Checking Linearizability and Sequential Consistency of a Distributed Database

CS4720 - Research in Program Analysis

2025/Q4

Project description

In this project, you will test a distributed database of your choice and check whether its executions satisfy linearizability or sequential consistency properties.

Background Information

Testing distributed systems

The executions of distributed systems involve complex interactions of their concurrent components, asynchrony in the exchange messages, possible network failures (e.g., delaying or dropping messages, isolating some nodes, partitioning the cluster of nodes), and component failures (e.g., node crashes). Software testing provides a practical method to exercise the system's behavior and check whether it satisfies its guarantees in the presence of asynchrony and failures.

In this project, you will generate test cases for distributed system executions using Jepsen [1], where the test cases inject network and process faults into the system executions. Then, you will check the linearizability or sequential consistency properties of the execution traces using the Knossos [2] tool.

Distributed database systems

Distributed database systems distribute a database across multiple nodes in a cluster to increase scalability and fault tolerance. They replicate the data on several different nodes, which improves performance (the nodes/replicas can concurrently serve client requests) and provides fault-tolerance (if a node/replica is not available, the other nodes/replicas can continue serving the clients). The systems are designed to operate on the replicated data objects concurrently and provide some guarantees on the consistency of the replicated data.

Linearizability and Sequential Consistency

Linearizability [3] and Sequential Consistency [4] are standard correctness properties for concurrent data structures and distributed databases. Database systems with linearizability or sequential consistency guarantee that their clients will have the illusion of working on a single copy of the data objects. Linearizability requires that there exists a total order of client operations where (i) the total order respects the real-time order of the non-concurrent operations and (ii) the behavior of the operations is consistent with the sequential specification of the data structure. Sequential Consistency relaxes linearizability in its requirement of real-time order of the non-concurrent operations. It requires that operations appear to take place in some total order, and that that order is consistent with the order of operations on each individual process.

System under test for your project:

You can test a distributed database system of your choice as the system under test (SUT), such as Cassandra, CockroachDB, etcd, MongoDB, NebulaGraph, Redis, Riak, YugaByte DB.

You can consider testing a database with weak consistency and isolation guarantees so that you can produce executions that violate strong consistency and evaluate the performance of different test configurations in exposing such executions.

Roadmap for the project:

The project involves the following steps:

- Familiarize yourself with the Jepsen fault injection tool [5] and take a look at the example projects [6] in detail.
- Set up a cluster of distributed system nodes running your selected SUT
- Generate test harnesses, i.e., concurrent client requests to submit to the system
- Run the test harnesses in the presence of asynchrony, network failures, and process failures (e.g., the executions with randomly delayed or dropped network messages, isolated processes, and crashing processes. You can generate your own test harnesses and inject faults into executions, or you can use an existing test generation tool, Jepsen [1].
  - Run the Jepsen tests with varying Nemesis fault injection configurations
- Collect the execution histories
- Check linearizability and sequential consistency of the collected histories. You can use the Knossos [2] or Elle [7] to check the properties.
- Evaluate the results of your tests:
  - Do any of the test executions violate linearizability?
  - Do any of the test executions violate sequential-consistency?
  - How do different Nemesis configurations affect test results?
  - Provide example execution traces and explain their linearizability/sequential consistency or violation scenarios.
- Provide a comprehensive documentation of findings, limitations, and recommendations for improvements
- Report if the methods find any previously unknown bugs in the system under test

Sample repository from a previous project:

Testing consistency of the rqlite distributed database (by Nienke Eijsvogel, Ruben van Baarle, Daan de Graaf GitHub Hacker News.

References

[1] Kyle Kingsbury. Jepsen. `http://jepsen.io/`.

[2] Kyle Kingsbury. Knossos. `https://github.com/jepsen-io/knossos/`.

[3] Maurice P Herlihy and Jeannette M Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(3):463–492, 1990.

[4] Leslie Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers c-28*, 9:690–691, 1979.

[5] Kyle Kingsbury. Jepsen talks. `https://jepsen.io/talks`.

[6] Kyle Kingsbury. Jepsen analyses. `https://jepsen.io/analyses`.

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