GRAIL
Checking Transaction Isolation Violations with Graph Queries
Dumbrava, Stefania; Jin, Zhao; Ozkan, Burcu Kulahcioglu; Qiu, Jingxuan

DOI
10.1145/3639478.3643094

Publication date
2024

Document Version
Final published version

Published in
Proceedings - 2024 ACM/IEEE 46th International Conference on Software Engineering

Citation (APA)

Important note
To cite this publication, please use the final published version (if applicable). Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights. We will remove access to the work immediately and investigate your claim.
GRAIL: Checking Transaction Isolation Violations with Graph Queries

Stefania Dumbrava
Zhao Jin
stefania.dumbrava@ensiie.fr
zhao.jin@ensiie.eu
ENSIIE
France

Burcu Kulahcioglu Ozkan
Jingxuan Qiu
b.ozkan@tudelft.nl
j.qiu-2@student.tudelft.nl
Delft University of Technology
The Netherlands

ABSTRACT
Distributed databases are surging in popularity with the growing need for performance and fault tolerance. However, implementing transaction isolation models on distributed databases is more challenging due to their sharding and replication. As a result, they can produce executions that violate their claimed isolation guarantees.

In this work, we propose a novel isolation model-agnostic approach that utilizes graph databases to efficiently detect isolation violations expressed as anti-patterns in transactional dependency graphs. To illustrate our approach, we introduce the GRAIL framework, implemented on top of the popular ArangoDB and Neo4j graph databases. GRAIL combines soundness guarantees and high performance with understandable, detailed counter-examples.

CCS CONCEPTS
• Information systems → Distributed database transactions;
• Software and its engineering → Empirical software validation.

KEYWORDS
distributed databases, transaction isolation, testing, graph queries

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:

1 INTRODUCTION
Database isolation levels describe the degree to which the updates of a running transaction are isolated from other concurrent transactions. At the strongest level, the transactions run in complete isolation, producing an execution where they run one after another. Serializability ensures that the effect of concurrent transactions is the same as running them serially in some order. While serializability provides strong guarantees, implementing it requires strong synchronization in the databases with sharding and replication.

Many databases support isolation levels weaker than serializability and aim to achieve higher performance by reducing the amount of isolation in concurrent transactions. These isolation levels are formally defined by the set of allowed and disallowed behaviors, e.g., the characteristics of anomalies prohibited in serializable databases.

Ensuring claimed isolation guarantees is, however, difficult as witnessed by the many violations discovered in popular distributed databases [10]. Moreover, checking the correctness of the executions for a given isolation level, e.g., for serializability or snapshot isolation, is generally NP-complete [13, 4]. Execution anomalies can be checked using dependency graphs, which model the read/write dependencies between transactions [1, 2], as specific patterns indicate violations of certain isolation guarantees. For example, the execution in Figure 1 is not serializable since it has a cyclic dependency between transactions. Moreover, the graph exhibits an anti-pattern that characterizes a write skew anomaly.

We aim to explore the efficiency of using graph database queries for checking such anti-patterns. Graph databases have been increasingly used to analyze relational datasets with their built-in support for modeling connectivity and providing query languages that can efficiently explore patterns in graph models. Therefore, we investigate: Given transaction dependencies and transactional isolation violation patterns naturally modeled by dependency graphs, how can we utilize graph database queries to detect isolation violations?

2 APPROACH
We introduce GRAIL, a new GRaph-based Isolation Level checking framework that, to the best of our knowledge, is the first to use graph queries to detect database isolation violations.

GRAIL’s architecture (Figure 2) consists of three main modules for History Collection, Graph Construction, and Cycle Detection:
ArangoDB and Neo4j-based checkers have low analysis times with the increasing size of execution histories. These works include the Knossos [11] linearizability checker, the Jepsen [9] to record the execution histories.

**4 RELATED WORK**


Graph Construction processes the collected histories to build the transaction dependency graph for each execution history.

Cycle Detection is the heart of GRAIL as it processes the dependence graphs and identifies their isolation anti-patterns. We introduce the proof-of-concept GRAIL implementation for checking serializability violations using graph queries written in Cypher [6] for Neo4j [12] and AQL (ArangoDB Query Language) for ArangoDB [7]. GRAIL’s Neo4j-based checker runs a Cypher query to detect all the cycles of a dependency graph. Its heuristic is based on the shortest path algorithm of Neo4j’s APOC library, which, for each given starting vertex, finds back paths connecting to it. We also implement the approach in AQL, for ArangoDB, to evaluate the performances of different graph database checkers. Similar to the Neo4j checker, it also uses a shortest path algorithm to detect cycles.

Further work will extend the checkers to detect anomalies for a spectrum of isolation levels, by running more extensive queries to filter the cycles and find specific isolation anti-patterns.

**3 EVALUATION**

Figure 3 presents some initial results for the performance and scalability of GRAIL’s ArangoDB and Neo4j checkers against that of the state-of-the-art Elle [3] tool for checking serializability of executions with an increasing collection time. As the plot shows, the ArangoDB and Neo4j-based checkers have low analysis times with smaller increases in history length. This is because graph databases are inherently efficient at processing graphs, and our checkers search for the shortest paths, returning an anti-pattern as soon as they find it, without further traversals to detect all the cycles.

As another advantage of using a graph database, GRAIL presents the detected cycle (which represents a violation, e.g., to serializability) to the user as an understandable counter-example. Figure 4 shows an example visualization of a serializability violation.

**4 RELATED WORK**

Checking isolation guarantees of database executions has gained attention due to the increasing distribution of database systems. These works include the Knossos [11] linearizability checker, the Gretchen [8] serializability checker that encodes the execution constraints based on the transactional specifications of [5], and Cobra [14] using an efficient SMT solver for checking graph properties. Unlike the state of the art, we design a transaction isolation checker that exploits graph databases for checking violation patterns on the dependency graphs. For this, we construct dependency graphs in graph databases and use graph queries for pattern-matching.

**5 CONCLUSIONS AND PERSPECTIVES**

We presented the initial results for checking serializability using GRAIL. The extended version of the work will explore an extensive set of graph queries for checking a spectrum of isolation levels.

**REFERENCES**


