**Week 11: Consistency**

**Overview**

The Consistency-Availability-(Network) Partitioning (CAP) theorem describes three fundamental properties of distributed systems and theoretical limits on the tradeoffs between these properties.  When Eric Brewer debuted the (at the time) hypothesis in 1998, it provided a simple, understandable model for analyzing and discussing distributed systems.  The hypothesis was proven formally by Seth Gilbert and Nancy Lunch in 2002, promoting it to a theorem.  Following lots of practical work by industry engineers and academic researchers, it was determined that the CP-AP tradeoff was not, in fact, a binary choice but rather a gradient.  Eric Brewer explained these insights in a 2012 retrospective.  In 2010, the CAP theorem was extended by Daniel J. Abadi to include tradeoffs between latency and consistency, resulting in the PACELC hypothesis, which was formally proven by Wojciech Golab in 2018.

The distributed hash table (Chord) that we reviewed is an example of a system that optimizes for availability.  Many distributed systems use a consistency-optimized component for managing cluster state.  For example, Kafka uses Apache ZooKeeper (which we'll read about this week), and Kubernetes uses etcd.   Consistent systems tend to have significantly lower throughput for writes since a majority of nodes need to confirm a write before it is acknowledged to the client.  Similarly, consistent systems generally don't scale beyond a handful of nodes.

A good example of the usage of a CP store is the use of [etcd by Kubernetes](https://kubernetes.io/docs/concepts/overview/components/" \t "_blank).  Kubernetes is controlled through the API server (a RESTful service); most components do not interact directly, but through the API server.  Using the API server, the user starts jobs, worker nodes report the status of their containers, and controllers dynamically adapt job states.  The API server manages the ground truth state of all components in the cluster.  The API server stores the cluster and job state in etcd.  By centralizing the cluster state, it prevents components from developing different interpretations of the cluster state.

A diagram of a cloud server

AI-generated content may be incorrect.

**Reflection Questions**

* Can you state the CAP theorem? What are its implications?
* In what ways have modern systems challenged our understanding of the CAP theorem and its implications?
* Can you state the PACELC theorem? What are its implications?
* What additional implications does the PACELC theorem have?
* What is Zookeeper? What are its use cases?
* What is Zookeepers data model? What operations can be performed?
* What is the use of the watch flag?
* What guarantees does Zookeeper provide?
* What is the difference between A-linearizability and linearizability?
* What do the authors consider distributed locks to be insufficient?
* Describe the distributed computing primitives (recipes) that can implemented on top of Zookeeper.
* Explain the steps in processing read and write requests by a Zookeeper cluster.
* How did the authors benchmark Zookeeper's throughput performance? How does throughput performance of the workloads scale with the number of nodes in the Zookeeper cluster and the ratio of reads and writes?