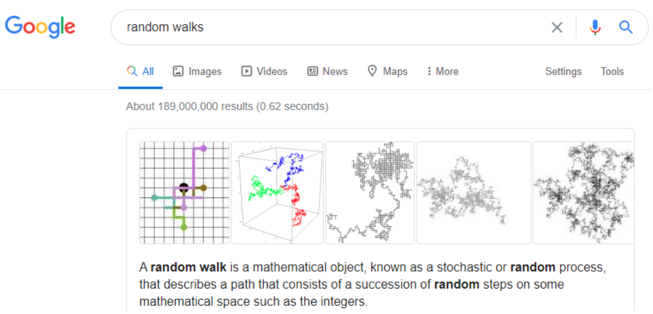
[‡] University of Science and Technology of China

IEEE MICRO 2022

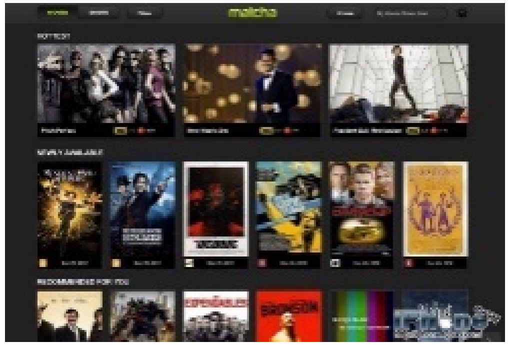
Graph applications

- Graphs are widely used in many applications

Social networks



Webpage links



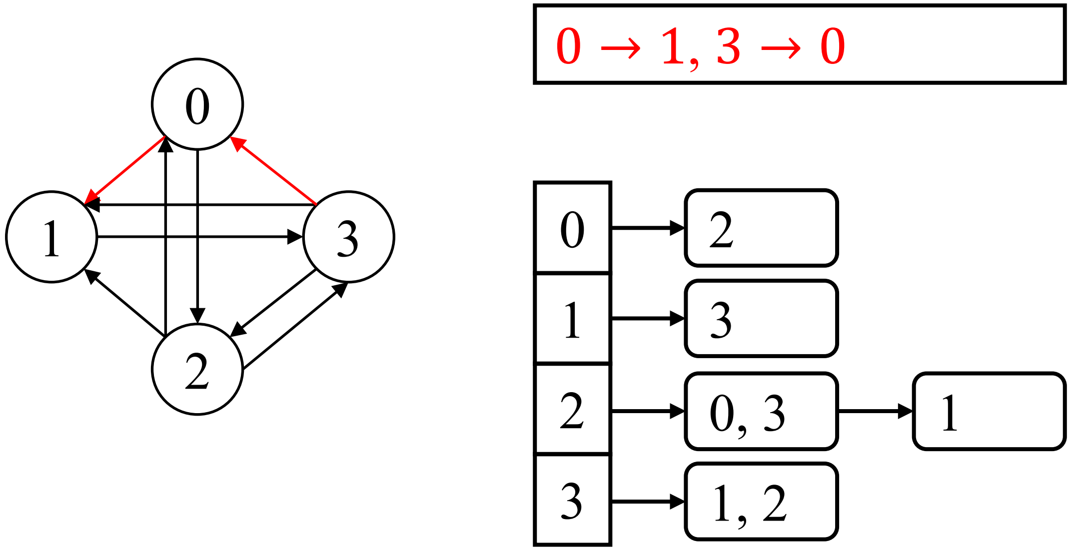
Recommendation systems

Graph analytics is one of the top 10 data and analytics technology trends [1]

[1] Gartner 's top 10 data and analytics technology trends for 2019 and 2021

Common graph storage formats for evolving graphs

- Edge list format and adjacency list format



→ Edge list format for fast edge ingesting

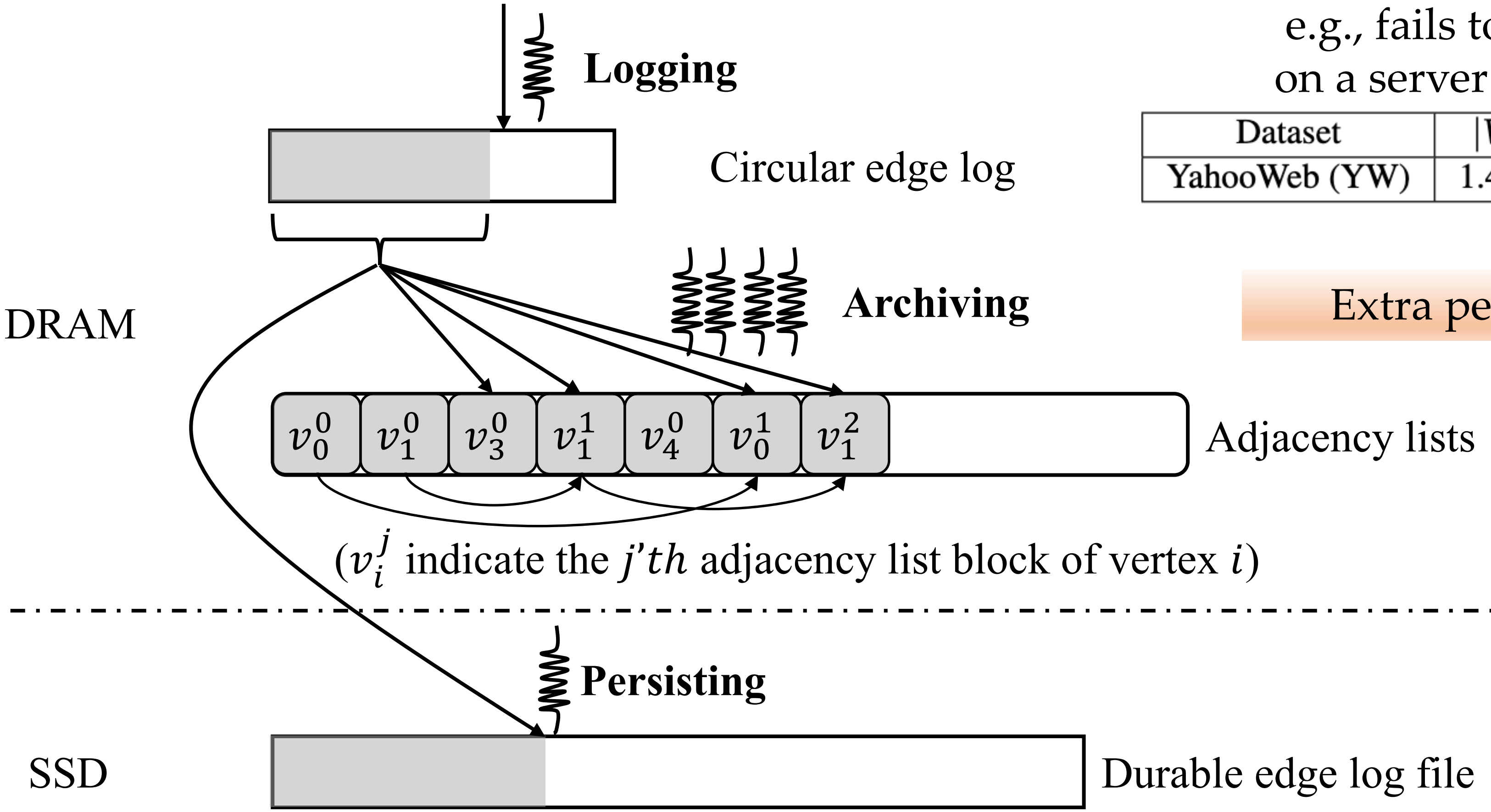
→ Adjacency list format for efficient vertex query

Hybrid store in SOTA memory graph storage systems, e.g., GraphOne[2]

[2] Pradeep Kumar and Howie Huang. Graphone: A data store for real-time analytics on evolving graphs. FAST 19

SOTA memory graph storage system – GraphOne[FAST19]

- Hybrid store model



Limited scalability

e.g., fails to run on YahooWeb on a server with 128GB DRAM

Dataset	$\|V\|$	$\|E\|$	Bin Size	CSR Size
YahooWeb (YW)	1.4B	6.6B	52.8GB	75.2GB

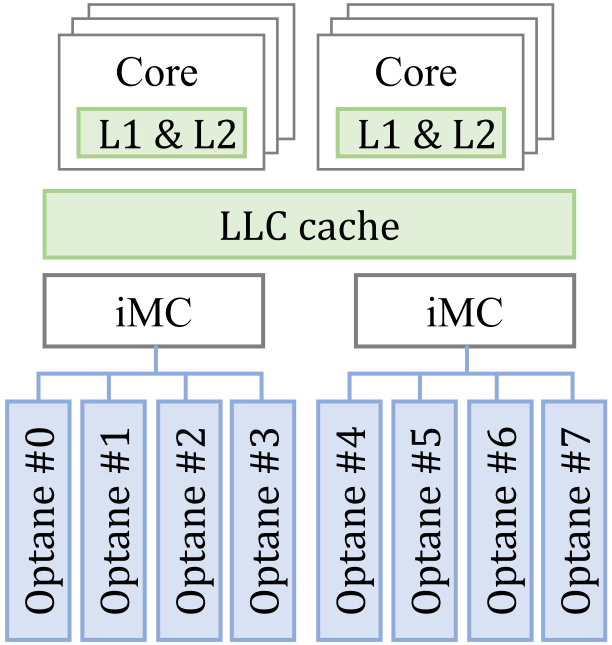
Extra persist cost

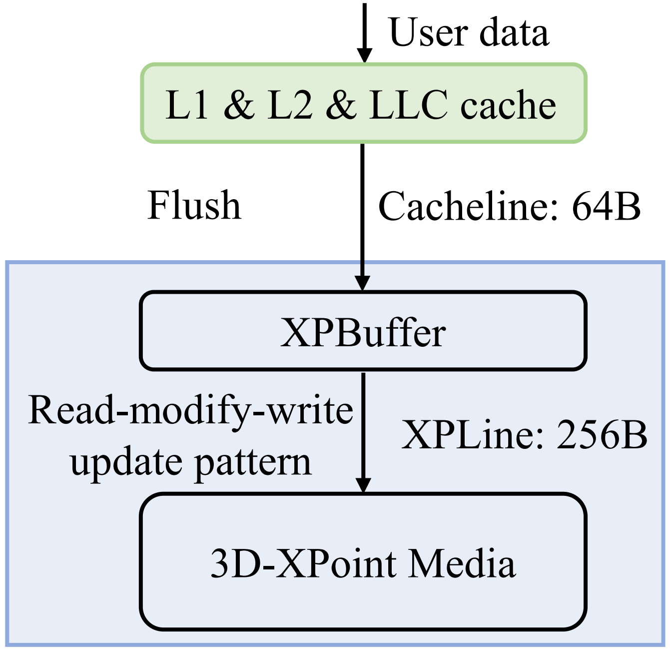
Emergency persistent memory (PMEM)

- Emergency PMEM
  - Larger capacity and non-volatility

→ **Provides us an opportunity to realize the scalable and high-performance graph stores.**

- Intel® Optane™ Persistent Memory 200 Series





Differences between DRAM:

1. Flush for persistence (in 64B cacheline size)

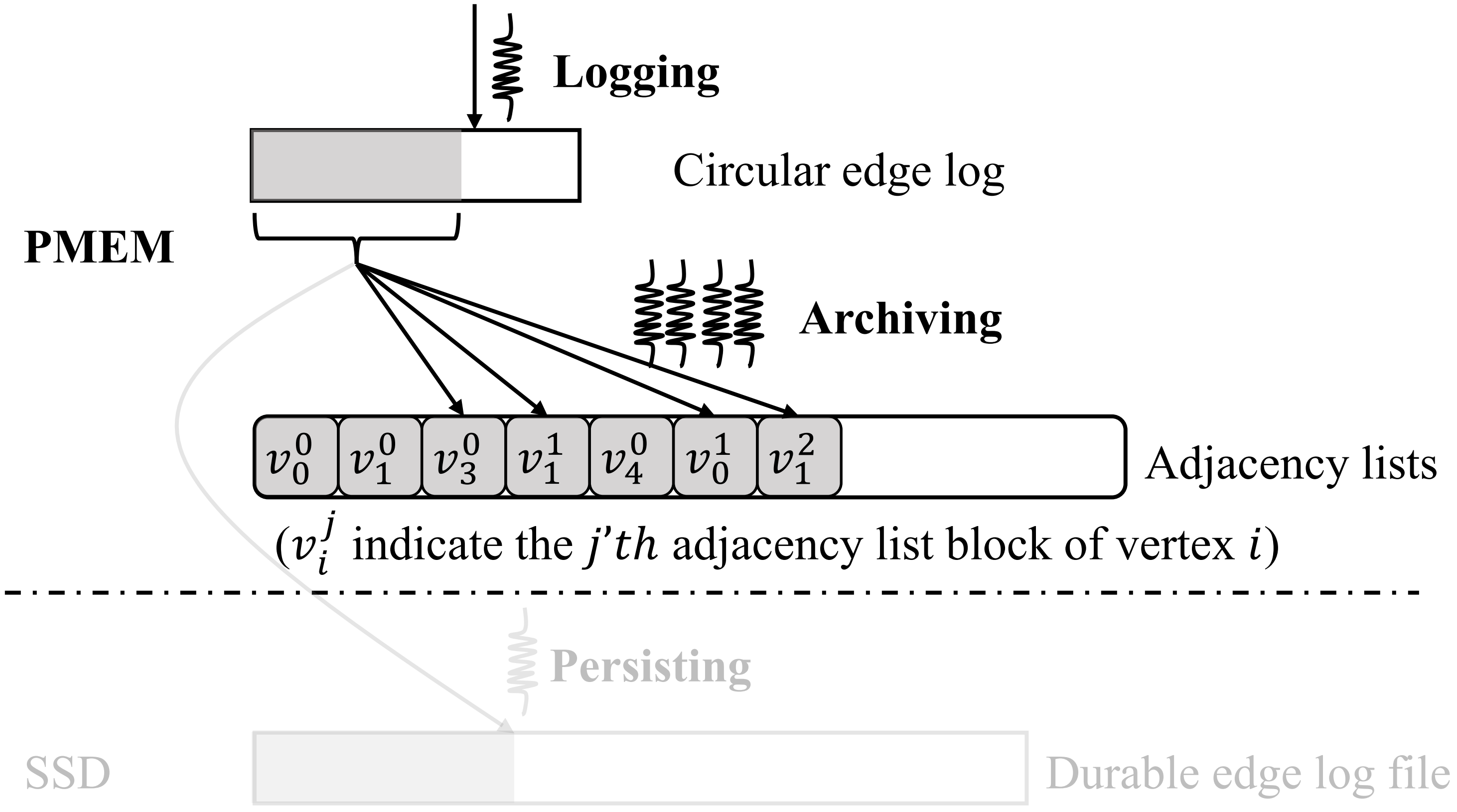
2. Physical access granularity is 256B (XPLine)

3. Read-modify-write update pattern

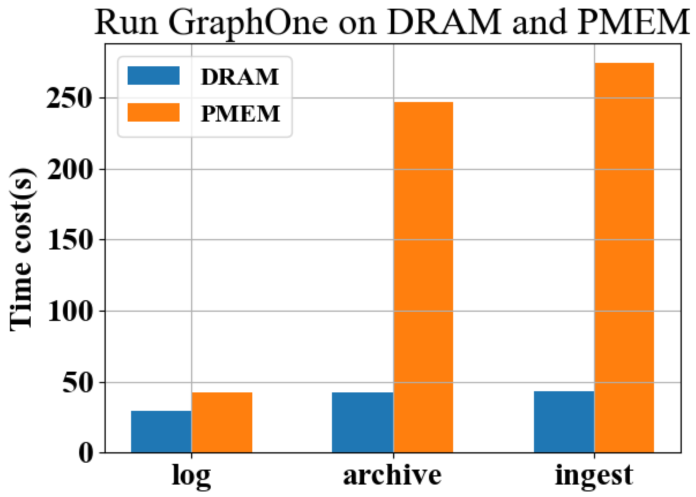
4. High cost for cross NUMA access

Migrate DRAM-based graph stores to PMEM

- GraphOne on PMEM



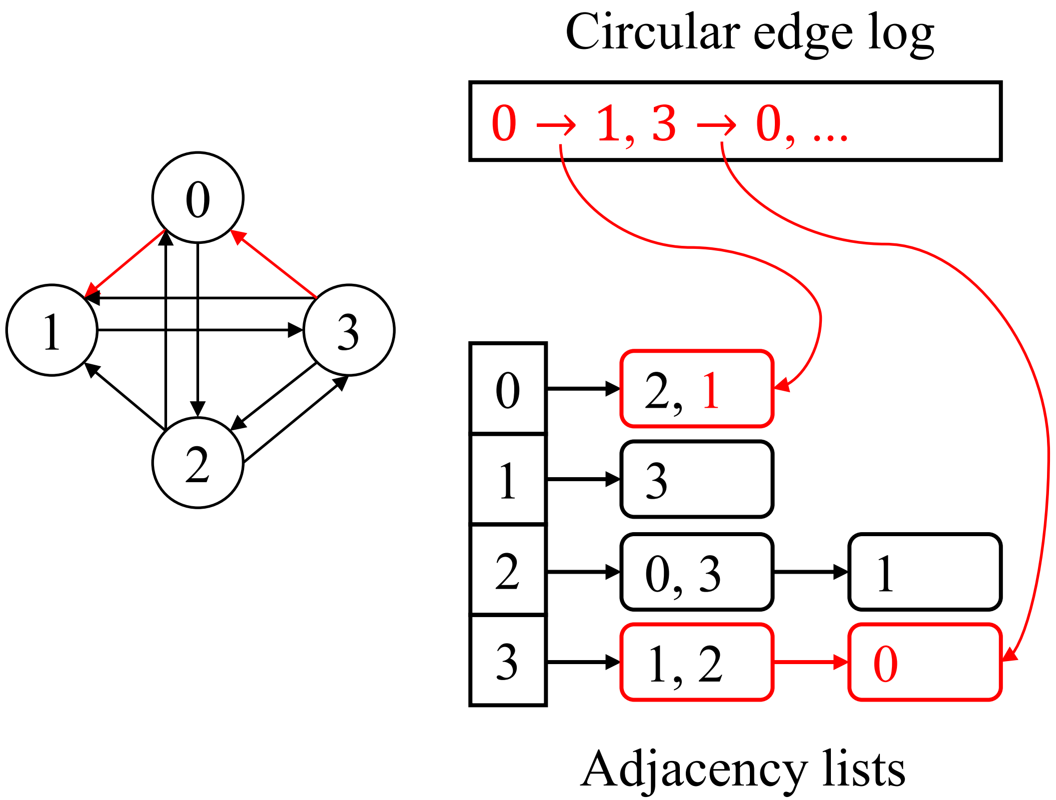
- **Large performance drop**
  - Import Friendster dataset



1.45x 5.88x 6.37x

Migrate DRAM-based graph stores to PMEM

- **Reasons**: Edge-centric global batched archiving + Remote PMEM accesses



⇨

Plenty of dense small random writes **(4B)**

⇩

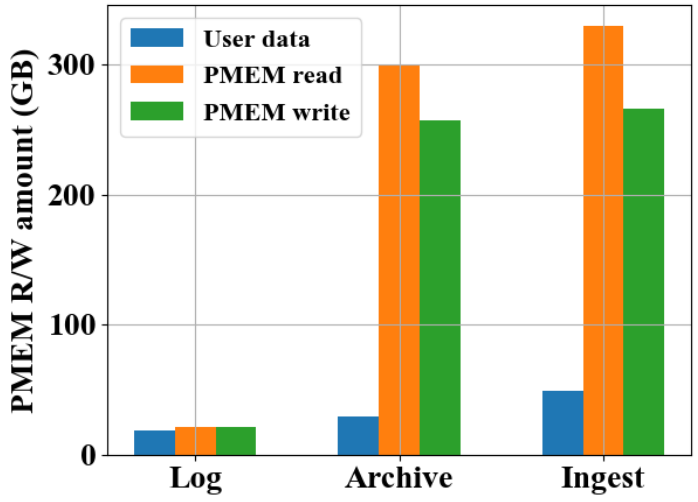
Each small write may cause an XPLine **(256B)** "read-modify-write" to PMEM

Remote PMEM accesses across NUMA nodes

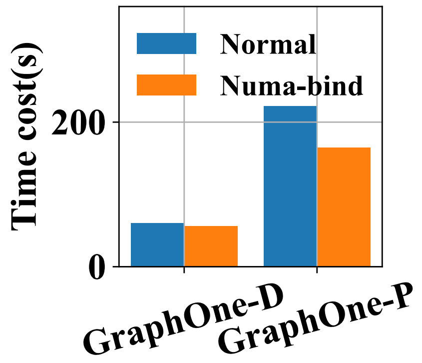
Migrate DRAM-based graph stores to PMEM

- Experimental validation -- Run GraphOne on Friendster

High read/write amplification in PMEM



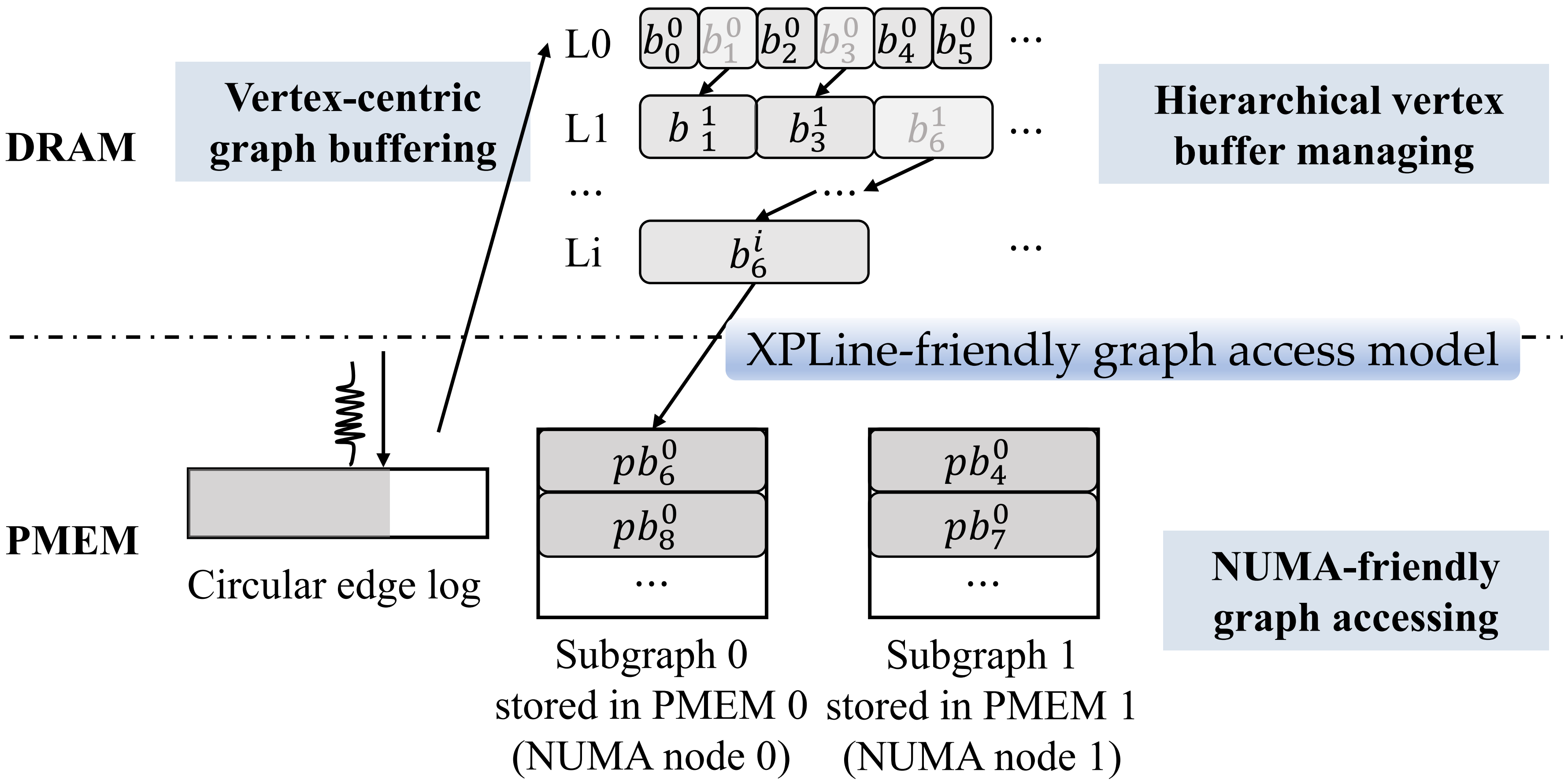
Costly remote PMEM accesses across NUMA nodes



Target: Reduce PMEM read/write cost for dynamic graph stores

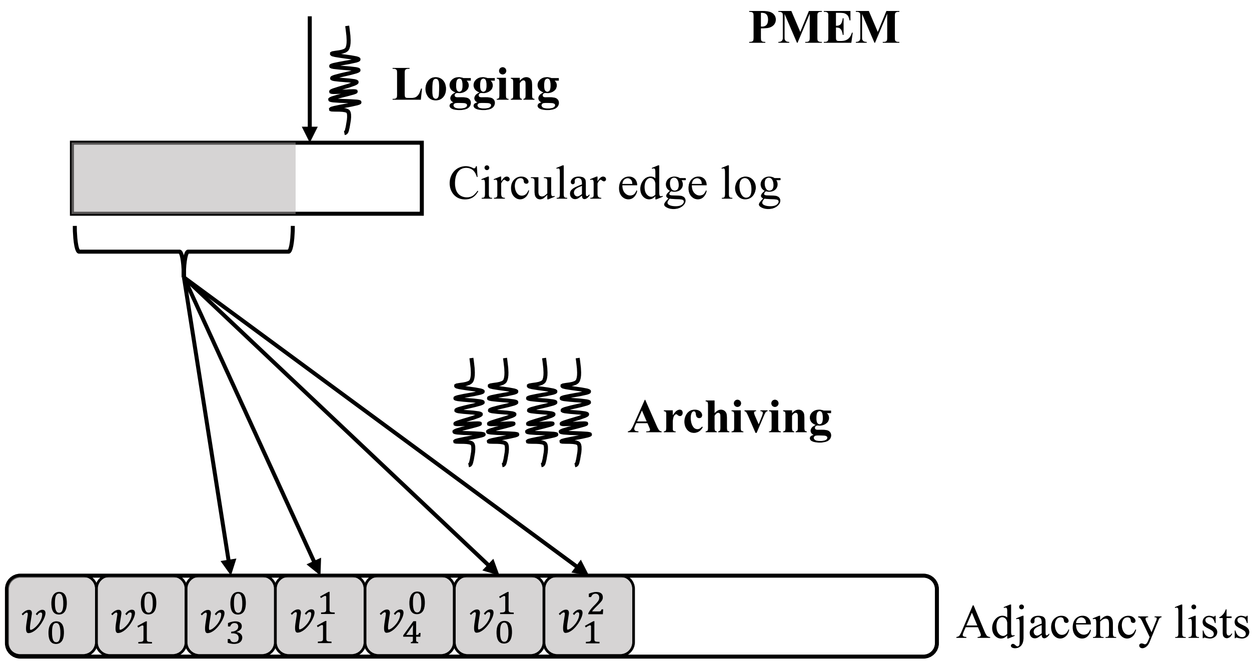
Idea: XPLine-friendly graph access model

- According to ``XPLine-centric access pattern" of Optane DIMM



Technique1: Vertex-centric graph buffering

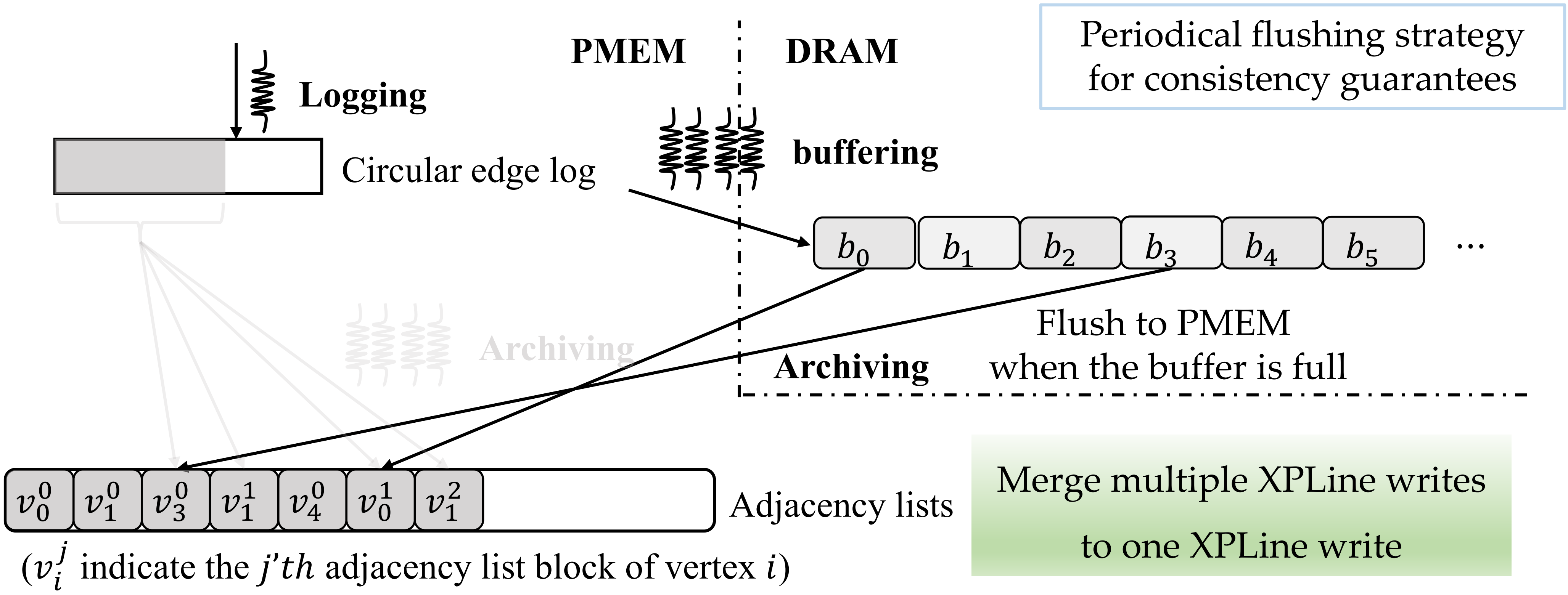
- Edge-centric global batched archive → **Vertex-centric local batched archive**



(v_i^j indicate the j'th adjacency list block of vertex i)

Technique1: Vertex-centric graph buffering

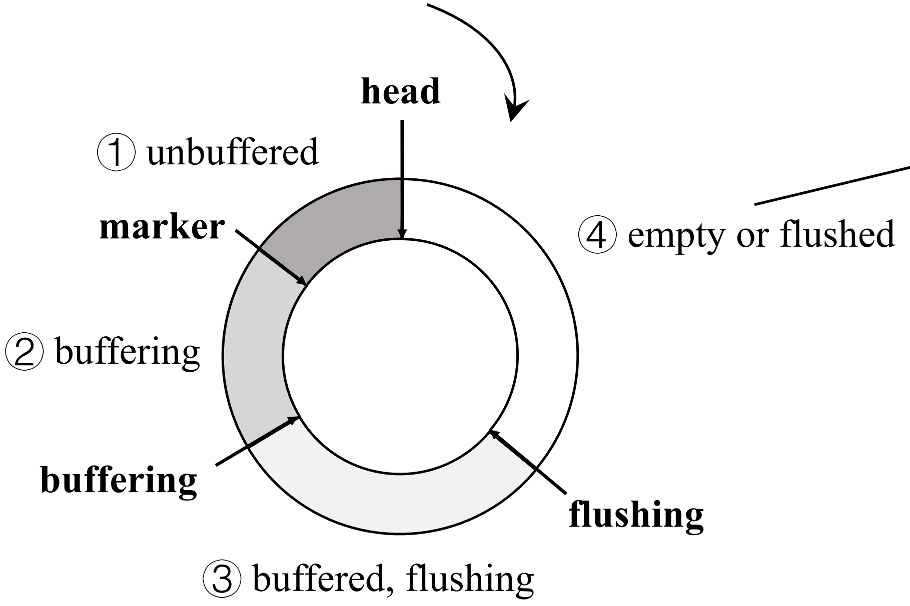
- Edge-centric global batched archive → **Vertex-centric local batched archive**



Technique1: Vertex-centric graph buffering

- **Consistency guaranteed circular edge log**

Write new coming edges in clockwise



Allow overwritten by new coming edges

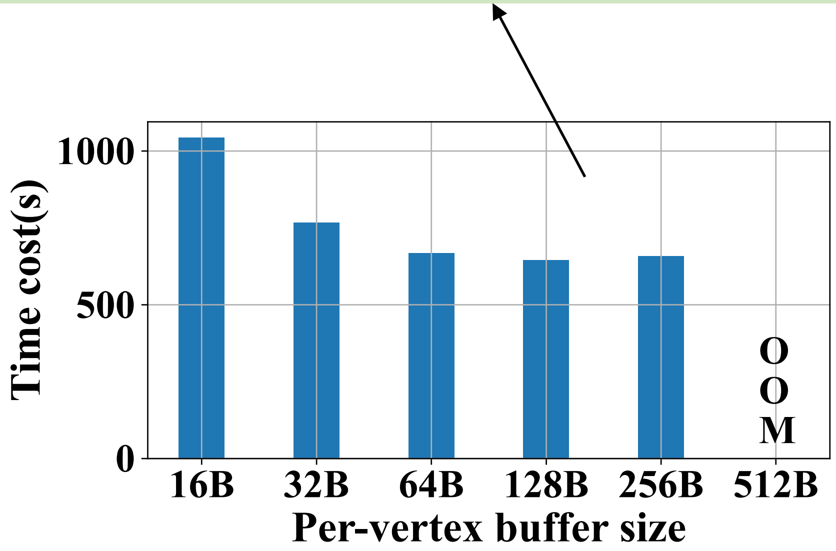
Once system crashes, we can recover the lost DRAM vertex buffers by traversing the edges between the **marker** and **flushing** positions

Wait for flushing before overwriting

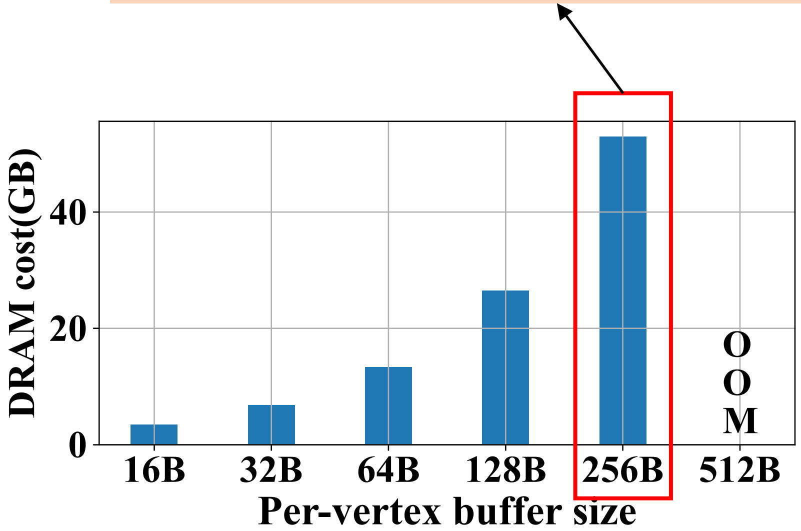
Technique1: Vertex-centric graph buffering

- Impact of buffer sizes for each vertex
  - Ingest YahooWeb graph (1.4B vertices and 6.6B edges)

Performance improved by large buffer sizes

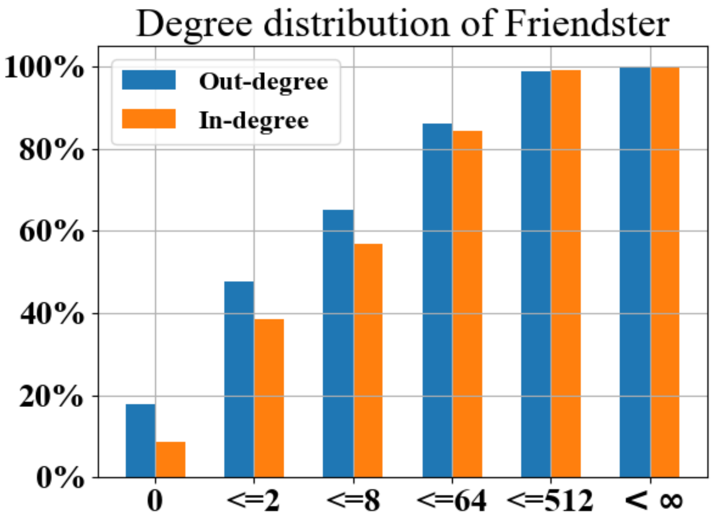


Cost more than 50GB DRAM space



Technique2: hierarchical vertex buffer managing

- Real-world graphs
  - **Power law degree distribution**



Large buffers for low-degree vertices → DRAM space waste

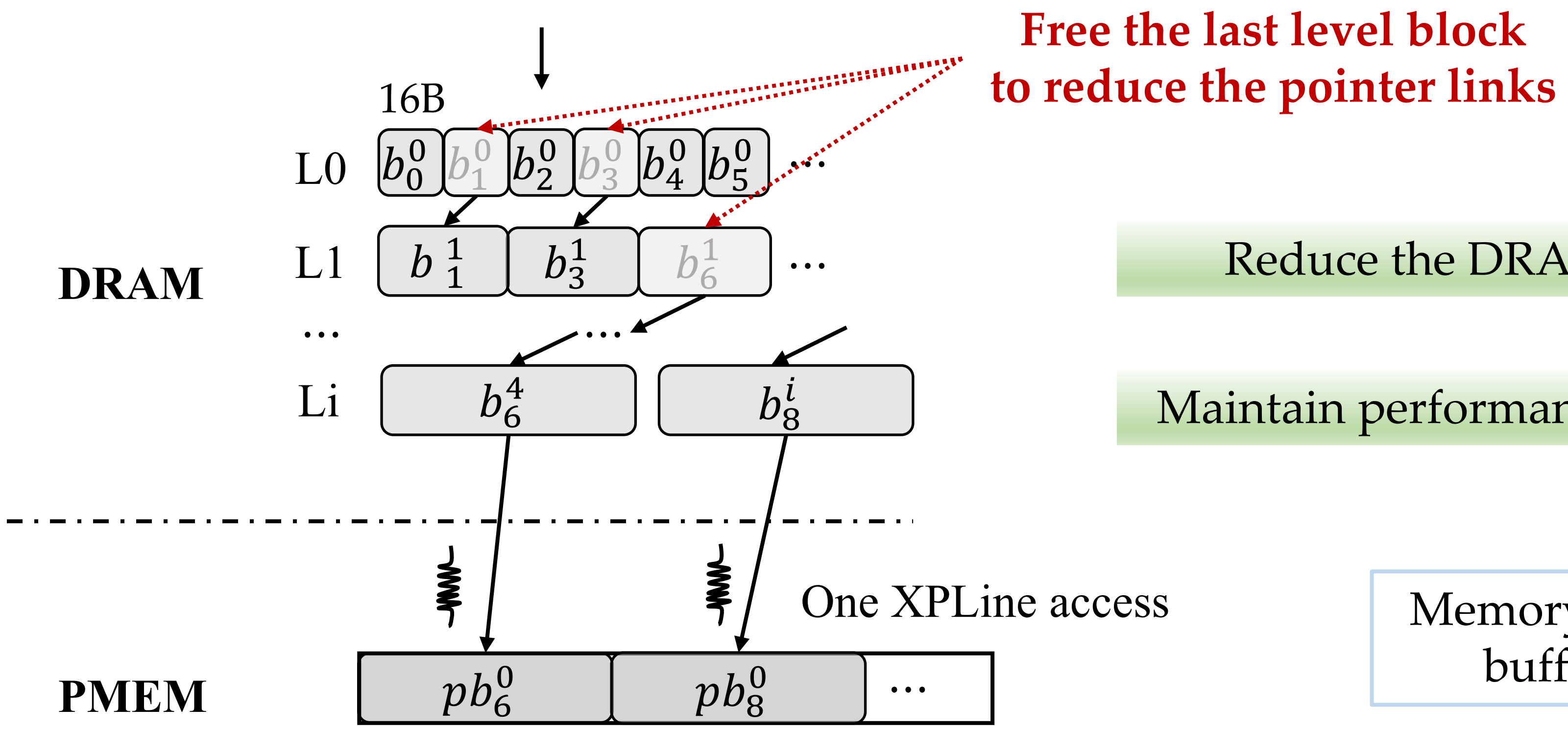
Small buffers for high-degree vertices → Limited benefit

⇩

Differentiated buffer sizes for vertices with different degrees

Technique2: hierarchical vertex buffer managing

- **Dynamicly adjust the buffer sizes**
  - According to the vertex degree changes



Reduce the DRAM cost

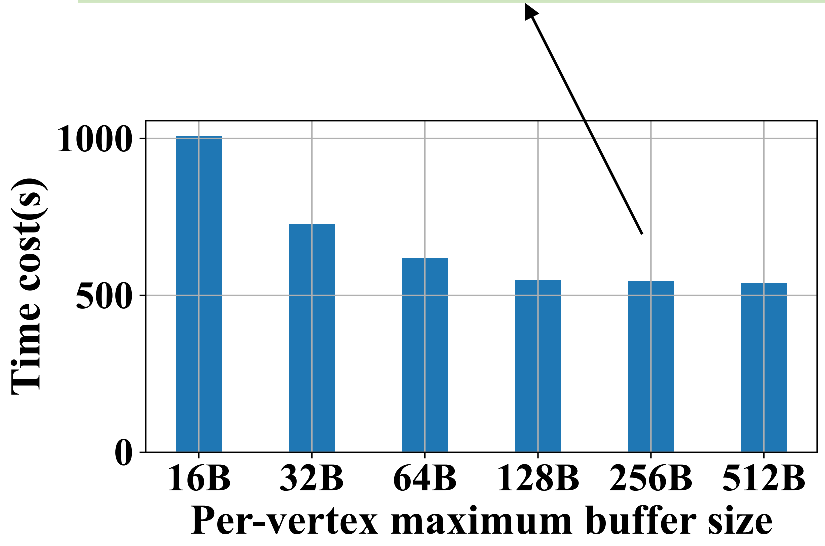
Maintain performance benefit

Memory pool based vertex buffer management

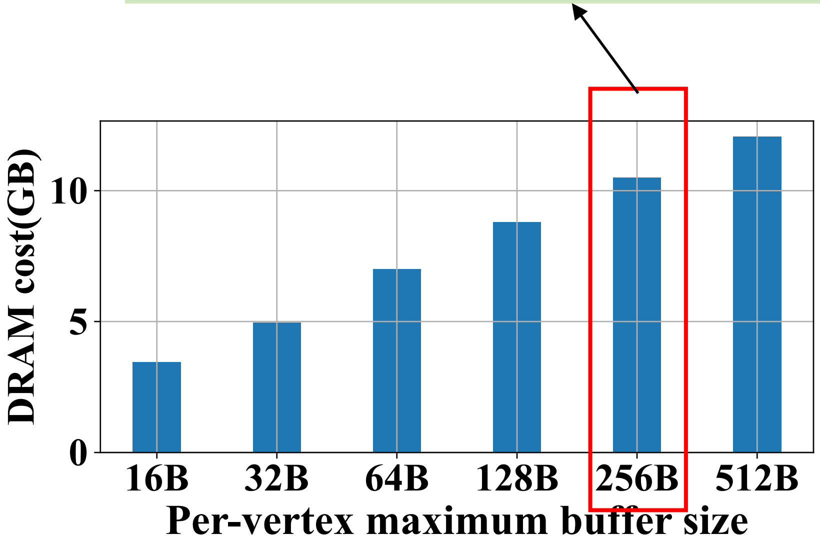
Technique2: hierarchical vertex buffer managing

- Impact of leveled buffer size setting
  - Ingest YahooWeb graph (1.4B vertices and 6.6B edges)

Keep the same performance benefit

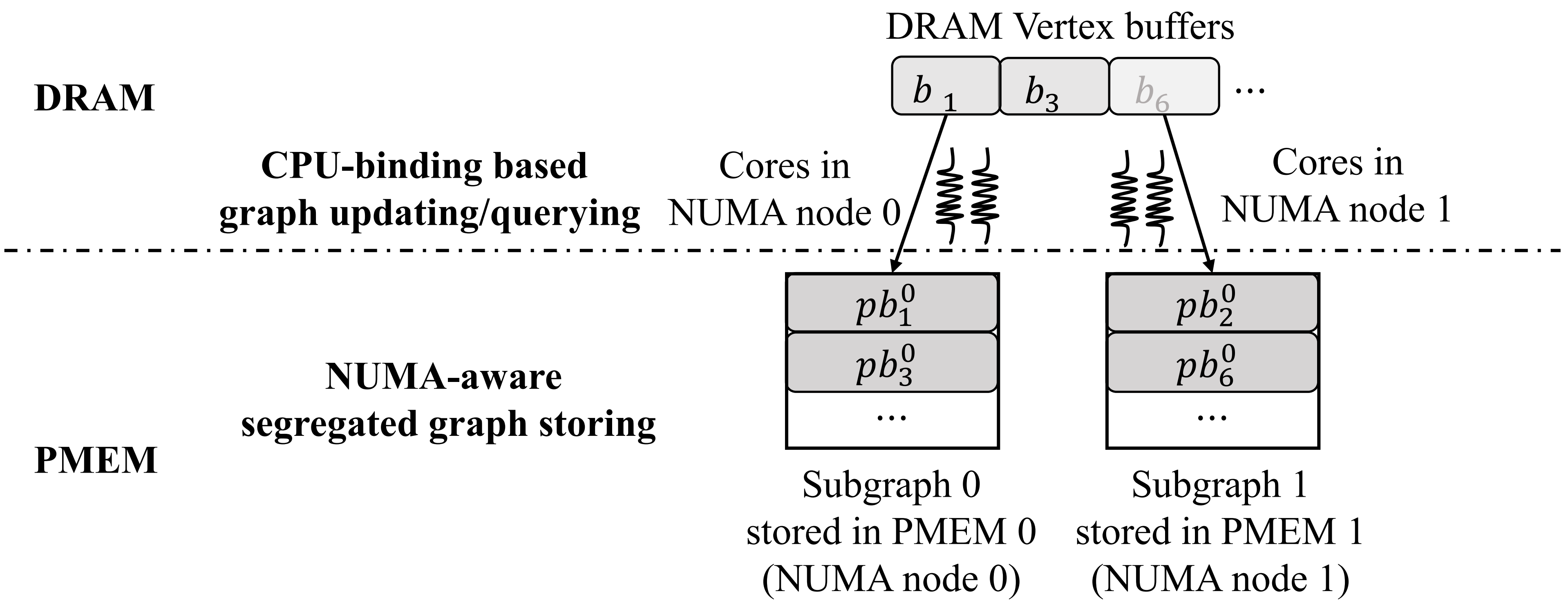


Reduced around 3/4 DRAM space cost



Technique3: NUMA-Friendly Graph Accessing

- **Avoid cross NUMA PMEM access**

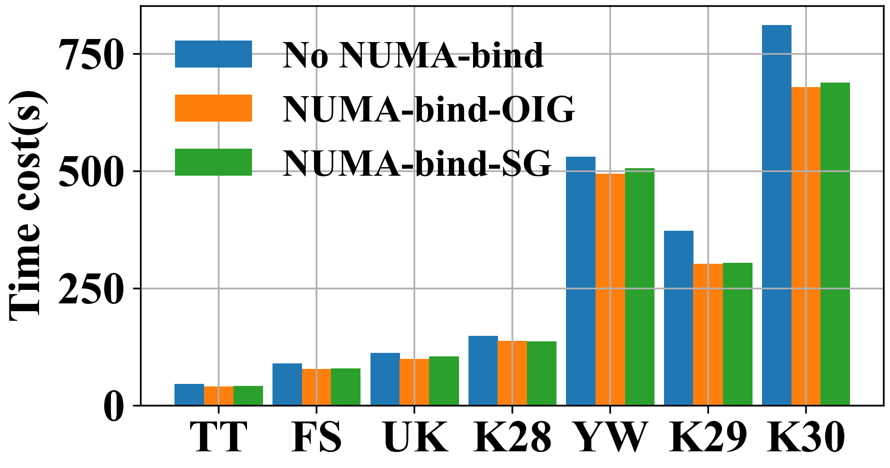


Technique3: NUMA-Friendly Graph Accessing

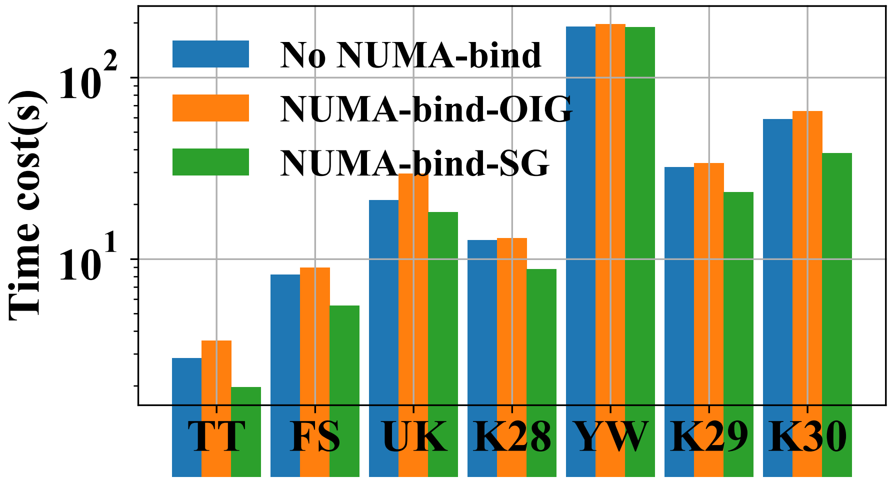
- Implementations of NUMA-Friendly Graph Accessing
  - Out/In-graph-based NUMA bind implementation (**NUMA-bind-OIG**)
    - ✓ Suit for two-socket systems
    - ✓ Store out-graph in PMEM 0 of NUMA node 0
    - ✓ Store in-graph in PMEM 1 of NUMA node 1
  - Sub-graph-based NUMA bind implementation (**NUMA-bind-SG**)
    - ✓ Suit for general P-socket systems
    - ✓ Divide whole graph into P sub-graphs
    - ✓ Store sub-graph p in PMEM p of NUMA node p

Technique3: NUMA-Friendly Graph Accessing

- Efficiency of NUMA-Friendly Graph Accessing
  - Ingest graph datasets, then conduct BFS algorithm



Graph ingest time



BFS time

Improve ingest performance by up to 23%

Improve BFS performance by up to 54%

Other optimizations and implementations

Periodical flushing for consistency guarantees.	Buddy-liked memory pool management.	Data Management Phases	Graph View Interfaces

More details are in the paper

- Prototype systems
  - **XPGraph**
  - **XPGraph-B, XPGraph-D**
    - ✓ Accommodate battery-backed and DRAM-only systems

Evaluation settings

- Testbed
  - A server with 2 sockets, each with 24 physical cores
  - 8 * 16GB = **128GB DRAM** + 8 * 128GB = **1TB PMEM**

- Graph datasets

Dataset	$\|V\|$	$\|E\|$	Bin Size	CSR Size
Twitter (TT)	61.6M	1.5B	12GB	12.4GB
Friendster (FS)	68.3M	2.6B	20.8GB	21.4GB
UKdomain (UK)	101.7M	3.1B	24.8GB	26.4GB
YahooWeb (YW)	1.4B	6.6B	52.8GB	75.2GB
Kron28 (K28)	256M	4B	32GB	36GB
Kron29 (K29)	512M	8B	64GB	72GB
Kron30 (K30)	1B	16B	128GB	144GB

Shuffle for ingestion

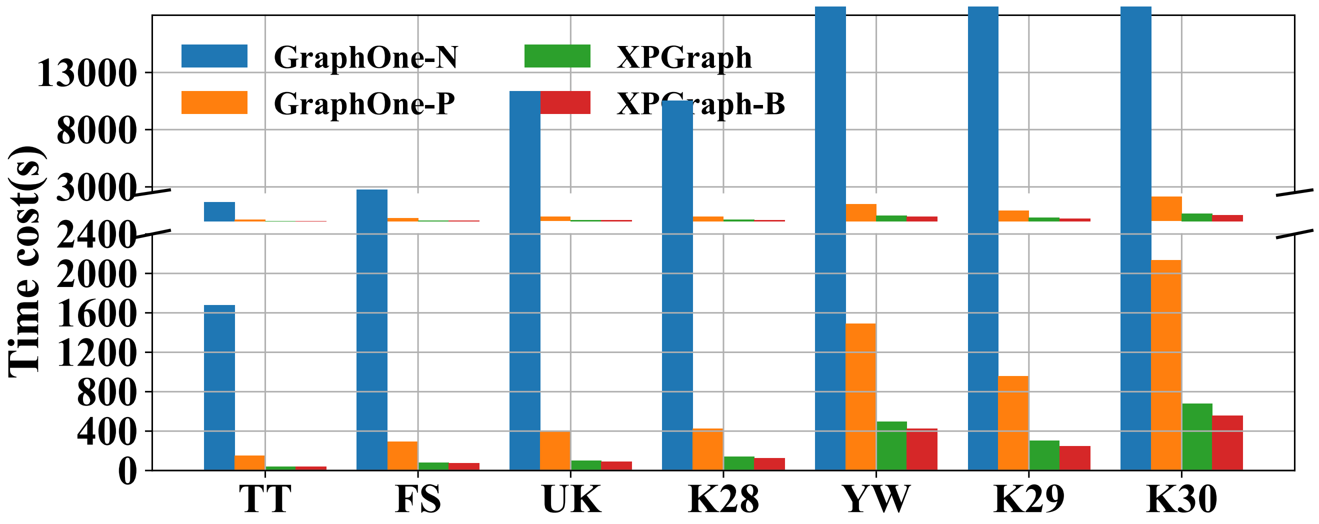
Evaluation settings

- Comparison systems
  - GraphOne-D
    - ✓ The original GraphOne that stores all data on DRAM.
  - GraphOne-P
    - ✓ Stores the edge log and adjacency lists on PMEM and keeps meta in DRAM
  - GraphOne-N
    - ✓ Stores the adjacency lists on PMEM by the NOVA file system

- Evaluation metrics
  - Graph ingesting performance
  - Graph query performance
    - ✓ 1-hop, BFS, PageRank, CC
  - Graph recovery performance

Evaluation1: Graph ingestion performance

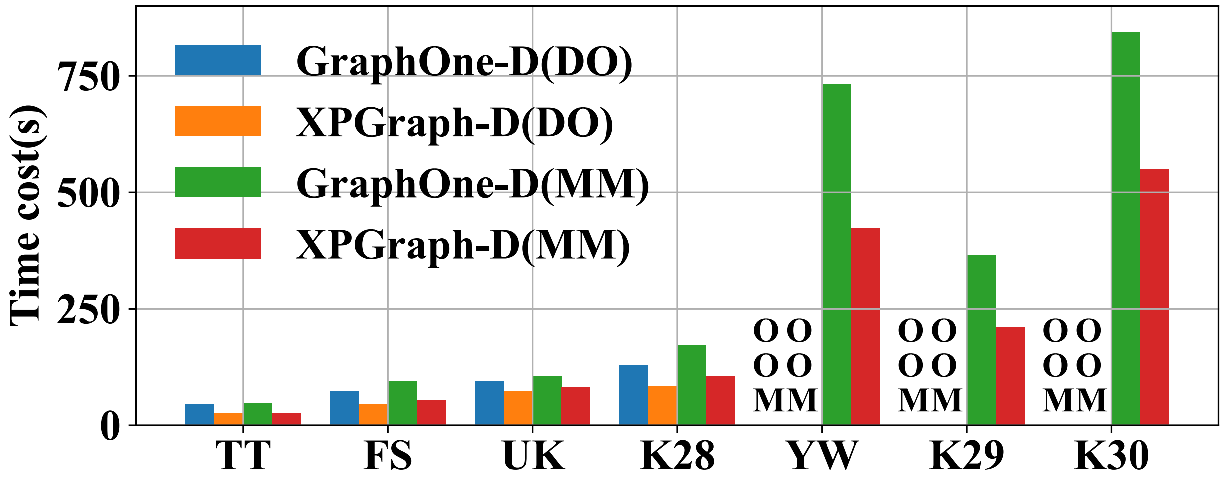
- Ingestion time cost for non-volatile systems



XPGraph achieves 3.01x-3.95x speedup upon GraphOne-P.
XPGraph-B can further improve the performance by up to 23% on top of XPGraph.

Evaluation1: Graph ingestion performance

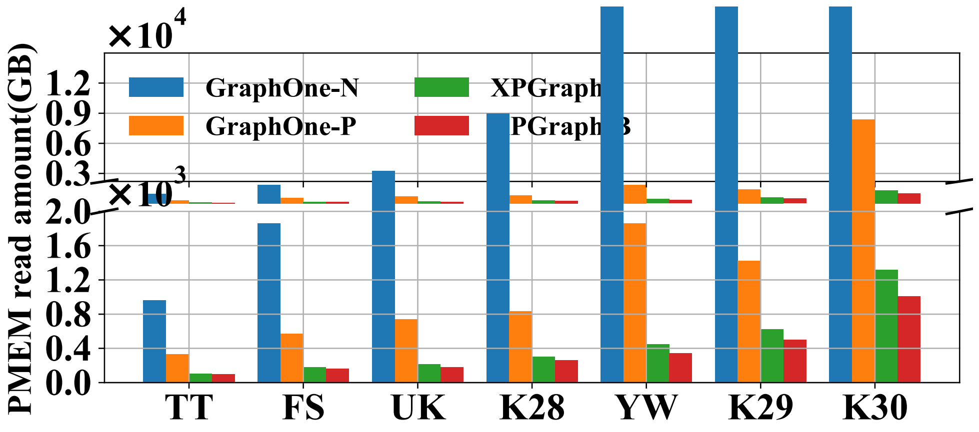
- Ingestion time cost for volatile systems



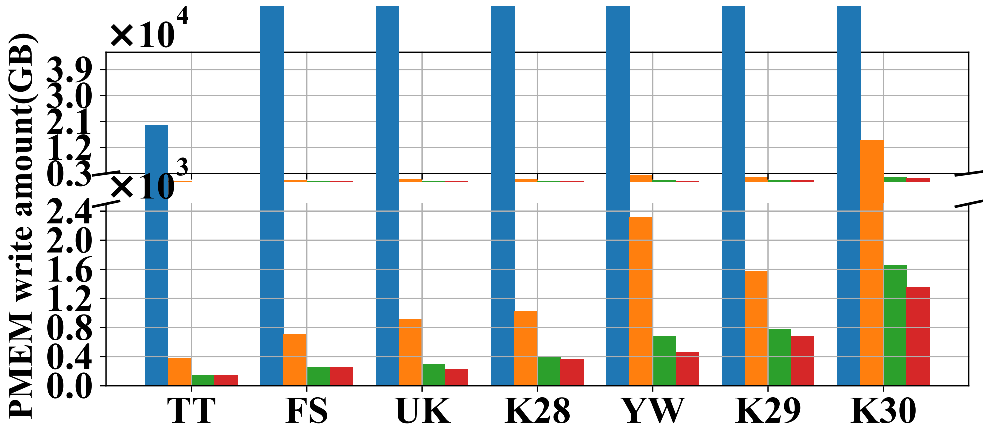
GraphOne-D and XPGraph-D can not run out on large graphs for DRAM-only systems. XPGraph-D always performs faster than GraphOne-D: the speedup is up to 73% for DRAM-only systems and 76% for PMEM-based systems with Optane in memory mode.

Evaluation1: Graph ingestion performance

- PMEM read and write data amount



PMEM read data amount

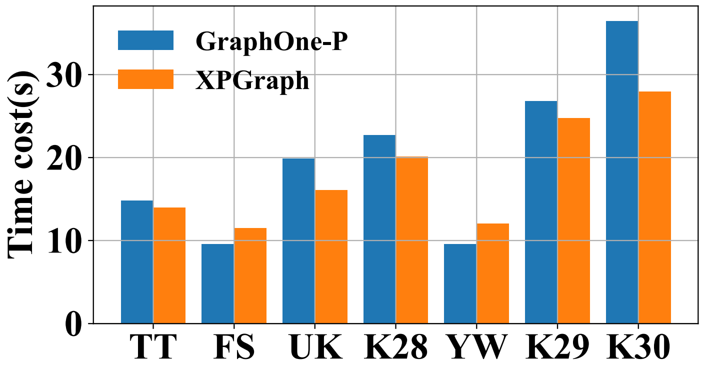


PMEM write data amount

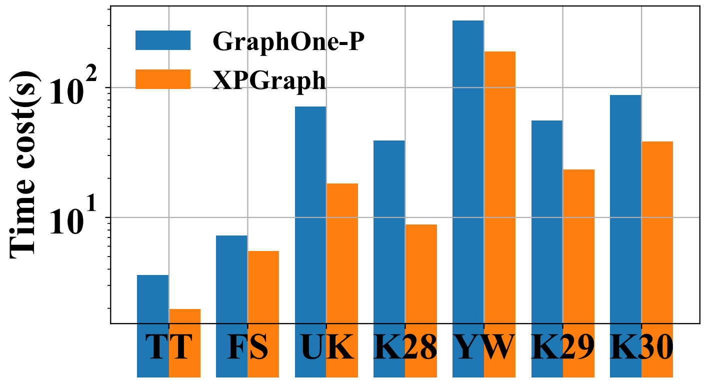
Compared with GraphOne-P, XPGraph greatly reduced the amount of PMEM read data by 2.29× to 4.17× and PMEM write data by 2.02× to 3.44×.

Evaluation2: Graph query performance

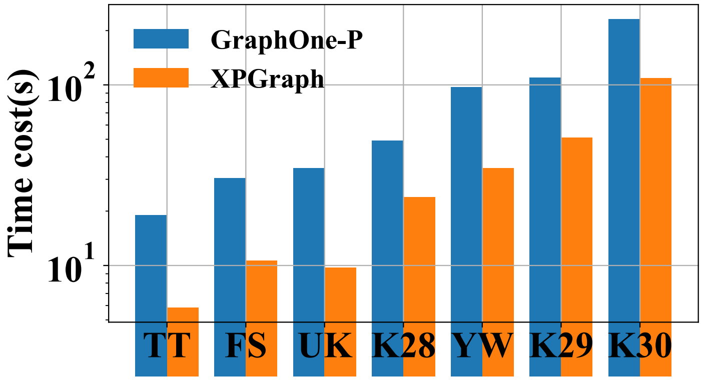
One-hop



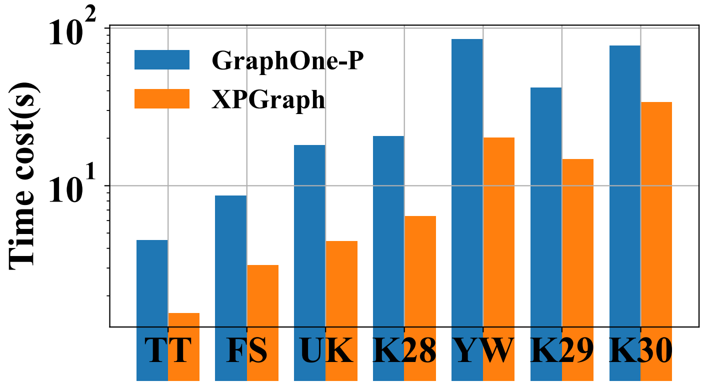
BFS



PageRank



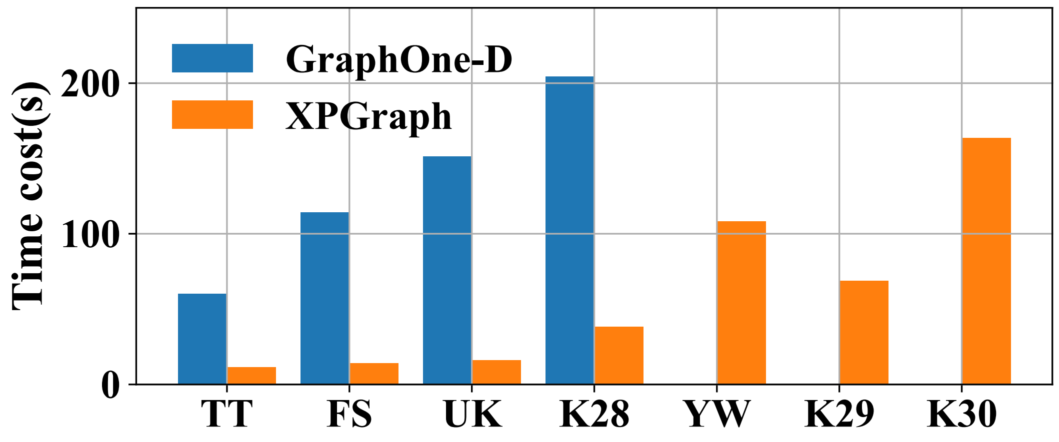
CC



XPGraph achieves up to 4.46×, 3.57×, and 4.23× speedup for BFS, PageRank, and CC respectively.

Evaluation3: Graph recovery performance

- Graph recover time cost



XPGraph achieves 5.20× to 9.47× higher recovery performance for the four relatively small graphs.
GraphOne-D can not run out on the larger three graphs, while XPGraph can realize the recovery in a reasonable time.

Conclusion

- **XPGraph**: a PMEM-based graph storage system for managing large-scale evolving graphs
  - Vertex-centric graph buffering
  - Hierarchical vertex buffer managing
  - NUMA-friendly graph accessing.

- More evaluation results and analysis are in the paper
- The source code is at https://github.com/ISCS-ZJU/XPGraph

Thanks for your attention!

Rui Wang @ ZJU
rwang21@zju.edu.cn