Introducing GART
Real-Time Online Graph Computation for SQL

Sijie Shen

Institute for Intelligent Computing

Alibaba

2024-08-31

Graph Computation for Relational Datasets

Data usually stored and updated in relational OLTP systems

Online Graph Computation: Graph data is 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

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

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

Requirements of Online Graph Computation

Performance

- Comparable to specific graph systems

Freshness

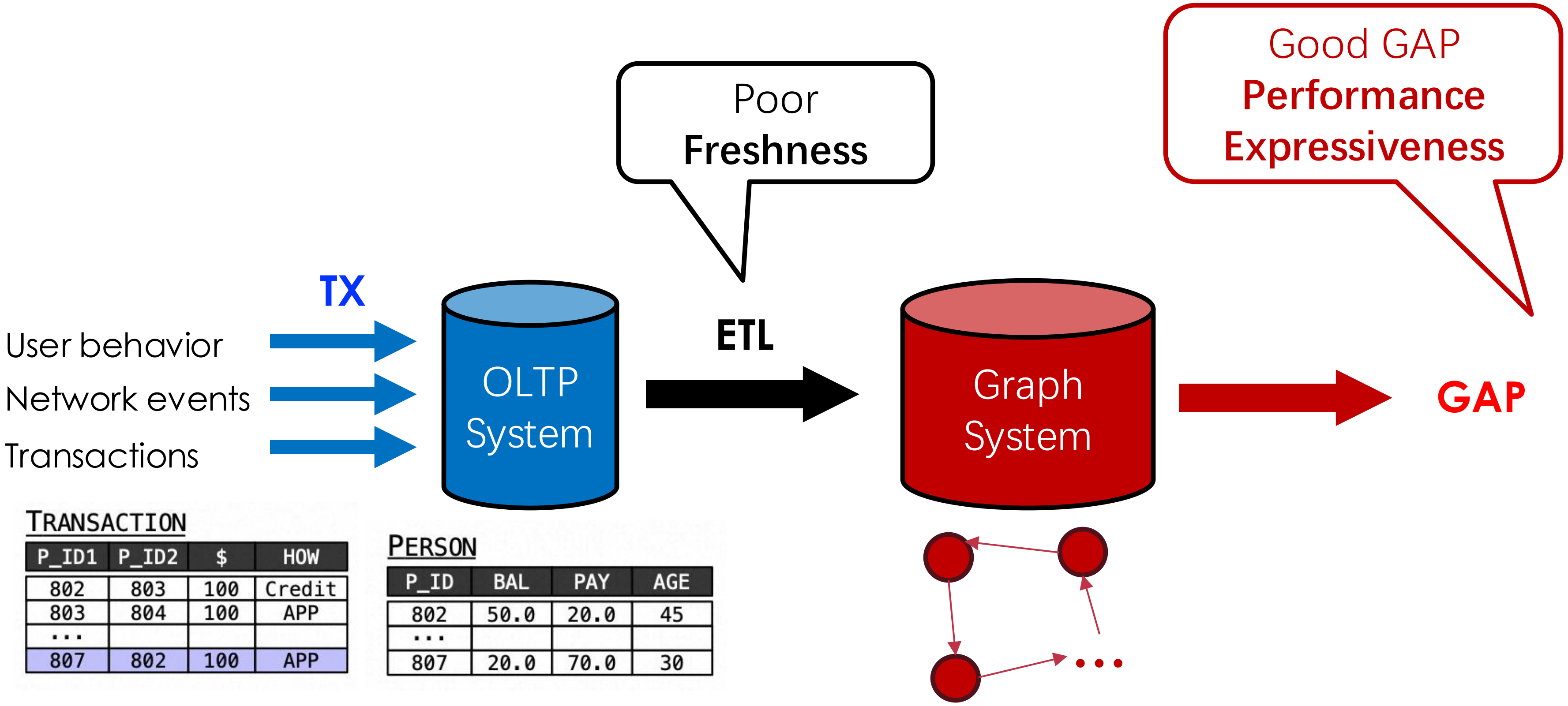
- Minimize the time gap between when data is committed and read

Expressiveness

- Sufficient graph representation for diverse graph workloads

Existing Solution 1/2: Processing on Offline Data

Combine OLTP systems with graph-specific systems



TRANSACTION

P_ID1	P_ID2	$	HOW
802	803	100	Credit
803	804	100	APP
...			
807	802	100	APP

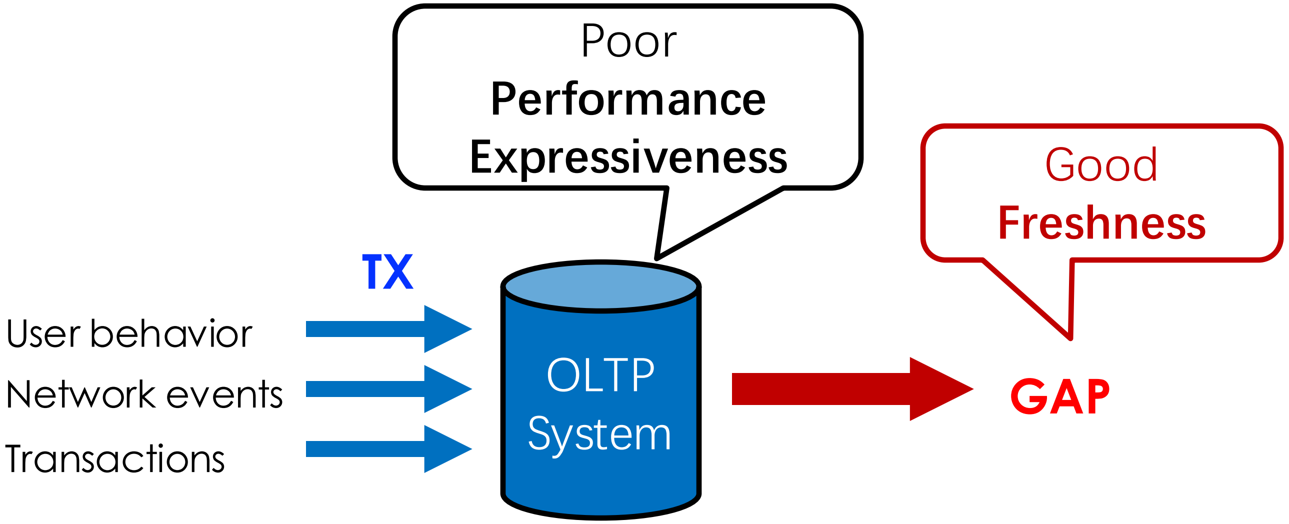
PERSON

P_ID	BAL	PAY	AGE
802	50.0	20.0	45
...			
807	20.0	70.0	30

Existing Solution 2/2: Processing on Online Data

OLTP systems support graph processing

- **Graph extension** in relational systems (SQL/PGQ, SQL Server)
- **Graph database** (Neo4j, TigerGraph)



neo4j

TigerGraph

```
SELECT a, b
FROM GRAPH_TABLE(student_network
  MATCH (a IS Person)-[e is knows]->(b is Person)
  COLUMNS(a.name as a, b.name as b))
```

SQL/PGQ

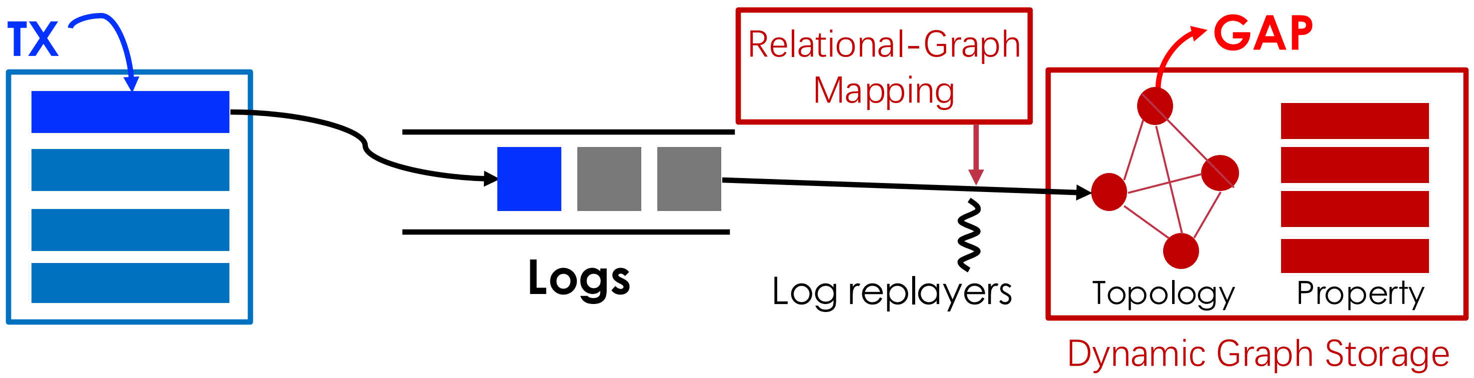
```
SELECT  Division.Name, Employee.Name
FROM    Division, Employee,
        Company Client, Company Merchant
MATCH   Division—[Employees]→Employee
        —[Clients]→Client,
        Employee—[Merchants]→Merchant
WHERE   Merchant.Location = 'Seattle' AND
        Client.Location = 'New York'
```

SQL Graph @ SQL Server

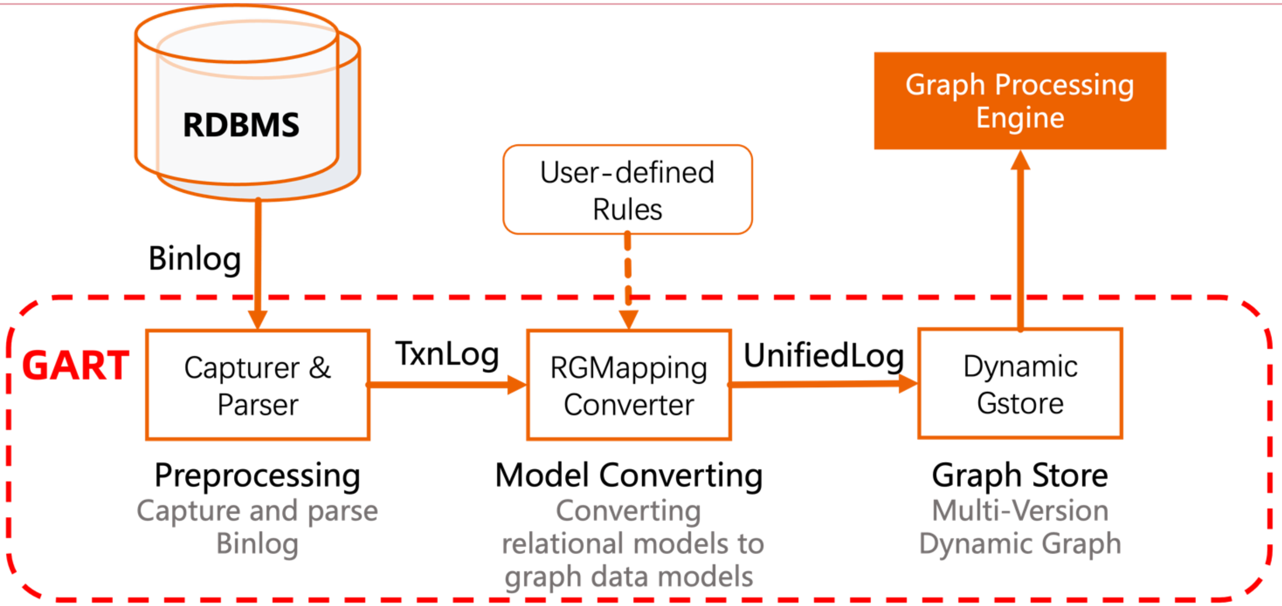
Our Approach: GART

GART: in-memory HTGAP system for dynamic GAP

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



Architecture & Workflow



Preprocess (Capture & Parser)

- Use transactional logs (e.g., binlog) to capture data changes

Model Convert (RGMapping Converter)

- GART not need to rewrite requests

Graph Store (Dynamic GStore)

- Support efficient read and write simultaneously

Preprocess: Log Capturer

Capture data changes from data sources by logs

- e.g., Binlogs in SQL systems
- Convert raw logs to TxnLogs with necessary data change information
- Now use Debezium (for MySQL, PostgreSQL, ...)


```
{
  "before": null,
  “after”: { “org_id”: “0”, “org_type”: “company”,
     "org_name": "Kam_Air",
     "org_url": "http://dbpedia.org/resource/Kam_Air" },
  "source": { "ts_ms": 1689159703811, "db": "ldbc",
     "table": "organisation" },
  "op": "c"
}
```

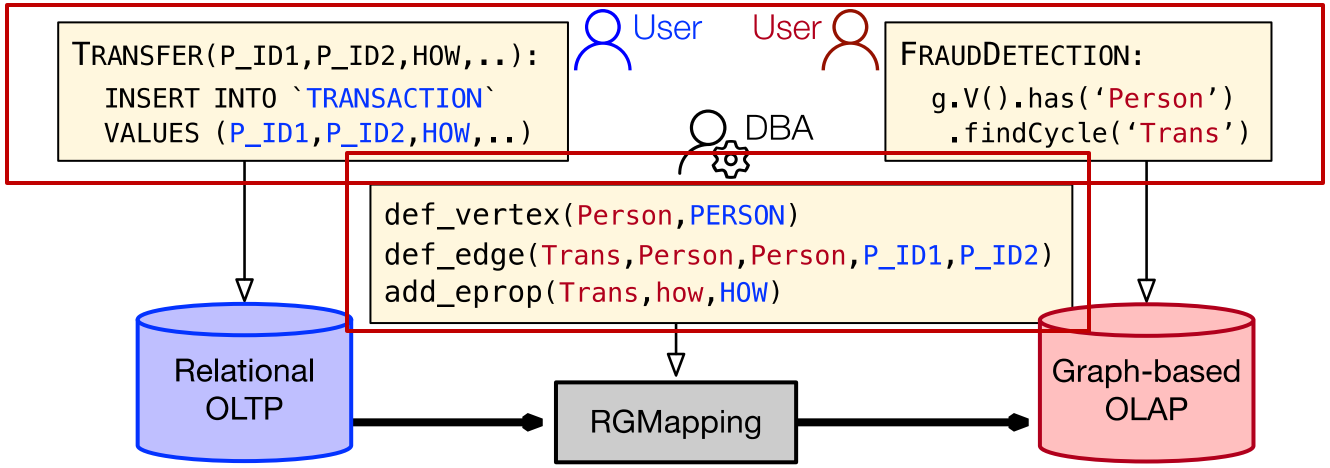
Model Convert

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

Extract property graph schema from relational schema

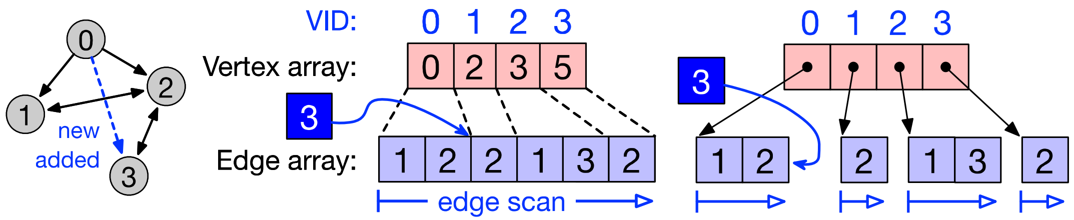
- Tables → Vertex or edge types
- Attributes → Properties
- Supported formats: SQL/PGQ, YAML, JSON, …

```
CREATE PROPERTY GRAPH ldbc
  VERTEX TABLES (
    "PERSON"
    KEY ( "p_id" )
    LABEL "person" PROPERTIES ( p_id AS "p_id", name AS "p_name" )
  )
  EDGE TABLES (
    "TRANSFER"
    SOURCE KEY ( "P_ID1" ) REFERENCES "PERSON"
    DESTINATION KEY ( "P_ID2" ) REFERENCES "PERSON"
    LABEL "transfer" PROPERTIES ( t_data AS "t_date" )
  )
```

Problems of Dynamic Graph Storage

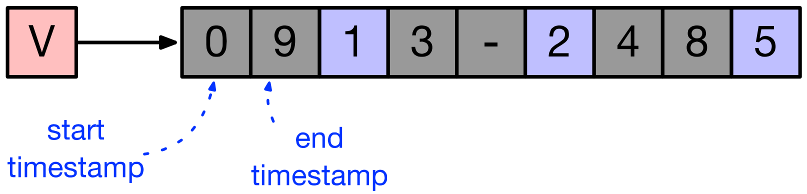
Topology

- 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 from Online Graph Computation

Embracing slight freshness trade-offs opens design optimization opportunities

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

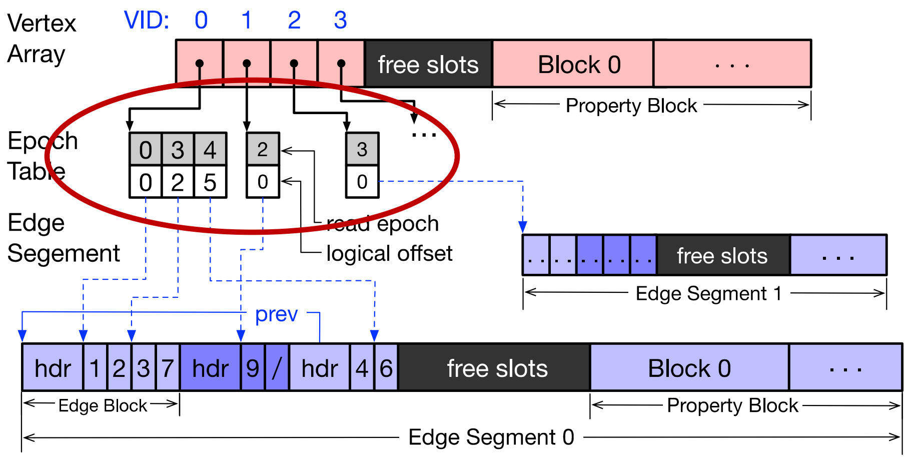
- Segmented edge store

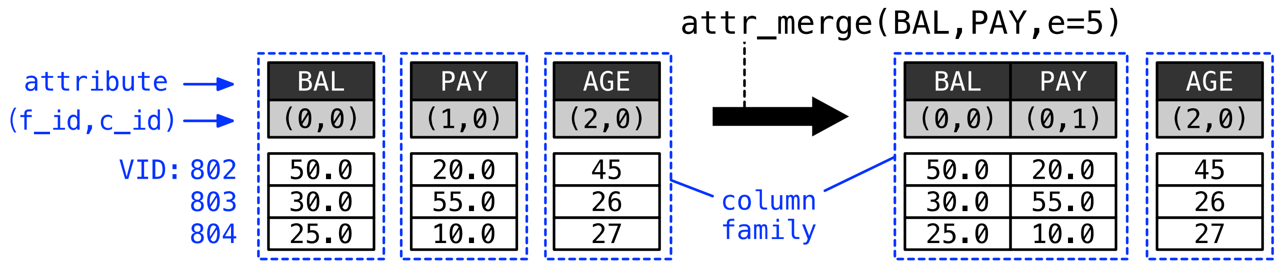
Coarse-grained MVCC

- Use epoch instead of timestamps

Flexible property storage

- User-defined property storage model





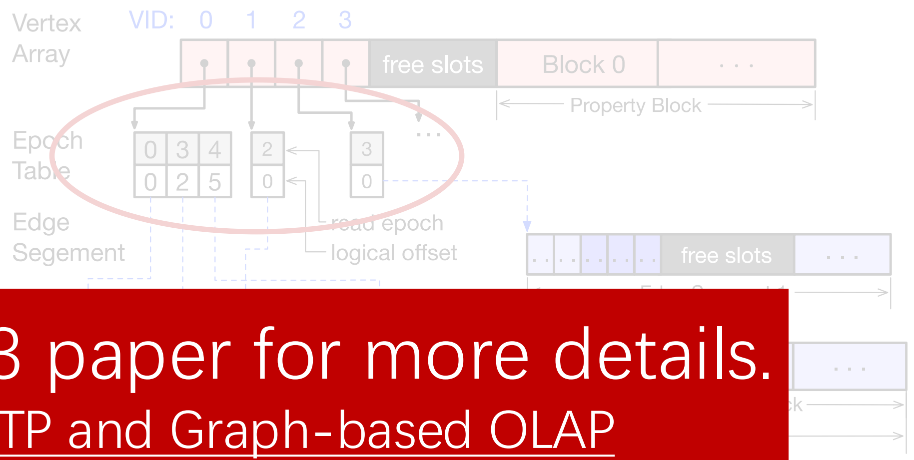
Graph Storage of GART

Efficient and mutable CSR

- Segmented edge store

Coarse-grained MVCC

- Use epoch instead of timestamps



Please refer to our USENIX ATC'23 paper for more details.

Bridging the Gap between Relational OLTP and Graph-based OLAP

Flexib

- User-defined property storage model

Evaluation

Testbed

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

Benchmark (extended for Online Graph Computation)

- 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
- **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

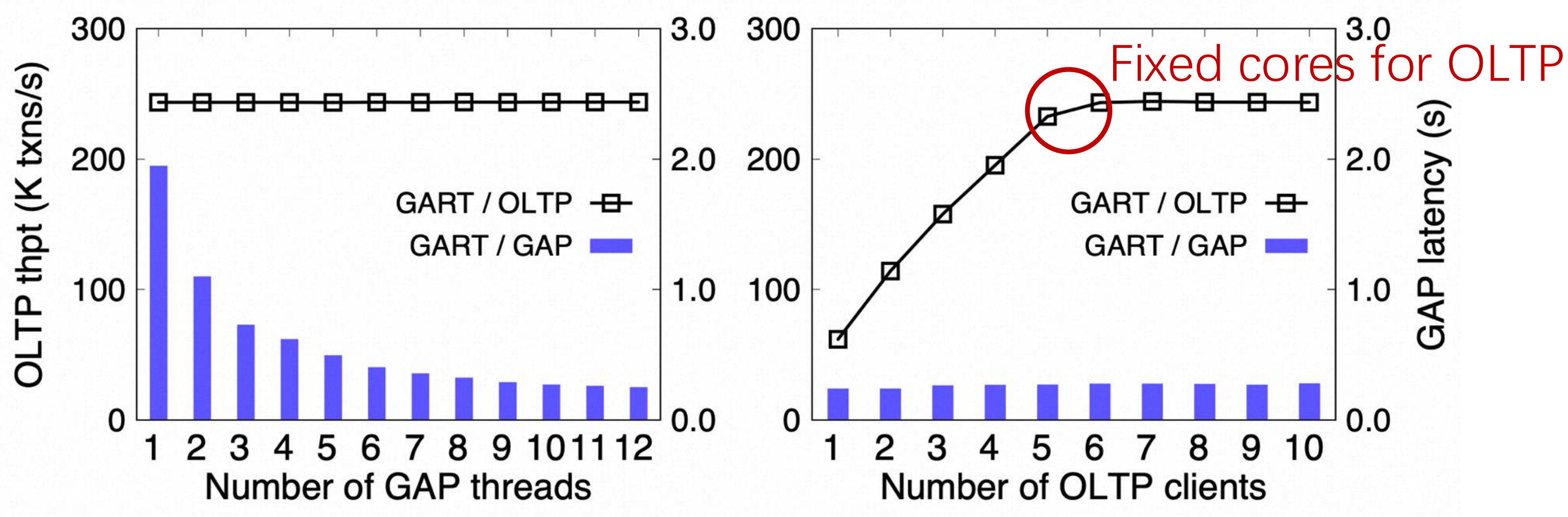
Workloads		LDBC SNB			
		GART	DH+GS	Neo4j	G/LG
OLTP ↑		1837 K	1929 K	3.5 K	1836 K
GA ↓	PR	377	309	5323	1276
	CC	362	312	4726	1137
	SSSP	513	433	4668	1381
GT ↓	IS-3	17.9	16.9	2.0	18.0
	BI-2	235	201	568	828
	BI-3	292	266	573	1278
GNN ↓	GCN	1097	940	×	1834
	GSG	1774	1443	×	2502
	SGC	779	717	×	1237
Freshness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

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