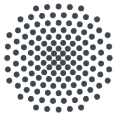


Universität Stuttgart
Master of Science

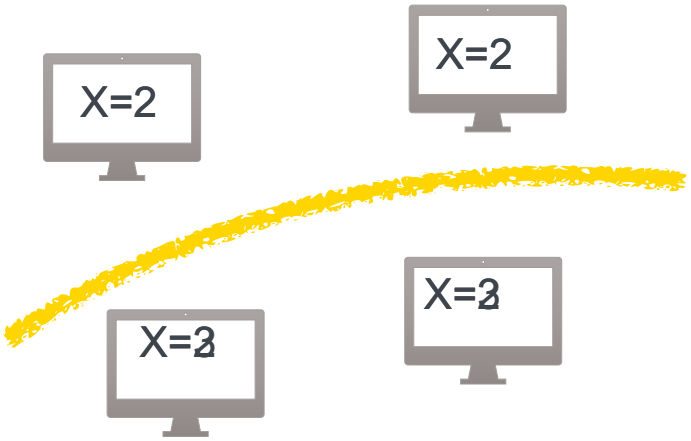
The CAP Theorem

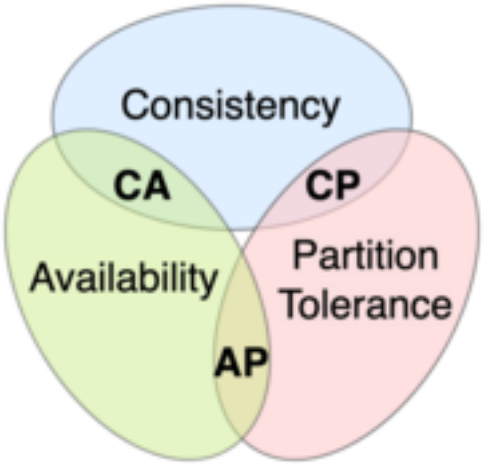
Marco Aiello

CAP Theorem

from Brewer's Conjecture

Any distributed data store can have at most two of the following properties, never all three:
Consistency, **A**vailability, **P**artition tolerance





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

History

- 1996 Birmann and Friedman “Trading consistency for availability in Distributed Systems
- 1998 Brewer gives the 2 out of three principle and poses the conjecture
- 2002 Gilbert and Lynch come with the formal proof
- 2012 PACELEC theorem

Gilbert & Lynch proof

Assumptions and definitions

- **Consistency:** any read operation that begins after a write operation completes must return that value, or the result of a later write operation
- **Availability:** every request received by a non-failing node in the system must result in a response
- **Partition Tolerance:** the network will be allowed to lose arbitrarily many messages sent from one node to another

Proof. Ab absurdum, let's assume that a PAC Algorithm exists. Consider then two nodes G_1 and G_2 holding a copy of object α with initial value v_0, then the value is altered to v_1 @G_2 and all messages between G_1 and G_2 are lost **(P)**, then a request to G_1 should **(A)** eventually return v_1 **(C)** but since there is no communication between the two nodes will return v_0, contradicting that a PAC Algorithm exists.

Gilbert & Lynch proof

what is possible

Consistent, Partion Tolerant

just delay (no availability)

Consistent, Available

just use a central controller for consistency (passive replication architecture)

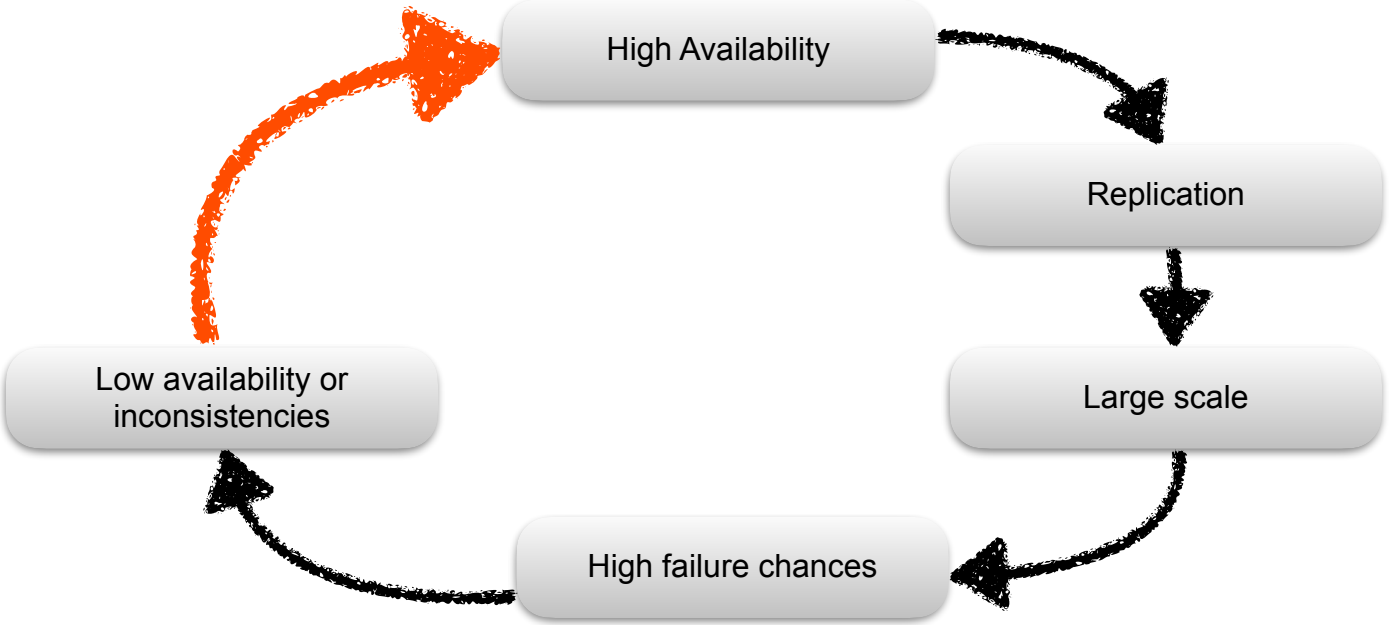
Available, Partition Tolerant

any request can be served

PAC in Practice, Abadi's extension

PACELEC

- Permanent partitions are uncommon
- In practice the tradeoff is between Consistency and Latency
- "an increase as small as 100 ms can dramatically reduce the probability that a customer will continue to interact or return in the future"
- latency above timeout (~2seconds) are the same as partitions



PACELEC Context

Abadi 2012

To understand modern DDBS design, it is important to realize the context in which these systems were built. Amazon originally designed Dynamo to serve data to the core services in its e-commerce platform (for example, the shopping cart). Facebook constructed Cassandra to power its Inbox Search feature. LinkedIn created Voldemort to handle online updates from various write-intensive features on its website. Yahoo built PNUTS to store user data that can be read or written to on every webpage view, to store listings data for Yahoo's shopping pages, and to store data to serve its social networking applications.

PACELEC Meaning

PACELC (pronounced *“pass-elk”*): if there is a partition (P), how does the system trade off availability and consistency (A and C); else (E), when the system is running normally in the absence of partitions, how does the system trade off latency (L) and consistency (C)?

Data update strategies

1. Data updates sent to all at the same time

   Active replication. With reliable TO one achieves consistency, but pays in latency. Lowering latency means loosing consistency.

2. Data updates sent to an agreed-upon location first

   Master node (passive replication). Three replication options:

   A. Synchronous, wait for all writes
   B. Asynchronous, optimistic, don’t wait (but all reads must go through the master
   C. A combination of A and C

3. Data updates sent to an arbitrary location first

   Any node can act as a master for an update

Examples

- The default versions of *Dynamo, Cassandra,* and *Riak* are **PA/EL** systems: if a partition occurs, they give up consistency for availability, and under normal operation they give up consistency for lower latency.
- Fully ACID systems such as *VoltDB/H-Store* and *Mega-store* are **PC/EC**: they refuse to give up consistency, and will pay the availability and latency costs to achieve it. *BigTable* and Google File System and related systems such as HBase are also PC/EC.
- *MongoDB* can be classified as a **PA/EC** system. In the baseline case, the system guarantees reads and writes to be consistent. However, MongoDB uses data replication option (2), and if the master node fails or is partitioned from the rest of the system, it stores all writes that have been sent to the master node but not yet replicated in a local rollback directory. Meanwhile, the rest of the system elects a new master to remain available for reads and writes
-

System's classification

from wikipedia

DDBS	P+A	P+C	E+L	E+C
BigTable/HBase		X		X
Cassandra	X		X	
Cosmos DB		X	X	
Couchbase		X	X	X
Dynamo	X		X	
DynamoDB		X	X	X
FaunaDB		X	X	X
Hazelcast IMDG	X	X	X	X
Megastore		X		X
MongoDB	X			X
MySQL Cluster		X		X
PNUTS		X	X	
PostgreSQL		X		X