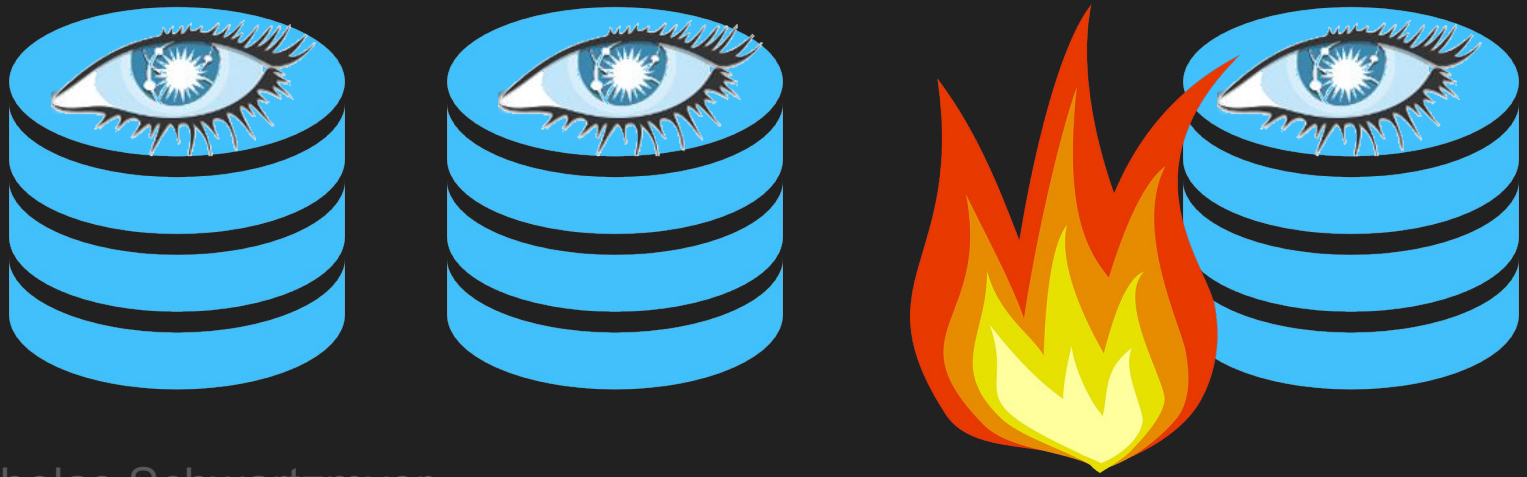
Store it, maybe?

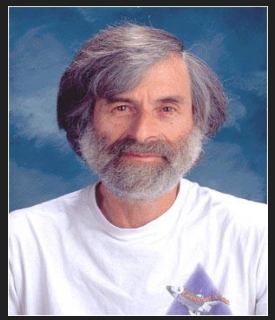
Partition testing Cassandra with Jepsen



Nicholas Schwartzmyer
Insight Data Engineering Fellowship
New York

What’s the value of this project?

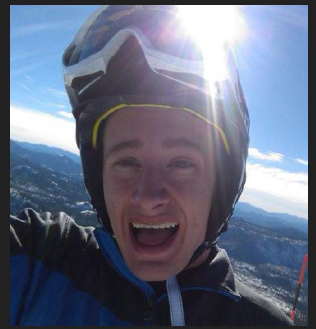
- Understanding system failure is vital to becoming a good engineer.
- Distributed systems are notoriously hard to reason about, even for seasoned engineers behind industry-standard distributed databases.
- Reason better while diving deep into Cassandra internals



Try to Think like a Distributed boss!

<-- Leslie Lamport

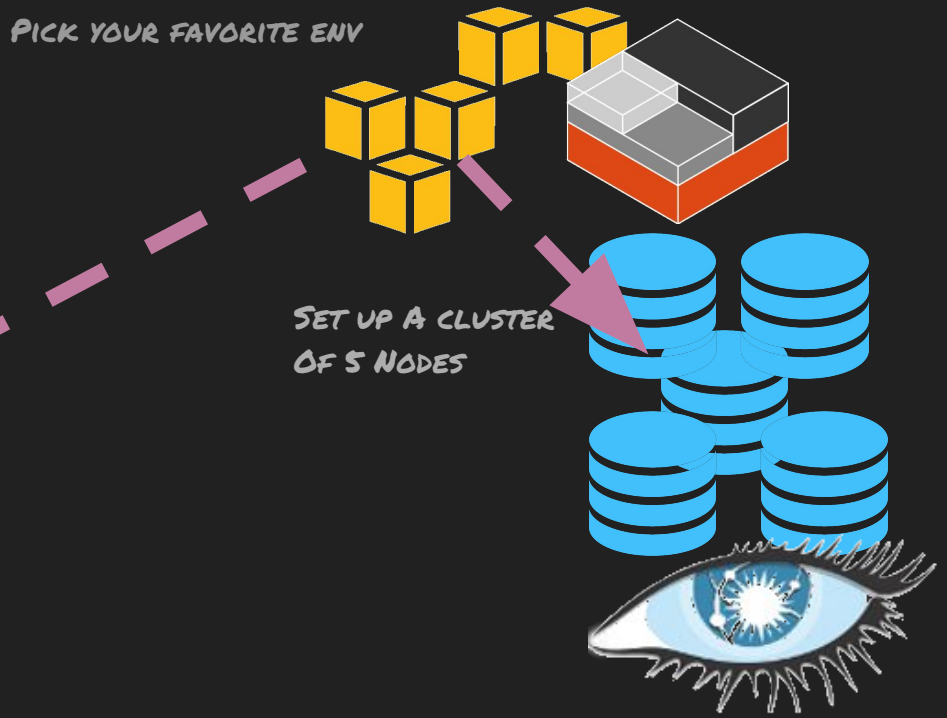
Kyle Kingsbury/Aphyr -->



Also...

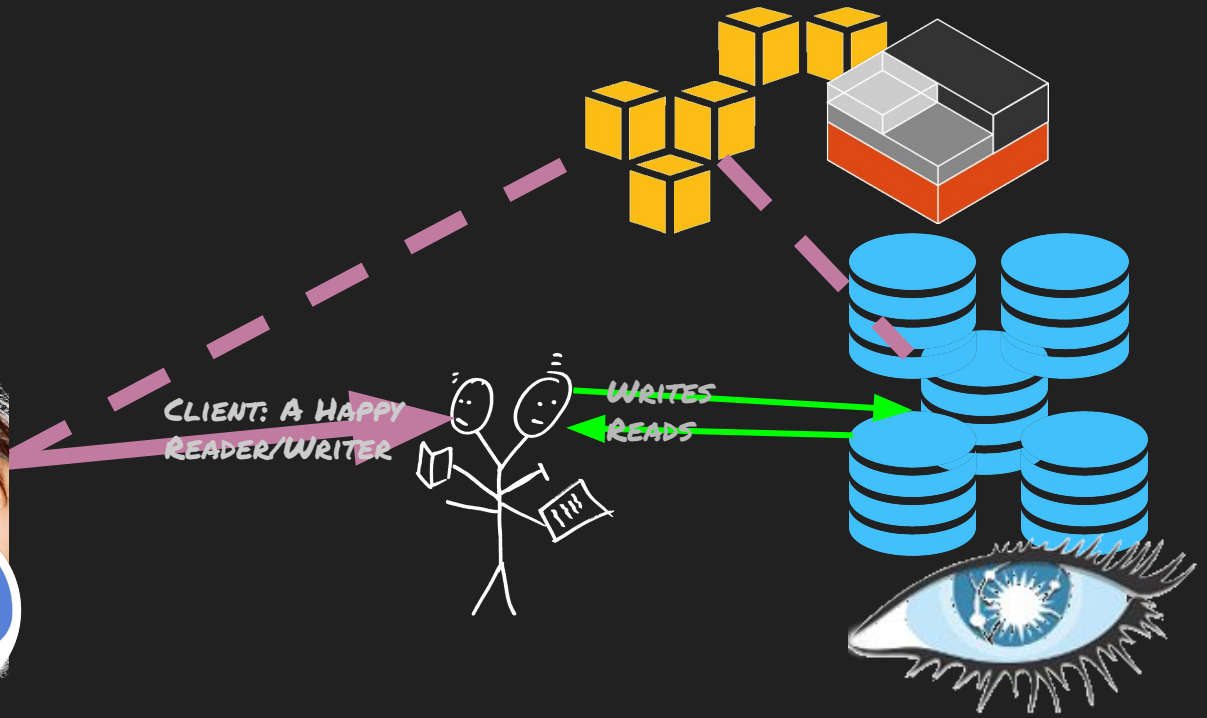
Data drives our companies. Losing or corrupting it is BAD!

Jepsen



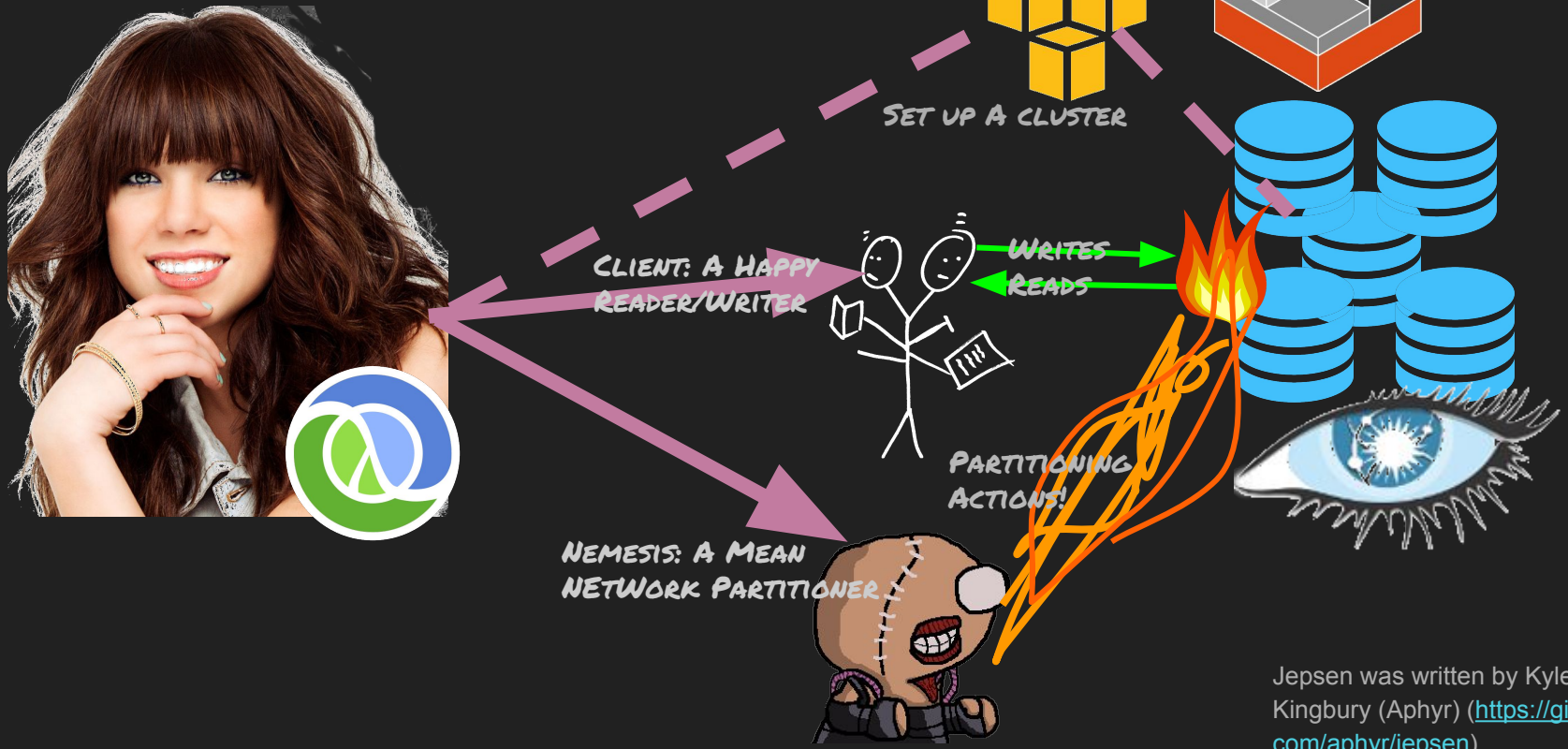
Jepsen was written by Kyle Kingbury (Aphyr)
(https://github.com/aphyr/jepsen)

Jepsen



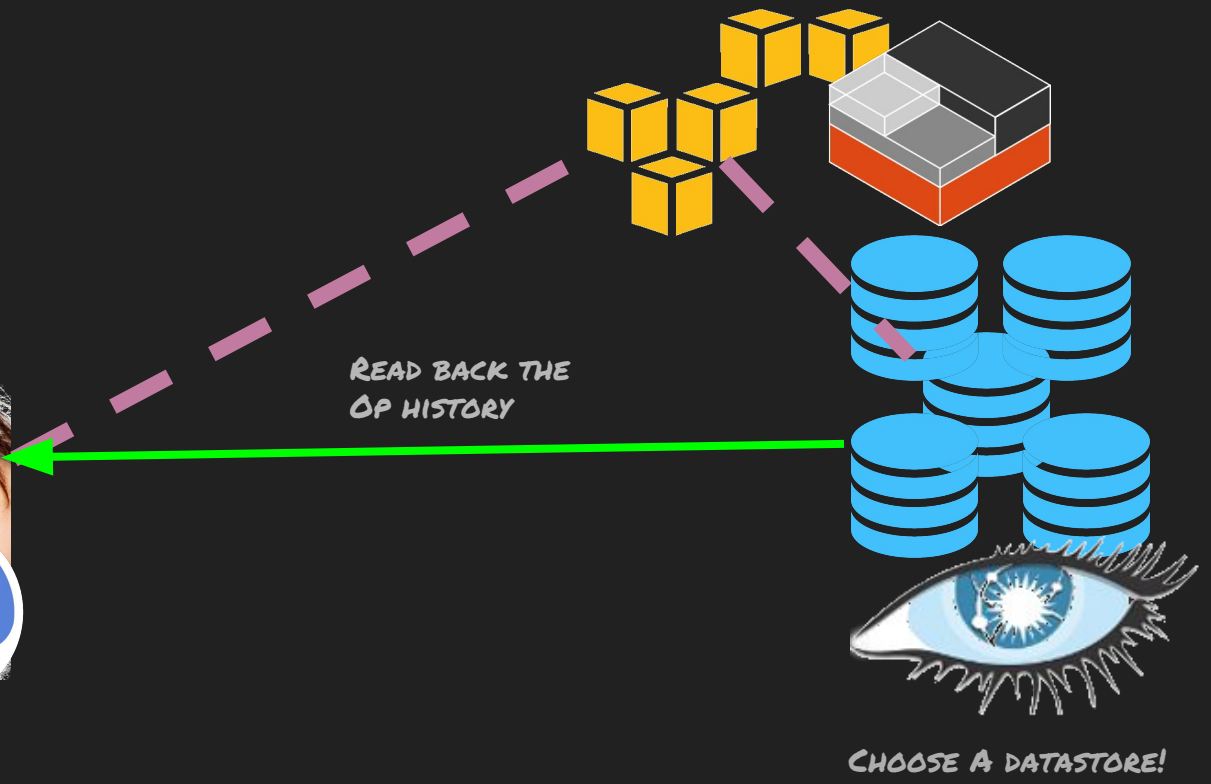
Jepsen was written by Kyle Kingbury (Aphyr) (https://github.com/aphyr/jepsen)

Jepsen



Jepsen was written by Kyle Kingbury (Aphyr) (https://github.com/aphyr/jepsen)

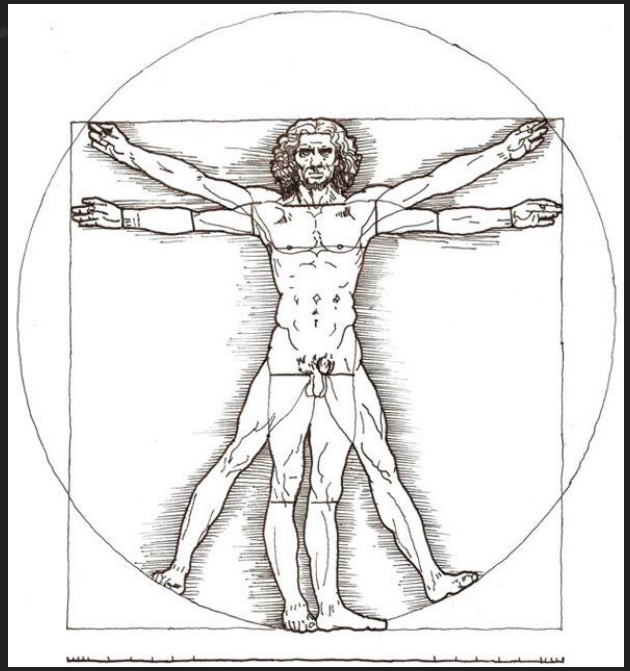
Jepsen



Jepsen was written by Kyle Kingbury (Aphyr) (https://github.com/aphyr/jepsen)

Jepsen

Model: The Ideal sequence of ops



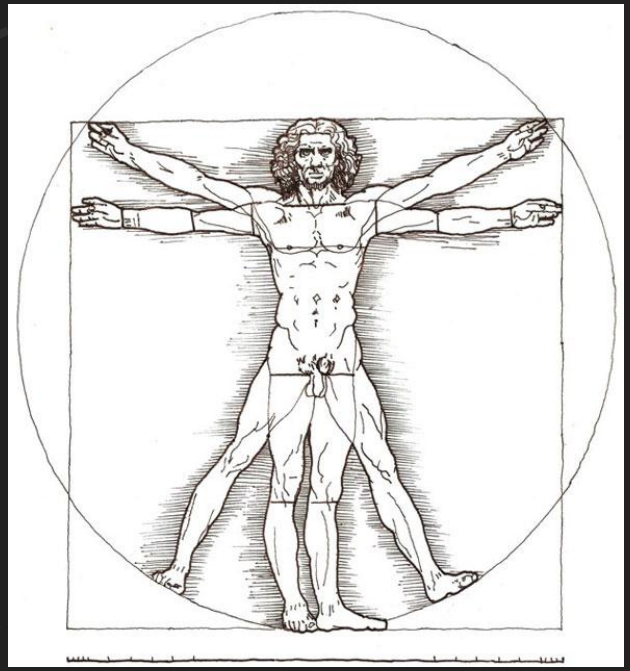
Jepsen was written by Kyle Kingbury (Aphyr) (https://github.com/aphyr/jepsen)

Jepsen

=

Check Model Against History

≠



Jepsen was written by Kyle Kingbury (Aphyr) (https://github.com/aphyr/jepsen)

Why test Cassandra?

- Most popular wide-column datastore*
- Emphasis on availability ⇝ *eventual consistency*
- EC ⇝ error-prone ordering decisions
- Existing tests ⇝ verifiable, incorrect
  (https://aphyr.com/posts/294-jepsen-cassandra)

*http://db-engines.com/en/ranking/wide+column+store

Test Plan

Used DataStax's existing tests
(http://www.datastax.com/dev/blog/testing-apache-cassandra-with-jepsen)

Cassandra 2.1.14 & 2.2.6

- Tests for 3.6 had compatibility issues
- Changelogs did not indicate work that would *weaken* fundamental data persistence
- 3.x important when EPaxos is released (https://issues.apache.org/jira/browse/CASSANDRA-6246)

Focus on stable state tests

- Lightweight transactions
- Batch inserts
- Counters
- Set & Map operations

Test Plan

Parameter tweaks:

- add/read/write consistency level
  ALL->QUORUM->TWO, SERIAL
- hinted_handoff: (true|false)
- batch only: ATOMIC (default)/UNLOGGED
  *UNLOGGED *not* tested by DataStax
- Keep replication factor = 3

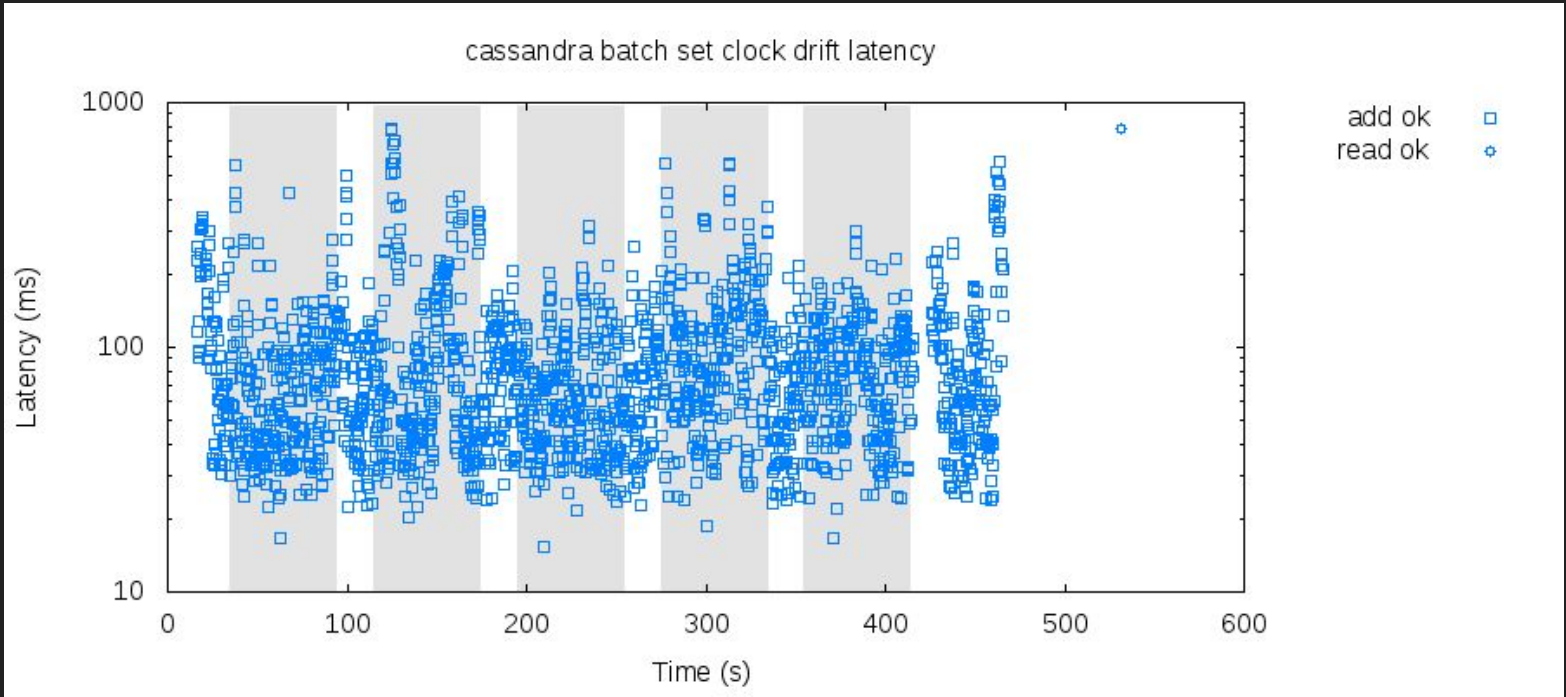
Results

All Results found here:
https://github.com/nps/jepsen/tree/master/cassandra/analysis

Let’s focus on the more interesting cases for now:

- Lightweight Transactions (LWT)
- Batch inserts (Atomic & Unlogged)

Results: Atomic Batch




```
INFO  jepsen.core - Everything looks good! ヽ(‘ー`)ノ
```

Fig1: qq, -hh

Results: Unlogged Batch

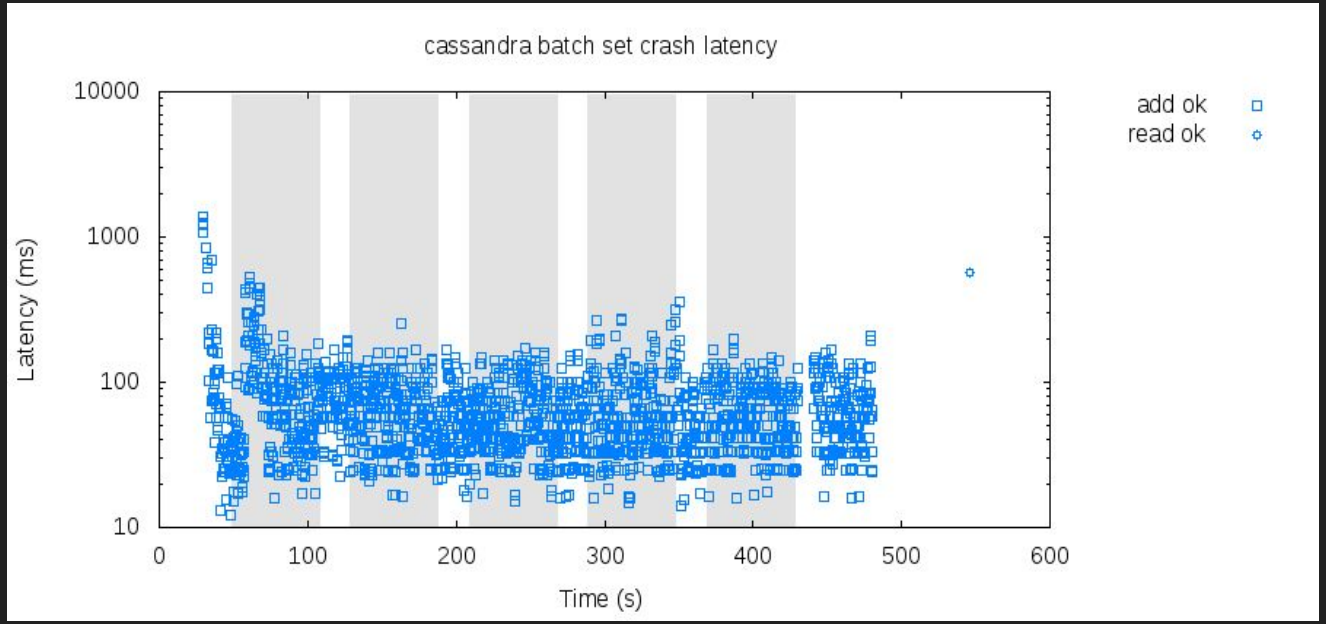


Fig: W=Q,R=Q,-hinted_handoff

Inconsistencies are possible if client & coordinator both suffer failures.

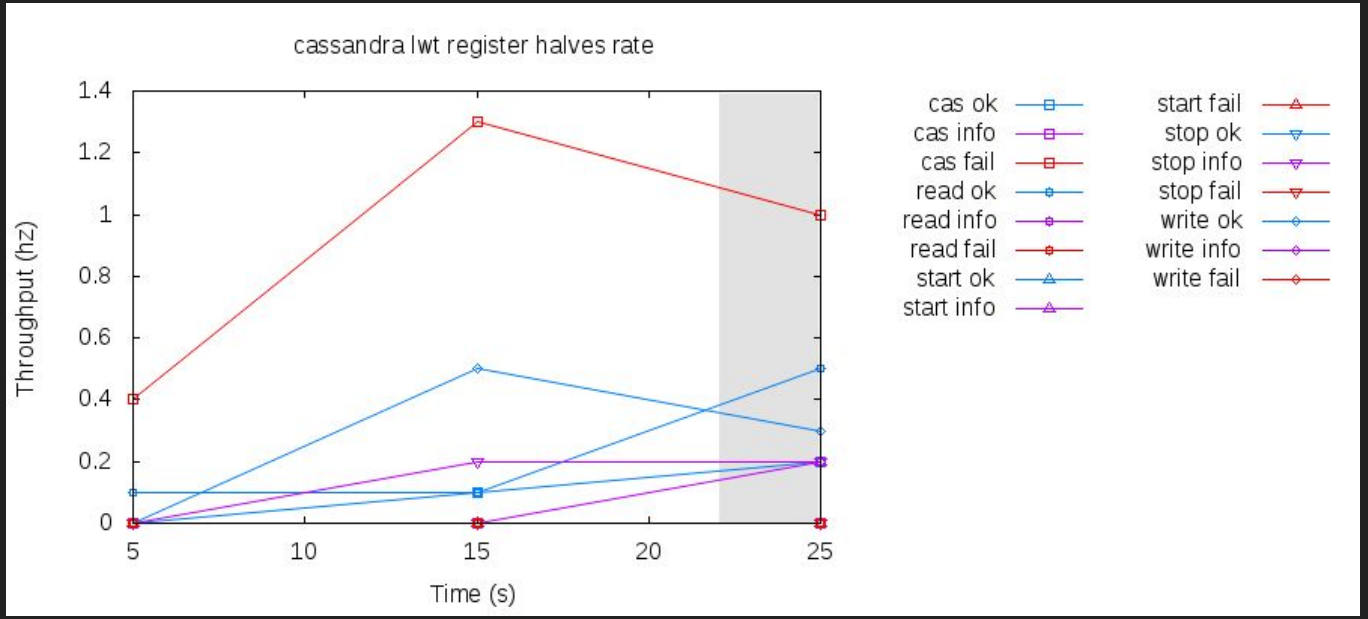
```
INFO  jepsen.core - Everything looks good! ヽ(‘ー`)ノ
```

Results: Lightweight Transactions

A promise *linearizable consistency**. A juicy claim…

Aha,FAILED CAS!

but not so fast...



*http://www.datastax.com/dev/blog/lightweight-transactions-in-cassandra-2-0

Results: Lightweight Transactions

With CAS, *operational* errors aren't necessarily *logical* errors!*

Provided we can read the previous known value, we can consider it legal.

```
61  (invoke! [this test op]
62    (case (:f op)
63      :cas (try (let [[v v'] (:value op)
64                      result (cql/update conn "lwt" {:value v'}
65                                         (only-if [[= :value v]])
66                                         (where [[= :id 0]]))]
67                  (if (-> result first ak)
68                    (assoc op :type :ok)
69                    (assoc op :type :fail :value (-> result first :value))))
70                (catch UnavailableException e
```

If cas wasn't applied, mark it as failed and set it to the value of 'value'

* See this JIRA ticket for one good discussion of the matter

Results: Lightweight Transactions

With CAS, *operational* errors aren't necessarily *logical* errors!*

Provided we can read the previous known value, we can consider it legal.

```
40    ; A failure; fill in either value.
41    :fail
42    (let [i           (get index (:process op))
43          _           (assert i)
44          invocation  (nth history i)
45          value       (or (:value invocation) (:value op))
46          [invocation' op'] (if (and (= :cas (:f op)) (number? (:value op)))
47                              [(assoc invocation :f :read :value (:value op))
48                               (assoc op :f :read :type :ok)]
49                              [(assoc invocation :value value) (assoc op :value value)])]
```

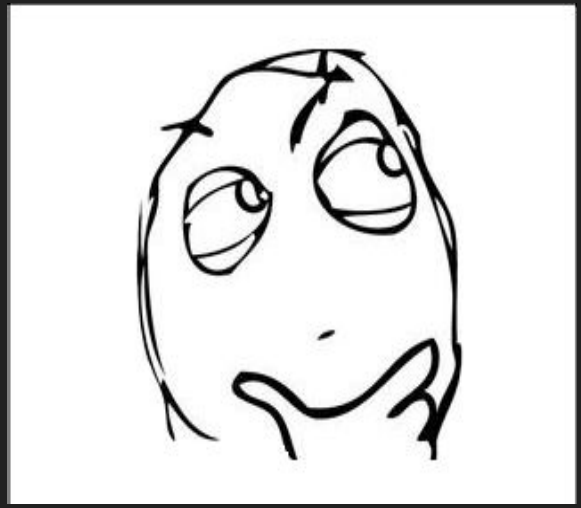
Ensure the failed cas is has a valid previous value and map it as a read

* See [this JIRA ticket] for one good discussion of the matter

Challenges!

The Ivory Tower kind

- Understanding Jepsen mechanics by reading through the code
- Picking up some Clojure to do so!
- Crash course in Cassandra fault tolerance model
- Researching distributed data structures and state models



Challenges!

The realities-of-working-with-software kind

- Getting Jepsen up-and-running
- Significant merge conflict and versioning issues with the tests

About Me

Nick Schwartzmyer

- MS, Computational Linguistics
- 8 years professional experience

I Like Hunting bugs!

Increasingly interested in Software Correctness

Command line geek

Intentionally blank

Test Plan

Nemesis types:

- Bridge partitions
- Random node isolations
- Clock skew
- Kill a node

Results: Atomic Batch

W=Q,R=ALL,+/-hinted_handoff:

```
INFO  jepsen.core - Everything looks good! ヽ(‘ー`)ノ
```

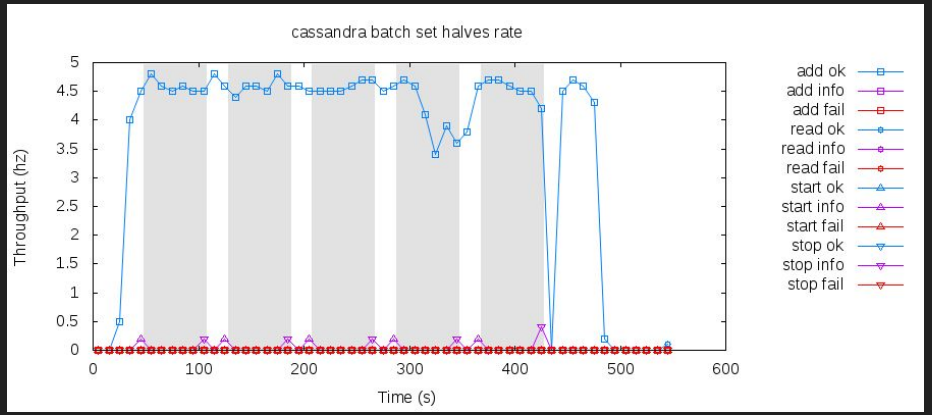
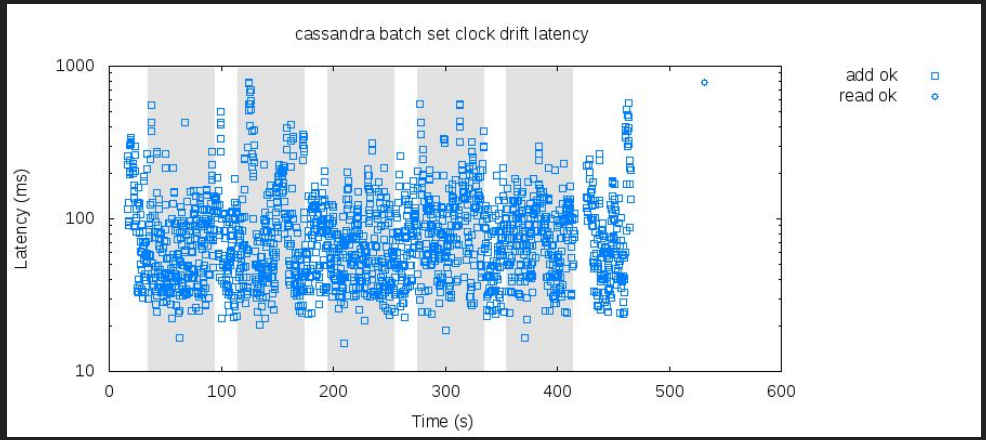


Fig1: qa, +hh



W=Q,R=Q,+/-hinted_handoff:

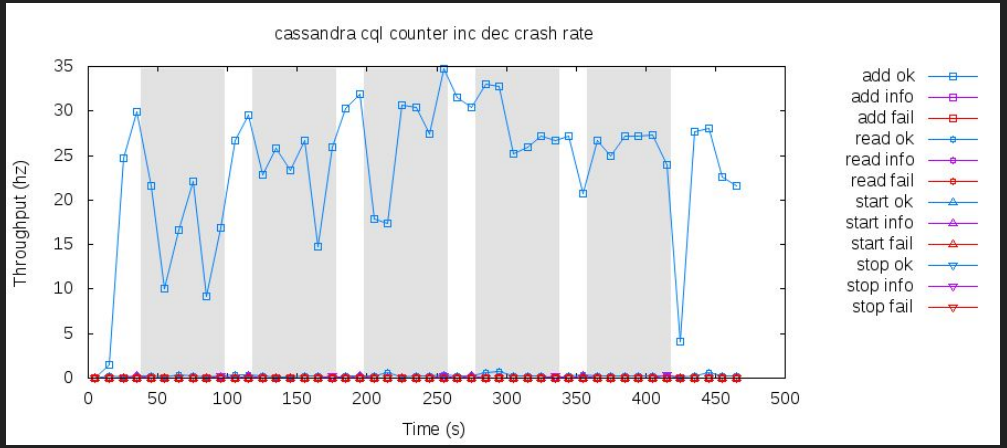
```
INFO  jepsen.core - Everything looks good! ヽ(‘ー`)ノ
```

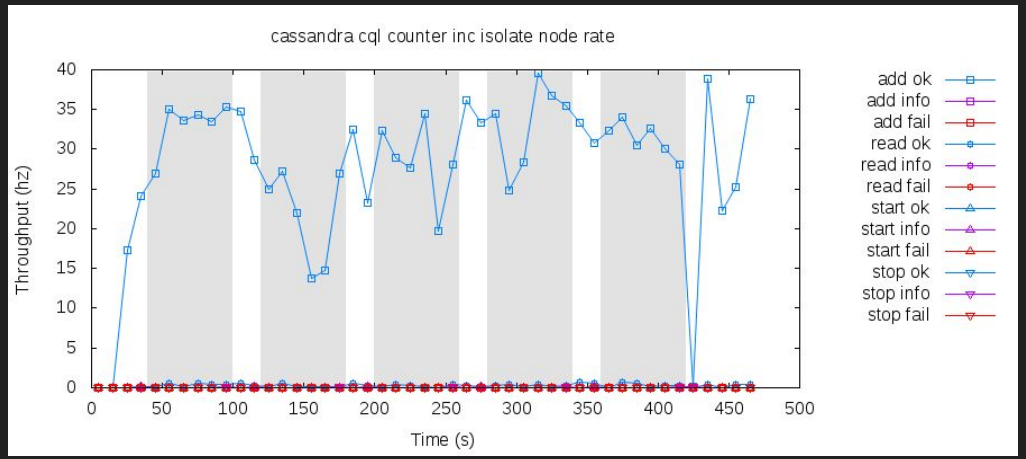
Fig2: qq, -hh

Results: Counters

Monotonic INC & INC/DEC tests:

```
INFO  jepsen.core - Everything looks good! ヽ(‘ー`)ノ
```

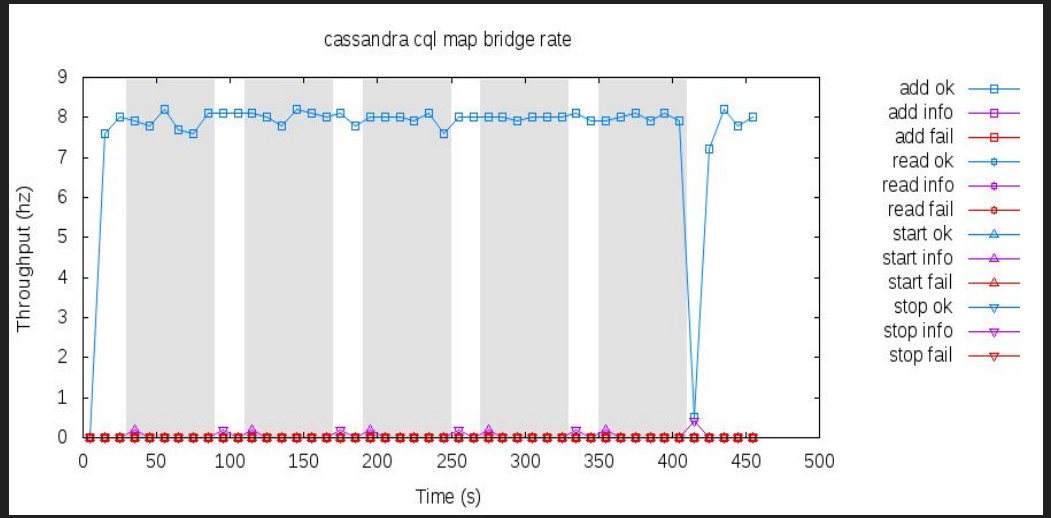




Results: Map & Set tests

```
INFO  jepsen.core - Everything looks good! ヽ(‘ー`)ノ
```

```
{:set
 {:valid? true,
  :lost "#{}",
  :recovered "#{}",
  :ok "#{0..3525}",
  :recovered-frac 0,
  :unexpected-frac 0,
  :unexpected "#{}",
  :lost-frac 0,
  :ok-frac 1},
 :perf
 {:latency-graph {:valid? true},
  :rate-graph {:valid? true},
  :valid? true},
 :valid? true}
results.edn (END)
```



In Summary

The original Jepsen tests
succeeded in making the
C* folks better at designing for failure

Testing makes our systems better

But finding nothing wrong is also kind of sad.

Future directions

Get tests working for C* 3.6

Compare with Riak

- Also Dynamo-based, old Jepsen test exist
- Richer CRDTs; what's the performance hit when partitioned?

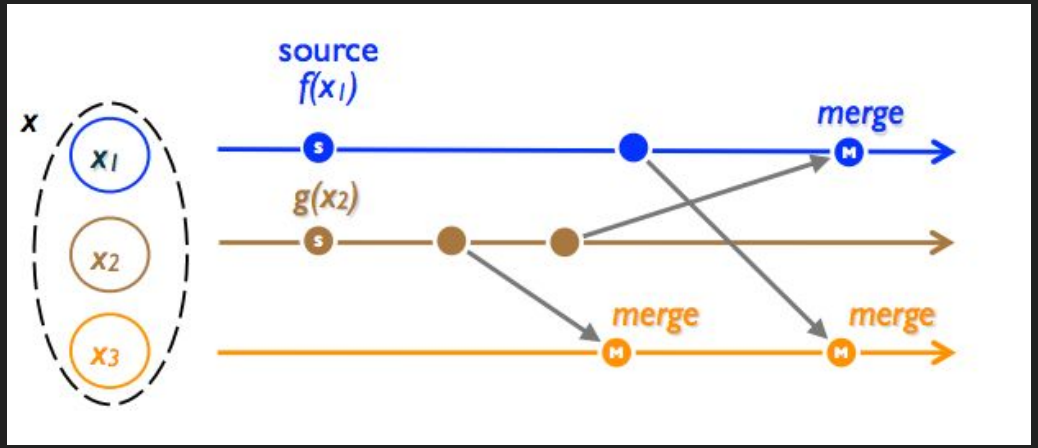
Write my own tests!

- HBase: Aphyr: "I wouldn't be at all surprised if HBase is terrible haha."
- Accumulo: Growing in popularity, NSA backed

(These are backed by Zookeeper, so already have known minimum constraints, though…)

In an even more distant future...

For fun, Implement a class of CRDT atop a datastore



Analyze how they impact with Jepsen tests