Blog

An Illustrated Proof of the CAP Theorem

The CAP Theorem is a fundamental theorem in distributed systems that states any distributed system can have at most two of the following three properties.

- **C**onsistency
- **A**vailability
- **P**artition tolerance

This guide will summarize Gilbert and Lynch's specification and proof of the CAP Theorem with pictures!

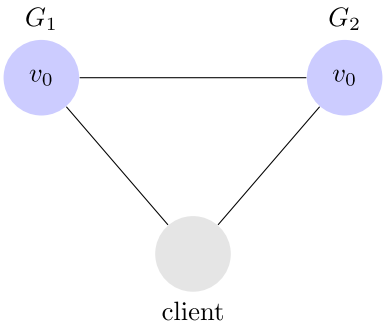
What is the CAP Theorem?

The CAP theorem states that a distributed system cannot simultaneously be consistent, available, and partition tolerant. Sounds simple enough, but what does it mean to be consistent? available? partition tolerant? Heck, what exactly do you even mean by a distributed system?

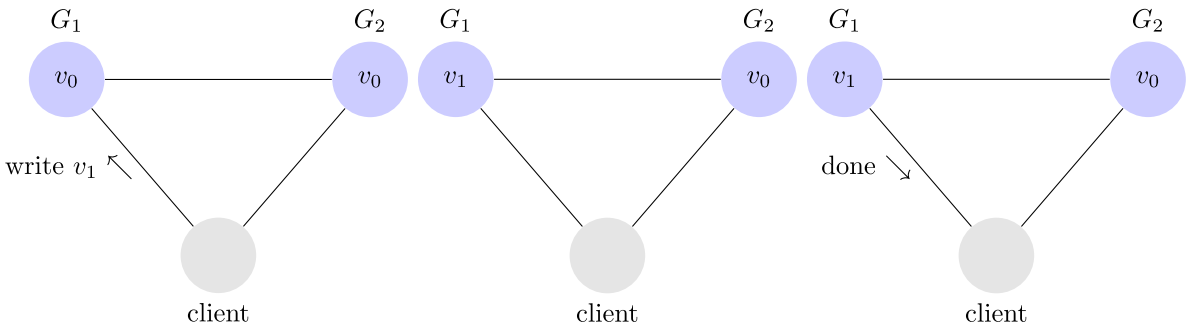
In this section, we'll introduce a simple distributed system and explain what it means for that system to be available, consistent, and partition tolerant. For a formal description of the system and the three properties, please refer to Gilbert and Lynch's paper.

A Distributed System

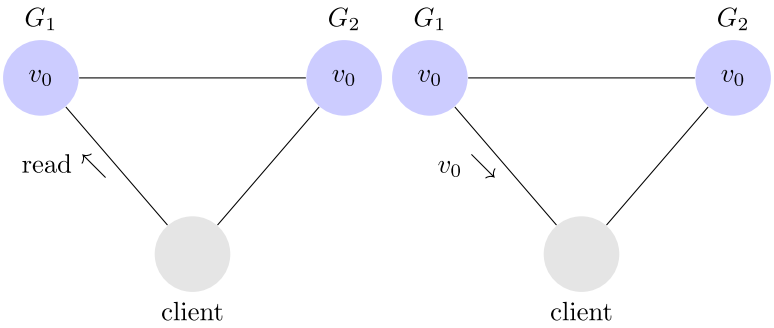
Let's consider a very simple distributed system. Our system is composed of two servers, G_1 and G_2. Both of these servers are keeping track of the same variable, v, whose value is initially v_0. G_1 and G_2 can communicate with each other and can also communicate with external clients. Here's what our system looks like.



A client can request to write and read from any server. When a server receives a request, it performs any computations it wants and then responds to the client. For example, here is what a write looks like.



And here is what a read looks like.



Now that we've gotten our system established, let's go over what it means for the system to be consistent, available, and partition tolerant.

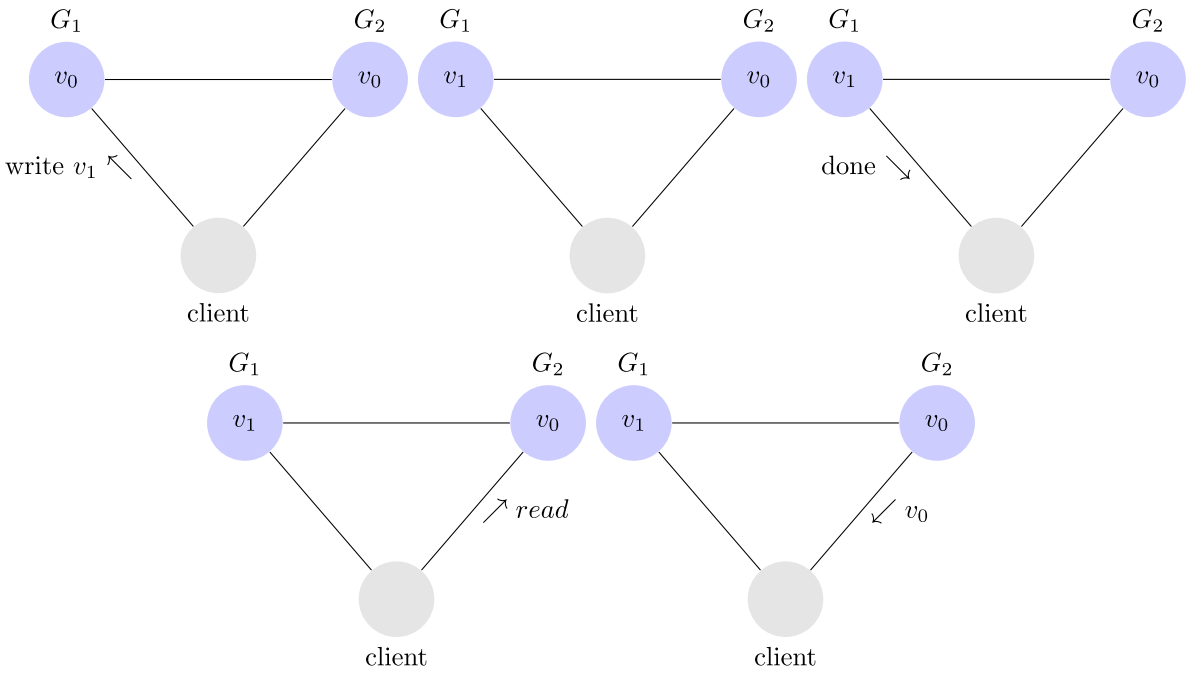
Consistency

Here's how Gilbert and Lynch describe consistency.

> any read operation that begins after a write operation completes must return that value, or the result of a later write operation

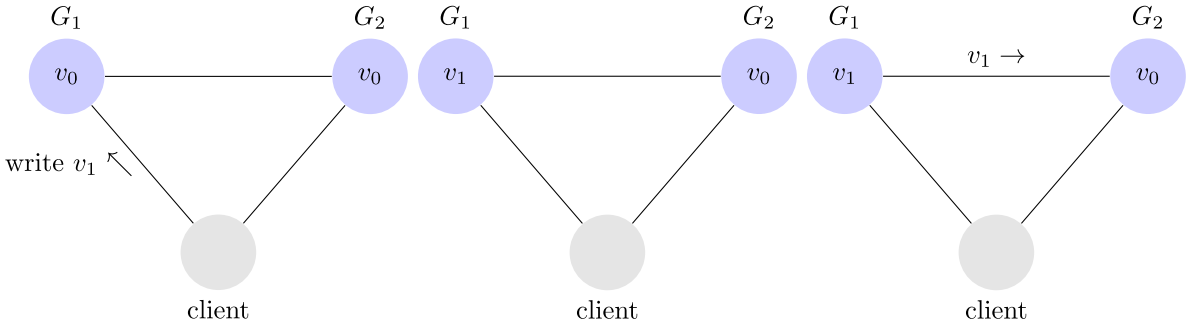
In a consistent system, once a client writes a value to any server and gets a response, it expects to get that value (or a fresher value) back from any server it reads from.

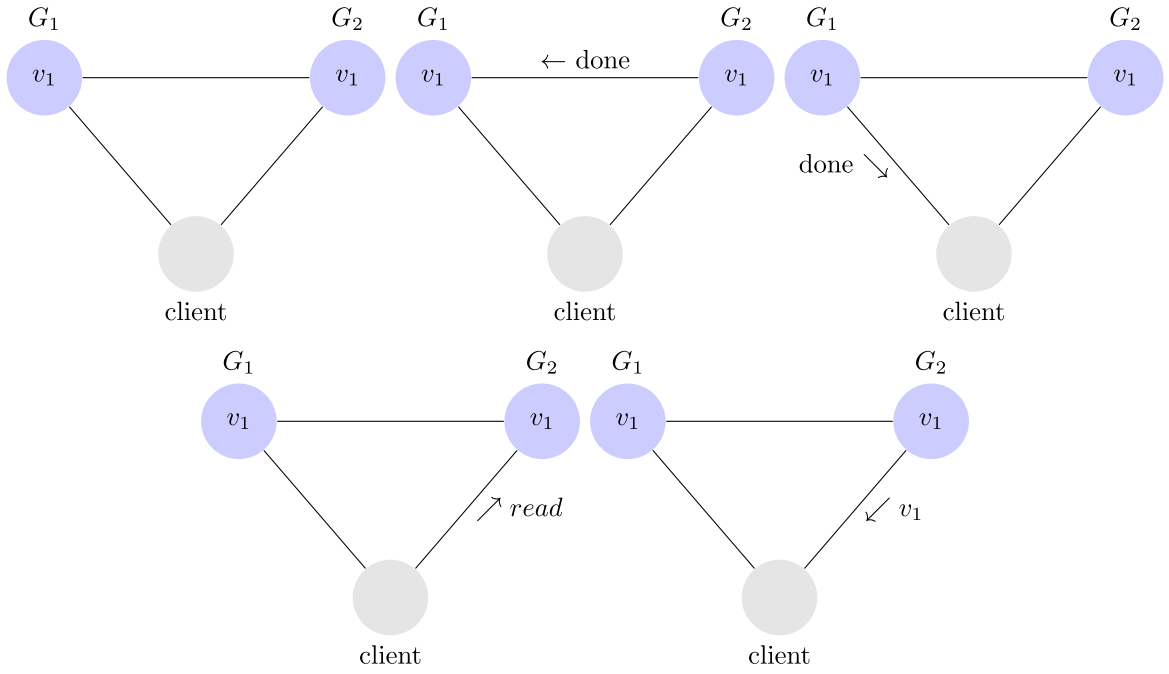
Here is an example of an **inconsistent** system.



Our client writes v_1 to G_1 and G_1 acknowledges, but when it reads from G_2, it gets stale data: v_0.

On the other hand, here is an example of a **consistent** system.





In this system, G_1 replicates its value to G_2 before sending an acknowledgement to the client. Thus, when the client reads from G_2, it gets the most up to date value of v : v_1.

Availability

Here's how Gilbert and Lynch describe availability.

> every request received by a non-failing node in the system must result in a response

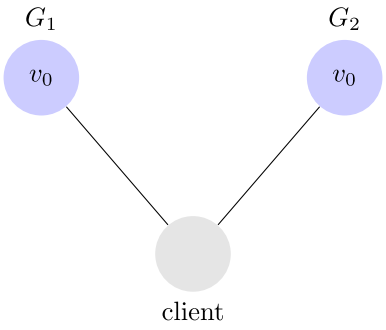
In an available system, if our client sends a request to a server and the server has not crashed, then the server must eventually respond to the client. The server is not allowed to ignore the client's requests.

Partition Tolerance

Here's how Gilbert and Lynch describe partitions.

> the network will be allowed to lose arbitrarily many messages sent from one node to another

This means that any messages G_1 and G_2 send to one another can be dropped. If all the messages were being dropped, then our system would look like this.

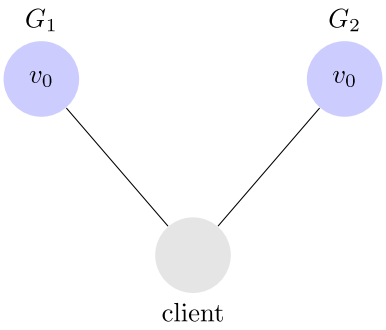


Our system has to be able to function correctly despite arbitrary network partitions in order to be partition tolerant.

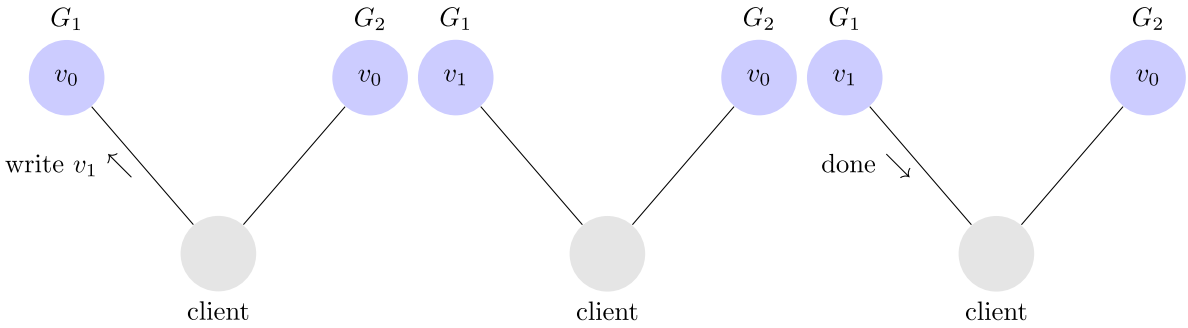
The Proof

Now that we've acquainted ourselves with the notion of consistency, availability, and partition tolerance, we can prove that a system cannot simultaneously have all three.

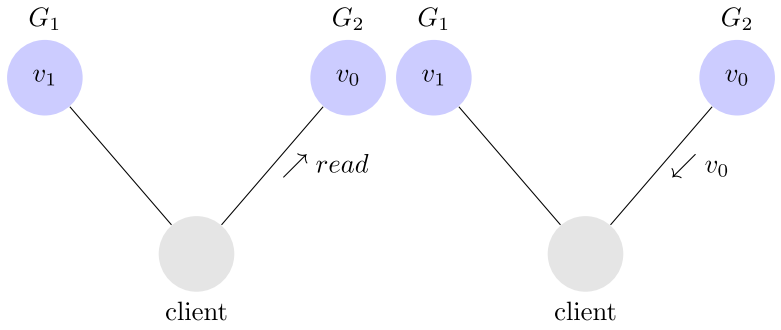
Assume for contradiction that there does exist a system that is consistent, available, and partition tolerant. The first thing we do is partition our system. It looks like this.



Next, we have our client request that v_1 be written to G_1. Since our system is available, G_1 must respond. Since the network is partitioned, however, G_1 cannot replicate its data to G_2. Gilbert and Lynch call this phase of execution α_1.



Next, we have our client issue a read request to G_2. Again, since our system is available, G_2 must respond. And since the network is partitioned, G_2 cannot update its value from G_1. It returns v_0. Gilbert and Lynch call this phase of execution α_2.



G_2 returns v_0 to our client after the client had already written v_1 to G_1. This is inconsistent.

We assumed a consistent, available, partition tolerant system existed, but we just showed that there exists an execution for any such system in which the system acts inconsistently. Thus, no such system exists.