6.824 FAQ for Object Storage on CRAQ: High-throughput chain replication for read-mostly workloads, by Jeff Terrace and Michael J. Freedman, USENIX 2009.

Q: How does CRAQ cope with network partition and prevent split brain?

A: CRAQ (and Chain Replication) do not themselves have a defense against partitions and split brain. In the short term, if a chain node doesn't respond, the other chain members have to wait. CRAQ and CR depend on a separate configuration service, which decides which servers make up each chain. The configuration service monitors the servers to form an opinion of who is alive, and every time a server seems to be unreachable, the configuration service decides which servers in the chain are still alive and tells those servers (and the clients) the new chain setup. Configuration services are typically built with Paxos or Raft or (in CRAQ's case) ZooKeeper so that they are fault tolerant and so that they themselves avoid split brain despite partition. At a high level, both GFS and VMware-FT follow this pattern (GFS's master monitors server liveness and picks primaries; VMware-FT's test-and-set service picks the sole server if one dies).

Q: What are the tradeoffs of Chain Replication vs Raft or Paxos?

A: Both CRAQ and Raft/Paxos are replicated state machines. They can be used to replicate any service that can be fit into a state machine mold (basically, processes a stream of requests one at a time). One application for Raft/Paxos is object storage -- you'll build object storage on top of Raft in Lab 3. Similarly, the underlying machinery of CRAQ could be used for services other than storage, for example to implement a lock server.

CR and CRAQ are likely to be faster than Raft because the CR head does less work than the Raft leader: the CR head sends writes to just one replica, while the Raft leader must send all operations to all followers. CR has a performance advantage for reads as well, since it serves them from the tail (not the head), while the Raft leader must serve all client requests.

However, Raft/Paxos and CR/CRAQ differ significantly in their failure properties. Raft (and Paxos and ZooKeeper) can continue operating (with no pauses at all) even if a minority of nodes are crashed, slow, unreliable, or partitioned. A CRAQ or CR chain must stop if something like that goes wrong, and wait for a configuration manager to decide how to proceed. On the other hand the post-failure situation is significantly simpler in CR/CRAQ; recall Figures 7 and 8 in the Raft paper.

Q: Would Chain Replication be significantly faster or slower than the kind of primary/backup used in GFS?

A: If there are just two replicas, there's probably not much difference. Though maybe CR would be faster for writes since the tail can send responses directly to the client; in a classic primary/backup scheme, the primary has to wait for the backup to acknowledge a write before the primary responds to the client.

If there are three or more replicas, the primary in a classic primary/backup system has to send each write to each of the replicas. If the write data is big, these network sends could put a significant load on the primary. Chain Replication spreads this networking load over all the replicas, so CR's head node might be less of a performance bottleneck than a classic primary. On the other hand maybe the client-observed latency for writes would be higher in CR.

Q: The paper's Introduction mentions that one could use multiple

chains to solve the problem of intermediate chain nodes not serving
reads. What does this mean?

A: In Chain Replication, only the head and tail directly serve client
requests; the other replicas help fault tolerance but not performance.
Since the load on the head and tail is thus likely to be higher than
the load on intermediate nodes, you could get into a situation where
performance is bottlenecked by head/tail, yet there is plenty of idle
CPU available in the intermediate nodes. CRAQ exploits that idle CPU
by moving the read work to them.

The Introduction is referring to this alternate approach. A data
center will probably have lots of distinct CR chains, each serving a
fraction (shard) of the objects. Suppose you have three servers (S1,
S2, and S3) and three chains (C1, C2, C3). Then you can have the three
chains be:

```
  C1: S1 S2 S3
  C2: S2 S3 S1
  C3: S3 S1 S2
```

Now, assuming activity on the three chains is roughly equal, the load on
the three servers will also be roughly equal. In particular the load of
serving client requests (head and tail) will be roughly equally divided
among the three servers.

This is a pretty reasonable arrangement; CRAQ is only better if it
turns out that some chains see more load than others.

Q: Is it enough to make CR strongly consistent if we restrict one client
to only read from the same node throughout a session?

A: This is not OK in CR, for at least two reasons. First, reading from
a CR node other than the tail may return a write that has not
committed; if some nodes then failed, that un-committed write might be
lost. It's not legal for a read to see a write that essentially never
occured. The second problem is that, if clients compare notes (which
they are allowed to do under linearizability), they may see that one
client sees a write, but another client that reads later in real time
does not see the write. That's a violation of linearizability as well.

Q: Is the failure model for CRAQ non-Byzantine?

A: CRAQ cannot handle Byzantine failures. Just fail-stop failures.

Few systems have a good story for Byzantine failures, and typically
have to make sacrifices in performance or flexibility when they do.
There are two main approaches I'm aware of. First, systems derived
from a paper titled Practical Byzantine Fault Tolerance (PBFT) by
Castro and Liskov; PBFT is like Raft but has more rounds of
communication and uses cryptography. Second, systems in which clients
can directly check the correctness of results that servers return,
typically by use of cryptographic hashes or signatures. This can be
tricky because clients need to defend against a server that returns
data whose signature or hash is correct, but is not the latest value.
Systems like this include SUNDR and Bitcoin.

Q: Is Chain Replication used by other systems?

A: Some examples: Amazon's EBS, Ceph's Rados, Google's Parameter
Server, COPS, and FAWN.

Q: What alternatives exist to the CRAQ model?

A: People use Chain Replication (though not CRAQ) fairly frequently.

People use quorum systems such as Paxos and Raft very frequently (e.g.
ZooKeeper, Google Chubby and Spanner and Megastore).

There are lots of primary/backup replication schemes that you can view
as similar to a Chain Replication chain with just two nodes, or with the
primary sending to all replicas directly (no chain). GFS is like this.

The main technique that people use to keep strong consistency but allow
replicas to serve reads is leases.

Q: Why does CRAQ keep the old clean object when it sees a write
and creates a new dirty object?

A: Suppose a node has clean version 1, and dirty version 2. If the
node receives a read from a client, it sends a "version query" to the
tail. If the tail hasn't seen version 2, the tail will reply with
version number 1. The node should then reply with the data for version
1. So it has to hold on to a copy of version 1's data until version 2
commits.