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*

Property Graph

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

Property Graph

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

Property Graph

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

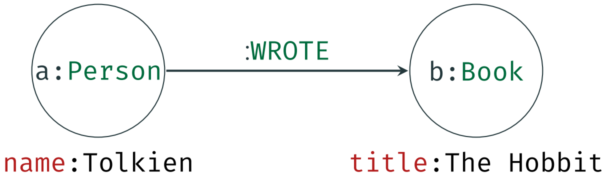


Figure 1: Vertices connected by an edge

Storage Layer Representation

- In the storage layer,
  - edge directionality does **not** exist
  - connected vertices store information about each other

Storage Layer Representation

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

Storage Layer Representation

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

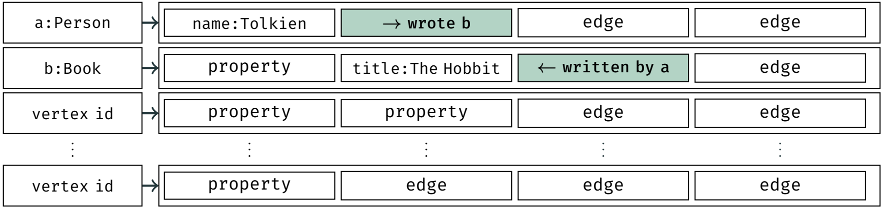


Figure 2: Edge storage layer representation

Reciprocal Consistency

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

Partition graph across machines in a cluster

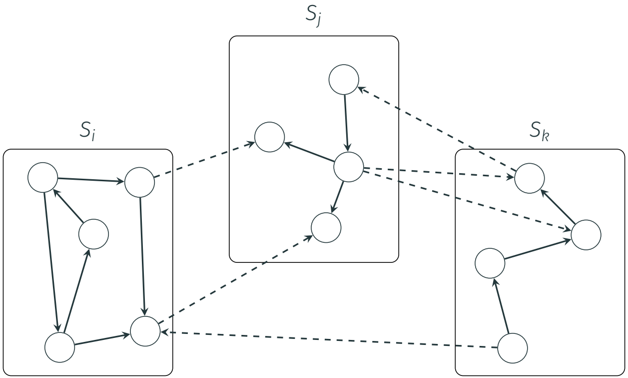


Figure 3: Partitioned graph

Distributed Graph Databases

A degree of concurrency control is needed for ensuring reciprocal consistency of distributed edges

Reciprocal Inconsistency

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

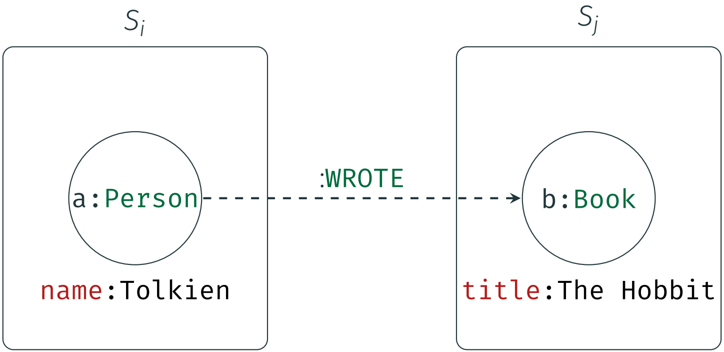
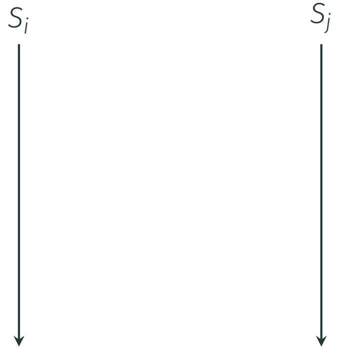


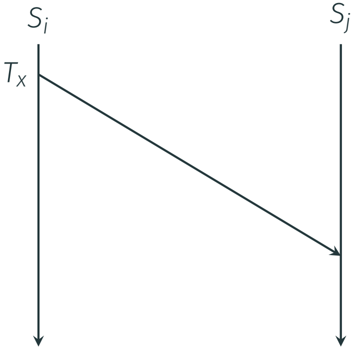
Figure 4: Distributed edge

Reciprocal Inconsistency



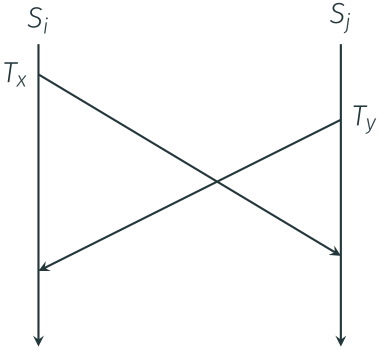
Reciprocal Inconsistency

- T_x deletes the edge



Reciprocal Inconsistency

- T_x deletes the edge
- T_y appends a property `{year:1937}`



The distributed edge is now *reciprocally inconsistent*

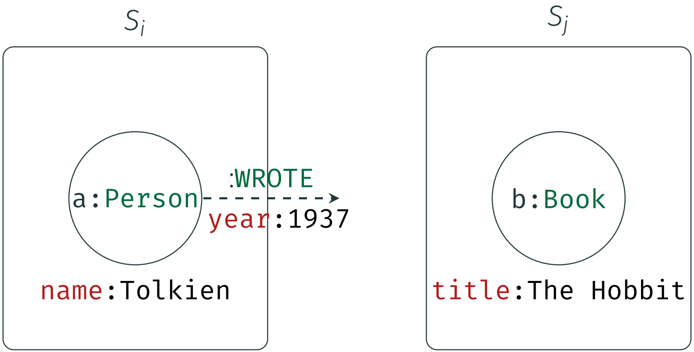


Figure 5: Reciprocally inconsistent distributed edge

Reciprocal Inconsistency

Storage layer consists of two inconsistent uni-directional edge pointers

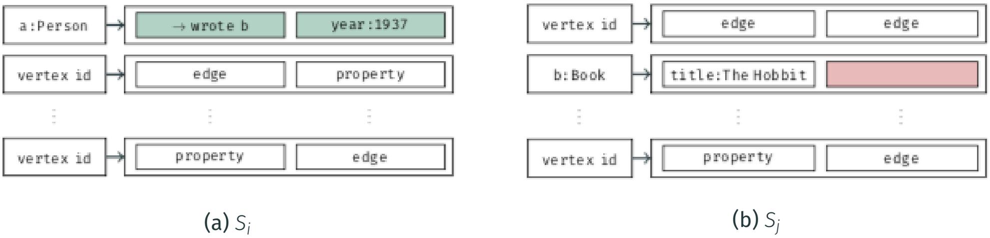


Figure 6: Edge storage layer representation

Reciprocal Inconsistency

- Reciprocally inconsistent edges are the source for *semantic corruption*

Reciprocal Inconsistency

- Reciprocally inconsistent edges are the source for *semantic corruption*
- Semantic corruption spreads through the database

Reciprocal Inconsistency

- Reciprocally inconsistent edges are the source for *semantic corruption*
- Semantic corruption spreads through the database
- **Motivated the design of a lightweight concurrency control protocol that preserves distributed edge reciprocal consistency**

Protocol Design Considerations

Design considerations:

Protocol Design Considerations

Design considerations:

1. Graph workloads exhibit high contention

Protocol Design Considerations

Design considerations:

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

Protocol Design Considerations

Design considerations:

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

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

Delta Protocol

- **Fact:** a transaction updating a distributed edge must update one edge pointer then **immediately** update the other

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

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 an update at one edge pointer and starting at the other can be estimated to be δ.

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 an update at one edge pointer and starting at the other can be estimated to be δ.
- **Parameter Selection:** Choose $\Delta > \delta$

Delta Protocol: Example

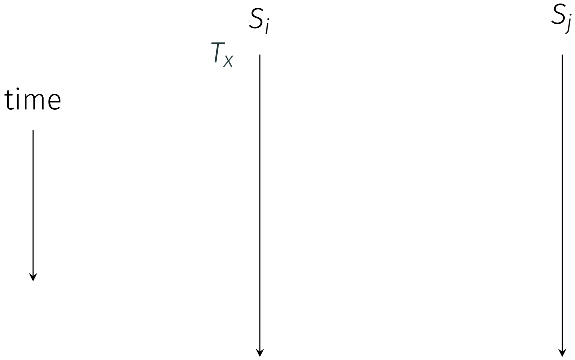
Rule: an update is permitted if the preceding update was done at least Δ time before.

Else, abort

Delta Protocol: Example

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

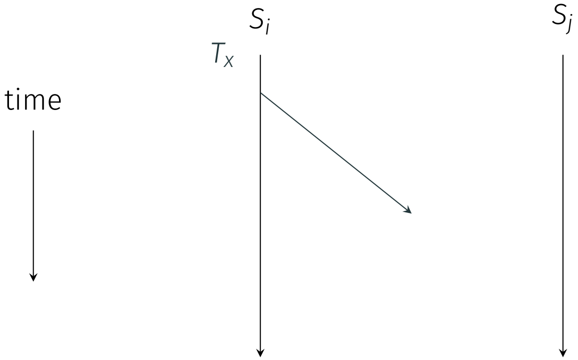
Else, abort



Delta Protocol: Example

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

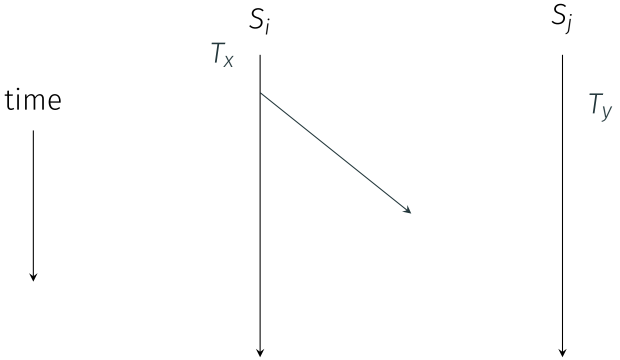
Else, abort



Delta Protocol: Example

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

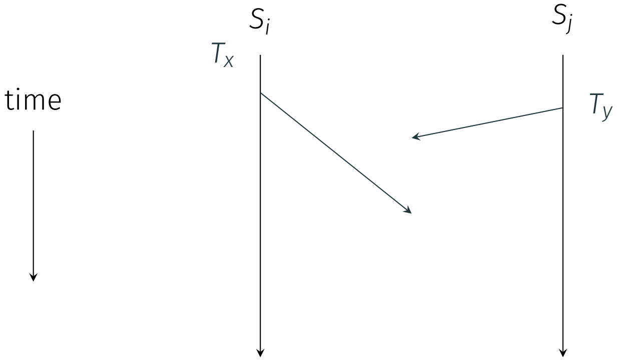
Else, abort



Delta Protocol: Example

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

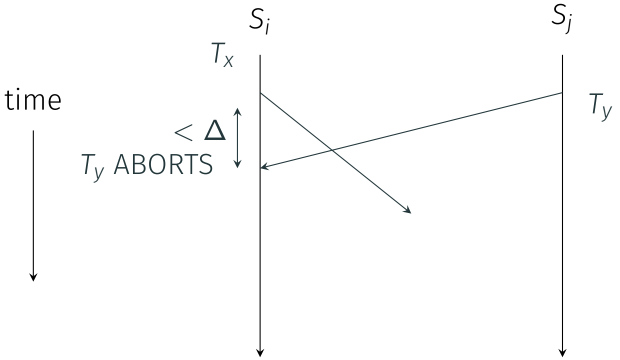
Else, abort



Delta Protocol: Example

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

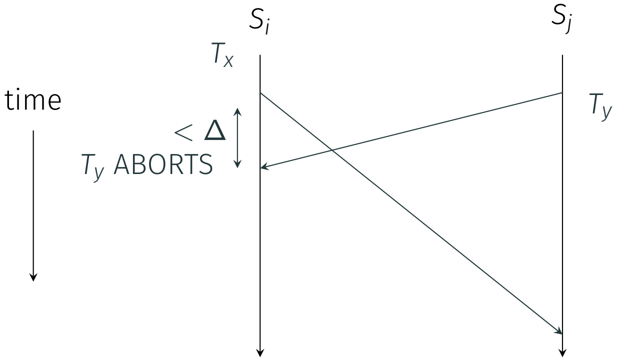
Else, abort



Delta Protocol: Example

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

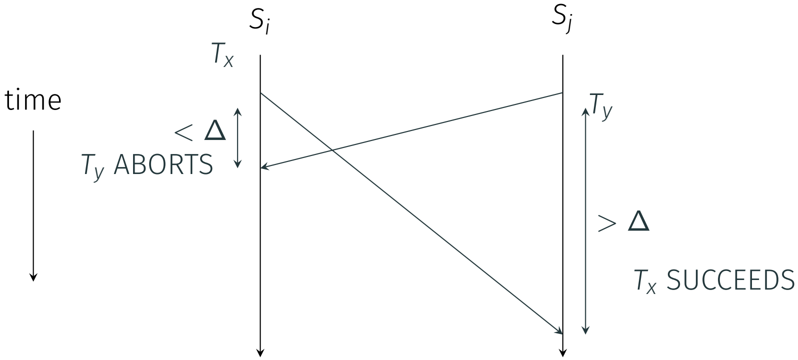
Else, abort



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 an update at one edge pointer and start at other remains less than Δ

Delta Protocol

- Reciprocal consistency is preserved if the time taken to complete an update at one edge pointer and start at other remains less than Δ
- If Δ is exceeded then reciprocal inconsistency and hence semantic corruption can occur

Delta Protocol

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

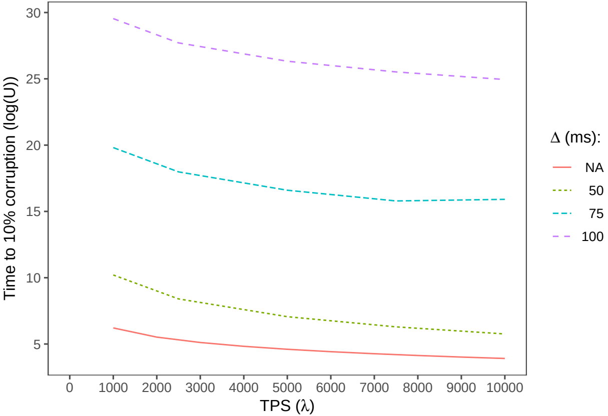
Performance Evaluation

Through simulations the following two metrics for various values of Δ were evaluated:

- Time taken for 10% of a large database to become semantically corrupt
- Fraction of transactions aborted per second

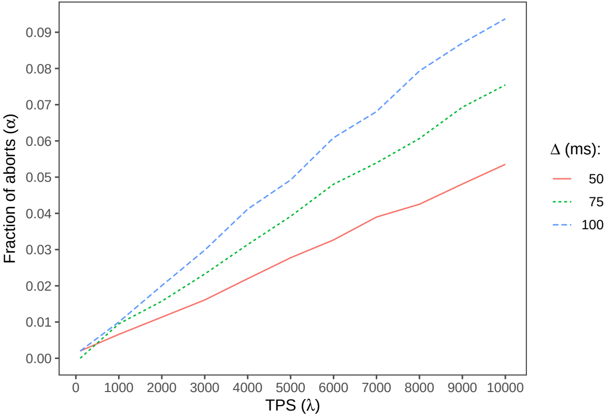
Time until 10% semantic corruption $log(U)$ vs Transaction Arrival Rate (λ)

For $\Delta = 50ms$, time taken for 10% of a large database to become semantically corrupt is between to 1-75 years



Fraction of Aborts (α) vs Transaction Arrival Rate (λ)

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



Summary

- Lack of concurrency control in a distributed graph database can lead to reciprocally inconsistent distributed edges
- Resulting in the spread of semantic corruption
- Delta protocol prevents reciprocal inconsistency given Δ is not exceeded
- Delta protocol significantly reduces the spread of semantic corruption at the cost of a very small fraction of aborts