IPADS@Shanghai Jiao Tong Univ.† and Johns Hopkins Univ.* http://ipads.se.sjtu.edu.cn/projects/wukong

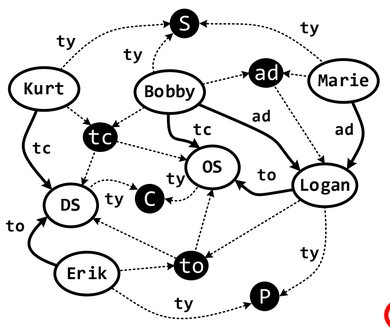
Fast and Concurrent Distributed RDF Queries using RDMA-assisted GPU Graph Exploration

Chang Lou*†, Siyuan Wang†, Rong Chen† and Haibo Chen†

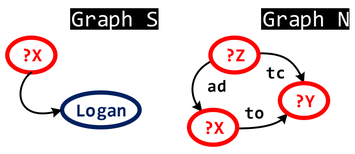
Motivation

Social Networks, Internet of Things and Business Intelligence applications model data as RDF Graphs and query with SPARQL query language.

Heterogeneity: selective and non-selective queries




```
SELECT ?X WHERE {
   ?X advisor Logan .
}                          SPARQL S
SELECT ?X ?Y ?Z WHERE {
   ?X teacherOf ?Y .
   ?Z takesCourse ?Y .
   ?Z advisor ?X .
}                          SPARQL N
```



Selective queries:
Light but frequent
e.g. Who is Logan's advisor?

Non-selective queries:
Infrequent but heavy
e.g. Who take their advisors' courses?

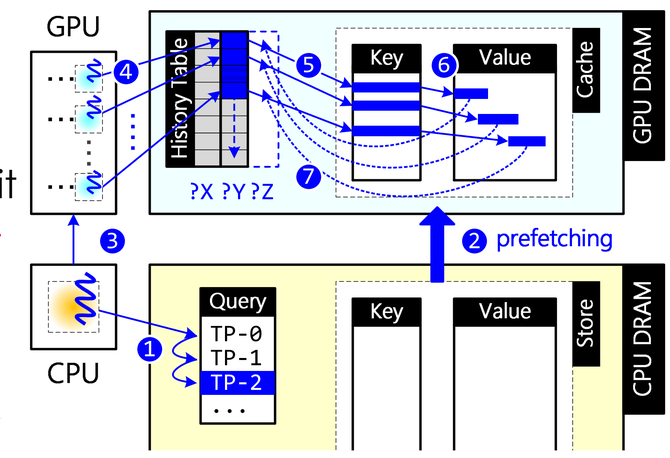
How to classify queries?
Graph traversal pattern

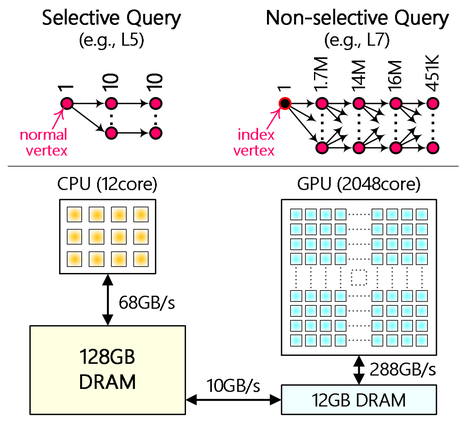
Problem: inefficient query processing on massive data parallelism and lack of execution isolation leads to 1) sub-optimal response time and 2) workload interference

Model & Solutions

Computation Model

Basic approach:
Leverage GPUs to exploit data parallelism in non-selective queries

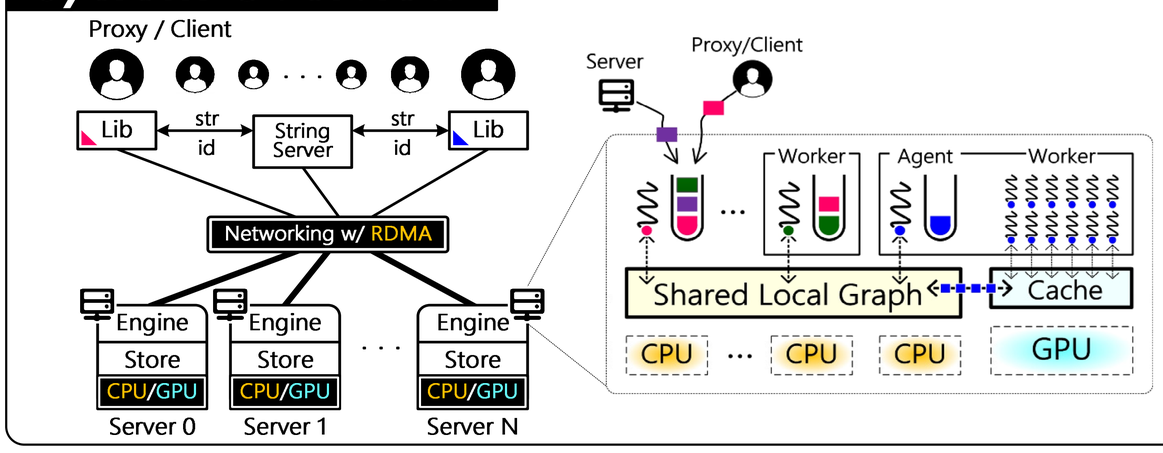




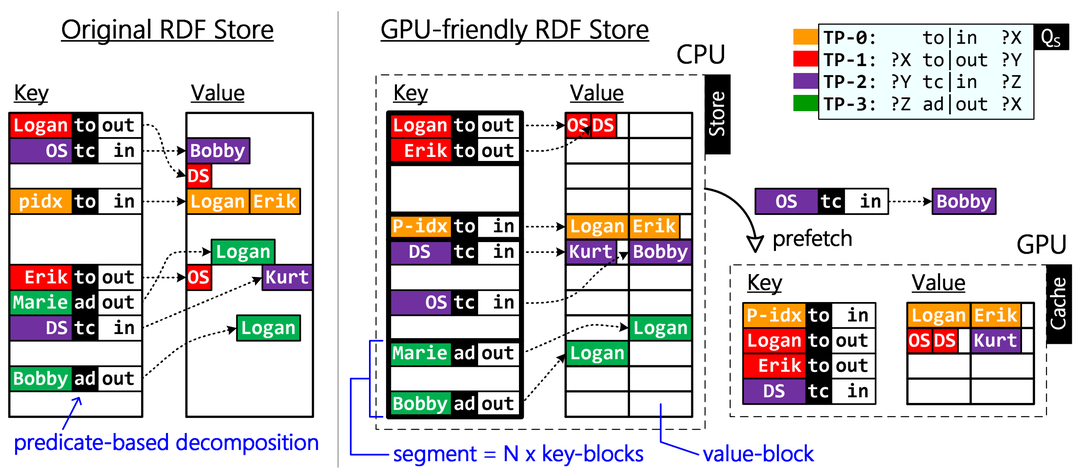
Challenges:

- Small GPU memory for large graphs
- Limited PCIe bandwidth for data movement
- Cross-node GPU communication for query distribution

System Overview

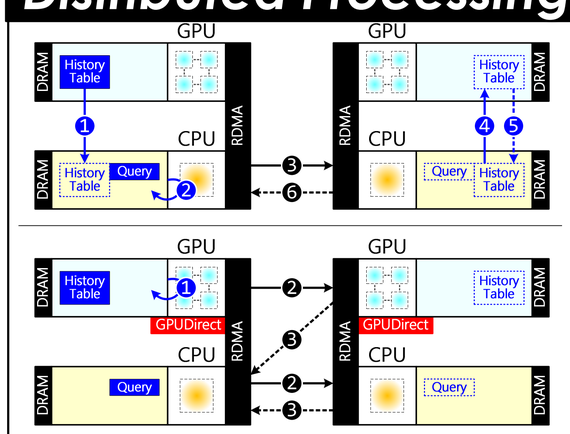


Graph Store



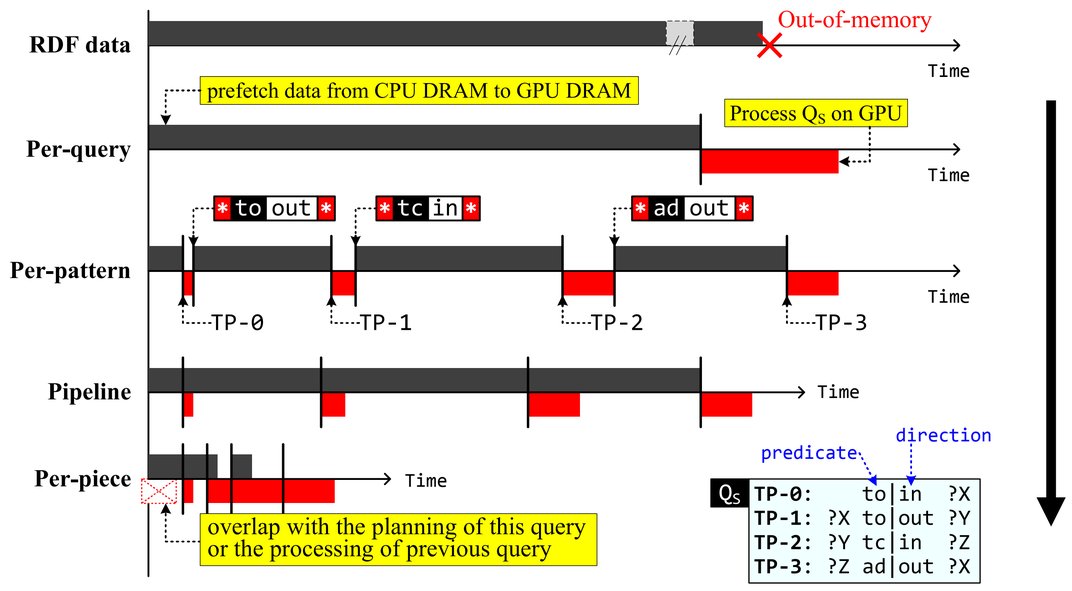
- Predicate-based grouping: partition the key space into multiple segments, which are identified by the combination of predicate and direction (e.g, ⟨pid,d⟩).
- Caching RDF store: splits each segment into multiple fixed-size blocks and allows to store them into discontinuous regions of the cache on GPU.
- Replacement policy: uses a look-ahead LRU-based policy to decide where to store the new prefetched value and key blocks.

Distributed Processing



- Parallel sub-query generation: leverages GPU to fast break history tables
- Direct sub-query distribution: adopts GPUDirect RDMA to avoid unnecessary data copy

Query Execution



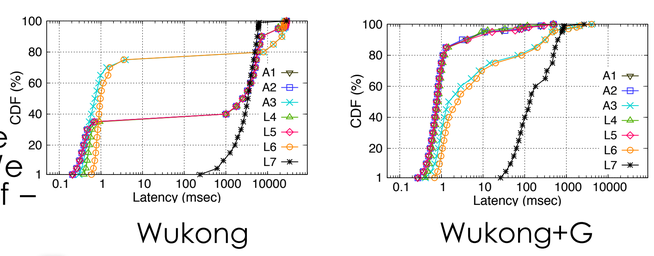
How we improve memory and time efficiency step by step:

- all graph scale: potential GPU memory overflow
- per-query scale : only cache the necessary data retained in GPU memory before running a query.
- per-pattern scale : only prefetches the triples with a certain predicate used by the next triple pattern.
- Pipeline: overlap the data movement and query execution time
- per-piece scale : further split predicates into multiple fixed-size blocks and cached them in a best-effort way.

Case Study

Background

We use Lehigh University Benchmark (LUBM) which includes both selective queries and non-selective queries. We compare our system with the state-of – art systems Wukong and TriAD.



Settings

Hardware Settings

- 5-node cluster, 12 cores each
- 56Gbps InfiniBand NIC

Benchmark Settings

- Single query latencies
- Mixed concurrent latency CDF

Note that datasets on 5 nodes is 4x larger than one node test.

Results

Performance (msec) on LUBM-10240

TriAD	L1	L2	L3	L4	L5	L6	L7
1 node	864	210	421	2.25	1.23	16.2	2,149
5 nodes	3,400	880	2,835	3.08	1.84	65.2	10,806
Wukong	**L1**	**L2**	**L3**	**L4**	**L5**	**L6**	**L7**
1 node	1,127	166	442	0.15	0.09	0.46	1,012
5 nodes	950	141	353	0.36	0.16	0.57	886
WukongG	**L1**	**L2**	**L3**	**L4**	**L5**	**L6**	**L7**
1 node	203	33	69	0.13	0.10	0.45	131
5 nodes	278	25	52	0.48	0.17	0.61	128

Summary

Wukong+G:

1. GPU-based RDF Query Execution
2. GPU-friendly RDF Graph Store
3. GPU & RDMA-accelerated Query Distribution

*This work was done while the author was in SJTU.