```
6.824 2020 Lecture 9: Chain Replication, CRAQ

CRAQ, Terrace and Freedman, USENIX 2009

What ideas?
  * Chain Replication, a very different approach from e.g. Raft.
  * CRAQ's ability to read from any replica AND have strong consistency.
  * Why can CRAQ do this but ZooKeeper/Raft/&c can't?
  * Implications for overall design of a CR/CRAQ system.

What is Chain Replication (CR)?
  (Renesse and Schneider, OSDI 2004)
  Influential: CRAQ and many others build on CR.
    Ceph, Parameter Server, COPS, FAWN.
  Goals: if client gets write reply, data is safe if even one server survives.
         and linearizable.
  S1, S2, S3, S4
  S1 is "head"
  S4 is "tail"
  Writes:
    Client sends to head
    Forwarded down the chain, in order
    Each server overwrites old data with new data
    Tail responds to client
  Reads:
    Client sends to tail
    Tail responds (no other nodes involved)

Why read only from the tail?

Intuition for linearizability of CR?
  When no failures, almost as if the tail were the only server.
    Head picks an order for writes, replicas apply in that order,
      so they will stay in sync except for recent (uncommitted) writes.
    Tail exposes only committed writes to readers.
  Failure recovery, briefly.
    Good news: every replica knows of every committed write.
    But need to push partial writes down the chain.
    If head fails, successor takes over as head, no commited writes lost.
    If tail fails, predecessor takes over as tail, no writes lost.
    If intermediate fails, drop from chain, predecessor may need to
      re-send recent writes.

Why is CR attractive (vs Raft)?
  Client RPCs split between head and tail, vs Raft's leader handles both.
  Head sends each write just once, vs Raft's leader sends to all.
  Reads involve just one server, not all as in Raft.
  Situation after failure simpler than in Raft (remember Figure 7).

Why is it attractive to let clients read any replica in CR?
  The opportunity is that the intermediate replicas may still
    have spare CPU cycles when the read load is high enough to saturate the tail.
  Moving read load from the tail to the intermediate nodes
    might thus yield higher read throughput on a saturated chain.
  The CRAQ paper admits there is at least one other way to skin this cat:
    Split objects over many chains, each server participates in multiple chains.
    C1: S1 S2 S3
    C2: S2 S3 S1
    C3: S3 S1 S2
  This works if load is more or less evenly divided among chains.
  It often isn't.
    Maybe you could divide objects into even more chains.
    Or use CRAQ's ideas.

Would it be correct (linearizable) to let clients read any replica in CR?
```

```
    No.
    A read could see uncommitted data, might disappear due to a failure.
    A client could see a new value from one replica,
      and then an older value from a different (later) replica.

  How does CRAQ support linearizable reads from any replica in the chain?
    (Figure 2/3, Section 2.3)
    Each replica stores a list of versions per object.
    One clean version, plus dirty version per recent write.
    Write:
      Client sends write to head.
      Replicas create new dirty version as write passes through.
      Tail creates clean version, ACKs back along chain, replicas turn "dirty" to "clean".
    Read from non-tail node:
      If latest version is clean, reply with that.
      Q: if latest is dirty, why not return most recent clean?
         (The Question. newer maybe already exposed to another reader!)
      Q: if latest is dirty, why not return that dirty version?
         (not committed, might disappear due to replica failure)
      If dirty, ask tail for latest version number ("version query").

  Intuition for why same as CR (i.e. linearizable) (assuming no failure):
    If replica has only clean, it MUST match tail, since no write has passed it.
    If replica has dirty, it asks tail, in which case it matches tail as well.

  Why can CRAQ serve reads from replicas linearizably but Raft/ZooKeeper/&c cannot?
    Relies on being a chain, so that *all* nodes see each
      write before the write commits, so nodes know about
      all writes that might have committed, and thus know when
      to ask the tail.
    Raft/ZooKeeper can't do this because leader can proceed with a mere
      majority, so can commit without all followers seeing a write,
      so followers are not aware when they have missed a committed write.

  Does that mean CRAQ is strictly more powerful than Raft &c?
    No.
    All CRAQ replicas have to participate for any write to commit.
    If a node isn't reachable, CRAQ must wait.
    So not immediately fault-tolerant in the way that ZK and Raft are.
    CR has the same limitation.

  Equivalently, why can't 2nd node take over as head if it can't reach the head?
    Partition -- split brain -- the 2nd node must wait patiently.

  How can we safely make use of a replication system that can't handle partition?
    A single "configuration manager" must choose head, chain, tail.
    Everyone (servers, clients) must obey or stop.
      Regardless of who they locally think is alive/dead.
    A configuration manager is a common and useful pattern.
      It's the essence of how GFS (master) and VMware-FT (test-and-set server) work.
      Usually Paxos/Raft/ZK for config service,
        data sharded over many replica groups,
        CR or something else fast for each replica group.
      Lab 4 works this way (though Raft for everything).
```