CAP theorem

CAP theorem **1**
- **What CAP means** **1**
  - Consistency 2
  - Availability 2
  - Partition tolerance 2
- **What “two of three” means** **2**
  - CP database 2
  - AP database 4
  - CA database 4
- **Why** **4**
  - The proof 5
- ***References** **7**
- **Follow-ups** **7**

What CAP means

A distributed system can deliver only two of three desired characteristics: consistency, availability, and partition tolerance (the ‘C,’ ‘A’ and ‘P’ in CAP).

A distributed system is a network that stores data on more than one node (physical or virtual machines) at the same time.

Consistency

Consistency means that all clients see the same data at the same time, no matter which node they connect to. For this to happen, whenever data is written to one node, it must be instantly forwarded or replicated to all the other nodes in the system before the write is deemed ‘successful.’

Availability

Availability means that any client making a request for data gets a response, even if one or more nodes are down. Another way to state this—all working nodes in the distributed system return a valid response for any request, without exception.

Partition tolerance

A partition is a communications break within a distributed system—a lost or temporarily delayed connection between two nodes. Partition tolerance means that the cluster must continue to work despite any number of communication breakdowns between nodes in the system.

What “two of three” means

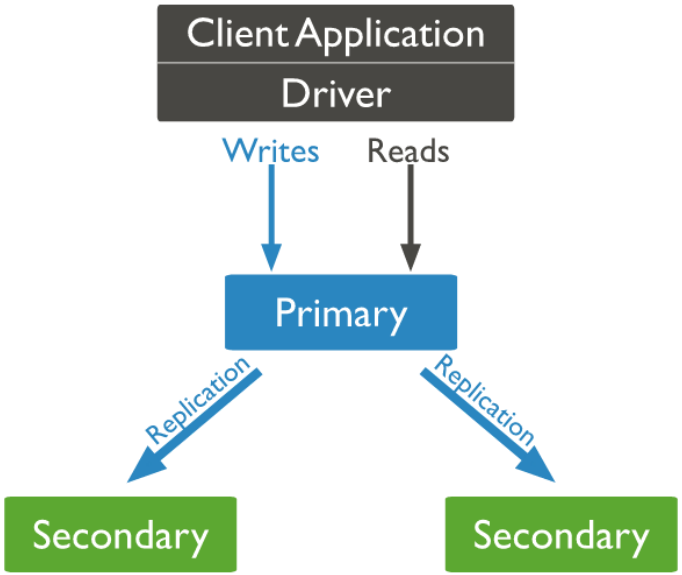
Today, NoSQL databases are classified based on the two CAP characteristics they support:

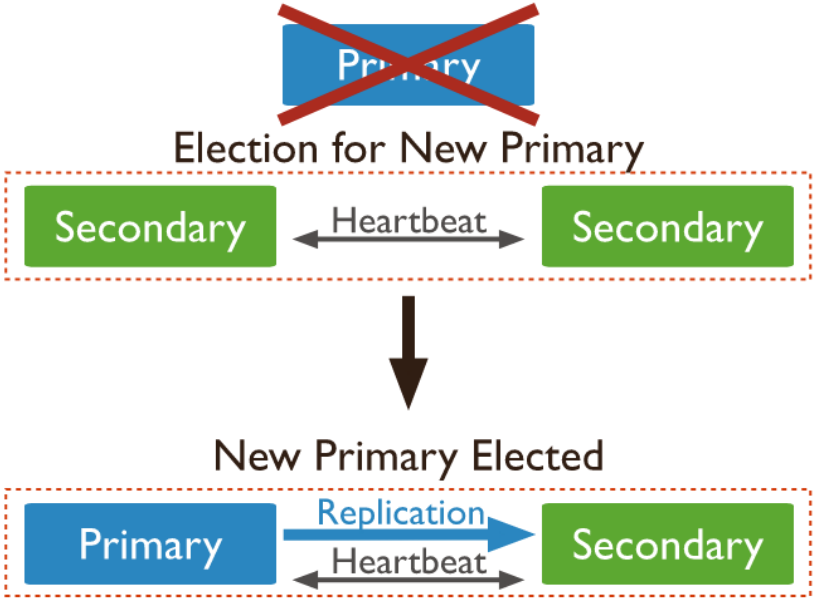
CP database

A CP database delivers consistency and partition tolerance at the expense of availability. When a partition occurs between any two nodes, the system has to shut down the non-consistent node (i.e., make it unavailable) until the partition is resolved. e.g. MongoDB.

MongoDB is a **single-master** system—each replica set (link resides outside IBM) can have only one primary node that **receives all the write** operations. All other nodes in the same replica set are secondary nodes that replicate the primary node's operation log and apply it to their own data set.

When the primary node becomes unavailable, the secondary node with the most recent operation log will be **elected as the new primary node**. Once all the other secondary nodes catch up with the new master, the cluster becomes available again. As clients **can't make any write requests** during this interval, the data remains consistent across the entire network.





AP database

An AP database delivers availability and partition tolerance at the expense of consistency. When a partition occurs, all nodes remain available but those at the wrong end of a partition might return an older version of data than others. (When the partition is resolved, the AP databases typically resync the nodes to repair all inconsistencies in the system.). e.g. Cassandra.

Unlike MongoDB, Cassandra has a **masterless(peer-to-peer)** architecture, and as a result, it has multiple points of failure, rather than a single one. Cassandra provides **eventual consistency** by allowing clients to write to any nodes at any time and reconciling inconsistencies as quickly as possible.

CA database

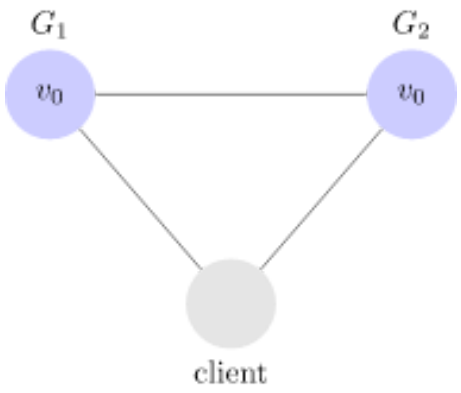
A CA database delivers consistency and availability across all nodes. It can't do this if there is a partition between any two nodes in the system, however, and therefore can't deliver fault tolerance.

In a distributed system, partitions can't be avoided. So, while we can discuss a CA distributed database in theory, for all practical purposes, **a CA distributed database can't exist**. However, this doesn't mean you can't have **a CA database for your distributed application** if you need one. Many **relational databases**, such as **PostgreSQL**, deliver consistency and availability and can be deployed to multiple nodes using replication.

Why

A distributed system looks like

Two servers, G1 and G2, keep track of the same variable, v, whose value is initially v0.
G1 and G2 communicate with each other and can also communicate with external clients.

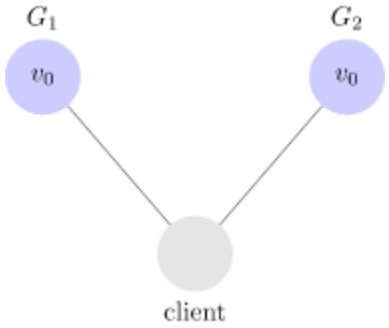


A client can request to write and read from any server.

The proof

Assume for contradiction that there does exist a system that is consistent, available, and partition tolerant.

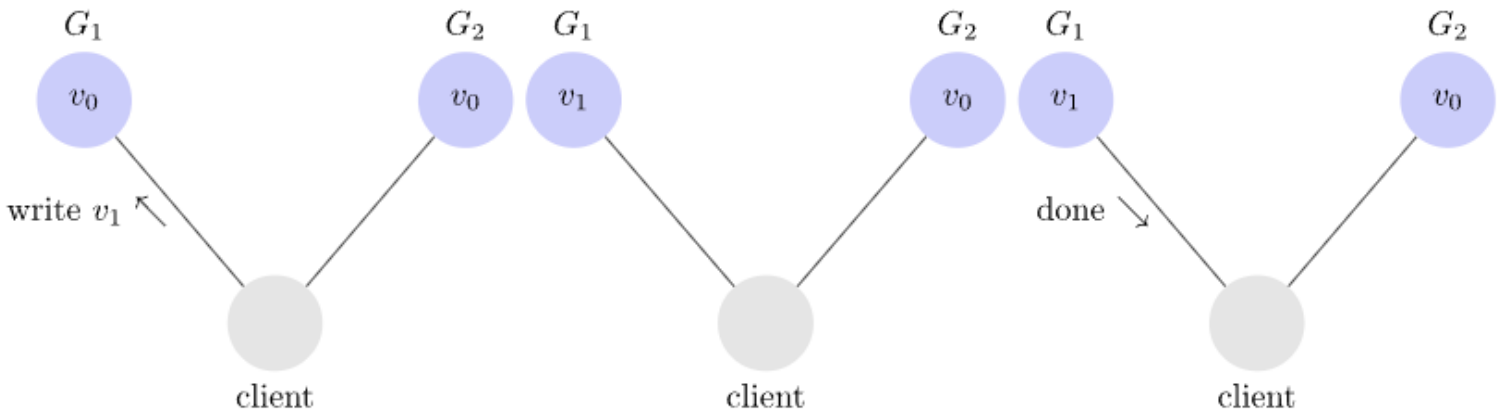
Let's partition our system:



Then have our client request that v1 be written to G1:

available → G1 respond

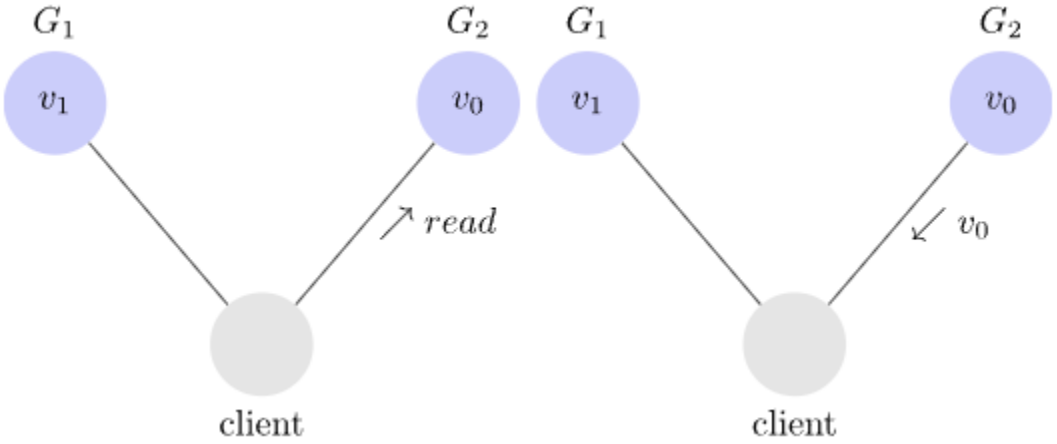
partitioned → G1 cannot replicate its data to G2



Finally, have our client issue a read request to G2:

available → G2 respond

partitioned → G2 cannot update its value from G1, returns v0 → inconsistent → NO SUCH SYSTEM EXISTS



*References

https://www.ibm.com/cloud/learn/cap-theorem
https://mwhittaker.github.io/blog/an_illustrated_proof_of_the_cap_theorem/

Follow-ups

Eventual Consistency
Master-master vs master-slave database architecture
NoSQL database