Jepsen VII
kyle Kingsbury
Please use the Scala Days app to rate sessions.
All happy databases are alike; each unhappy database is unhappy in its own way. — Leo Tolstoy
Anna
Concurrenina
Kyle Kingsbury
@aphyr

I break databases!
Every is thing fine!
Databases!
Queues!
Discovery!
THE HORROR
Split Brain
Broken Foreign keys
write ok read ≠ read ok

DB →

Anomalies
How do you know if a system is safe?
Measure your Systems
Jepsen

github.com/aphyr/jepsen
Environment

System
-INVARİANTS-
client: \( w \xrightarrow{w'} \ r \xrightarrow{r'} \ w \)
Clients generate random operations and apply them to the system.
1. Generate ops
2. Record history
3. Verify history is consistent w/model
Partitions!
"So, what have you found?"
Riak

LWW \rightarrow \text{lost writes}

CRDTs \rightarrow \text{safe}
Mongo

Data loss at all

Write concerns
Redis Sentinel

Split brain, massive write loss
Cassandra

- LWW write loss
- Row isolation broken
- Transaction deadlock
  data loss
NuoDB

Beat CAP by buffering all requests in RAM during partition
Kafka

In-sync Replica Set could shrink to 0 nodes, causing msg loss.
Zookeeper

Works.
etcd / Consul

stale reads
Elastic search

Loses documents in every class of partition tested.
RabbitMQ

Split brain, massive message loss
Aerospike

Claims "ACID", was really LWW.
Elasticsearch 1.5.0

Still loses data in every test case
MongoDB 2.6.7

stale reads

dirty reads
Chronos

-Breaks forever after losing quorum
"Snapshots" weren't preserved
First-committe-wins not preserved
Read locks broken
RethinkDB

- Basic tests passed
- Reconfiguration could destroy cluster in rare cases
Be bad, but at least don't be a liar, a deceiver!
VOLTDB 6.3
Distributed SQL Database

All data in RAM (but persistent)
Basic SQL queries

Txns as Java stored procedures
Intended for high-throughput, mostly sharded transactions
Keystroke

abc ... m ... r ...

Partitions

a ... l  m ... q  r ... z
A partition

TXN Queue

SPI

replicas
Across Partitions

MPI

SPI

SPI

SPI
- Txns in 2 partition scale ~linearly
- Txns across partitions have const throughput
Claim: all txns are strict serializable!
Serializability

time →
Linearizability
Strict Serializable

- Real-time
- Multi-object
VoltDB 6.3 exhibits several violations of strict serializability.
Stale Reads
Dirty Reads
Lost Updates
Stale Reads
Node 1

Node 2

network partition

w4
For a given partition

SPI orders updates
...but not reads
Solution:

Push all txns through the SPI ordering path
Dirty Reads
$T_1 \quad w3...\text{abort}$

$T_2 \quad r3$
Lost Updates
$T_1: \; W_1 \xleftarrow{\text{lost!}}$

$T_2: \; \rho_0$
Waiting: [n2, n3]
n2 down
n3 down
Waiting: []

Cluster too small!
"Zookeeper" watches are asynchronous; race condition between ack & shutdown
Solution:
Make cluster shrink
$\text{shutdown one atomic step}$
Still loses acknowledged writes...
On crash recovery, Volt picks longest log as authoritative.
fail

\[ n_1 \]

\[ n_2 \]

\[ n_3 \]

\[ n_1 \text{ has longest log, so } n_2 \text{ txns lost} \]
Solution:
Reconstruct cluster state from fault logs
Multi-Partition Transactions

No faults found
MPI fully serializes read and write txns, may prevent divergence
VoltDB 6.4

Passes all Jepsen tests for strict serializability
Future work

- Formal models
- Partial failure
- Clock skew
Distributed SQL DB for IOT, analytics, etc
“after having written its record, the record will be consistent and immediately available when selecting this primary key across the cluster.”
App

CRATE

Elastic Search
ES 1.1.0: extensive data loss
ES 1.5.0: less extensive data loss
Crate uses 1.7.
Elasticsearch 1.7

- Stale reads
- Dirty reads
- Lost updates
OK but how bad are dirty reads anyway?
Sequential updates
Are versions actually distinct?
- Write unique values
- Read value, version
{
  :value 1, :version 2
}

{
  :value 3, :version 2
}
value
version
add 3

10

1

8

2

13

2

2

3

9

sub 2

add 1

(fails)

(dirty read)
$10 + 3 + 1 = 14 \neq 9$

$10 - 2 + 3 + 1 = 12 \neq 9$

Preserves garbage -2
Loses ok update +3
If you read, modify, write, use Read Committed.
lt gets worse!
It turns out that Crate will lose plain old inserts, too.
Not Crate's fault! It's an ES issue...
What about using a newer Elasticsearch?
Elastic closed

#7572: "network partitions can cause documents to be lost"
Resiliency page:
Lost documents is fixed in 5.0.0
5.0.0-alpha5

- dirty reads
- lost updates
- replica divergence
#20031
Dirty reads:

no plan to fix

⇒ no safe updates
Lost updates:

#20384: Can promote stale primaries. Partial fix in 5.0.0. Full: 6.0?
Replica Divergence

Unacked docs aren't resynchronized until the next replica recovery
still unaddressed
as of Dec 2016
(5.0.2)
Recommendations

- Don't use ES as a system of record.
- Don't make up your own consensus algo.
Crate & ES are well suited for machine data where loss is OK!
- Scale-out SQL
- Postgres wire protocol
- Like Spanner*
- Hybrid Logical Clocks
Txns in a Raft cluster are linearizable.
Cockroach cheats on reads with time-based leases

⇒ Needs reliable timeouts
Txns across n Raft clusters use a custom protocol w/1-HLC

⇒ Needs synced clocks
- Writes commit once replicated.
- Reads on contested keys may block up to clock skew threshold.
insert A

insert B

select *

[]
insert A

insert B

select *

[A]
insert A

insert B

select *

[A, B]
insert A

select *

insert B

Serializable!  [B]  Nonlinearizable
Cockroach Labs tested with Jepsen

- Clock-skew detector needed to be stricter
- SQL timestamps derived from KV timestamps
We have to go Deeper
We found two new issues together!
<table>
<thead>
<tr>
<th></th>
<th>table 1</th>
<th></th>
<th>table 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>
Select max;
insert max + 1
**table 1**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

**table 2**

<table>
<thead>
<tr>
<th>0</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

2 copies!
Time stamp

---

Cache

- Stores set of keys read by a txn
- Prevents read keys from modification
Txns with same timestamp can steal each other's ts cache entries
Fix: clear txn id for conflicting reads

⇒ Allows read-read-read, but subsequent writes fail
Problem #2

insert 1;
insert 2;
insert 3;
...

...
Select *

{0, 1, 2, 3, 4, 5, ... 3}

whaaaat.
- Single-stmt txhs can be executed in 1 phase
Now distinguishes between retryable and non-retryable errors.
CockroachDB now passes its test suite!

The world needs more serializable SQL DBs
- But remember, clocks **MUST** be synchronized.

- Performance **still** a sticking point—**it's** ß
Recap
Read the docs!

CAREFULLY
Then test it for yourself
"Strict"

"Strong"

"ACID"
Be Formal

&

Be Specific
Figure out the invariants your system needs
Consider your failure modes
Process

Crash

#kill -9 1234
Node failure

- AWS terminate
- Physical power switch
Clock Skew

# date 10280000

# fake time ...
GC/IO Pause

# killall -s STOP foo

# killall -s CONT foo
Network Partition

# iptables -j DROP

# tc qdisc ... delay ... drop ...
Test your Systems

end to end
If you look for perfection, you will never be content.
- Property testing
- High-level invariants
- With distsys failure modes
Thanks

VoltDB
Crate.io
Cockroach Labs
Boaz Leskes
Peter Alvaro

Funded Research
http://jepsen.io
Please Remember to rate this talk

Thank you!

#scala.days
val x = 123456789
val y = 0

x - (if (true) \{ x \} else \{ y \})
val x = 1234567891
val y = 0

x - (if (true) {x} else {y})

⇒ -3.0