Weaver: A High Performance, Transactional Graph Database Based on Refinable Timestamps

By Dubey et al.

Presented by: Ishank Jain

Department of Computer Science

02/12/2019

UNIVERSITY OF WATERLOO
FACULTY OF MATHEMATICS

CONTENT

- Related work
- Research question
- Method
- Challenges
- Results
- Future work
- Questions

RELATED WORK

- Offline Graph Processing Systems
- Online Graph Databases
- Temporal Graph Databases
- Consistency Models
- Concurrency Control

RESEARCH QUESTION

- Existing systems either operate on offline snapshots, provide weak consistency guarantees, or use expensive concurrency control techniques that limit performance.
- The **key challenge** in a transactional system is to ensure that distributed operations taking place on different machines follow a **coherent timeline**.

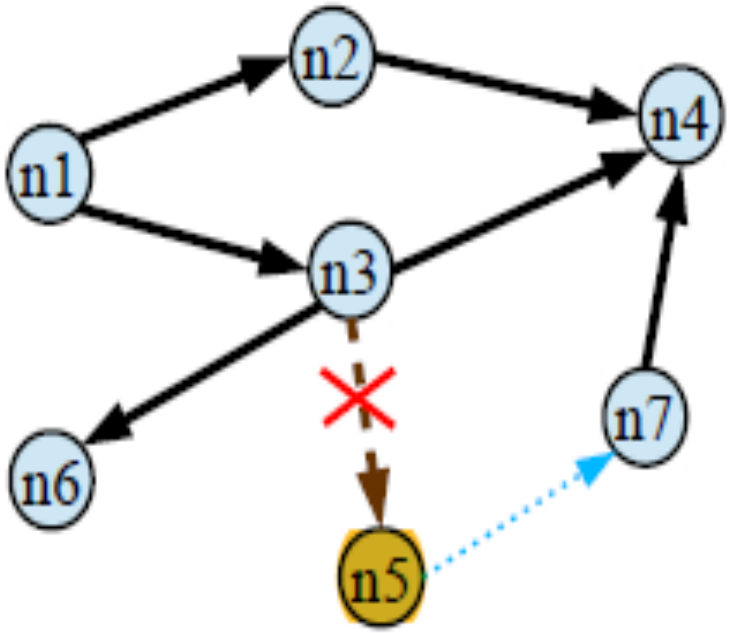
PROBLEM EXAMPLE

- Path discovery query

  n3 -> n5: removed

  n5 -> n7: added

  n1 -> n7 ?



REDIFINALBLE TIMESTAMPS

- This technique Couples a**) coarse-grained vector timestamps** b) a **fine-grained timeline oracle** to pay the overhead.
- Fine-grained timeline oracle is used for ordering only the potentially-conflicting reads and writes.

```
begin_weaver_tx()
photo    = create_node()
own_edge = create_edge(user, photo)
assign_property(own_edge, "OWNS")
for nbr in permitted_neighbors:
  access_edge = create_edge(photo, nbr)
  assign_property(access_edge, "VISIBLE")
commit_weaver_tx()
```

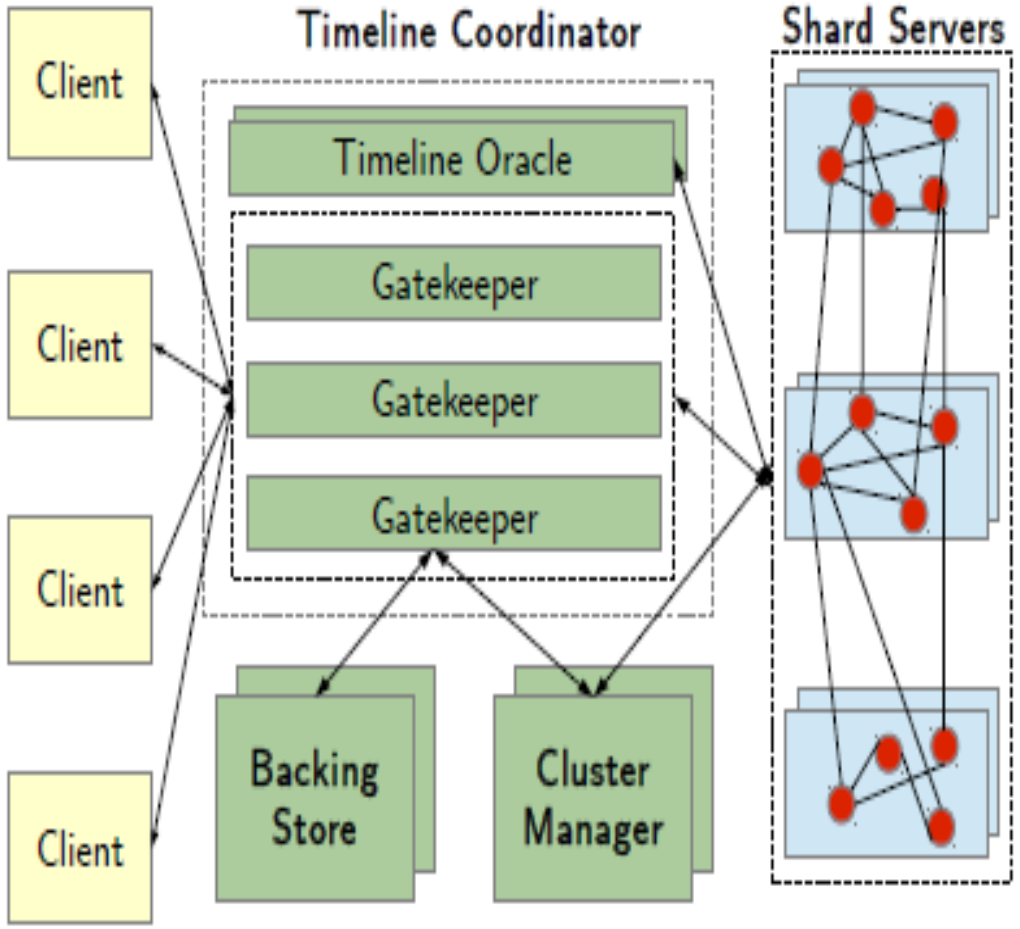
NODE PROGRAM

- Uses scatter-gather like property.
- Node programs are sometimes stateful.
- Node program state is garbage collected after the query terminates on all servers.
- Consistency: Weaver delays execution of a node program at a shard until after execution of all preceding and concurrent transactions.
- Supports transitivity.

```
node_program(node, prog_params):
 nxt_hop = []
 if not node.prog_state.visited:
  for edge in node.neighbors:
   if edge.check(prog_params.edge_prop):
    nxt_hop.append((edge.nbr,prog_params))
  node.prog_state.visited = true
 return nxt_hop
```

ARCHITECTURE

- **Shard Servers**: The shard servers are responsible for executing both node programs and transactions on the in-memory graph data.



ARCHITECTURE

- **Backing Store**:
  - Use HyperDex Warp as backing store.
  - Data recovery in case of failure.
  - Directs transactions on vertex.

Warp: Lightweight Multi-Key Transactions for Key-Value Stores

Robert Escriva[†], Bernard Wong[‡], Emin Gün Sirer[†]
[†] *Computer Science Department, Cornell University*
[‡] *Cheriton School of Computer Science, University of Waterloo*

Abstract

Traditional NoSQL systems scale by sharding data across multiple servers and by performing each operation on a small number of servers. Because transactions necessarily require coordination across multiple servers, NoSQL systems often explicitly avoid making transactional guarantees in order to avoid such coordination. Past work in this space has relied either on heavyweight protocols, such as two-phase commit or Paxos, or clock synchronization to perform this coordination.

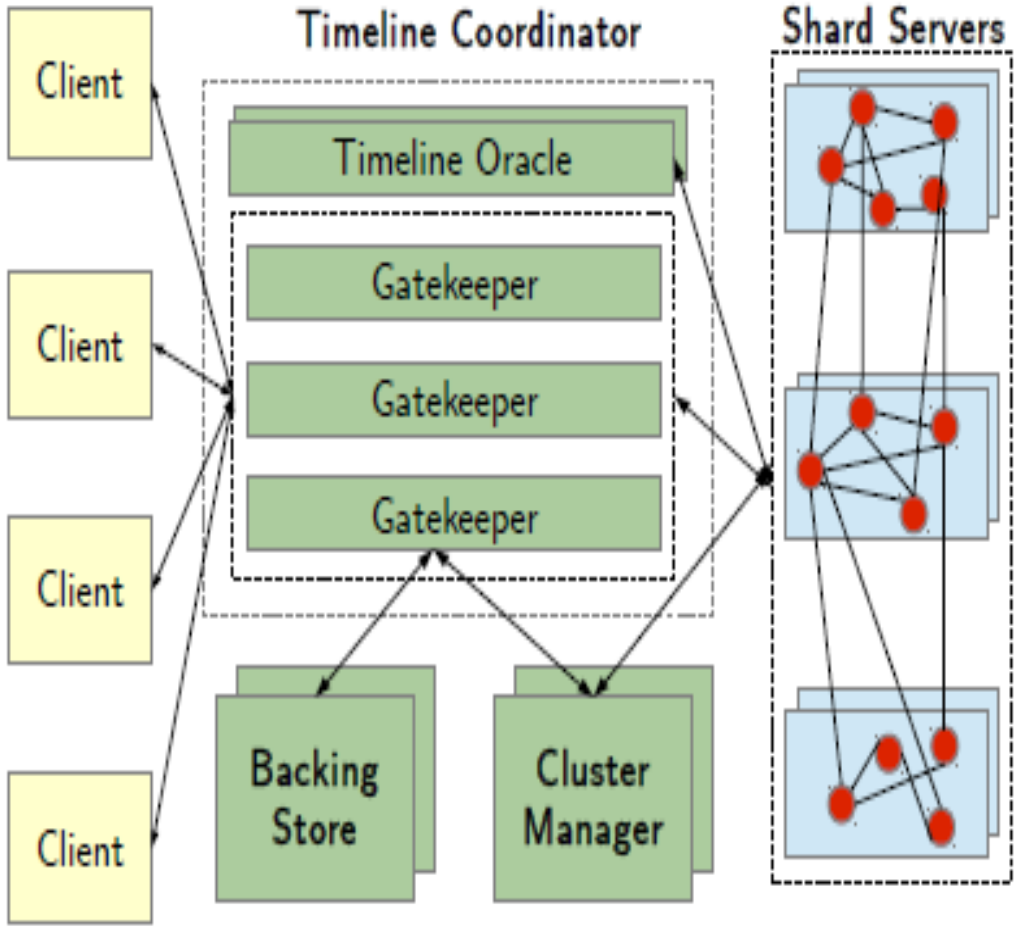
This paper presents a novel protocol for providing ACID transactions on top of a sharded data store. Called linear transactions, this protocol allows transactions to execute in natural arrival order unless doing so would violate serializability. We have fully implemented linear transactions in a commercially available data store. Experiments show that Warp achieves 3.2× higher throughput than Sinfonia's mini-transactions on the standard TPC-C benchmark with no aborts. Further, the system

Google's Megastore [5] shard the data across different Paxos groups based on their key, thereby gaining scalability, but incur higher coordination costs for actions that span multiple groups. An alternative approach, pursued in Calvin [43], is to serialize all operations using a consensus protocol and use deterministic execution to improve performance. Google's Spanner [12] relies on the TrueTime API to assign timestamps to transactions without cross-server synchronization. Compared to traditional NoSQL systems with simple and scalable designs, these systems introduce *spurious coordination* between transactions. Spurious coordination is when a transaction processing protocol unnecessarily delays or reorders transactions' execution in order to enforce an order between transactions that could be applied in natural arrival order. Coarse-grained consensus groups and centralized sequencers both exhibit varying degrees of spurious coordination.

This paper introduces Warp, a NoSQL system that

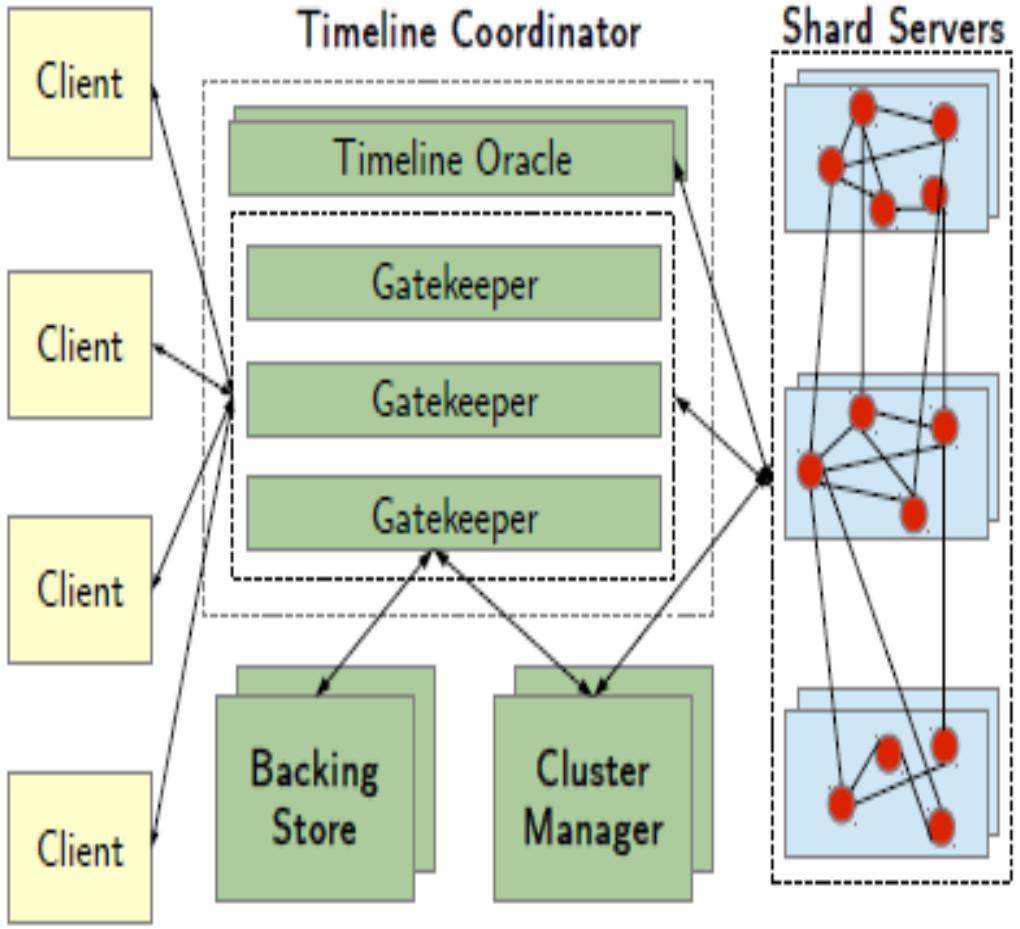
ARCHITECTURE

- **Timeline Coordinator**:
  - Gatekeeper
  - Timeline oracle



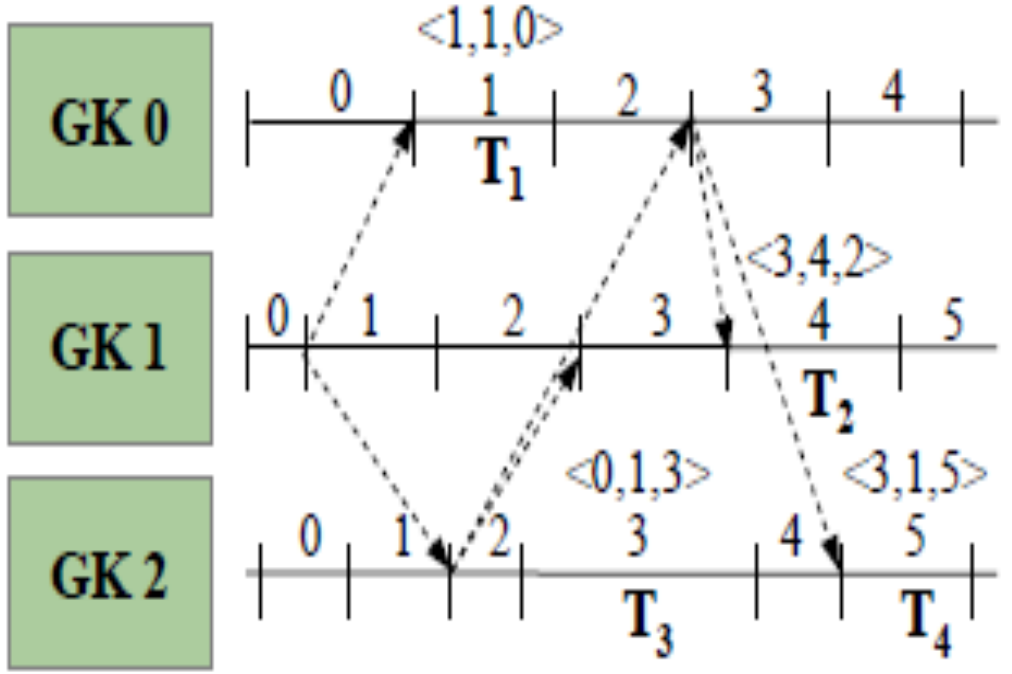
ARCHITECTURE

- **Cluster Manager**:
  - Failure detection,
  - System reconfiguration.



PROACTIVE ODERING USING GATEKEEPERS

- Vector clock.
- Maintains a happens-before **partial order** bet refinable timestamps.
- Synchronization period.



$T_1\langle 1,1,0\rangle \prec T_2\langle 3,4,2\rangle$ and $T_3\langle 0,1,3\rangle \prec T_4\langle 3,1,5\rangle$. T_2 and T_4 are concurrent and require fine-grain ordering only if they conflict.

PROACTIVE ODERING USING GATEKEEPERS

Timestamps in Message-Passing Systems That Preserve the Partial Ordering

Colin J. Fidge
Department of Computer Science, Australian National University, Canberra, ACT.

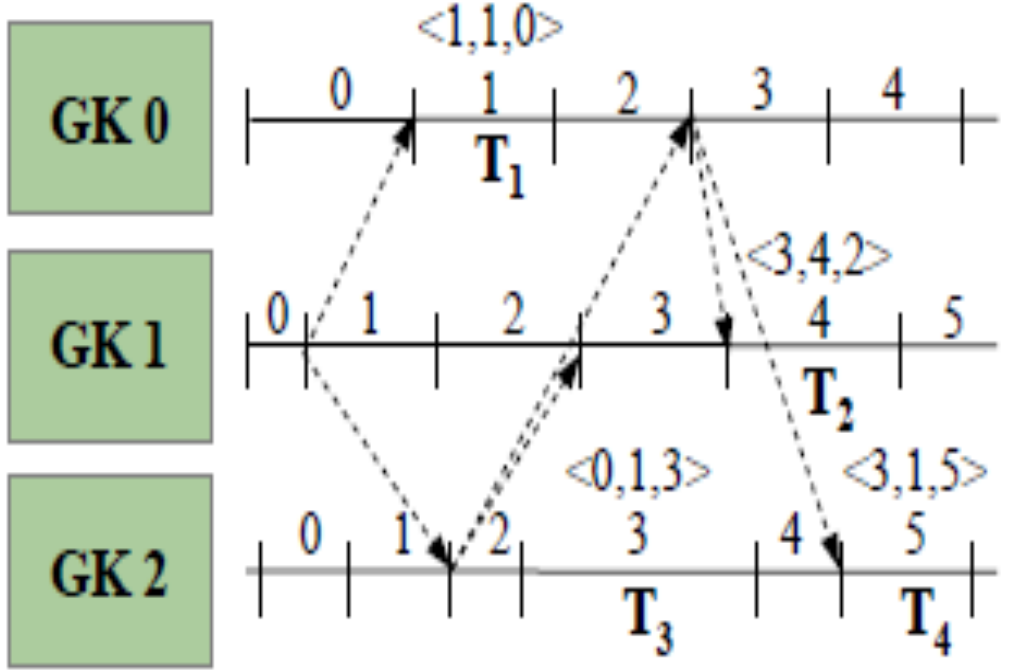
ABSTRACT

Timestamping is a common method of totally ordering events in concurrent programs. However, for applications requiring access to the global state, a total ordering is inappropriate. This paper presents algorithms for timestamping events in both synchronous and asynchronous message-passing programs that allow for access to the partial ordering inherent in a parallel system. The algorithms do not change the communications graph or require a central timestamp issuing authority.

Keywords and phrases: concurrent programming, message-passing, timestamps, logical clocks
CR categories: D.1.3

REACTIVE ORDERING BY TIMELINE ORACLE

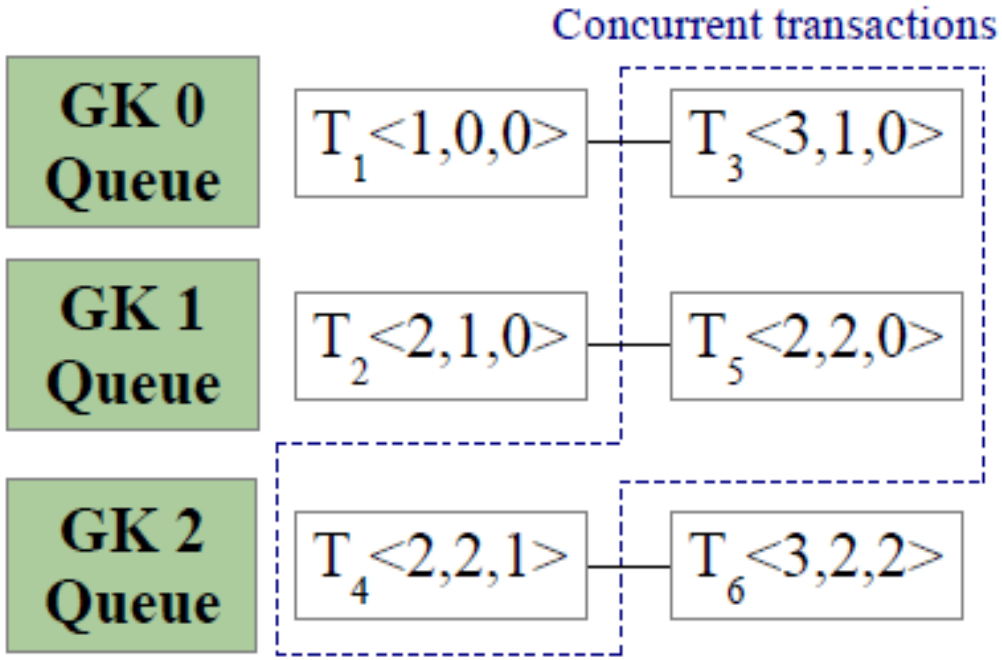
- Timeline oracle:
  - Guarantees graph remains acyclic.
- Event dependency graph and new event creation.



$T_1\langle 1,1,0\rangle \prec T_2\langle 3,4,2\rangle$ and $T_3\langle 0,1,3\rangle \prec T_4\langle 3,1,5\rangle$. T_2 and T_4 are concurrent and require fine-grain ordering only if they conflict.

TRANSACTIONS

- Transaction executed on backing store to ensure validity.
- FIFO channels,
- NOP transactions



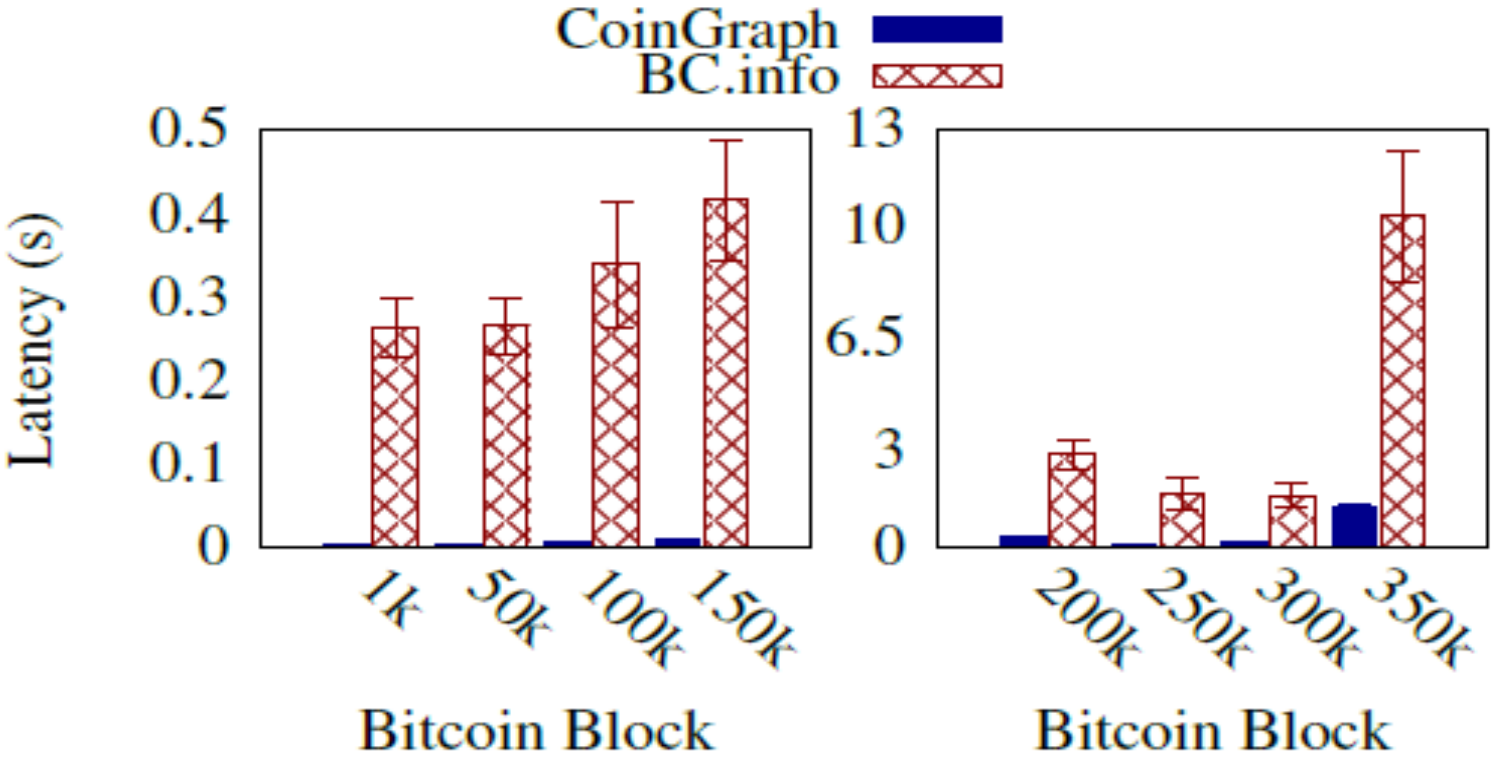
FAULT TOLERANCE

- Graph data persistently stored on backing store.
- All node programs, are re-executed by Weaver with a fresh timestamp after recovery.
- To maintain monotonicity of timestamps on gatekeeper failures, a backup gatekeeper restarts the vector clock for the failed gatekeeper.

GRAPH PARTITIONING & CACHING

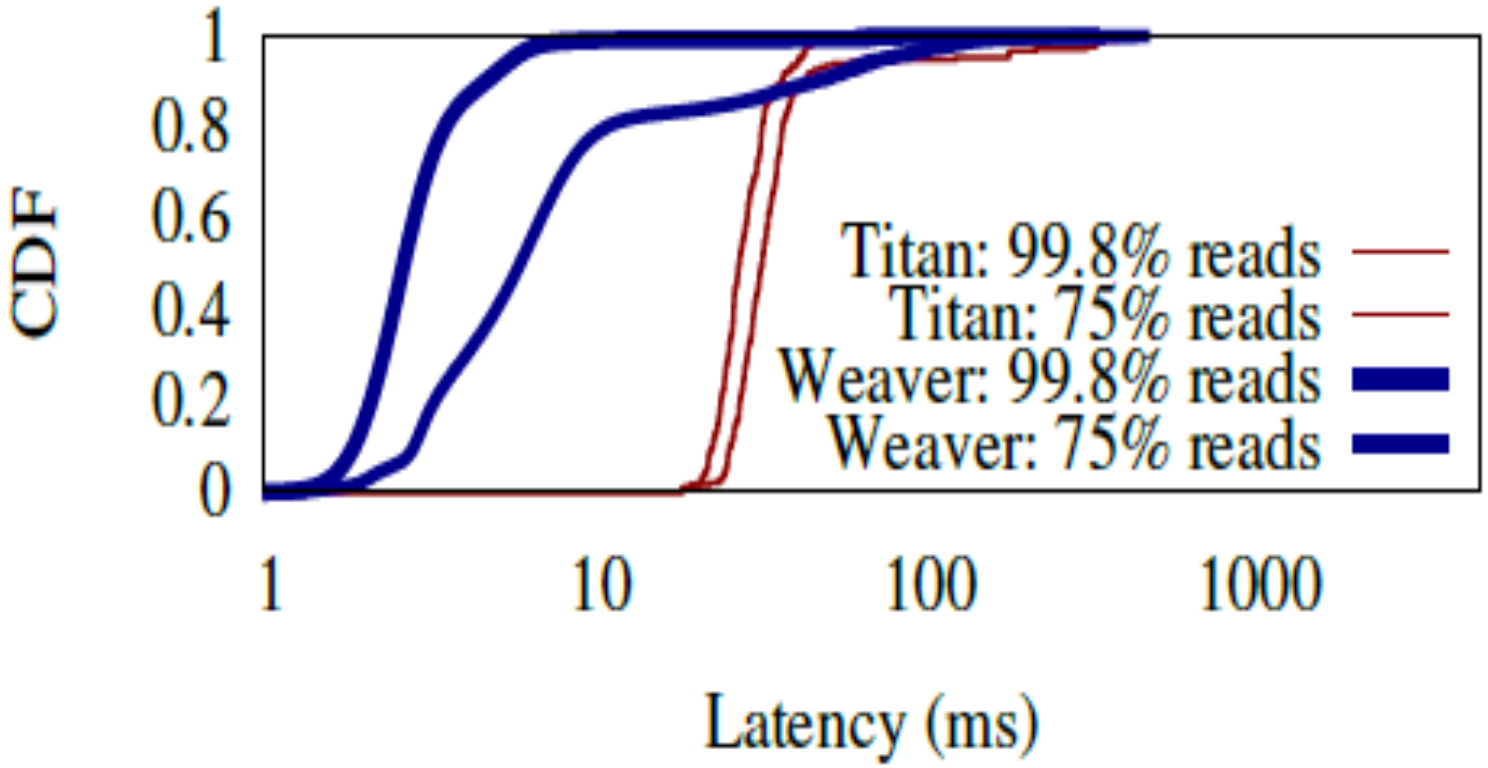
- Streaming graph partitioning algorithms:
  - To reduce communication overhead.
- Caching analysis for path discovery:
  - Path stored in cache at each vertex
  - Path deleted from cache once an edge in path deleted.

EVALUATION



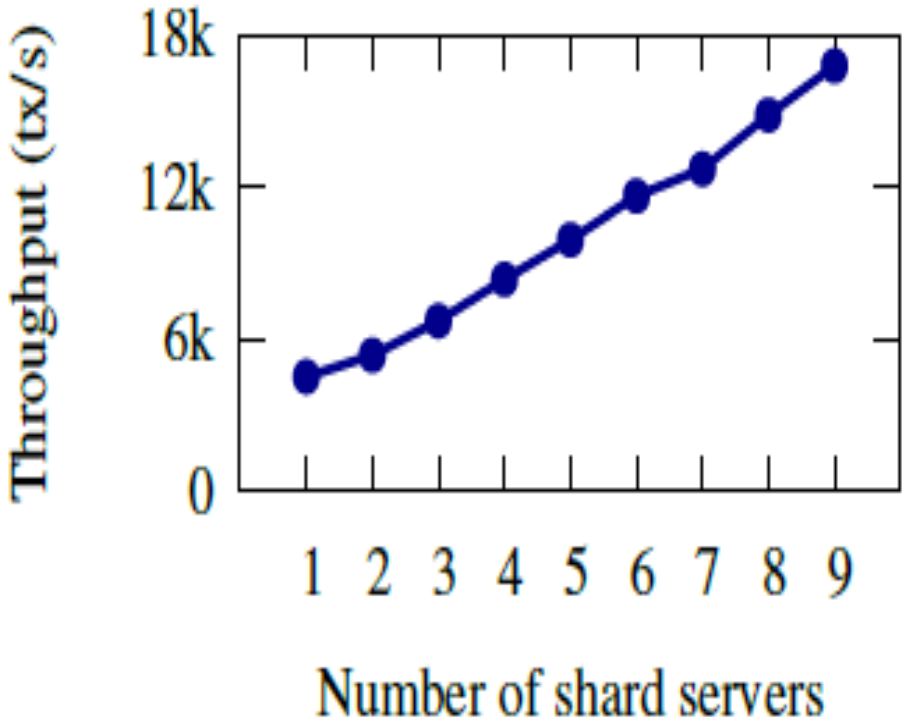
Average latency (secs) of a Bitcoin block query in blockchain application.

EVALUATION



Transaction latency for a social network workload on the LiveJournal graph.

EVALUATION



Shows almost linear scalability with the number of shards

RESULTS

- Weaver enables CoinGraph to execute Bitcoin block queries 8x faster than Blockchain.info.
- outperforms Titan by 10.9x on social network workload and outperforms GraphLab by 4x on node program workload
- Weaver scales linearly with the number of gatekeeper and shard servers for graph analysis queries.

IMPORTANT POINTS

- Proactive costs due to periodic synchronization messages between gatekeepers, and the reactive costs incurred at the timeline oracle needs to be carefully balanced.
- As synchronization period increases, the reliance on the timeline oracle increases.
- TrueTime system assumes no network or communication latency, so a system synchronized with average error bound ε will necessarily incur a mean latency of 2ε.
- Number of shard servers and gatekeepers in shard are the potential bottleneck for the query throughput. As synchronization period increases, the reliance on the timeline oracle increases.

QUESTIONS

- Why is node program allowed to visit a vertex multiple times in the weaver model ?
- The graph data in shard severs are kept in-memory, will keeping all data in-memory increase performance at expense of cost?
- Does creation of new event by timeline oracle in anyway effect the model ? (adding overheads)

REFERENCE

Ayush Dubey, Greg D. Hill, Robert Escriva, and Emin Gün Sirer. Weaver: a high-performance, transactional graph database based on refinable timestamps. Proc. VLDB Endow. 9(11): 852-863, 2016.