CAP

Summary By: Natasha Kubiak
ECE 531
SUMMER 2022

The CAP theorem was formulated first by Eric Brewer, so it is also referred to as Brewer's theorem. This states that any distributed data store can only provide at most two of the following three guarantees.

Consistency: Every read receives the most recent write, or an error.
Availability: Every request receives a (non-error) response, without the guarantee that it contains the most recent write.
Partition Tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes.

Through this discovery, Brewer found out that any system was able to satisfy two requirements, but not all three. The CAP theorem can be thought of as a triangle with each node representing one of the values in the CAP Theorem. Two nodes cannot coexist without another node being left out due to inconsistency. When nodes are able to communicate then it is possible to conserve that values of for example, Consistency and Availability, but you would have to forfeit the third node which is Partition Tolerance. Databases should not only want to preserve availability first, and consistency second, but also to the ACID, and BASE theorems The ACID and BASE theorems represent two design philosophies, which are at opposite ends of the consistency-availability spectrum. ACID focuses on consistency, while BASE captures availability. Normally, wide scale cloud systems will utilize a mix of both of these philosophies.

ACID

Atomicity: All systems benefit from atomic operations.

Consistency: a transaction pre-serves all the database.

Isolation: FOr ACID isolation, it can operate at most one side during a partition.

Durability: Running aCID transactions on each side of a partition makes recovery easier and enables a framework for compensating transactions that can be used for recovery from a partition.

The CAP theorem does not account for latency, CAP takes place during a period where a program must make a *partition decision,* either: cancel the operation and thus decreased availability OR proceed with the operation and risk inconsistency.Another challenge is for designers to be able to mitigate a partitions effect on consistency and availability, by carefully managing partitions. This management contains the following steps:

The challenging case for designers is to mitigate a partition's effects on consistency and availability. The key idea is to manage partitions very explicitly, including not only detection, but also a specific recovery process and a plan for all of the invariants that might be violated during a partition. This management approach has three steps: detect the start of a partition, enter an explicitly partition mode that may limit some operations and initiate partition recovery when communication is restored. It is a challenge to decide what to limit, and what your system can maintain under those limits. There is a risk to maintaining one thing, and then violating another.

Next there is a challenge of how to proceed when tackling mistakes made during partitioning. The system will typically discover the violation made during recovery, and can then implement a fix. So how does a system fix these partition mistakes? There are several different ways a system can potentially fix a mistake, like "last writer wins" or in the case that human intervention is needed. An example of a mistake and a potential fix could be if a computer accidentally executes something twice during a partition, then as a fix, the system may be able to distinguish that these two orders are duplicates and cancel one. However this is given if the computer happens to have a "history" of the mistake from a past experience

A system designer should keep this all in mind when creating a system, and strategize in a way where constituency and availability is not compromised in an optimized manner,