Weekly Research Progress Report

Student: Sungho Hong **Date:**7/13/2020 **# of hrs worked this week:** 30

Problems discussed in last week's report

1. **Exploring Jepsen I/O**
    a. Example of testing etcd with Jepsen I/O
    b. Possibility of integrating Jepsen to RamCloud
    c. Checking anomalies with Jepsen

2. **Debugging anomalies from ClusterPerf**
    a. Debugged results

Elle: Inferring Isolation Anomalies from Experimental Observations Jepsen 2020

Background

- Existing solutions require hand-proven invariants.
- General checkers verify linearizability
- Serializability checking is NP-complete
    - cannot use real-time constraints to reduce the search space

Challenge

- Database may not have any concept of a version order
- Database may not expose that ordering information to clients
- Cannot deduce the version in real-time order
    - Optimistic and multi-version may commit earlier versions later in time.
- Cannot Identify write-write dependency
    - Database could have executed T1 and T2 in either order with equivalent

Solution

- **Elle == Isolation checker**
    - Introduce recoverability and traceability to infer dependencies between observed transactions
    - Elle infers a transaction dependency graph over experimentally recorded histories, and searches for cycles in that graph.
        - G0 (dirty write),
        - G1a (aborted read), G1b (intermediate read)

        - G1c (cyclic information flow)
        - G-single (read skew)
        - G2-item (anti-dependency cycle)

- **Counter & Sets**

$$T_0\colon\ read(x, \{0\})$$
$$T_1\colon\ add(x, 1)$$
$$T_2\colon\ add(x, 2)$$
$$T_3\colon\ read(x, \{0, 1, 2\}$$

    - Read-read dependency: T0 → T3
    - Write-read dependency: T1 → T3, T2 → T3
    - Anti-read dependency: T0 → T1 T0 → T2

- **List**
    - add order to our values by letting each version be a list
    - a write appends a unique value
    - Write-write dependency:
        - If there is a read of {0, 1} or {0, 2}

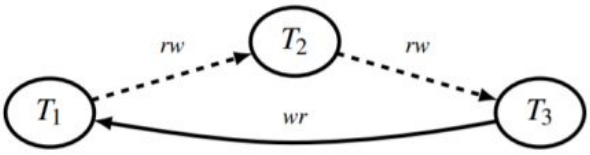
- **Traceability**
    - Restricting our histories to the objects with list
    - Get the largest version of the object
    - Reason about the version order using the results of individual read operations

- **Recoverability**
    - Identify aborted and intermediate versions
    - Restricting observations to a single write per object makes recovery trivial.
        - Counters and sets are difficult to recover in general: a set like {1, 2} could have resulted either from a write of 1 or 2.

Test case of PostgreSQL Jepsen 2020

Background

- PostgreSQL added support for serializable snapshot isolation (SSI)
- SSI extends SI by checking for dangerous structures
    - a pair of adjacent read-write dependencies between three transactions.



- Preventing these dangerous structures, in addition to snapshot isolation's normal rules, yields only serializable executions

Challenge

- Transactions executed with serializable isolation on a single PostgreSQL instance were not serializable

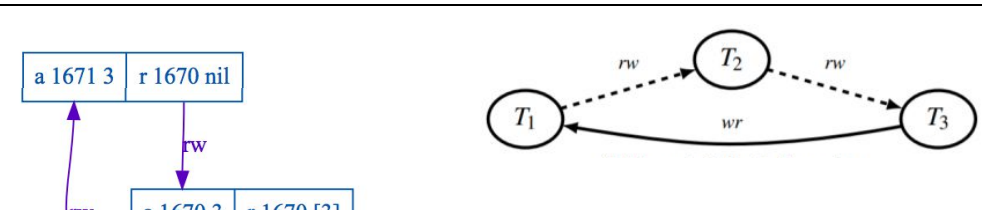
Design

- Kill PostgreSQL processes in random order
- Generate randomized transactions of append and read operations
- Use Elle isolation checker to infer a transaction dependency graph over recorded histories

Results

- PostgreSQL allows write-skew, and dangerous structures

a pair of transactions which read one key and write another, each failing to observe the other's effects



Three transactions involve a read-only transaction which precedes, via two adjacent rw anti-dependencies, a transaction which wrote state which the read-only transaction observed.

Conclusion

- This code has gone essentially untouched since the introduction of serializable snapshot isolation in 2011

Integration with RamCloud

1. Using Jepsen + Elle
    a. Implement the client for Ramcloud
    b. Additionally benefit from intensive test-case scenarios from Jepsen

2. Using Elle (bypass Jepsen)
    a. Record the history and convert them into Jepsen history
    b. Can use the test cases provided by RamCloud

Reference Materials

Comparison between Linearizability and Serializability

Linearizability	**Serializability**
- Based on time - Deterministic - Provides a real-time (i.e., wall-clock) guarantee on the behavior of a set of single operations	- Program order - Freedom to interleave operations as long as the ordering from each client is preserved - Guarantees that the execution of a set of transactions is equivalent to some serial execution (total ordering) of the transactions.