Bridging the Gap between Relational OLTP and Graph-based OLAP

Sijie Shen\textsuperscript{1,2}, Zihang Yao\textsuperscript{1}, Lin Shi\textsuperscript{1}, Lei Wang\textsuperscript{2}, Longbin Lai\textsuperscript{2}, Qian Tao\textsuperscript{2}, Li Su\textsuperscript{2}, Rong Chen\textsuperscript{1,3}, Wenyuan Yu\textsuperscript{2}, Haibo Chen\textsuperscript{1}, Binyu Zang\textsuperscript{1}, and Jingren Zhou\textsuperscript{2}

1. Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University
2. Alibaba Group
3. Shanghai AI Laboratory
Graph Analytical Processing (GAP)

Social Network

Knowledge Graph

Transportation Network

**Graph analysis:** PageRank, SSSP, …

**Graph traversal:** business intelligence, …

**Graph learning:** Graph Neural Network, …
Dynamic **GAP** for **Relational** Datasets

Data usually stored and updated in relational OLTP systems

**Dynamic GAP**: GAP while data updating

**Inefficient** graph operation in relational systems

- Rewrite graph queries by relational operations[1]
- Join: cost & large intermediate results

**SQL**

```sql
WITH RECURSIVE descendants AS
(
  SELECT person
  FROM tree
  WHERE person='Thurimbert'
  UNION ALL
  SELECT t.person
  FROM descendants d, tree t
  WHERE t.parent=d.person
)
SELECT * FROM descendants;
```

**Cypher**

```cypher
MATCH path=(n:Person {name: 'Thurimbert'})-[*-]>(n)
RETURN n;
```

Requirements of Dynamic GAP

Performance

- Comparable to specific OLTP and GAP systems

Freshness

- Minimize the time gap between when data is committed and read
Existing Solutions
Existing Solution 1/2: GAP on Offline Data

Combine OLTP systems with graph-specific systems
Existing Solution 2/2: GAP on Online Data

OLTP systems support graph processing

- **Graph extension** in relational systems (DB2, SQL Server, SAP HANA)
- **Graph database** (Neo4j, TigerGraph)

![Diagram showing OLTP System, User behavior, Network events, Transactions, Good Freshness, Poor Performance, GAP, SQL Graph @ SQL Server](image)

SQL Graph @ SQL Server

```sql
SELECT Division.Name, Employee.Name
FROM Division, Employee,
     Company Client, Company Merchant
MATCH Division-[Employees]->Employee
    -[Clients]->Client,
    Employee-[Merchants]->Merchant
WHERE Merchant.Location = 'Seattle' AND
     Client.Location = 'New York'
```
Opportunity: HTAP

**Hybrid Transactional and Analytical Processing**
- Perform **real-time queries** on data generated by **transactions**

**Loosely coupled** architecture based on transactional **logs**
- **Performance**: isolated and dedicated engines
- **Freshness**: AP data restored from (sync.) logs recording data updates

![Diagram showing TX, Logs, and AP with log replayers]
General Idea: From HTAP to HTGAP

**Hybrid Transactional and Graph Analytical Processing**

- Real-time graph queries on data generated by transactions
Our Approach: GART

GART: in-memory HTGAP system for dynamic GAP

- **Relational-Graph Mapping**: data model conversion
- **Dynamic Graph Storage**: write (log replay) & read (GAP)
Unique Design Goals

Transparent data model conversion
- Instead of interface conversion (GART not need to rewrite requests)

Efficient dynamic graph storage
- Performance of both operations is important
- Guarantee read locality when supporting updates
Unique Design Goals

Transparent data model conversion
- Instead of interface conversion (rewrite requests)

Efficient dynamic graph storage
- Performance of both operations is important
Interfaces

Data manipulation interfaces

- User: write **requests** as if on the specific engines

Graph extraction interfaces

- DBA: define data model **conversion (only once)**
RGMapping: Graph Extraction Interfaces

Define **vertex and edge types** by relational tables

- **Vertex**: dedicate table name
- **Edge**: dedicate table name and primary key
- **Property**: dedicate column name

```python
# Definition for vertices
def def_vertex(vtype, table):
    add_vprop(vtype, vprop, attr)

# Definition for edges
def def_edge(etype, src_vtype, dst_vtype, pk) # 1-to-m
    def_edge(etype, src_vtype, dst_vtype, src_pk, dst_pk) # m-to-m
    add_eprop(etype, eprop, attr)
```
Unique Design Goals

Transparent data model conversion
- Instead of interface conversion (rewrite requests)

Efficient dynamic graph storage
- Performance of both operations is important
Problems of Dynamic Graph Storage

Topology
- CSR (immutable): Good edge locality
- Adjacency list: poor edge locality from adjacent vertices

Fine-grained MVCC
- Timestamps for each edge
- Break spatial and temporal locality

Property
- No efficient property storage model for all GAP workloads
Key Insights about HTGAP

Required freshness is sufficient for updating compact structure

- Time gap between write (OLTP) and read (GAP)
- E.g., tens-of-ms freshness

GAP latency much longer than the required freshness

- Fine-grained MVCC is not necessary
- GAP latency (more than 10x of freshness)

Access pattern of properties is nearly fixed

- User can decide how to store different properties
Graph Storage of GART

Efficient and mutable CSR
- Segmented edge store

Coarse-grained MVCC
- Use epoch instead of timestamps

Flexible property storage
- User-defined property storage model
Efficient and Mutable CSR

Preserve the locality of edge scan

Edge Segment

- Store edges for a group of vertices
- Tradeoff: read & write performance
- Fixed initial size (16KB) with free slots
- Full: double size

Reallocation of segments

- Each vertex store edge in the same segment
- Logic offset prevent from metadata updating when resize
Graph Storage of GART

Efficient and mutable CSR

- Segmented edge store

Coarse-grained MVCC

- Use epoch instead of timestamps

Flexible property storage

- User-defined property storage model

Please refer to our paper for more details.
Evaluation

Testbed

- 2x dual-socket machines (OLTP server & GAP server under HTGAP workloads)

Benchmark (extended for HTGAP)

- LDBC Social Network Benchmark (SNB)
- TPC-C [refer to our paper]

GAP Workloads

- Graph analytics (GA): PR (PageRank), CC (Connected Components), SSSP (Single Source Shortest Path)
- Graph traversal (GT): LDBC SNB IS-3, BI-2, and BI-3
- Graph neural network (GNN): GCN, GSG, and SGC
Overall Performance

Comparing targets

- **Offline**: DrTM+H with GraphScope (DH+GS)
  - Same OLTP and GAP engines as GART
  - Graph database
  - Adjacency-list-based storage

- **Online**: Neo4j

- **Replace storage by LiveGraph: G/LG**
  - General-used dynamic graph storage

### Workloads

<table>
<thead>
<tr>
<th>Workloads</th>
<th>LDBC SNB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GART</td>
</tr>
<tr>
<td>OLTP ↑</td>
<td>1837 K</td>
</tr>
<tr>
<td>PR</td>
<td>377</td>
</tr>
<tr>
<td>GA ↓ CC</td>
<td>362</td>
</tr>
<tr>
<td>SSSP</td>
<td>513</td>
</tr>
<tr>
<td>IS-3</td>
<td>17.9</td>
</tr>
<tr>
<td>GT ↓ BI-2</td>
<td>235</td>
</tr>
<tr>
<td>BI-3</td>
<td>292</td>
</tr>
<tr>
<td>GNN ↓ GCN</td>
<td>1097</td>
</tr>
<tr>
<td>GSG</td>
<td>1774</td>
</tr>
<tr>
<td>SGC</td>
<td>779</td>
</tr>
<tr>
<td>Freshness ↓</td>
<td>18</td>
</tr>
</tbody>
</table>
Overall Performance

OLTP & GAP performance

- Comparable with offline solution (DH+GS)
- OLTP 525x online solution (Neo4j)

Freshness (18ms)

- Comparable with online solution (Neo4j)
- 872x improvement with DH+GS

<table>
<thead>
<tr>
<th>Workloads</th>
<th>LDBC SNB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GART</td>
</tr>
<tr>
<td>OLTP ↑</td>
<td>1837 K</td>
</tr>
<tr>
<td>PR</td>
<td>377</td>
</tr>
<tr>
<td>GA ↓</td>
<td>362</td>
</tr>
<tr>
<td>CC</td>
<td>513</td>
</tr>
<tr>
<td>SSSP</td>
<td>17.9</td>
</tr>
<tr>
<td>IS-3</td>
<td>235</td>
</tr>
<tr>
<td>GT ↓</td>
<td>292</td>
</tr>
<tr>
<td>BI-2</td>
<td>1097</td>
</tr>
<tr>
<td>BI-3</td>
<td>1774</td>
</tr>
<tr>
<td>GNN ↓</td>
<td>779</td>
</tr>
<tr>
<td>Freshness ↓</td>
<td>18</td>
</tr>
</tbody>
</table>
GAP Breakdown

GART outperforms G/LG due to storage

Breakdown analysis (e.g., PageRank)

- **Topology**: 6.2x (82% of improvement)
- **Property**: 3.0x
- **Computation**

<table>
<thead>
<tr>
<th>Workloads</th>
<th>LDBC SNB</th>
<th>GART</th>
<th>DH+GS</th>
<th>Neo4j</th>
<th>G/LG</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLTP ↑</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR</td>
<td>1837 K</td>
<td>1929 K</td>
<td>3.5 K</td>
<td>1836 K</td>
<td></td>
</tr>
<tr>
<td>GA ↓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC</td>
<td>362</td>
<td>312</td>
<td>4726</td>
<td>1137</td>
<td></td>
</tr>
<tr>
<td>SSSP</td>
<td>513</td>
<td>433</td>
<td>4668</td>
<td>1381</td>
<td></td>
</tr>
<tr>
<td>GT ↓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BI-3</td>
<td>17.9</td>
<td>16.9</td>
<td>2.0</td>
<td>18.0</td>
<td></td>
</tr>
<tr>
<td>BI-2</td>
<td>235</td>
<td>201</td>
<td>568</td>
<td>828</td>
<td></td>
</tr>
<tr>
<td>BI-3</td>
<td>292</td>
<td>266</td>
<td>573</td>
<td>1278</td>
<td></td>
</tr>
<tr>
<td>GNN ↓ GNN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GCN</td>
<td>1097</td>
<td>940</td>
<td>×</td>
<td>1834</td>
<td></td>
</tr>
<tr>
<td>GSG</td>
<td>1774</td>
<td>1443</td>
<td>×</td>
<td>2502</td>
<td></td>
</tr>
<tr>
<td>SGC</td>
<td>779</td>
<td>717</td>
<td>×</td>
<td>1237</td>
<td></td>
</tr>
<tr>
<td>Freshness ↓</td>
<td>18</td>
<td>15683</td>
<td>5</td>
<td>25</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Storage</th>
<th>Topology</th>
<th>Property</th>
<th>Computation</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR</td>
<td>GART</td>
<td>107 (28%)</td>
<td>163 (44%)</td>
<td>107 (28%)</td>
</tr>
<tr>
<td></td>
<td>G/LG</td>
<td>658 (52%)</td>
<td>486 (38%)</td>
<td>132 (10%)</td>
</tr>
<tr>
<td>BI-3</td>
<td>GART</td>
<td>10 (3%)</td>
<td>178 (61%)</td>
<td>104 (36%)</td>
</tr>
<tr>
<td></td>
<td>G/LG</td>
<td>40 (3%)</td>
<td>1115 (87%)</td>
<td>123 (10%)</td>
</tr>
<tr>
<td>SGC</td>
<td>GART</td>
<td>36 (5%)</td>
<td>477 (61%)</td>
<td>266 (34%)</td>
</tr>
<tr>
<td></td>
<td>G/LG</td>
<td>236 (19%)</td>
<td>732 (59%)</td>
<td>269 (22%)</td>
</tr>
</tbody>
</table>
Performance Isolation

Increase the number of GAP and OLTP clients

Performance degradation

- OLTP: 1%
- GAP: 12% (overhead of version checking)
Conclusion & Thanks

GART: in-memory HTGAP system for dynamic GAP
- Transparent data model conversion by RGMapping
- Efficient dynamic graph storage with good locality

Open Source: https://github.com/GraphScope/GART
Experimental Code: https://github.com/SJTU-IPADS/vegito/tree/gart

Contact: shensijie.ssj@alibaba-inc.com