Preserving Reciprocal Consistency in Distributed Graph Databases

Jack Waudby[1], Paul Ezhilchelvan[1], Jim Webber[2] & Isi Mitrani[1]

April 27, 2020

[1]Newcastle University

[2]Neo4j

Property Graph

- Graph databases model data as a *property graph*
- Vertices represent entities and edges the relationships between entities
- Edges are **always** directional

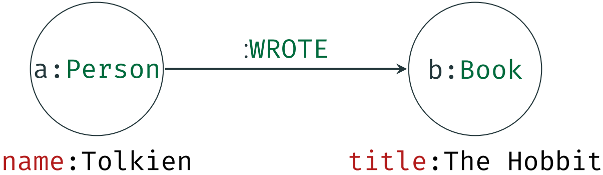
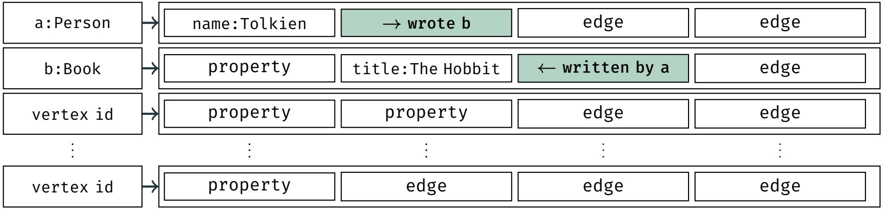


Figure 1: Vertices connected by an edge

Storage Layer Representation

- In the storage layer,
  - edge directionality does **not** exist
  - connected vertices store information about each other
- Bidirectional edge traversal speeds up query performance



a:Person	name:Tolkien	→ **wrote b**	edge	edge
b:Book	property	title:The Hobbit	← **written by a**	edge
vertex id	property	property	edge	edge
⋮	⋮	⋮	⋮	⋮
vertex id	property	edge	edge	edge

Figure 2: Edge storage layer representation

Reciprocal Consistency

When the adjacency list entries for a given edge refer to each other in a complementary manner, that edge is ***reciprocally consistent***

Distributed Graph Databases

Partition graph across machines in a cluster

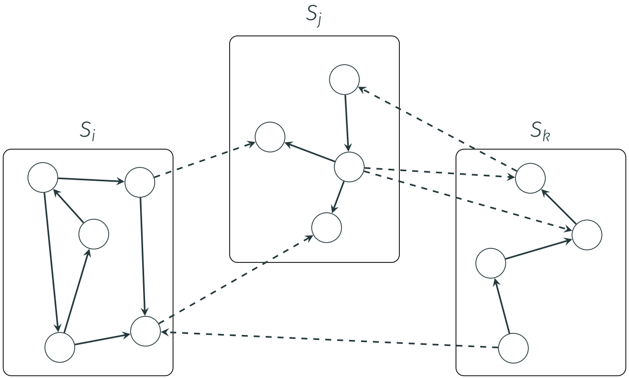


Figure 3: Partitioned graph

Distributed Graph Databases

Some concurrency control is needed for ensuring reciprocal consistency of distributed edges

Reciprocal Inconsistency

Distributed edge *ab* indicates `Tolkien` wrote `The Hobbit`

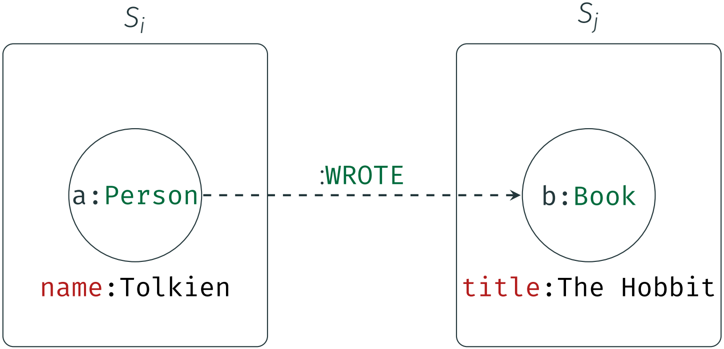
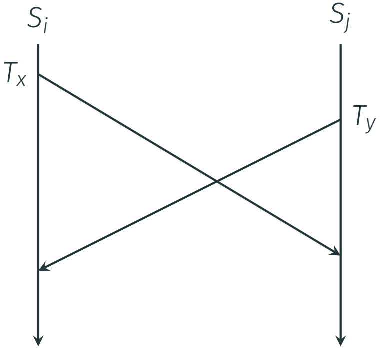


Figure 4: Distributed edge

Reciprocal Inconsistency

- T_x deletes the edge
- T_y appends a property `year`



Reciprocal Inconsistency

The distributed edge is now reciprocally inconsistent

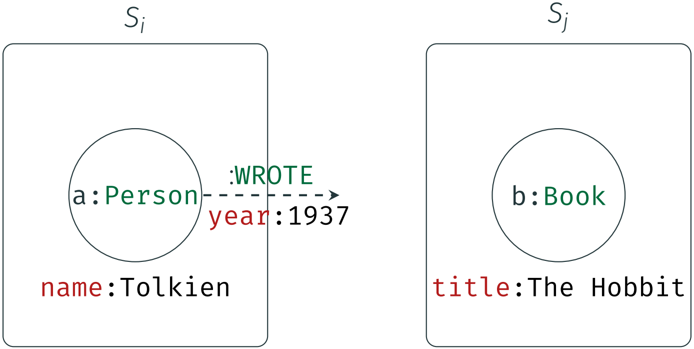
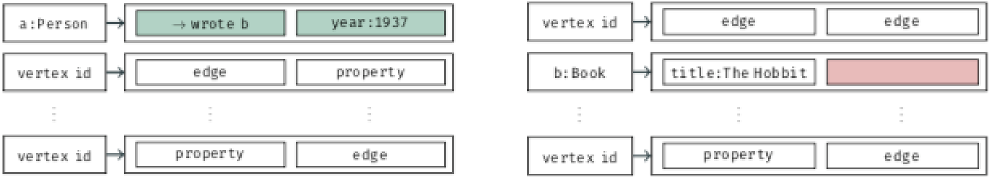


Figure 5: Reciprocally inconsistent distributed edge

Reciprocal Inconsistency

Storage representation consists of two inconsistent unidirectional edges



(a) S_i

(b) S_j

Figure 6: Storage representation

Reciprocal Inconsistency

- Reciprocally inconsistent edges are the source for *semantic corruption*
- Semantic corruption spreads until the database becomes *operationally corrupt*
- **Motivated the design of a lightweight protocol that preserves reciprocal consistency**

Protocol Design Considerations

Design Considerations:

1. Graph workloads exhibit high contention.
2. Graph transactions tend to be long-lived than those in other databases.

Protocol must permit multiple updates on the same record provided they are they are *sufficiently* apart in time in ensure reciprocal consistency

Delta Protocol

- Fact: a transaction updating a distributed edge must update one edge pointer then **immediately** update the other.
- Rule: an update is permitted if the immediately preceding update was done at least Δ time before. Else, abort.
- Assumption: the time interval that elapses between completing at update at one end and starting at the other can be estimated, δ. Choosing $\Delta > \delta$.

Delta Protocol: Example

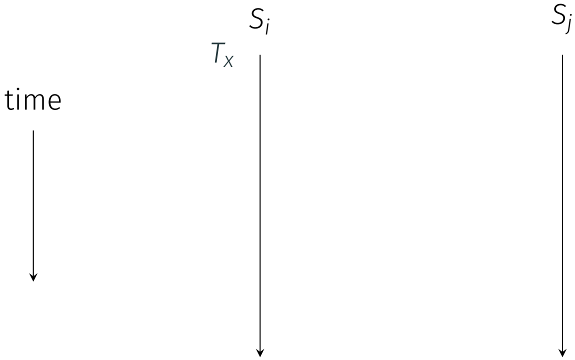
Rule: an update is permitted if the preceding update was done at least Δ time before.

Else, abort

Delta Protocol: Example

Rule: an update is permitted if the preceding update was done at least Δ time before.

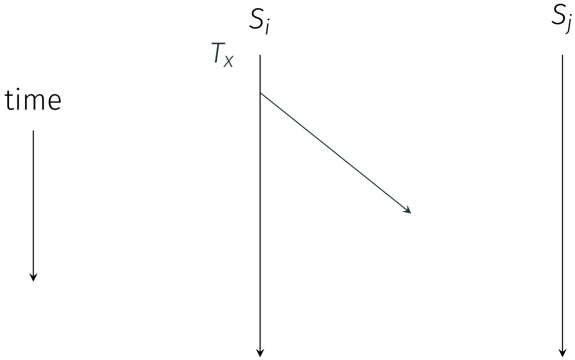
Else, abort



Delta Protocol: Example

Rule: an update is permitted if the preceding update was done at least Δ time before.

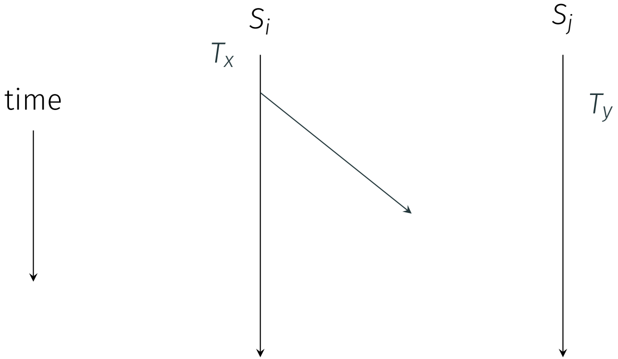
Else, abort



Delta Protocol: Example

Rule: an update is permitted if the preceding update was done at least Δ time before.

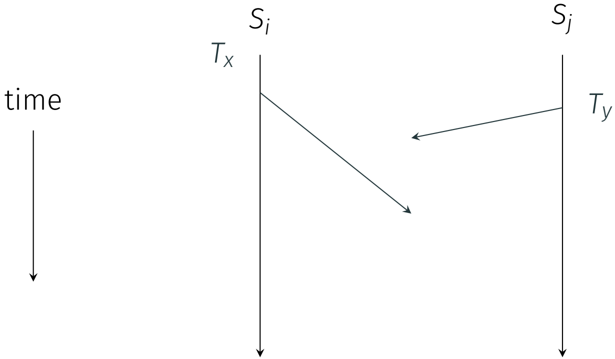
Else, abort



Delta Protocol: Example

Rule: an update is permitted if the preceding update was done at least Δ time before.

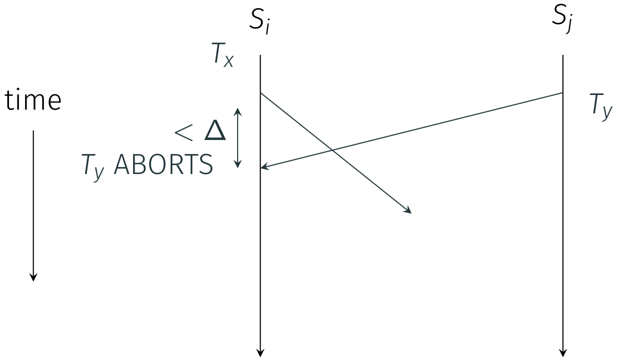
Else, abort



Delta Protocol: Example

Rule: an update is permitted if the preceding update was done at least Δ time before.

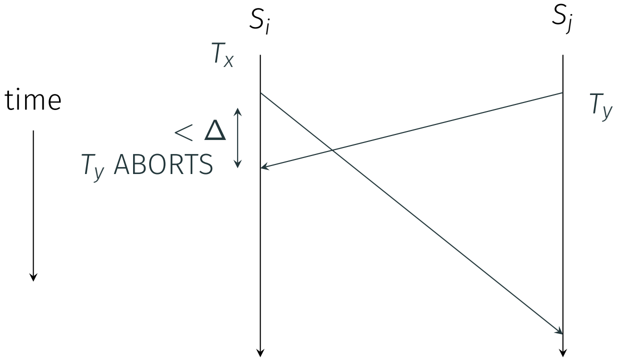
Else, abort



Delta Protocol: Example

Rule: an update is permitted if the preceding update was done at least Δ time before.

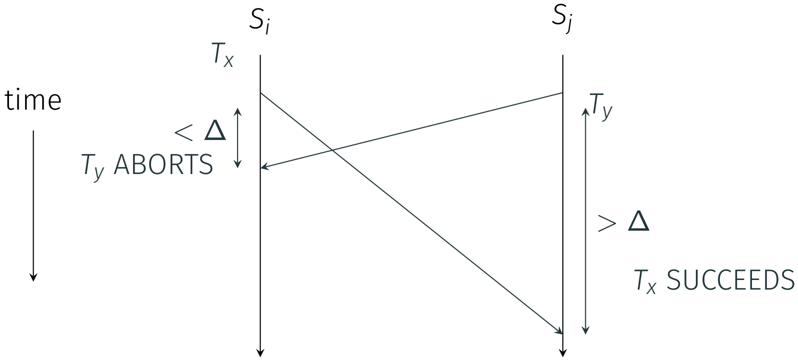
Else, abort



Delta Protocol: Example

Rule: an update is permitted if the preceding update was done at least Δ time before.

Else, abort



Delta Protocol

- Reciprocal consistency is preserved if the time taken to complete am update at one end and start at other end remains less than Δ
- If Δ is exceeded then reciprocal inconsistency can occur
- Setting a large Δ tends to preserve consistency but leads to more aborted transactions

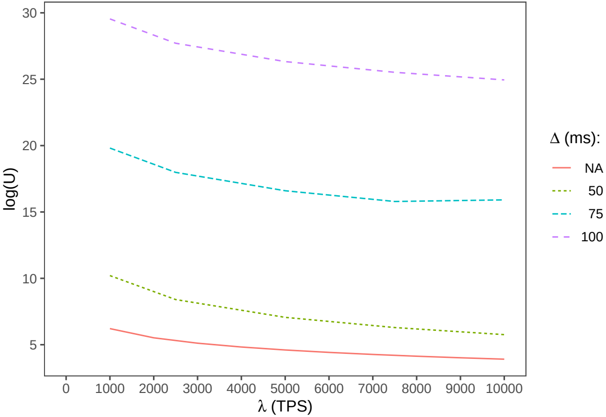
Performance Evaluation

Two metrics the following two metrics for various values of Δ:

- Time taken for 10% of a large database to be corrupt
- Number of transactions aborted per second

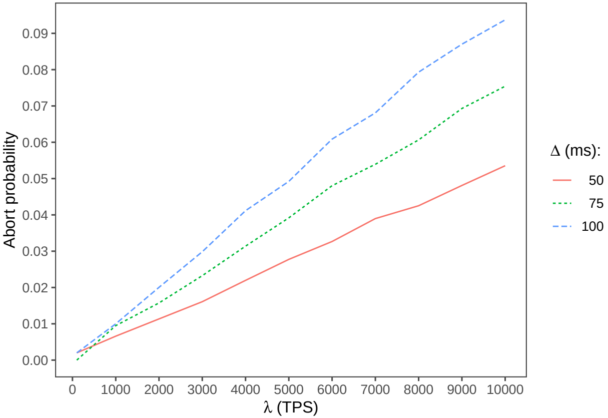
Time until 10% database corruption $log(U)$ vs Transaction Arrival Rate (λ)

For $\Delta = 50ms$, time taken for 10% database corruption is between to 1-75 years



Fraction of Aborts vs Transaction Arrival Rate (λ)

For $\Delta = 50ms$, the fraction of aborts is between 1 – 5%



Summary

- Lack of concurrency control can lead to reciprocally inconsistent edges
- Semantic corruption spreads quickly in a graph database
- Delta protocol prevents reciprocal inconsistency given the bound Δ in not violated
- Delta protocol significantly reduces the time until operational corruption at the cost of some aborts