CAP Theorem

14 February 2012

1 CAP Theorem

Content

The content here is derived from "CAP 12 years later: How the rules have changed", by E. Brewer
http://www.computer.org/portal/web/csdl/doi/10.1109/MC.2012.37

CAP Theorem

The CAP theorem states that any networked shared-data system can have at most two of three desirable properties:

- consistency (C) equivalent to having a single up-to-date copy of the data;
- high availability (A) of that data (for updates); and
- tolerance to network partitions (P).

CAP simplified

The easiest way to understand CAP is to think of two nodes on opposite sides of a partition. Allowing at least one node to update state will cause the nodes to become inconsistent, thus forfeiting C.

Likewise, if the choice is to preserve consistency, one side of the partition must act as if it is unavailable, thus forfeiting A.

Only when nodes communicate is it possible to preserve both consistency and availability, thereby forfeiting P.

Insight

CAP prohibits only a tiny part of the design space: perfect availability and consistency in the **presence** of partitions, which are rare.

By explicitly handling partitions, designers can optimize consistency and availability, thereby achieving some tradeoff of all three.

Insight...

Although designers still need to choose between consistency and availability when partitions are present, there is an incredible range of flexibility for handling partitions and recovering from them.

The modern CAP goal should be to maximize combinations of consistency and availability that make sense for the specific application. Such an approach incorporates plans for operation during a partition and for recovery afterward.

CAP space

- No reason to forfeit C or A when the system is not partitioned
- Choice between C and A at subsystem-, operation-, data-, user-level
- All 3 properties are more continuous than binary

CAP-Latency connection

In practice, partition is a time bound on communication.
Failing to achieve consistency within the time bound implies a partition and thus a choice between C and A for this operation.

These concepts capture the core design issue with regard to latency: Are two sides moving forward without communication?

There is no global notion of a partition, since some nodes might detect a partition, and others might not.

Managing partitions

When partitions are present or perceived, a strategy that detects partitions and explicitly accounts for them is in order:

1. detect partitions,
2. enter an explicit partition mode that can limit some operations, and
3. recover to restore consistency & compensate for mistakes during a partition.

Managing partitions. . .

Once the system enters partition mode, two strategies are possible.

1. Limit some operations, thereby reducing availability
2. Record extra info about the operations to help during partition recovery

Availability during partitions

The designer must decide whether to maintain a particular invariant during partition mode or risk violating it with the intent of restoring it during recovery. E.g., duplicate keys –easy to restore.

The best way to track the history of operations on both sides is to use version vectors (vector clocks), which capture the causal dependencies among operations.

Partition recovery

Recovery goals:

- the state on both sides must become consistent, and
- compensate the mistakes made during partition mode

Using commutative operations is the closest approach to a general framework for automatic state convergence.
Commutativity implies the ability to rearrange operations into a preferred consistent global order. Unfortunately, using only commutative operations is harder than it appears.

Commutative Replicated Data Types (CRDTs)

CRDTs converge after a partition by

- ensuring that all operations during a partition are commutative, or
- representing values on a lattice and ensuring that all operations during a

partition are monotonically increasing with respect to that lattice.

The latter approach converges state by moving to the maximum of each side's values. It is a formalization and improvement of what Amazon does with its shopping cart: after a partition, the converged value is the union of the two carts, with union being a monotonic set operation. The consequence of this choice is that deleted items may reappear.

Develop new CRDTs

Technically, CRDTs allow only locally verifiable invariants -a limitation that makes compensation unnecessary but that somewhat decreases the approach's power. However, a solution that uses CRDTs for state convergence could allow the temporary violation of a global invariant, converge the state after the partition, and then execute any needed compensations.

Self-stabilization approach