Debugging and Extending

Distributed Coordination Systems

@rgs_
member of @twitterSRE
ZooKeeper committer

agenda

- Twitter infrastructure
- how does it all communicate
- ZooKeeper: good, not great & bad parts
- the use cases
- pushing ZooKeeper to its limits
- debugging your pipelines
- bleeding edge features + extensions
- contributing back

twitter infrastructure

- erstwhile monolithic app
- now, thousands of services that need to:
    - discover each other
    - share config data
    - distribute & discover sharded datasets
    - contend for distributed locks
    - elect primaries, secondaries, etc
    - … and more!

how does it all communicate

- Apache ZooKeeper provides low-level primitives for:
    - maintaining & propagating configuration
    - distributed synchronization
    - group membership
    - liveness checks

apache zookeeper (the good parts)

- well maintained recipes for locking, leader election, sharding, leases, etc
- it’s been battle tested over the years (almost venerable at this point)
- libraries available for most languages (C, Java, Python, Go)

apache zookeeper (not great parts)

- session handling is easy to get wrong
- without careful thought, apps can bloat the number of watches/reads/writes needed
- not hard for apps going into GC turbulence to DoS a cluster
- session setup (in 3.4) is expensive, thundering herds are problematic
- no eager checks for ACLs, exists(), ...

apache zookeeper (the bad parts)

- throttling support is rudimentary at scale
- no way to introspect writes which timed out
- no client-side stats/introspection
- server-side stats collection mechanism is in-band
- arcane serialization format (jute)

the use cases

- /configs (runtime decisions: what to do?)
- /services (discovery: where is it?)
- /locks (acquisition: who owns it?)

though: do we really need the same consistency model for all three?

service discovery: the easy part

```
from kazoo.client import KazooClient
cli = KazooClient("cluster.tld:2181")
cli.create(
 "/services/api/%s:9090" % socket.gethostname(),
 "http/json",
 ephemeral=True
)
```

service discovery: the hard part

- pings every ⅓ sessionTimeout
- discovery != availability (end-to-end health checks)
- partitions (caching strategy needed)
- n+1 reads on every upstream instance startup
- updates are ~expensive
- no way to check `create()` timeouts

distributed locking: the easy part

```
cli= KazooClient("cluster.tld:2181")
lock = cli.Lock(
  "/locks/db/users/shards/00",
  socket.gethostname()
)
with lock:
   # serve shard
```

distributed locking: the hard part

- liveness check: you still need an external healthcheck
- when should you yield the lock?
    - on disconnect: might be too noisy
    - on session expiration: might never converge

config dist: the easy part

```
def updates_cb(*args, **kwargs):
   # something

cli.get_children(
 "/configs/features", watch=updates_cb
)
```

config dist: the hard part

- size constraints
- versioning
- blocking in the event thread

pushing ZooKeeper to its limits

- x-DC locks with < 1.sec timeouts
- configuration distribution to > hundreds of thousands clients
- one of the biggest mesos production clusters

debugging your pipelines

- ideally, we’d release frequently enough to all clusters with more stats & visibility on every release
- lots of legacy clients didn’t take elections that well
- ad-hoc & client-side introspection needed
- out-of-band monitoring would help

bleeding edge

- we’ve been running 3.5 (trunk+patches) for ~2 years
- local sessions are great, you should use them (modulo bugs)
- when possible, check at the edge not at the leader (i.e.: ACL checks, NoNode, etc)

extensions

- given the wide variety of clients (using a combination of versions, languages), backwards compatibility is a must
- we retrofitted backpressure by reusing the liveness parts of the protocol (i.e.: ping replies & watches)
- to enhance visibility we tagged requests

contributing back

- released zktraffic: tools to analyze the ZooKeeper protocol:

  https://github.com/twitter/zktraffic
- uncovered many client & server-side bugs, we sent patches upstream
- we’ll drive the next ZK release (3.4.7)
- hopefully sync with upstream soon

questions

- tweet @TwitterSRE, @rgs_
- we hang out at #zookeeper (FreeNode) too
- rgs@twitter.com

bonus slide: read-only support

- for ~1.5 years we did use extended read-only support (ZOOKEEPER-1607). It generally works well, though it needs a client-side change
- it would be nice to upstream this change, as a complement to the current read-only support

bonus slide: Netty/Finagle

- there is server-side Netty support in ZK (default is NIO)
- we decided to use Finagle Futures instead