Context

Let's imagine we wanted to build a data center storage system. We have to decide on a huge amount of design points; one of the most critical is the underlying consistency of the data and the availability of the storage system.

Today: what are the implications of the various choices? More specifically, what data consistency and availability is possible, given that our systems experience failures? Also, what are the implications of different design choices on performance, in particular latency?

CAP theorem

- Consistency: atomic, linearizable data items (each write appears to happen immediately across all nodes)
- Availability: always get a response if your message goes through; no hanging
- Partition tolerance: can lose messages (varying degrees)

Brewer's PODC keynote 2000: you can have at most two of these properties

→ go through thinking highlighting why
→ A & P: must satisfy write, even though can't propagate it to a remote replica
→ C & P: must propagate write before returning; therefore blocked under partition
→ A & C: not really possible, since partitions happen in practice no matter what

(Gilbert & Lynch provided a proof of CAP. The bottom line is scary: even with bounded message delays, if you lose messages arbitrarily, writes may not be propagated correctly and you'll get stale data. Don't need partitions to cause problems, just message loss.)

Incorrect interpretation of CAP:

> Failures (P) happen. Therefore, we can't have both consistency and availability. Since we want a highly available system, we must choose to relax its consistency in the common case.

Correct interpretation of CAP:

> When there is no failure (common case), it is possible to provide applications with both consistency and availability. However, under the uncommon case of failure, need to sacrifice one of the two. For high availability systems, this means permitting inconsistencies, and potentially reconciling conflicts later on.

If you choose a system that prefers A to C, then it is helpful to explicitly detect when you enter a partition, and record additional information (e.g., causality history via vector clocks) to help the programmer detect and reconcile conflicts once the partition is repaired.

Performance and consistency

C/A is not the whole story – latency also matters tremendously. It turns out that providing strong consistency is expensive from the point of view of latency, since it involves coordination across partitions and replicas.

If, instead, you avoid the need to coordinate by relaxing consistency, you can get much better latency: update locally, let the update propagate out slowly to replicas in the background. (Eventual consistency, and as it turns out, causal consistency)

A related issue: the choice between C & A happens when a write operation experiences a timeout. At that point, the system designer has to decide whether:

- to allow the write to proceed, and risk inconsistency
- to deny the write, and sacrifice availability

Different systems make different choices. But, CAP is fundamentally related to latency.