RethinkDB

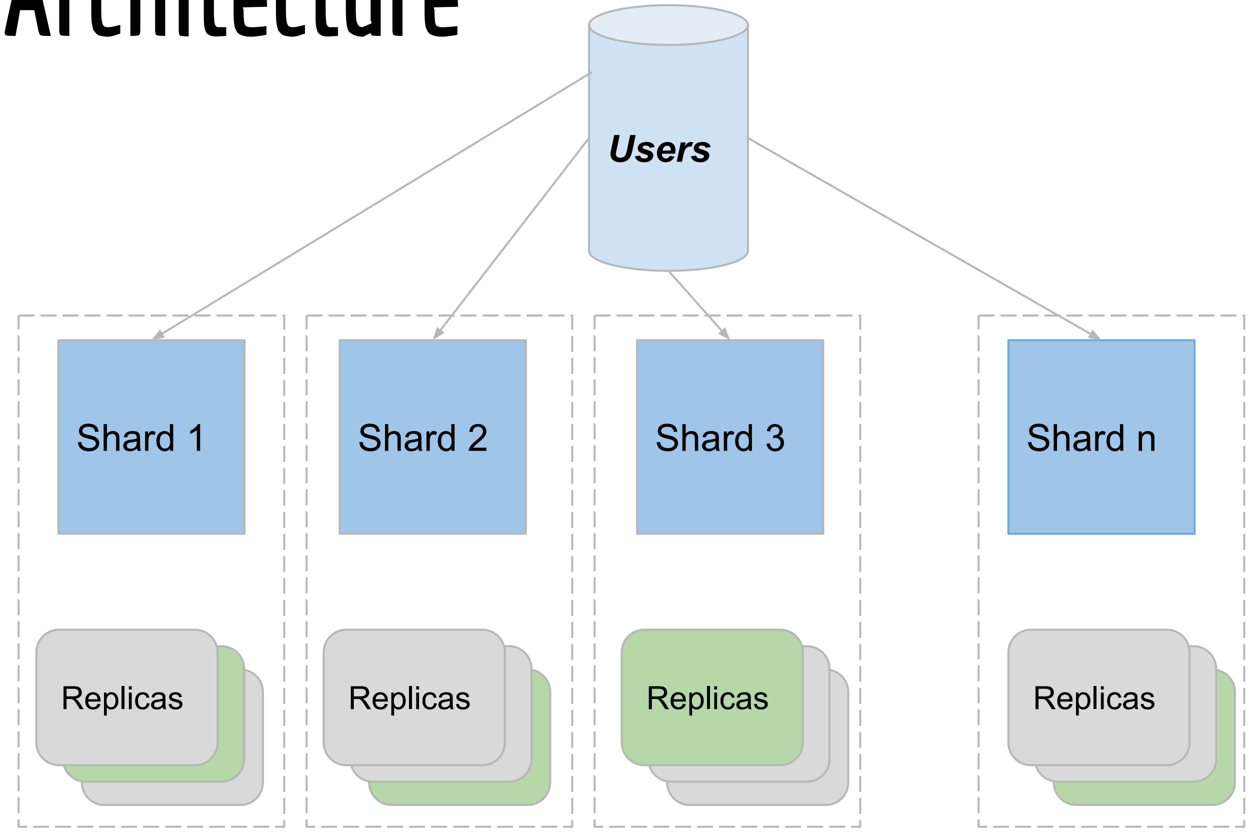
RethinkDB - Jepsen Test

Evaluating consistency semantics of RethinkDB - an upcoming open-source database for the real-time web

What is RethinkDB?

- Started by two CS students of the Stony Brook University in 2009
- Design Goal: Make building realtime applications easier
- Features
    - *Document Oriented (NoSQL)*
    - *ReQL as a query language*
    - *Pull-architecture*
    - *Indices + Joins!*
    - *Distributed. Easy-to-scale. Uses MapReduce under the hood*
- Built from scratch in C++ and completely open-source on Github

Architecture

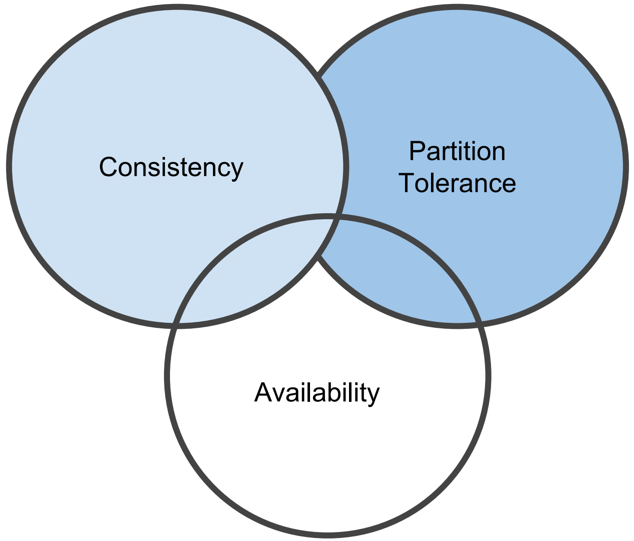


Every shard is assigned to a single authoritative primary replica

Reads / Writes get routed to the primary replica

Consistency Guarantees

- RethinkDB chooses to maintain data consistency over availability in network partitions
- In essence, it is a **CP (consistent - partition tolerant)** system.
- More like MongoDB (CP) and unlike Riak / Dynamo (AP)



Dealing with Partitions

When a primary replica fails

- If a quorum can be attained
  - One of the alive replicas is selected arbitrarily as a primary
  - Brief period of unavailability - but no data loss
  - No split brain - if old primary comes back up, it'll return to being a primary
- If a quorum cannot be attained
  - The system becomes unavailable
- Client experience depends on the side of the netsplit they are on
  - If on the side of the quorum - no change
  - Else writes and up-to-date read queries will become unavailable

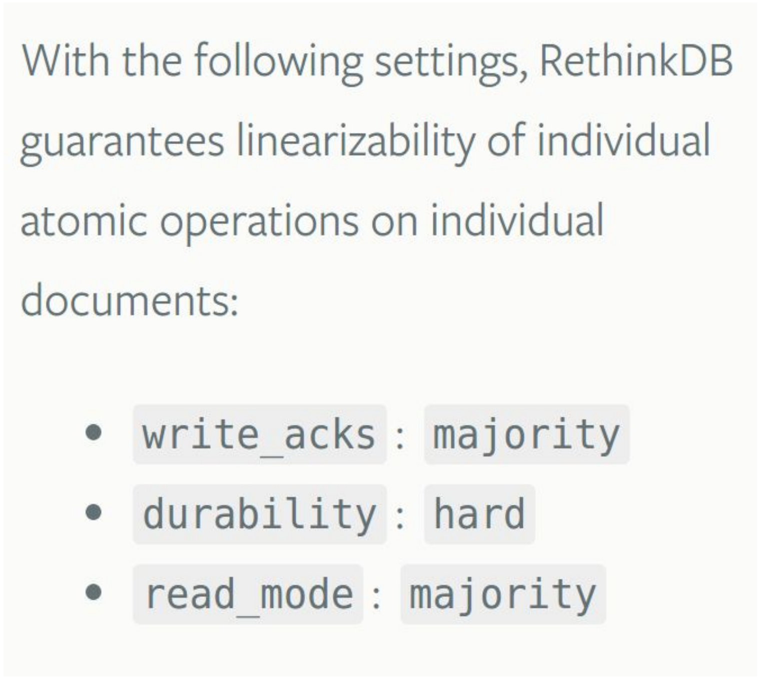
Configuration Parameters

- Write acknowledgements
  - *How does RethinkDB confirm that a write was successful?*
  - **Majority** - Majority of replicas confirm successful writes
  - **Single** - Single replica confirms a write.

- Durability
  - *How does each node in the cluster claim successful writes?*
  - **Hard** - Data committed to disk
  - **Soft** - Data stored in memory

- Read mode
  - *Where to read the data from?*
  - **Single** - Return values from memory in primary replica
  - **Majority** - Return values that are safely committed on disk on a majority of replicas
  - **Outdated** - Return values from memory of an arbitrarily selected replica

RethinkDB - Consistency Semantics



With the following settings, RethinkDB guarantees linearizability of individual atomic operations on individual documents:

- `write_acks`: `majority`
- `durability`: `hard`
- `read_mode`: `majority`

Source: RethinkDB Documentation

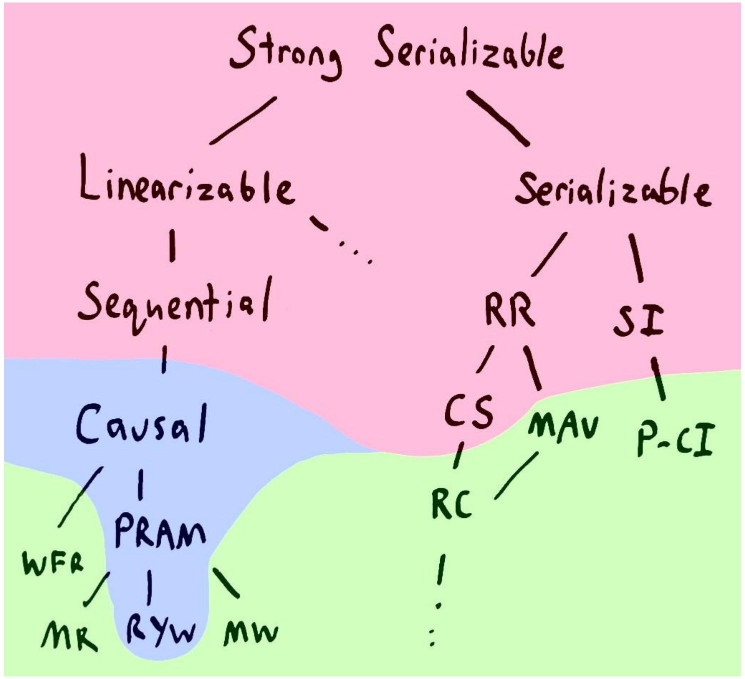


Image Courtesy: Aphyr

Test Setup

- Docker in Docker
  - 6 Docker containers
  - Running Debian Jessie
- Amazon Web Services
  - 1 EC2 Instance - 160 GB Memory and 40 vCPUs
- RethinkDB
  - Version Number: 2.2.0
- Jepsen
  - Clojure library for simulating failures, running tests
  - Version Number: 0.0.7-SNAPSHOT
- Knossos
  - Linearizability checker written as a Clojure library
  - Version Number: 0.2.4-SNAPSHOT

Writing Jepsen Tests for Rethinkdb

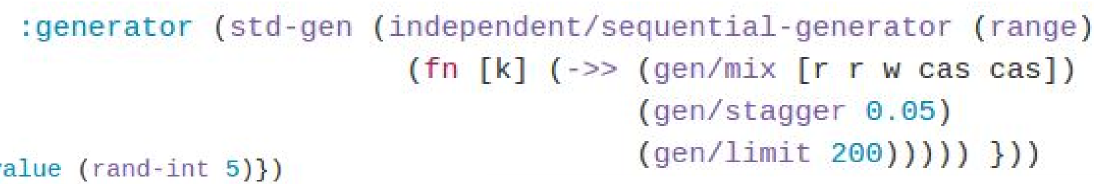
*"Jepsen is a clojure library that provides functionality to **setup** a distributed system, **run** a bunch of **operations** against that system and **verify** that the history of those operations make sense."*

Jepsen tests are comprised of three fundamental parts:

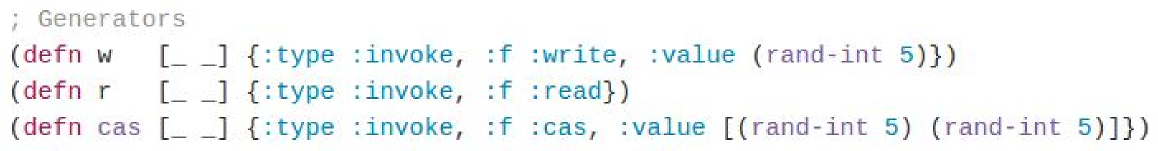
1. **Generator** A stateful object that generates operations for processes to run
2. **Client** Called by processes to run operations on the distributed system
3. **Checker** Validates whether a history of operations is correct w.r.t some model

Jepsen Generators - RethinkDB

- The job of the generator is to generate a bunch of randomly ordered operations for the processes to execute.
- Operations are mixed, randomly permuted and staggered over a time limit



```
:generator (std-gen (independent/sequential-generator (range)
                     (fn [k] (->> (gen/mix [r r w cas cas])
                                  (gen/stagger 0.05)
                                  (gen/limit 200))))) }))
```

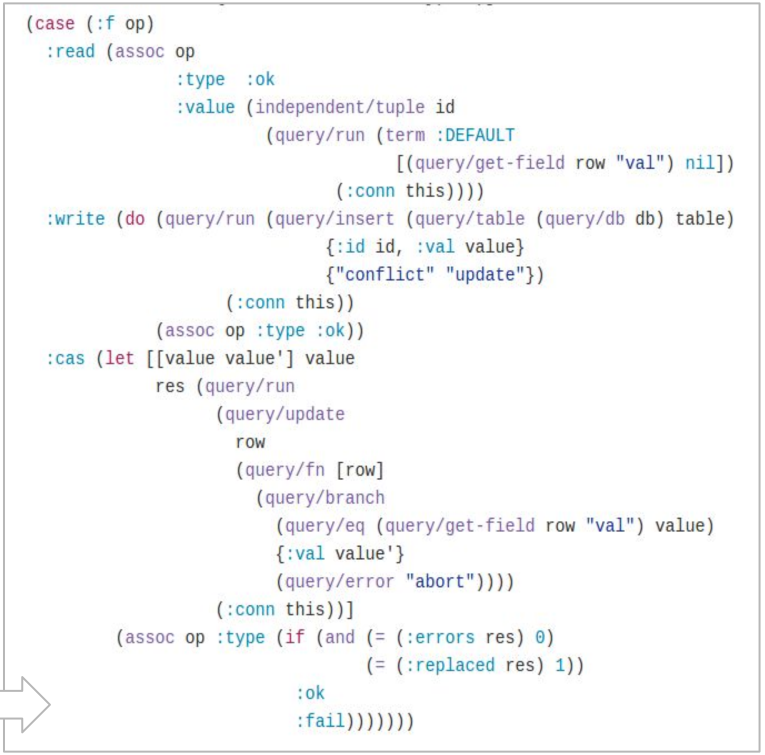


```
; Generators
(defn w   [_ _] {:type :invoke, :f :write, :value (rand-int 5)})
(defn r   [_ _] {:type :invoke, :f :read})
(defn cas [_ _] {:type :invoke, :f :cas, :value [(rand-int 5) (rand-int 5)]})
```

- gen/stagger - introduces a time delay between operations.
- gen/mix - returns a mix of operations randomly chosen from the list.
- gen/limit - the time limit for which the operations are selected.
- r/w/cas - read, write and cas operations performed on the database.

Jepsen Client - RethinkDB

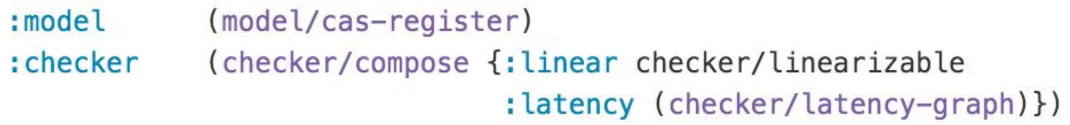
- Interfaces with rethinkdb server
- setup! - bootstraps the table and database with the appropriate `write-mode` setting
- teardown! - cleans up after the test ends.
- read-mode configuration specified at each individual operation.
- Returns corresponding **`:ok, :fail`** or **`:info`**



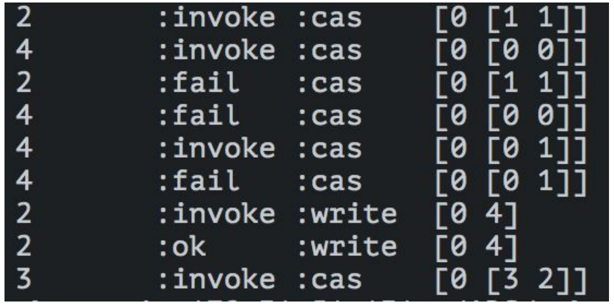
```
(case (:f op)
  :read (assoc op
               :type  :ok
               :value (independent/tuple id
                        (query/run (term :DEFAULT
                                     [(query/get-field row "val") nil])
                                   (:conn this))))
  :write (do (query/run (query/insert (query/table (query/db db) table)
                                      {:id id, :val value}
                                      {"conflict" "update"})
                        (:conn this))
             (assoc op :type :ok))
  :cas (let [[value value'] value
             res (query/run
                   (query/update
                     row
                     (query/fn [row]
                       (query/branch
                         (query/eq (query/get-field row "val") value)
                         {:val value'}
                         (query/error "abort"))))
                   (:conn this))]
         (assoc op :type (if (and (= (:errors res) 0)
                                  (= (:replaced res) 1))
                           :ok
                           :fail))))))
```

Jepsen Checker - Knossos

Knossos is linearizability checker that given a **history of operations** by a set of clients, and some single-threaded model, attempts to show that the history is **not linearizable** with **respect to that model**.



```
:model       (model/cas-register)
:checker     (checker/compose {:linear checker/linearizable
                               :latency (checker/latency-graph)})
```



```
2         :invoke :cas    [0 [1 1]]
4         :invoke :cas    [0 [0 0]]
2         :fail   :cas    [0 [1 1]]
4         :fail   :cas    [0 [0 0]]
4         :invoke :cas    [0 [0 1]]
4         :fail   :cas    [0 [0 1]]
2         :invoke :write  [0 4]
2         :ok     :write  [0 4]
3         :invoke :cas    [0 [3 2]]
```

CAS-Register

A simple compare-and-set register that acts as an abstract model to mimic database behavior. Supports three operations

- *Read*: Read value from register
- *Write(x)*: Write value x to register
- *CAS(x, y)*: If x set y, else exception

Linearizability: That every read sees the most recently written value.

Validating Linearizability with Knossos

Problem

Taking a history with *pairs* of (invoke, ok) operations, and finding an equivalent history of *single* operations which is consistent with the model. This equivalent single-threaded history is called a ***linearization***; a system is ***linearizable*** if at least one such history exists.

Solution (simplistic)

Try all possible permutations of the concurrent history until we either find a linearization and stop, or fail to find one and report that the concurrent history is not linearizable.

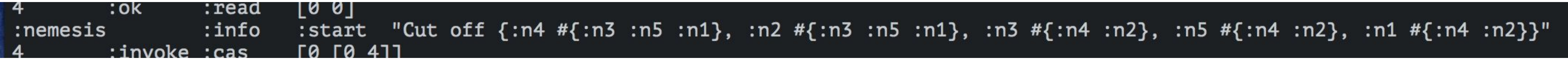
Knossos *does this plus a bunch of other optimizations*

Causing Partitions with Nemesis

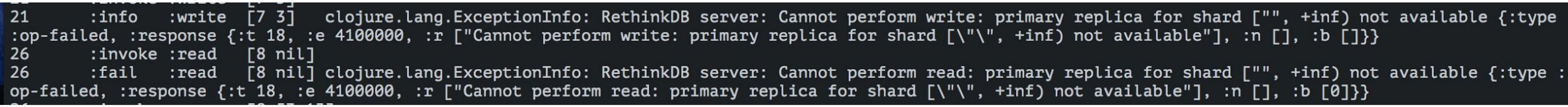
- Part of the Jepsen library which is responsible for causing partitions (havoc) in the system.
- Works by removing `IPTable` entries between nodes and modifying `/etc/hosts` file

```
:nemesis    (nemesis/partition-random-halves)
```

- Splits the cluster of 5 nodes randomly into two halves.



```
4        :ok      :read   [0 0]
:nemesis          :info   :start  "Cut off {:n4 #{:n3 :n5 :n1}, :n2 #{:n3 :n5 :n1}, :n3 #{:n4 :n2}, :n5 #{:n4 :n2}, :n1 #{:n4 :n2}}"
4        :invoke :cas    [0 [0 4]]
```

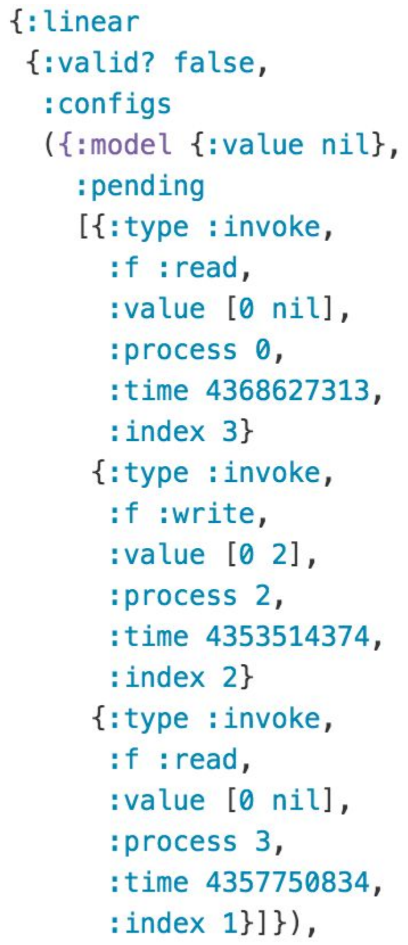


```
21      :info   :write  [7 3]   clojure.lang.ExceptionInfo: RethinkDB server: Cannot perform write: primary replica for shard ["", +inf) not available {:type
:op-failed, :response {:t 18, :e 4100000, :r ["Cannot perform write: primary replica for shard [\"\", +inf) not available"], :n [], :b []}}
26      :invoke :read   [8 nil]
26      :fail   :read   [8 nil] clojure.lang.ExceptionInfo: RethinkDB server: Cannot perform read: primary replica for shard ["", +inf) not available {:type :
op-failed, :response {:t 18, :e 4100000, :r ["Cannot perform read: primary replica for shard [\"\", +inf) not available"], :n [], :b [0]}}
```

Test Results

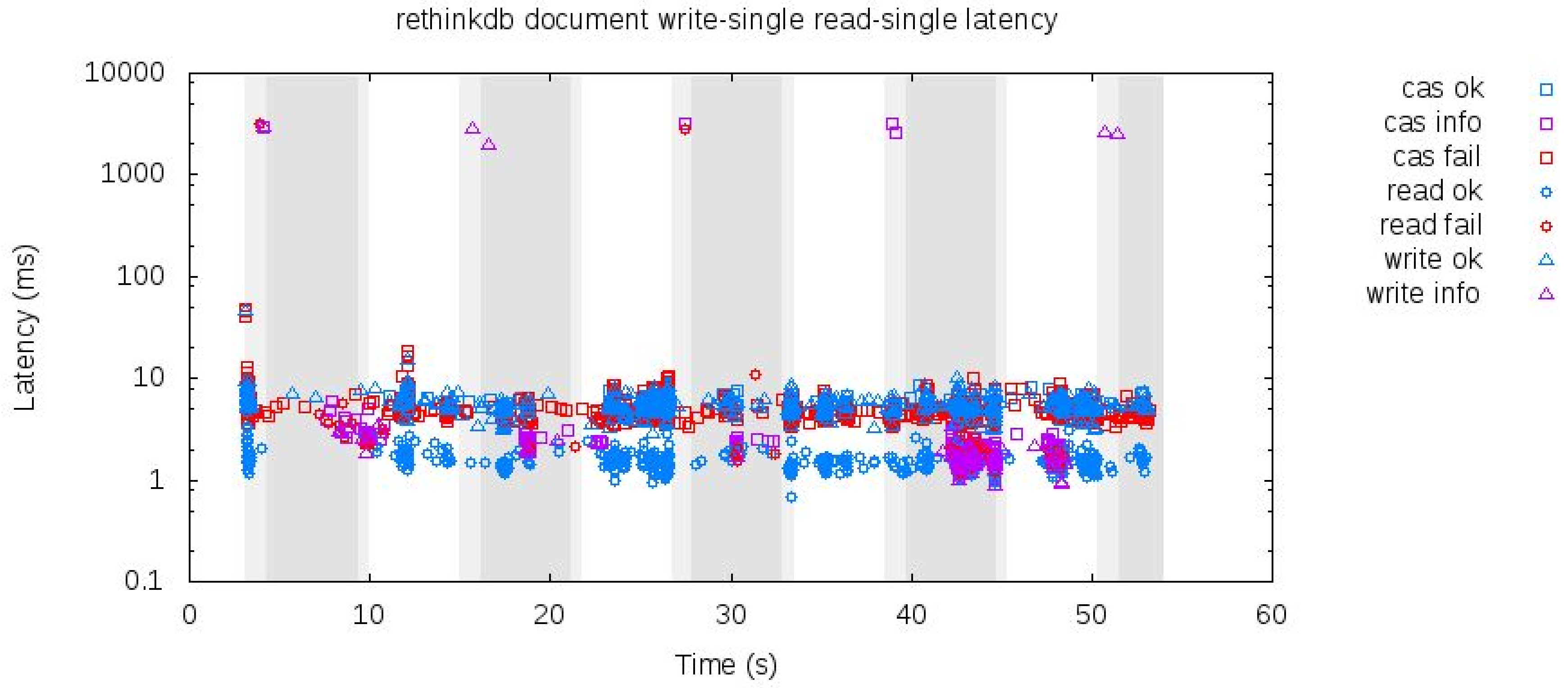
read_mode	write_mode	linearizable?
Single	Single	False
Majority	Majority	False

For both the strongest and the weakest configuration provided by RethinkDB, Jepsen reports that the system is not **linearizable**.

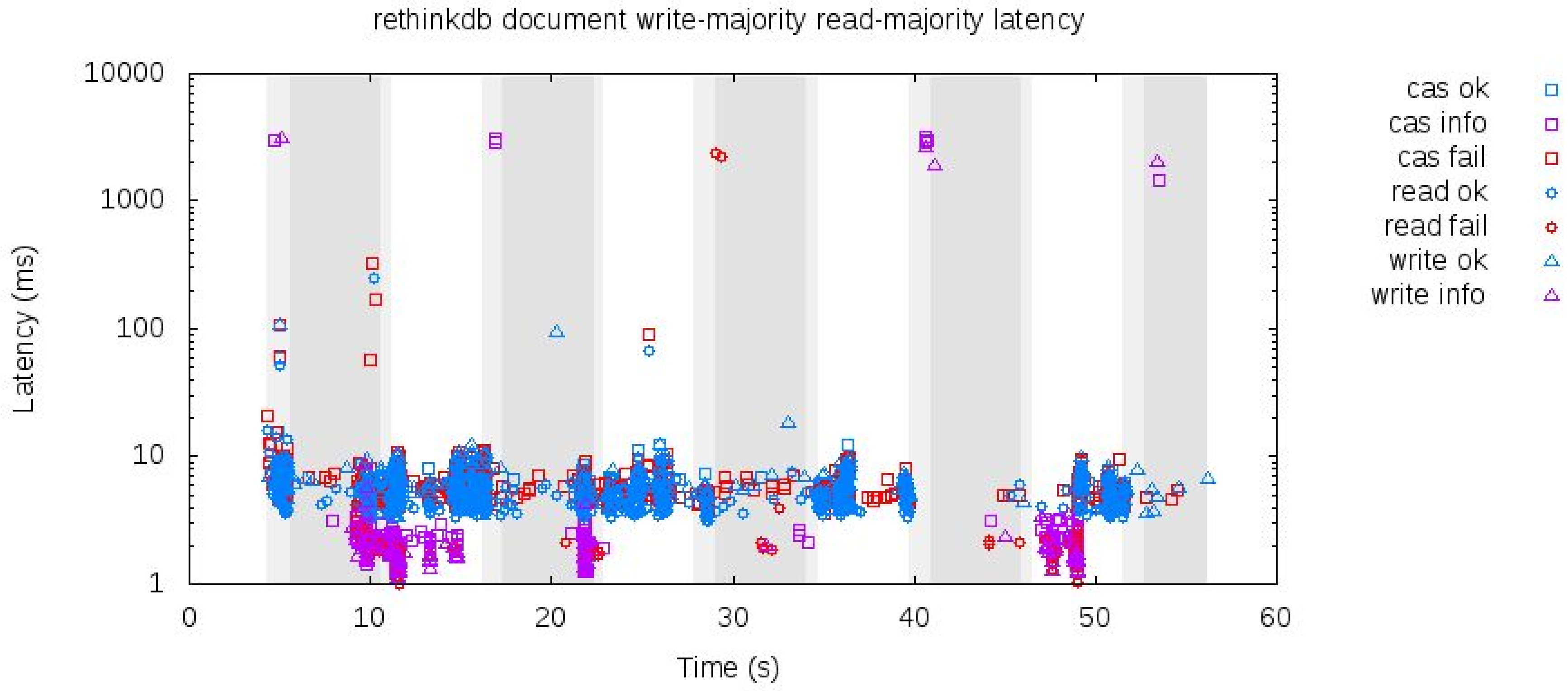


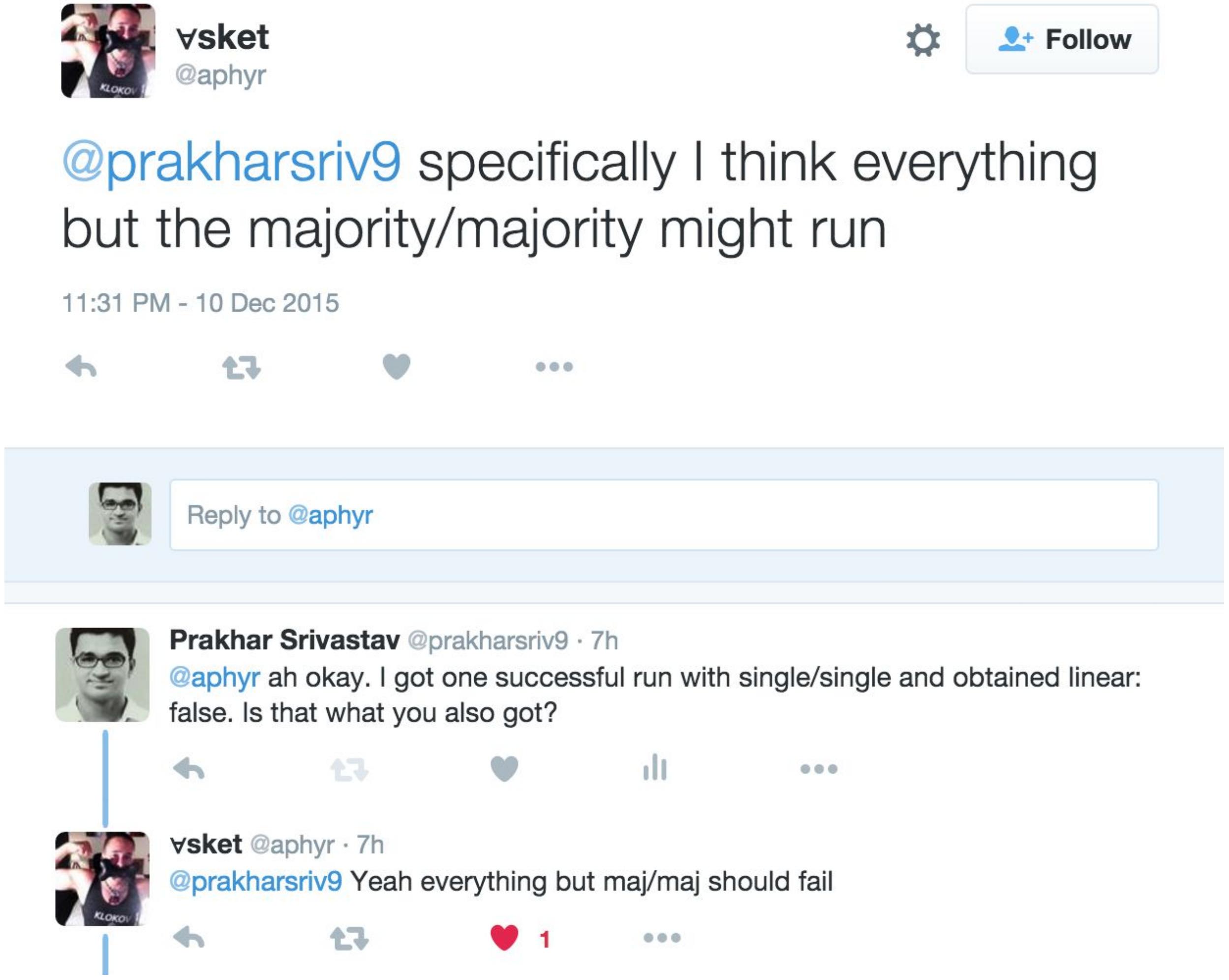
```
{:linear
 {:valid? false,
  :configs
  ({:model {:value nil},
    :pending
    [{:type :invoke,
      :f :read,
      :value [0 nil],
      :process 0,
      :time 4368627313,
      :index 3}
     {:type :invoke,
      :f :write,
      :value [0 2],
      :process 2,
      :time 4353514374,
      :index 2}
     {:type :invoke,
      :f :read,
      :value [0 nil],
      :process 3,
      :time 4357750834,
      :index 1}]}),
```

Graphs - Single / Single



Graphs - Majority / Majority





∀sket
@aphyr

Follow

@prakharsriv9 specifically I think everything but the majority/majority might run

11:31 PM - 10 Dec 2015

Reply to @aphyr

Prakhar Srivastav @prakharsriv9 · 7h
@aphyr ah okay. I got one successful run with single/single and obtained linear: false. Is that what you also got?

∀sket @aphyr · 7h
@prakharsriv9 Yeah everything but maj/maj should fail

1

DEMO

Shortcomings & Future Work

- Validate more configurations e.g. Single / Majority, Majority / Single
- Add tests for validating guarantees around split-brain
- Make tests more thorough by running for longer duration (very resource intensive) and for variety of timing settings
- Mapping of different configurations to weaker consistency semantics

Future Work

- Preferred ideal: Jepsen analysis like MongoDB

Lessons Learnt

- Analyzing distributed systems is hard
- No amount of RAM/CPU is ever enough

Questions?

- Prakhar Srivastav (ps2894)
- Ayush Jain (aj2672)

} Team Members

Code, scripts and tests are available on Github: http://github.com/prakhar1989/ADS