How we Test a Distributed System

LiuQi | PingCAP

About me

- LiuQi / co-founder & CEO @ PingCAP
- Working on TiDB / TiKV projects

Shit happens

Are you ready to handle those shits ?

Are you ready to handle those shits ?

Always be prepared.

Are you ready to handle those shits ?

Test **everything**.

Or

When I mean everything...

- Unit tests
  - Coverage
- Performance tests
  - Bench Whole system
  - Bench Each layer
  - Bench Functions
  - Compatibility tests
- Integration tests
- Fuzz tests

Automate everything.

Well, still not enough

- Profile everything
  - Function
  - Disk IO
  - Network
  - CPU
- Enable profiling even online
  - May be a once-in-a-lifetime chance. CPU is burning but we may never know why.
- **Design** for testing or **Die** without good tests

Hold on, who is the tester?

- Testing is owned by the entire team. It is a culture, not a process.
- Quality comes from solid engineering.
- Stop talking and go build things.
- Don’t hire too many testers.
- Are testers software engineers? Yes.
- Hiring good people is the first step. And then keep them challenged.

Random kill

What doesn’t kill you makes you stronger.

Error injection

If there is an error, make that error happen again.

Fault injection

If there is a failure, make that failure happen again.

Event order

If there are ordered events, reorder.

Do not tell me you are right, prove it.

AWS: [Use of Formal Methods at Amazon Web Services]

Applying TLA+ to some of our more complex systems

System	Components	Line count (excl. comments)	Benefit
S3	Fault-tolerant low-level network algorithm	804 PlusCal	Found 2 bugs. Found further bugs in proposed optimizations.
	Background redistribution of data	645 PlusCal	Found 1 bug, and found a bug in the first proposed fix.
DynamoDB	Replication & group-membership system	939 TLA+	Found 3 bugs, some requiring traces of 35 steps
EBS	Volume management	102 PlusCal	Found 3 bugs.
Internal distributed lock manager	Lock-free data structure	223 PlusCal	Improved confidence. Failed to find a liveness bug as we did not check liveness.
	Fault tolerant replication and reconfiguration algorithm	318 TLA+	Found 1 bug. Verified an aggressive optimization.

Do not tell me you are right, prove it.

The Most Frequently Asked Question On learning about TLA+:

Q: “How do we know that the executable code correctly implements the verified design?”

A: The answer is that we don’t.

Why we need a proof ?

- If the design is broken, the code is almost certainly broken.

Why we need a proof ?

- Formal methods help engineers to find strong invariants, so formal methods can help to improve assertions, which help improve the quality of code

TLA+ in TiDB

- Transaction model is based on Google Percolator.

We make lots of optimizations. So

- Transaction model should be proved by TLA+.

A proof is not enough.

Create a daemon that checks all the constraints.

Found several serious RocksDB bugs:

https://github.com/facebook/rocksdb/pull/2799

https://github.com/facebook/rocksdb/pull/3017

Let’s talk about fail point

- Fail points are used to add code points where errors may be injected in a user controlled fashion.

Fail point

- Fail point in freebsd looks like:

```
sysctl debug.fail_point.foobar="5*return(5)->0.1%return(22)"
        For 5 times, return 5.  After that, 1/1000th of the time, return
        22.

sysctl debug.fail_point.foobar="0.1%5*return(5)"
        Return 5 for 1 in 1000 executions, but only 5 times total.

sysctl debug.fail_point.foobar="1%*sleep(50)"
        1/100th of the time, sleep 50ms.

sysctl debug.fail_point.foobar="1*return(5)[pid 1234]"
        Return 5 once, when pid 1234 executes the fail point.
```

fail-rs

- The idea came from freebsd
- Built for rust

fail-rs: why did we build it?

- Make failure injection more precise.
- Make writing test cases easier by injecting failure when invoking the libraries API.

fail-rs: how does it help?

- We use it in our unit tests to test known cases.
- We wrote different scenarios to test it continuously and randomly.

fail-rs: how does it work?

- Design overview
  - Use the Rust macro to make injection easy and completely optimised for release.
  - Basically a global map that stores the failpoint name and errors to be injected. When program runs into failpoint, it checks if there is any action, and then execute it.

fail-rs

- An example: we want to simulate database write failure or write with high latency

```
fail_point!("write", |e| Err(e.into()));
db.write(wb)?;
```

  - write=return(“err”) to simulate write failure
  - write=sleep(1000) to simulate busy IO

But i am a go programmer?

- gofail: created by CoreOS
- Used to test etcd.
- Also used in TiDB

But i am a go programmer?

- Are you still fighting with error handling ?

```
if err != nil  {

}
```

Does it really work if error happens ?

```
if err != nil {
        // handle error
    // is the current branch coverd by your test?
}
```

gofail in TiDB

```
25  // committing primary region task.
26  func (s *testCommitterSuite) TestFailCommitPrimaryRpcErrors(c *C) {
27          gofail.Enable("github.com/pingcap/tidb/store/tikv/mocktikv/rpcCommitResult",
    `return("timeout")`)
```

Jepsen

- Run Nemesis to disturb a cluster
- Record request and response
- Verify the linearizability of history

Focus on network and linearizability testing

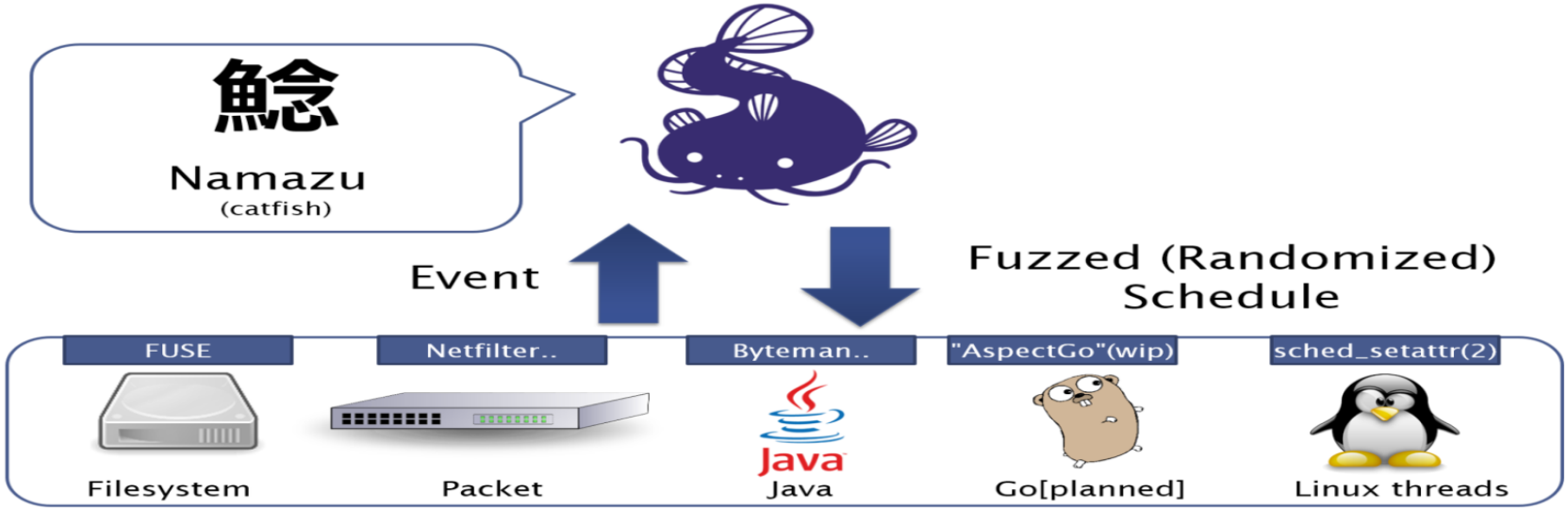
TiDB with Jepsen

- parts: network partition
- start-stop: send SIGSTOP to some nodes
- start-kill: send SIGKILL to some nodes

Each time when there is an update in the TiDB code, we will internally trigger CI to execute Jepsen tests

Namazu

- Filesystem Inspector - Use Fuse to delay or inject faults
- Ethernet Inspector - Use iptables to drop or delay packages
- Process Inspector - Change scheduler attributes



Namazu

Issue	Reproducibility (traditional)	Reproducibility (Namazu)	Note
YARN-4548	11%	82%	Used Namazu process inspector.
ZOOKEEPER-2080	14%	62%	Used Namazu Ethernet inspector. Blog article and repro code are available.
YARN-4556	2%	44%	Used Namazu process inspector.
YARN-5043	12%	30%	Used Namazu process inspector.
ZOOKEEPER-2137	2%	16%	Used Namazu process inspector.
YARN-4168	1%	8%	Used Namazu process inspector.
YARN-1978	0%	4%	Used Namazu process inspector.
etcd #5022	0%	3%	Used Namazu process inspector.

We also improved reproducibility of some flaky etcd tests (to be documented).

Inside TiKV

- Random scheduler
  - Homemade chaos monkey
- Always Check Sum / Hash

Fuzzing tools

- Is it really necessary?

Fuzzing tool: american fuzzy lop

Mozilla Firefox [1] [2] [3] [4]	Internet Explorer [1] [2] [3] [4]	Apple Safari [1]
Adobe Flash / PCRE [1] [2] [3] [4]	sqlite [1] [2] [3] [4]...	OpenSSL [1] [2] [3] [4] [5] [6] [7]
LibreOffice [1] [2] [3] [4]	poppler [1]	freetype [1] [2]
GnuTLS [1]	GnuPG [1] [2] [3] [4]	OpenSSH [1] [2] [3]
PuTTY [1] [2]	ntpd [1]	nginx [1] [2] [3]
bash (post-Shellshock) [1] [2]	tcpdump [1] [2] [3] [4] [5] [6] [7] [8] [9]	JavaScriptCore [1] [2] [3] [4]

File system is stable, until

Time to first bug

Filesystem	Time	Type
Btrfs	5s	BUG()
Ext4	2h	BUG()
F2fs	10s	BUG()
Gfs2	8m	Double free
Hfs	30s	Page Fault
Hfsplus	25s	Page Fault
Nilfs2	1m	Page Fault
Ntfs	4m	Soft lockup
Ocfs2	15s	BUG()
Reiserfs	25s	BUG()
Xfs	1h45m	Soft lockup

Fuzzing tools

- Random Query Generator
  - https://launchpad.net/randgen
  - Found 7 bugs of TiDB
  - run desired test:
    - ./gentest.pl --dsn dbi:mysql:host=127.0.0.1:port=4000:user=root:database=test --grammar=conf/optimizer_subquery_portable.yy --gendata --thread=1

Schrodinger

- Use configuration to define cases
- Run cases in Kubernetes
- Run Nemesis to disturb a cluster
  - Reorder network packages
  - Isolate node and network partition
  - Kill / Restart / Pause node randomly
  - Slow down IO operation / Return error
  - More……
- Validate cases
- Web visualization

Keep calm because Shit still happens

Thanks

We are hiring!

Q&A