Call me maybe: Zookeeper

2013/09/23
Software Network Distributed Systems Jepsen Zookeeper

In this Jepsen post, we'll explore Zookeeper. Up next: NuoDB.

Zookeeper, or ZK for short, is a distributed CP datastore based on a consensus protocol called ZAB. ZAB is similar to Paxos in that it offers linearizable writes and is available whenever a majority quorum can complete a round, but unlike the Paxos papers, places a stronger emphasis on the role of a single leader in ensuring the consistency of commits.

Because Zookeeper uses majority quorums, in an ensemble of five nodes, any two can fail or be partitioned away without causing the system to halt. Any clients connected to a majority component of the cluster can continue to make progress safely. In addition, the linearizability property means that all clients will see all updates in the same order–although clients may drift behind the primary by an arbitrary duration.

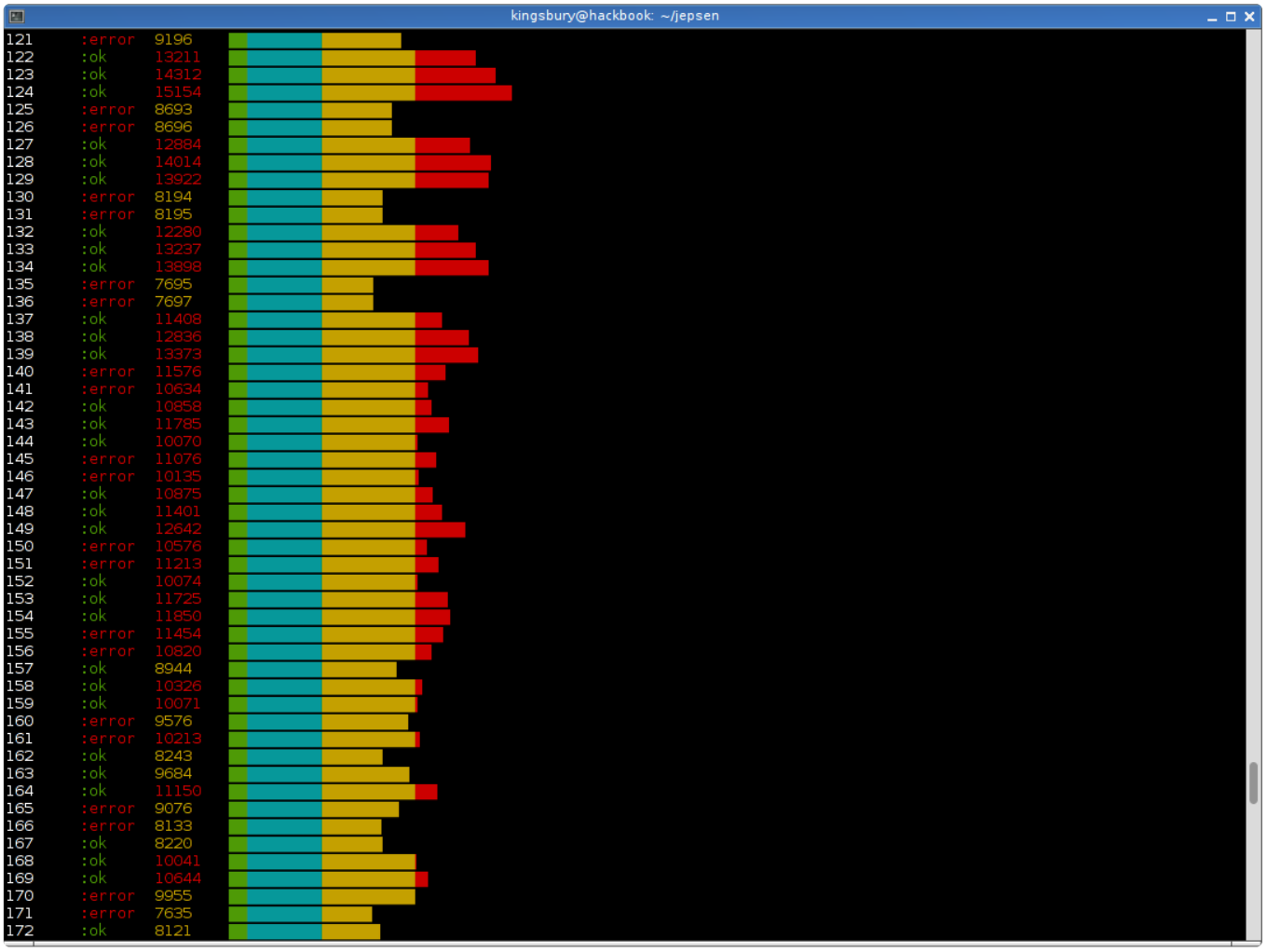
This safety property comes at a cost: writes must be durably written to a disk log on a majority of nodes before they are acknowledged. In addition, the entire dataset must fit in memory. This means that Zookeeper is best deployed for small pieces of state where linearizability and high availability is critical. Often, ZK is used to track consistent *pointers* to larger, immutable data stored in a different (perhaps AP) system; combining the safety and scalability advantages of both. At the same time, this strategy reduces the availability for writes, since there are two systems to fail, and one of them (ZK) requires majority quorums.

ZNode linearizability

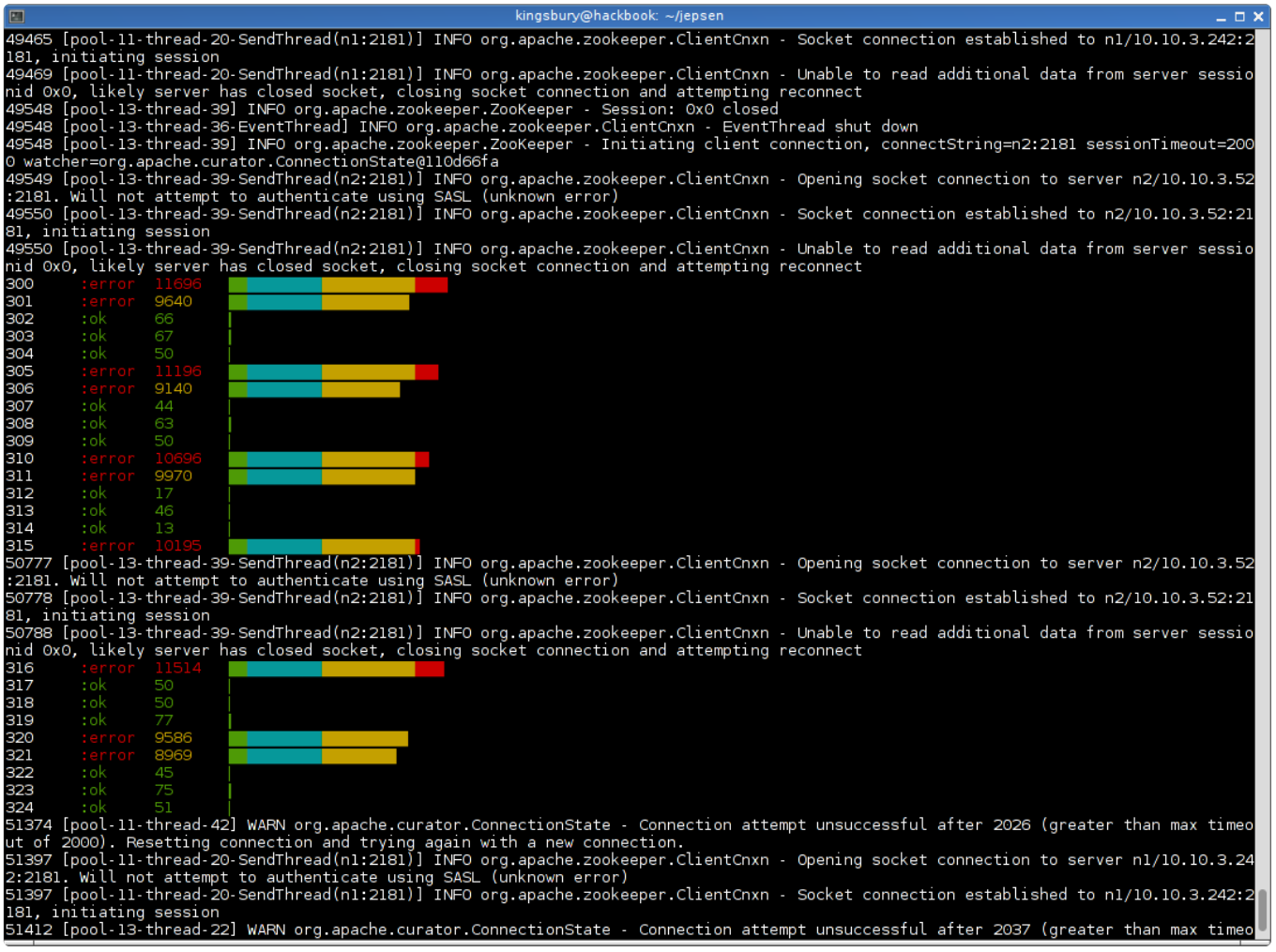
In this test, five clients use a Curator DistributedAtom to update a list of numbers. The list is stored as a single serialized znode, and updates are applied via a CaS loop: atomically reading, decoding, appending the appropriate number, enoding, and writing back iff the value has not changed.

```
(let [curator (framework (str (:host opts) ":2181") "jepsen")
      path    "/set-app"
      state   (distributed-atom curator path [])]
  (reify SetApp
    (setup [app]
      (reset!! state []))

    (add [app element]
      (try
        (swap!! state conj element)
        ok
        (catch org.apache.zookeeper.KeeperException$ConnectionLossException e
          error)))

    (results [app]
      @state)

    (teardown [app]
      (delete! curator path)))))
```

Initially, the ZK leader is n1. During the test, we partition [n1 n2] away from [n3 n4 n5], which means the leader cannot commit to a majority of nodes–and consequently, writes immediately block:



After 15 seconds or so, a new leader is elected in the majority component, and writes may proceed again. However, only the clients which can see one of [n3 n4 n5] can write: clients connected to [n1 n2] time out while waiting to make contact with the leader:



When the partition is resolved, writes on [n1 n2] begin to succeed right away; the leader election protocol is stable, so there is no need for a second transition during recovery.

Consequently, in a short test (~200 seconds, ~70 second partition, evenly distributed constant write load across all nodes) ZK might offer 78% availability, asymptotically converging on 60% (3/5 nodes) availability as the duration of the partition lengthens. ZK has never dropped an acknowledged write in any Jepsen test. It also typically yields 0-2 false positives: likely due to writes proxied through n1 and n2 just prior to the partition, such that the write committed, but the acknowledgement was not received by the proxying node.

As with any experiment, we can only disconfirm hypotheses. This test demonstrates that in the presence of a partition and leader election, Zookeeper is able to maintain the linearizability invariant. However, there could be other failure modes or write patterns which would *not* preserve linearizability–I just haven't been able to find them so far. Nonetheless, this is a positive result: one that all CP datastores should aim for.

Recommendations

Use Zookeeper. It's mature, well-designed, and battle-tested. Because the consequences of its connection model and linearizability properties are subtle, you should, wherever possible, take advantage of tested recipes and client libraries like Curator, which do their best to correctly handle the complex state transitions associated with session and connection loss.

Also keep in mind that linearizable state in Zookeeper (such as leader election) does not guarantee the linearizability of a system which *uses* ZK. For instance, a cluster which uses ZK for leader election might allow multiple nodes to be the leader simultaneously. Even if there are no simultaneous leaders at the same wall-clock time, message delays can result in logical inconsistencies. Designing CP systems, even with a strong coordinator, requires carefully coupling the operations in the system to the underlying coordinator state.

Next up: NuoDB.

Finally, a success story :) Could you elaborate though on the false positives? Does the leader also proxy writes? Thanks!

Paul Evans, on 2013/09/27

I think you mean “false negatives” in the paragraph near the end:

It also typically yields 0-2 false positives: likely due to writes proxied through n1 and n2 just prior to the partition, such that the write committed, but the acknowledgement was not received by the proxying node.

Writes that were successful, but the requesting client was never informed of such.

Marco Serafini, on 2013/09/29

Nice post! About the differences between Paxos and Zab, it is mainly due to the fact that Zookeeper uses passive replication. You can have a look here for more details http://arxiv.org/pdf/1308.2979v3

Thomas Beck, on 2014/03/09

But what happened to the two leaders once the partition was resolved? Did n1 drop his lead responsibility?

Ganesh Chandrasekaran, on 2014/04/17

@Thomas The n1 lost its leadership as soon as there was a partition because there are only 2 nodes on its side (n1 and n2). The requirement is that there should be (n/2)+1 available in the cluster so in this case it would be 3.

Tim B, on 2014/05/20

Hi, great post.

Do you know what happens if the net connection between two nodes goes down in a three node ensemble? Say n1 and n2 can communicate, and n2 and n3 can communicate, but n1 cannot reach n3 - that’s not a partition or node failure. Can n1 sync with n3 via n2 forwarding messages?

Post a Comment

Name

Email

Http

Supports github-flavored markdown for [links](http://foo.com/), *emphasis*, _underline_, `code`, and > blockquotes. Use ```clj on its own line to start a Clojure code block, and ``` to end the block.

Post Comment