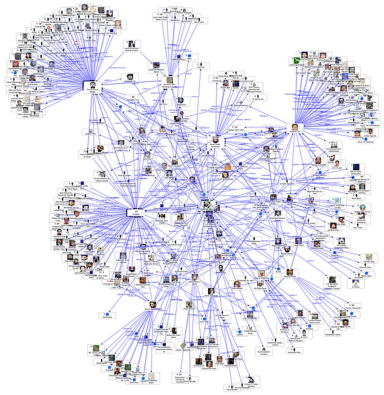
Bridging the Gap between Relational OLTP and Graph-based OLAP

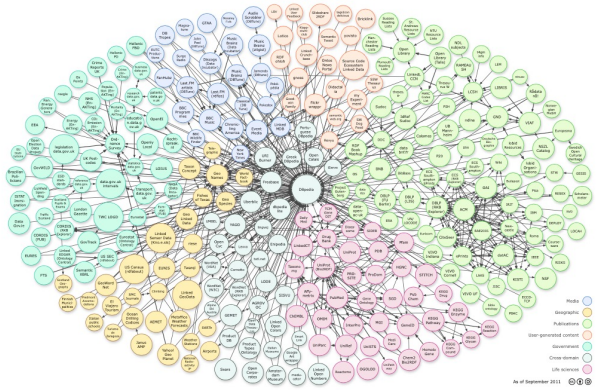
Sijie Shen[1,2], Zihang Yao[1], Lin Shi[1], Lei Wang[2], Longbin Lai[2], Qian Tao[2], Li Su[2], Rong Chen[1,3], Wenyuan Yu[2], Haibo Chen[1], Binyu Zang[1], and Jingren Zhou[2]

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

```
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

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

[1] https://memgraph.com/blog/graph-database-vs-relational-database

Requirements of Dynamic GAP

Performance

- Comparable to specific OLTP and GAP systems

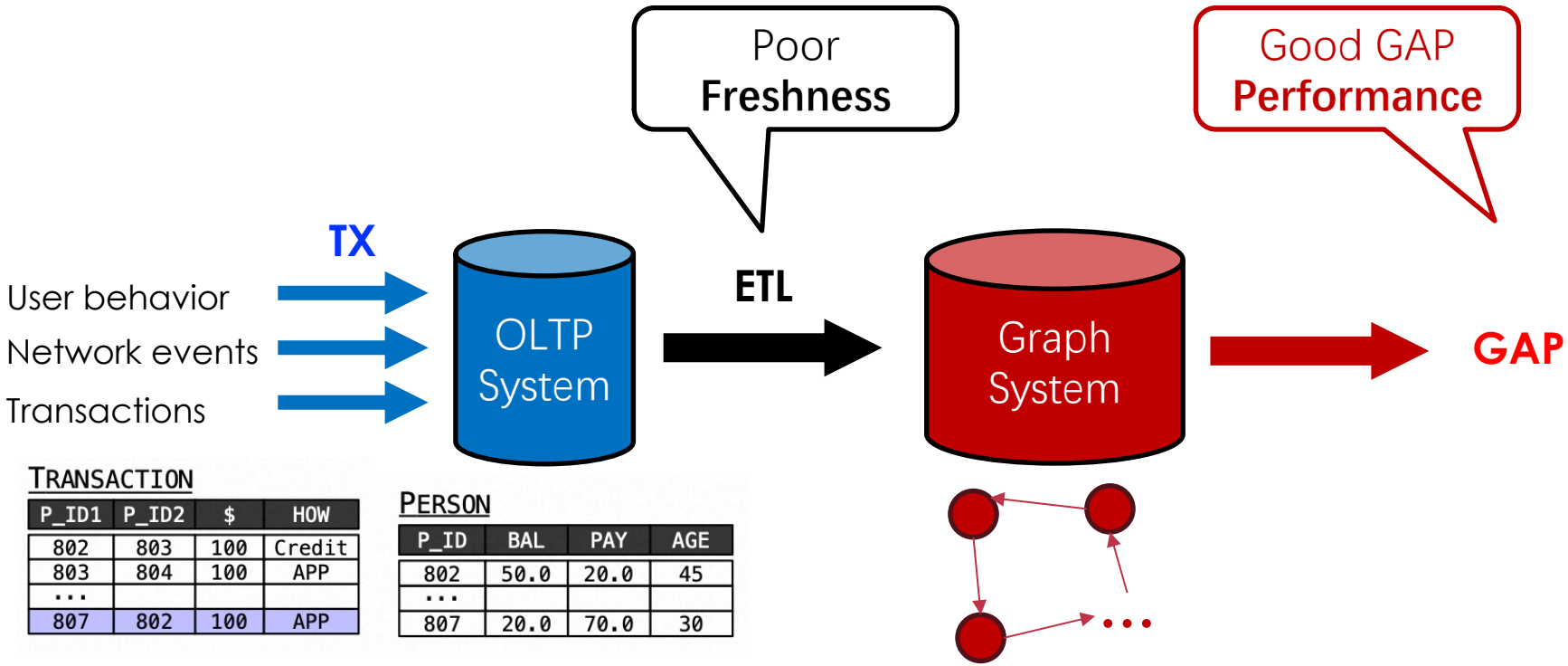
Freshness

- Minimize the time gap between when data is committed and read

Existing Solutions

Existing Solution 1/2: GAP on Offline Data

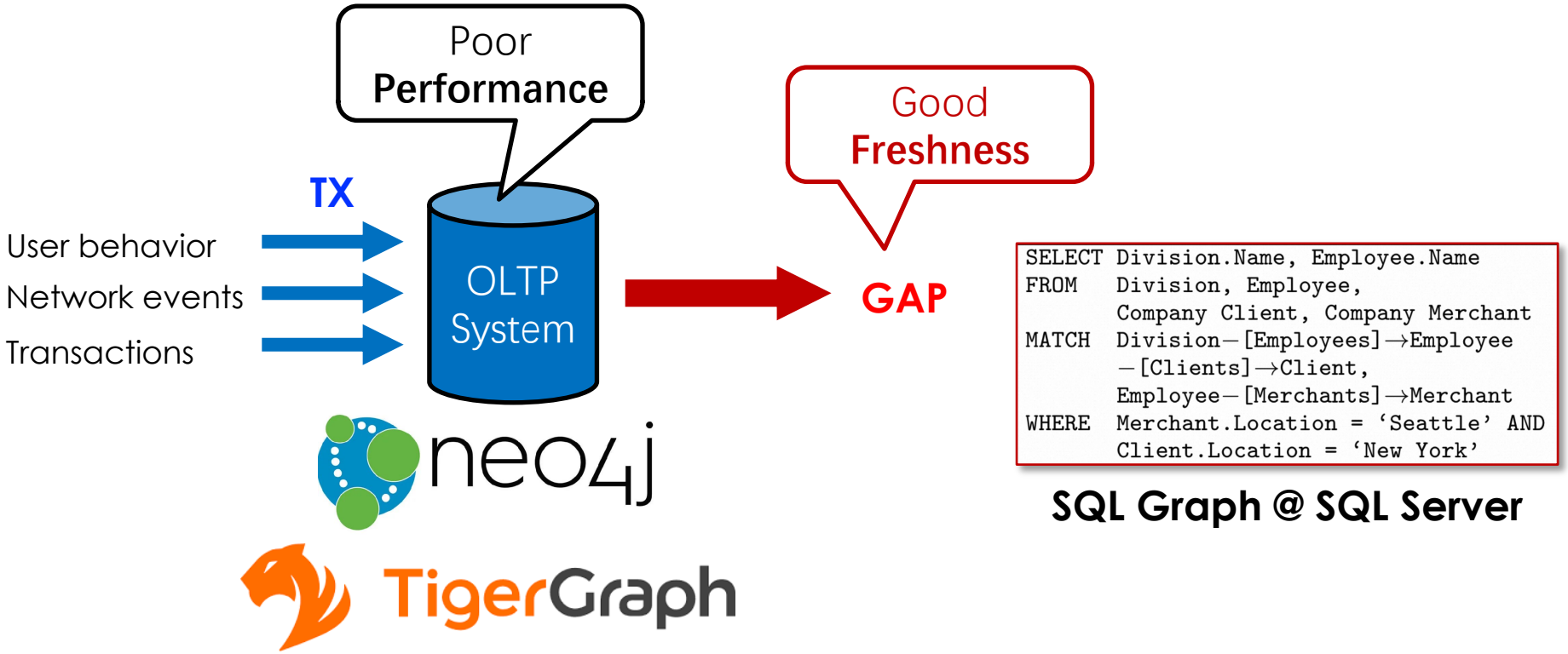
Combine OLTP systems with graph-specific systems



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)




```
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'
```

SQL Graph @ SQL Server

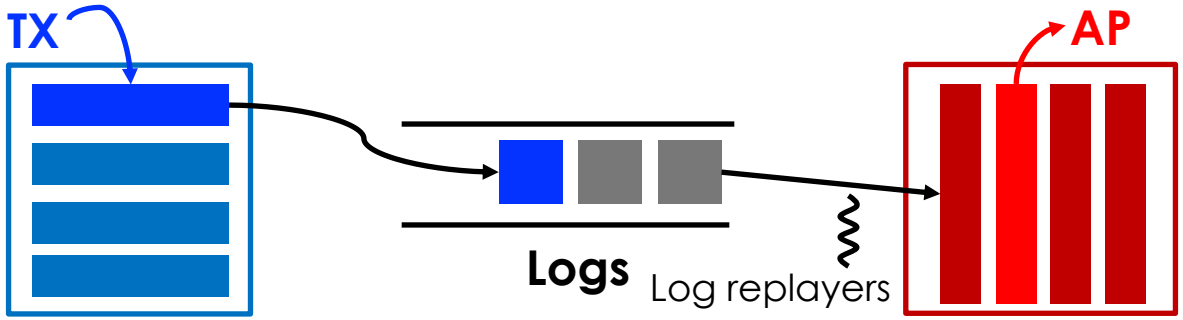
Opportunity: HTAP

Hybrid **T**ransactional and **A**nalytical **P**rocessing

- Perform real-time queries on data generated by transactions

Loosely coupled architecture based on transactional **logs**

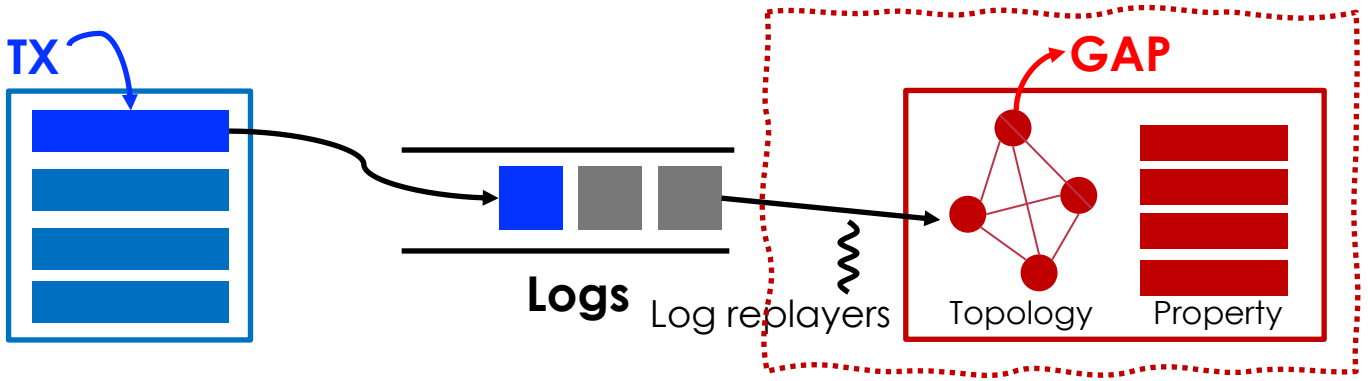
- **Performance**: isolated and dedicated engines
- **Freshness**: AP data restored from (sync.) logs recording data updates



General Idea: From HTAP to HTGAP

Hybrid **T**ransactional and **G**raph **A**nalytical **P**rocessing

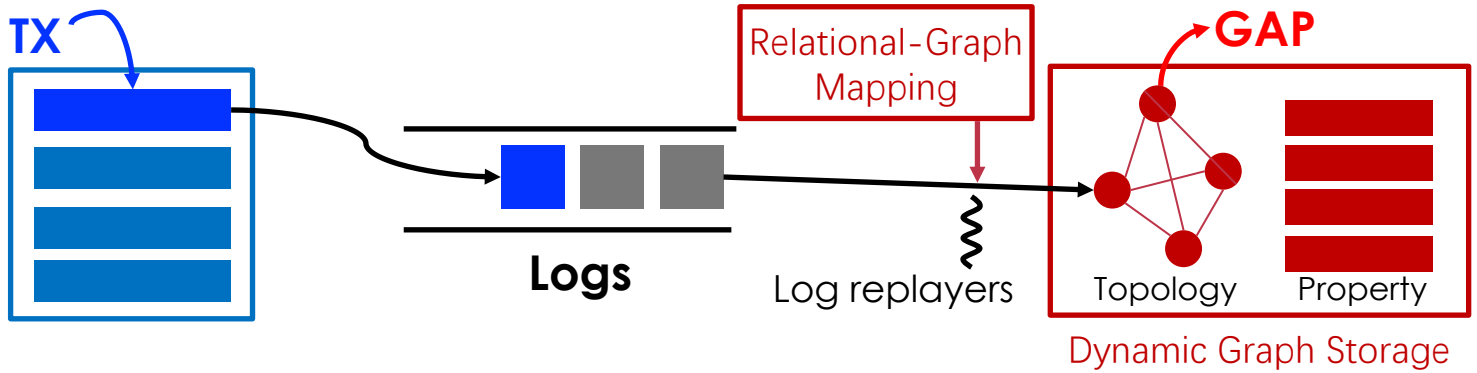
- Real-time graph queries on data generated by transactions



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)



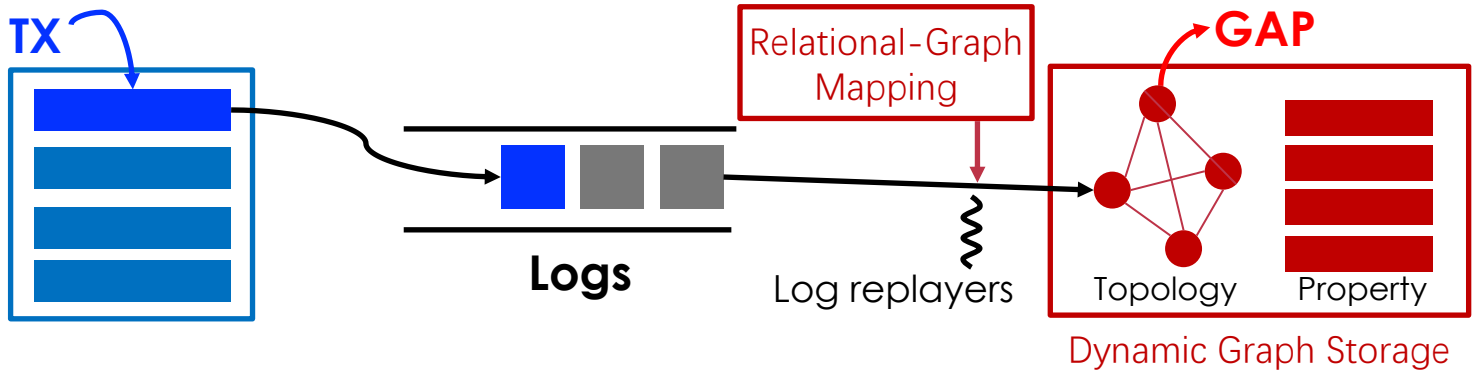
Unique Design Goals

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



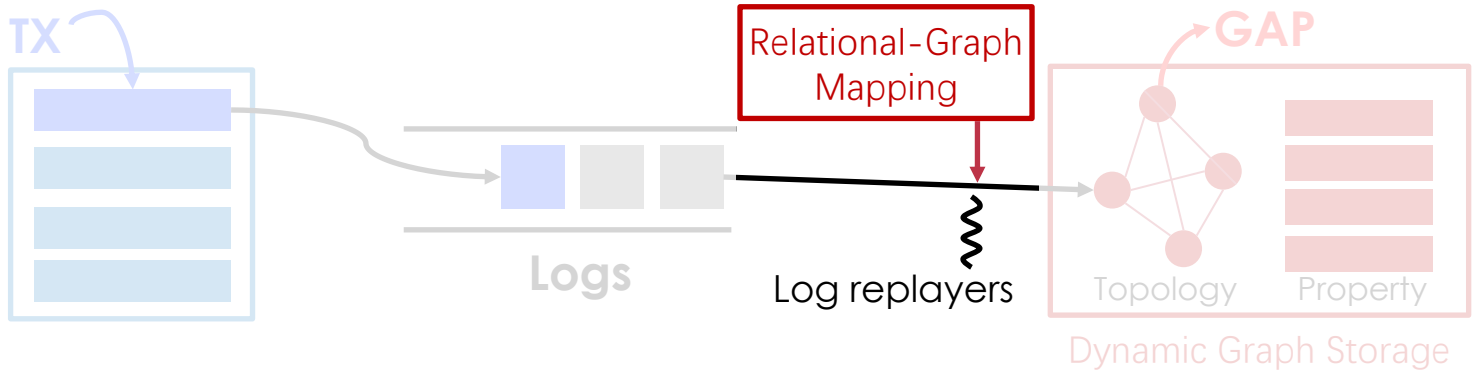
Unique Design Goals

Transparent data model conversion

- Instead of interface conversion (rewrite requests)

Efficient dynamic graph storage

- Performance of both operations is important



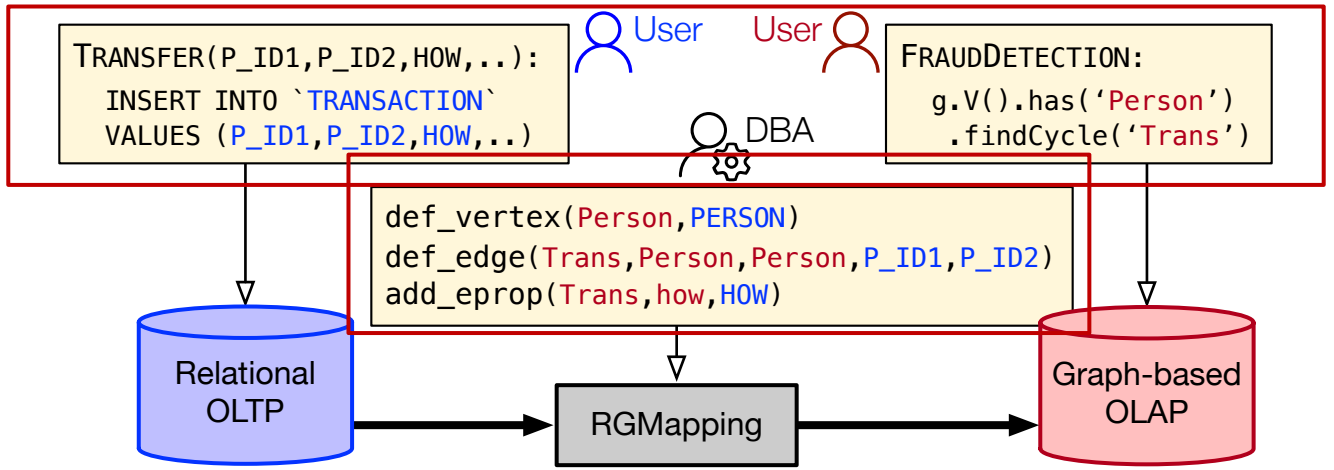
Interfaces

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

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

# Definition for edges
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)
```

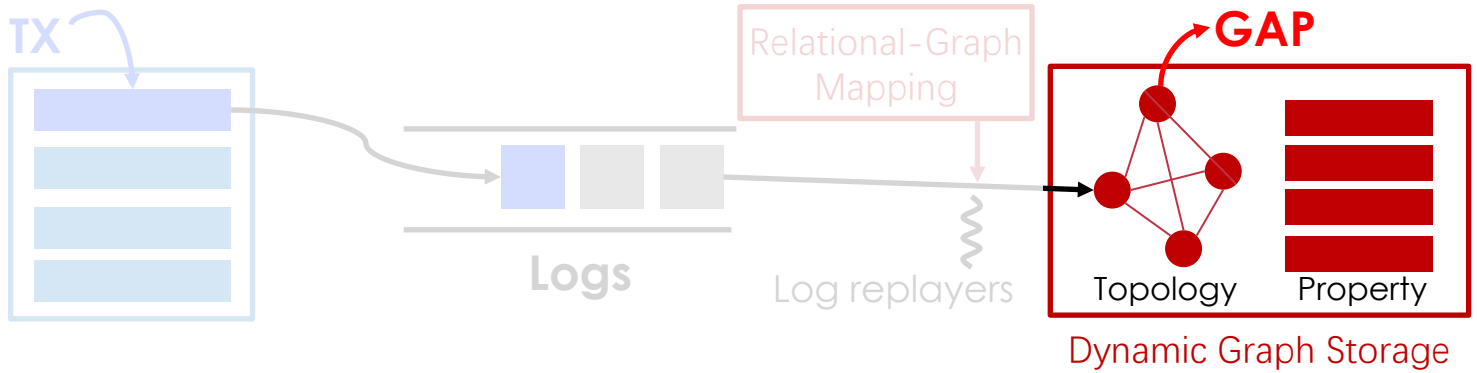
Unique Design Goals

Transparent data model conversion

- Instead of interface conversion (rewrite requests)

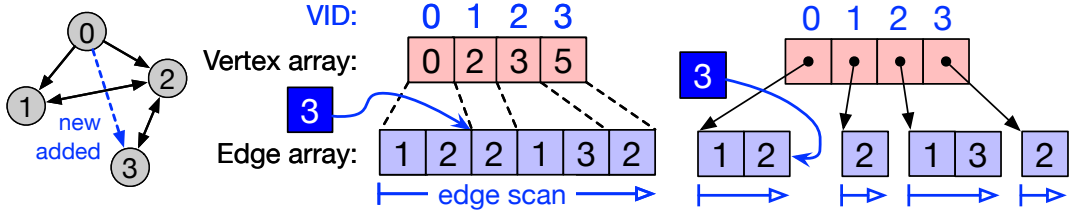
Efficient dynamic graph storage

- Performance of both operations is important



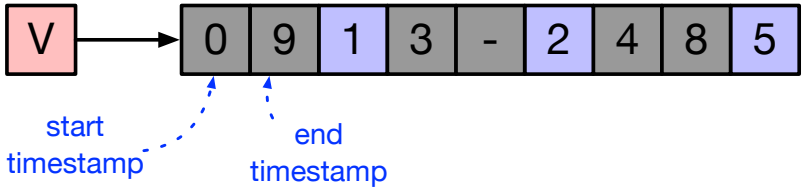
Problems of Dynamic Graph Storage

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

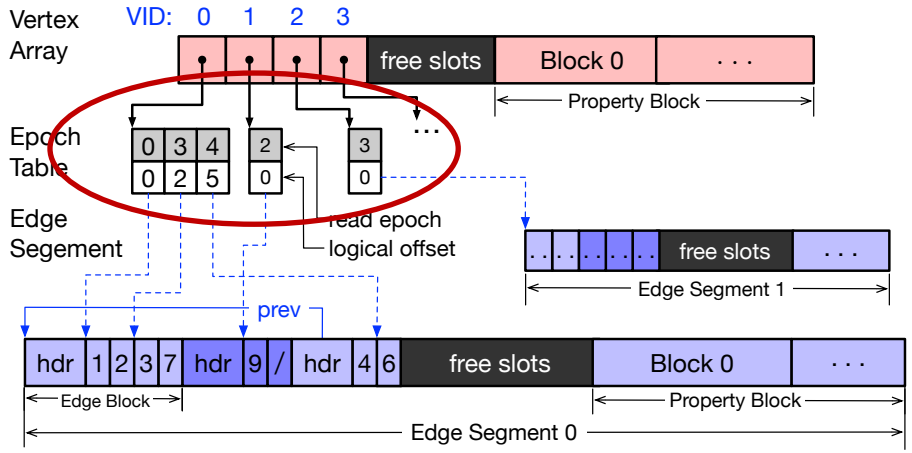
- Segmented edge store

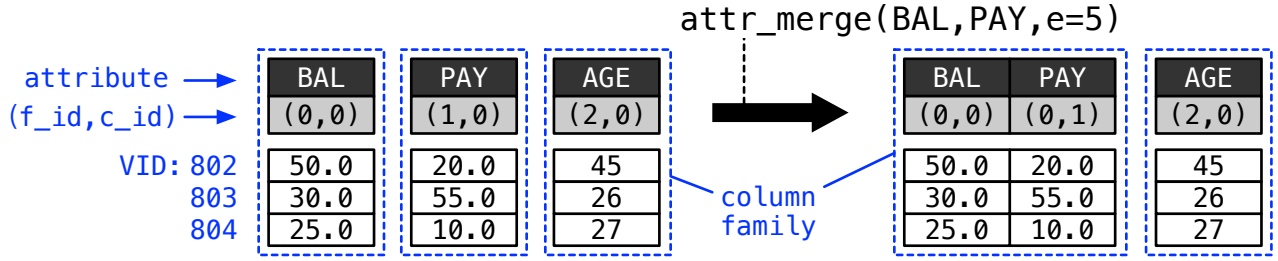
Coarse-grained MVCC

- Use epoch instead of timestamps

Flexible property storage

- User-defined property storage model



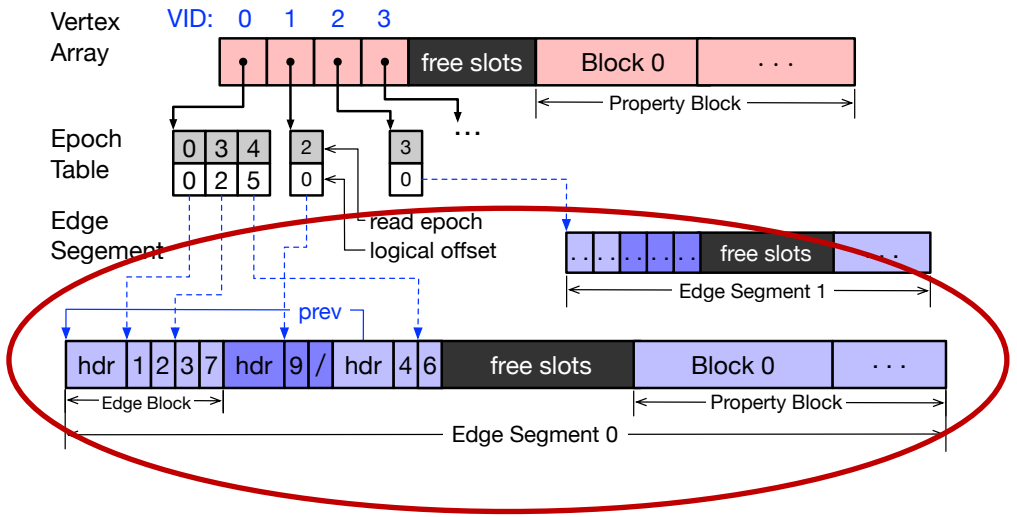


Efficient and Mutable CSR

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

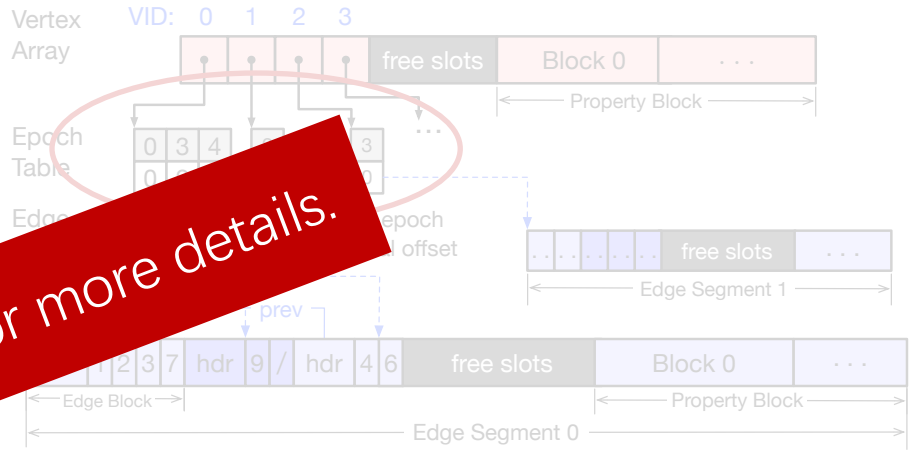
- Segmented edge store

Coarse-grained MVCC

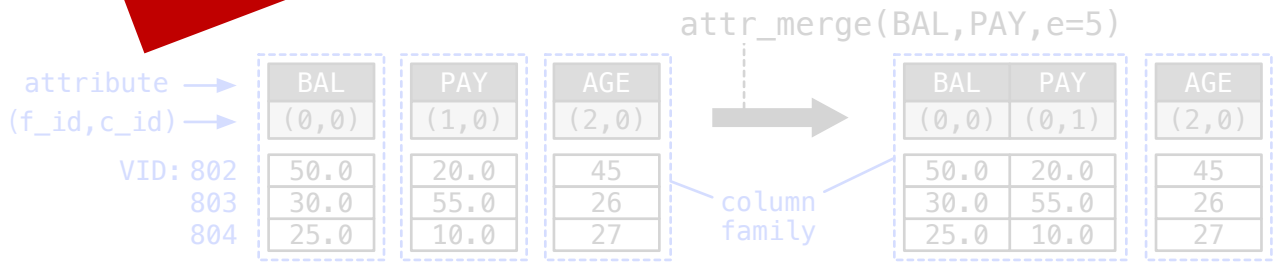
- Use epoch instead of timestamps

Flexible property storage

- User-defined property graph 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

Same OLTP and GAP engines as GART

- Offline: DrTM+H with GraphScope (DH+GS)
- Online: Neo4j

Graph database
Adjacency-list-based storage

- Replace storage by LiveGraph: G/LG

General-used dynamic graph storage

Workloads		LDBC SNB			
		GART	DH+GS	Neo4j	G/LG
OLTP ↑		1837 K	1929 K	3.5 K	1836 K
GA ↓	PR	377	309	5323	1276
	CC	362	312	4726	1137
	SSSP	513	433	4668	1381
GT ↓	IS-3	17.9	16.9	2.0	18.0
	BI-2	235	201	568	828
	BI-3	292	266	573	1278
GNN ↓	GCN	1097	940	×	1834
	GSG	1774	1443	×	2502
	SGC	779	717	×	1237
Freshness ↓		18	15683	5	25

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

Workloads		LDBC SNB			
		GART	DH+GS	Neo4j	G/LG
OLTP ↑		1837 K	1929 K	3.5 K	1836 K
GA ↓	PR	377	309	5323	1276
	CC	362	312	4726	1137
	SSSP	513	433	4668	1381
GT ↓	IS-3	17.9	16.9	2.0	18.0
	BI-2	235	201	568	828
	BI-3	292	266	573	1278
GNN ↓	GCN	1097	940	×	1834
	GSG	1774	1443	×	2502
	SGC	779	717	×	1237
Freshness ↓		18	15683	5	25

GAP Breakdown

GART outperforms G/LG due to storage

Breakdown analysis (e.g., PageRank)

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

	Storage	Topology	Property	Computation	Total
PR	GART	107 (28%)	163 (44%)	107 (28%)	377
	G/LG	658 (52%)	486 (38%)	132 (10%)	1276
BI-3	GART	10 (3%)	178 (61%)	104 (36%)	292
	G/LG	40 (3%)	1115 (87%)	123 (10%)	1278
SGC	GART	36 (5%)	477 (61%)	266 (34%)	779
	G/LG	236 (19%)	732 (59%)	269 (22%)	1237

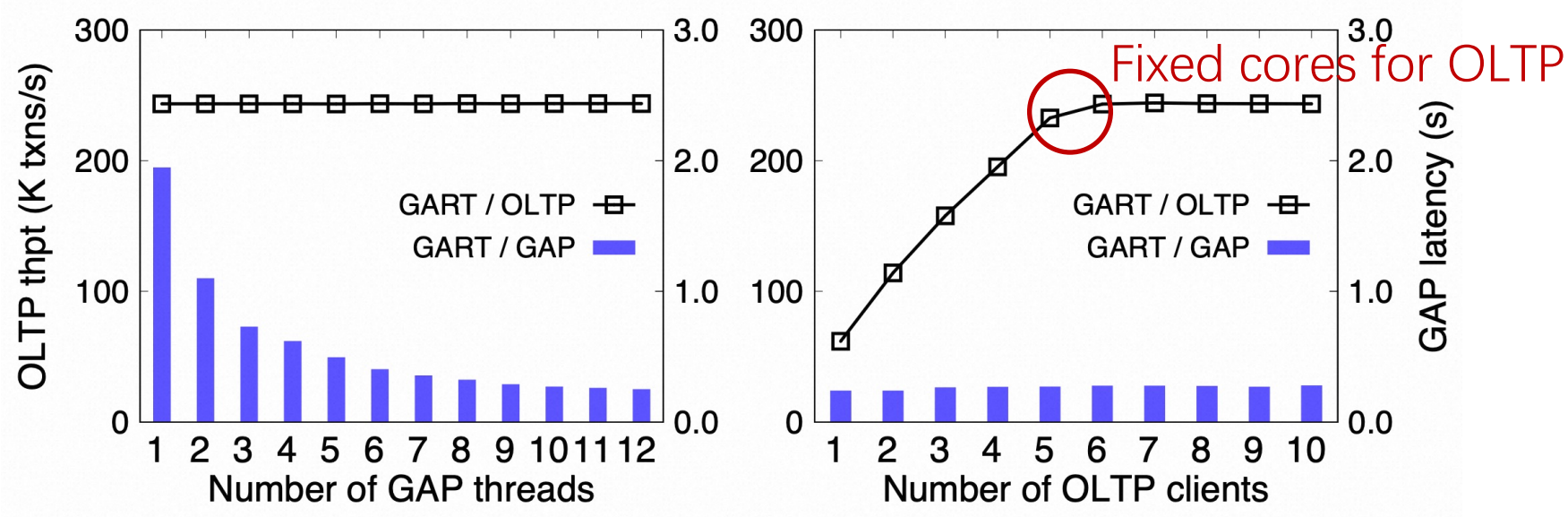
Workloads		LDBC SNB			
		GART	DH+GS	Neo4j	G/LG
OLTP ↑		1837 K	1929 K	3.5 K	1836 K
GA ↓	PR	377	309	5323	1276
	CC	362	312	4726	1137
	SSSP	513	433	4668	1381
GT ↓	IS-3	17.9	16.9	2.0	18.0
	BI-2	235	201	568	828
	BI-3	292	266	573	1278
GNN ↓	GCN	1097	940	×	1834
	GSG	1774	1443	×	2502
	SGC	779	717	×	1237
Freshness ↓		18	15683	5	25

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