Serializability checking for MongoDB clients

Master Thesis Project Description

Johannes Baum

Supervisor: Lucas Brutschy

Department of Computer Science, ETH Zürich

October 28, 2016

1 Introduction

To increase scalability and availability, replicated data stores have become an important part of modern distributed systems. While non-replicated data stores usually provide strong consistency, the majority of replicated stores only provide variants of eventual consistency [1], which guarantees that

1. updates are eventually propagated to other replicas
2. replicas observing the same set of operations reflect the same state

Providing only eventual consistency, replicated data stores demand more caution by the developer using them in an application. Eventual consistency allows behaviors of the data store which are not possible under strong consistency. If strong consistency is required by the application, the developer is now responsible for guaranteeing it. Serializability [2] is a property that if it holds for an execution of a weakly consistent application allows to reason about it as if it would provide strong consistency. Brutschy et al. propose a serializability criterion for clients of data stores which only provide eventual consistency and suggest two polynomial-time algorithms to check whether their criterion holds on a given dynamic program execution.

This project is intended to provide a tool that instruments any NodeJS [3] application using MongoDB [4]. Running the application then records any database operations triggered by the application. These records are then being used to apply the algorithm by [5], deciding on their serializability criterion and giving details on violations. The software stack consisting of NodeJS and MongoDB has been chosen because it is commonly used for implementing distributed systems.

MongoDB is a document-based database that guarantees eventual consistency in a certain configuration. This is sufficient to apply the serialization criterion mentioned above if the algebraic properties commutativity and absorption [5] of the database operations will be defined.

1.1 Serializability criterion

The mentioned serializability criterion assumes eventual consistency and is relying on the following relations:

- **Program order** (po)

- **Arbitration order (ar):** Order in which update conflicts are eventually resolved by the system.
- **Visibility (vi):** An update is visible to a query iff it was applied to the queried replica before the query was executed.
- **Dependency ($\oplus$):** A query depends on a visible update iff the result of the query would change had the update become invisible.
- **Anti-dependency ($\ominus$):** A query anti-depends on an update that is not visible if the query result would change had the update become visible.

If the graph denoting the union of the four relations po, ar, $\oplus$, $\ominus$ of an execution forms a cycle then serializability is violated.

2 Core Goals

The core goals of this project are to build a tool that instruments NodeJS applications using MongoDB and analyzes dynamic executions using the serializability criterion suggested by [5]. This can be split into the following tasks:

- Recording database operations of dynamic executions of NodeJS applications using MongoDB. The NodeJS applications have to be instrumented such that they record the database traffic with MongoDB in a way that allows to extract the needed relations from the recordings.
- Define algebraic properties commutativity and absorption for MongoDB database operations. These definitions are necessary to obtain the dependencies and anti-dependencies with the algorithm proposed by [5]
- Determine the arbitration order of MongoDB. Therefore the resolution strategy of conflicting updates of MongoDB has to be examined.
- Analyze recorded traces using the mentioned algorithm. Based on the recorded traces the relations po and vi have to be determined. These relations and the arbitration order serve as input for the algorithm which then points out possible serializability violations.
- Evaluate the effectiveness of the analysis on a set of open source projects. A selection of open source projects using NodeJS and MongoDB have to be instrumented and run. The generated traces have to be analyzed as described to find serializability violations in these projects.
- Manual inspection of the reported alarms, categorizing them into true and false alarms. The serializability violations detected by the algorithm have to be checked manually. This allows to evaluate the quality of the detected violations.
- The tool should reach a state that allows a developer to use it for a custom project without much effort.

3 Extensions

In the following, possible extensions to the core goals are described. Since it is not possible to explore all of them in detail within the frame of this thesis we will decide which ones to tackle during the project.

1. Further increasing the precision of the serializability criterion. This can either be achieved by extending the formal framework given by [5] or by figuring out a way to filter out certain cycles in the dependency serialization graph which cause false positives.

2. Check the validity of the recorded queries. For each query we execute all updates visible to that query followed by the query itself on an empty database. If the result of that execution differs from the result of the recorded dynamic execution this means that there is an error in the analysis. Replaying also ensures that our necessary assumptions on the configuration of the database (to provide eventual consistency) are satisfied. Depending on the complexity of the queries we may simulate the database by an in-memory data-structure instead of setting up an empty database before replaying each query. This whole process implies that before recording the dynamic execution of the applications, the database must be empty.

3. Increase the chance of detecting serializability violations by implementing stress tests for MongoDB. Because of the possibility of a serializability violation occurring in a dynamic run being very small, we aim on implementing stress tests for MongoDB to increase that possibility. These stress tests can for example simulate network partitions to increase the chance of a serializability violation being recorded.

References

[1] Sebastian Burckhardt, *Principles of Eventual Consistency*, vol. 1, now publishers, October 2014.

[2] Christos H Papadimitriou, "The serializability of concurrent database updates," *Journal of the ACM (JACM)*, vol. 26, no. 4, pp. 631–653, 1979.

[3] "Node.js," `https://nodejs.org/en/`, Accessed: 2016-09-27.

[4] "Mongodb," `https://www.mongodb.com/`, Accessed: 2016-09-27.

[5] Lucas Brutschy, Dimitar Dimitrov, Peter Müller, and Martin Vechev, "Serializability for eventual consistency: Criterion, analysis and applications," 2016.