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

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

ACM Reference Format:
Stefania Dumbrava, Zhao Jin, Burcu Kulahcioglu Ozkan, and Jingxuan Qiu. 2024. GRAIL: Checking Transaction Isolation Violations with Graph Queries. In *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3639478.3643094

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.

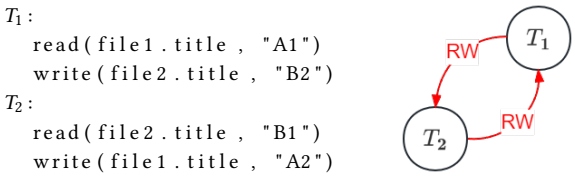

Figure 1: An execution with a cyclic dependency graph.

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:

ICSE-Companion '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0502-1/24/04
https://doi.org/10.1145/3639478.3643094

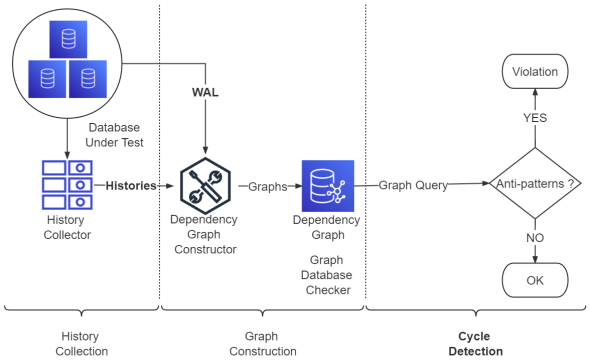

Figure 2: GRAIL's architecture.

History Collection initiates multiple transactions on a sequence of concurrent workers in a distributed database cluster. We use Jepsen [9] to record the execution histories.

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

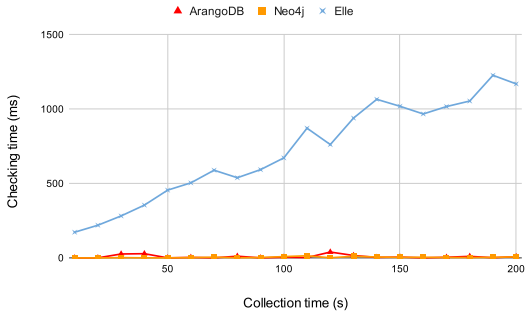

Figure 3: Runtime for checking serializability anomalies in the increasing size of execution histories.

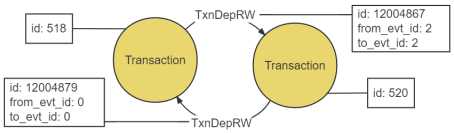

Figure 4: An example violation.

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

[1] Atul Adya. 1999. *Weak consistency: a generalized theory and optimistic implementations for distributed transactions.* Ph.D. Dissertation. Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science.

[2] Atul Adya, Barbara Liskov, and Patrick E. O'Neil. 2000. Generalized isolation level definitions. In *Proc. of the 16th Int. Conf. on Data Engineering, San Diego, California, USA, February 28 - March 3, 2000.* IEEE Computer Society, 67–78.

[3] Peter Alvaro and Kyle Kingsbury. 2020. Elle: inferring isolation anomalies from experimental observations. *Proc. VLDB Endow.*, 14, 3, 268–280.

[4] Ranadeep Biswas and Constantin Enea. 2019. On the complexity of checking transactional consistency. *Proc. ACM Program. Lang.*, 3, OOPSLA, 165:1–165:28.

[5] Andrea Cerone, Giovanni Bernardi, and Alexey Gotsman. 2015. A framework for transactional consistency models with atomic visibility. In *26th Int. Conf. on Concurrency Theory, CONCUR 2015* (LIPIcs). Vol. 42, 58–71.

[6] Nadime Francis et al. 2018. Cypher: an evolving query language for property graphs. In *SIGMOD Conference.* ACM, 1433–1445.

[7] ArangoDB Inc. 2023. Arangodb. https://www.arangodb.com/. (2023).

[8] Kyle Kingsbury. 2022. Gretchen: offline serializability verification, in clojure. (2022). Retrieved July, 2023 from https://github.com/aphyr/gretchen.

[9] Kyle Kingsbury. 2022. Jepsen. (2022). Retrieved July, 2023 from http://jepsen.io/.

[10] Kyle Kingsbury. 2022. Jepsen analyses. (2022). Retrieved July, 2023 from https://jepsen.io/analyse.

[11] Kyle Kingsbury. 2022. Knossos. (2022). Retrieved July, 2023 from https://github.com/jepsen-io/knossos.

[12] Neo4j. 2023. Neo4j. https://neo4j.com/. (2023).

[13] Christos H. Papadimitriou. 1979. The serializability of concurrent database updates. *J. ACM*, 26, 4, 631–653. https://doi.org/10.1145/322154.322158.

[14] Cheng Tan, Changgeng Zhao, Shuai Mu, and Michael Walfish. 2020. Cobra: making transactional key-value stores verifiably serializable. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020.* USENIX Association, 63–80.