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

Graph applications

- Graphs are widely used in many applications

  Social networks  Webpage links  Recommendation systems

[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]

SOTA memory graph storage system – GraphOne[FAST19]

- **Hybrid store model**

  - **Logging**
  - **Archiving**
  - **Persisting**

  - **Circular edge log**
  - **Adjacency lists**

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

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

  - Extra persist cost

| Dataset       | $|V|$  | $|E|$  | Bin Size | CSR Size |
|---------------|------|------|---------|---------|
| YahooWeb (YW) | 1.4B | 6.6B | 52.8GB  | 75.2GB  |
Emergency persistent memory (PMEM)

- Emergency PMEM
  - Larger capacity and non-volatility

**Intel® Optane™ Persistent Memory 200 Series**

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

**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

Logging

Circular edge log

PMEM

Archiving

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

Persisting

SSD

Durable edge log file

➢ Large performance drop

• Import Friendster dataset

Run GraphOne on DRAM and PMEM

<table>
<thead>
<tr>
<th>Operation</th>
<th>DRAM</th>
<th>PMEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log</td>
<td>50</td>
<td>75</td>
</tr>
<tr>
<td>Archive</td>
<td>100</td>
<td>150</td>
</tr>
<tr>
<td>Ingest</td>
<td>200</td>
<td>250</td>
</tr>
</tbody>
</table>

1.45x 5.88x 6.37x
**Migrate DRAM-based graph stores to PMEM**

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

![Diagram](image)

- **Circular edge log:** 0 → 1, 3 → 0, ...

- **Adjacency lists**
  - 0: 2, 1
  - 1: 3
  - 2: 0, 3
  - 3: 1, 2

- **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: XPLLine-friendly graph access model

- According to ``XPLLine-centric access pattern” of Optane DIMM
Technique 1: Vertex-centric graph buffering

- Edge-centric global batched archive $\rightarrow$ Vertex-centric local batched archive

Diagram:

- **Logging**
  - Circular edge log
  - Adjacency lists
  - PMEM

- **Archiving**

$(v_i^j)$ indicate the $j^{th}$ adjacency list block of vertex $i$.
Technique 1: Vertex-centric graph buffering

- Edge-centric global batched archive $\rightarrow$ Vertex-centric local batched archive

- Circular edge log

- DRAM buffering

- PMEM

- Archiving

- Archiving when the buffer is full

- Flush to PMEM

- Merge multiple XPLine writes to one XPLine write

- Periodical flushing strategy for consistency guarantees

(v_i^j$ indicate the j'th adjacency list block of vertex i)
**Technique 1: Vertex-centric graph buffering**

- **Consistency guaranteed circular edge log**
  - Write new coming edges in clockwise
  - Allow overwritten by new coming edges
  - Wait for flushing before overwriting

Once system crashes, we can recover the lost DRAM vertex buffers by traversing the edges between the **marker** and **flushing** positions.
Technique 1: 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
Technique 2: hierarchical vertex buffer managing

- Real-world graphs
  - Power law degree distribution

![Degree distribution of Friendster](image)

- Large buffers for low-degree vertices $\rightarrow$ DRAM space waste
- Small buffers for high-degree vertices $\rightarrow$ Limited benefit

Differentiated buffer sizes for vertices with different degrees
Technique 2: hierarchical vertex buffer managing

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

  ![Diagram of buffer management](image)

  - Reduce the DRAM cost
  - Free the last level block to reduce the pointer links
  - Maintain performance benefit
  - One XPLine access
  - 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
Technique 3: NUMA-Friendly Graph Accessing

- Avoid cross NUMA PMEM access

![Diagram showing NUMA-aware segregated graph storing and CPU-binding based graph updating/querying]
Technique 3: 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
Technique 3: 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
**Evaluation 1: 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.
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.
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

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 $5.20 \times$ to $9.47 \times$ 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](https://github.com/ISCS-ZJU/XPGraph)
Thanks for your attention!

Rui Wang @ ZJU
rwang21@zju.edu.cn