CAP Theorem: Revisited

Written by [Robert Greiner (/about)](/about) on August 14, 2014
4 minute read

Note: *Close to two months ago, I wrote a [blog post explaining the CAP Theorem (http://robertgreiner.com/2014/06/cap-theorem-explained/)](http://robertgreiner.com/2014/06/cap-theorem-explained/). Since publishing, I've come to realize that my thinking on the subject was quite outdated and is no longer applicable to the real world. I've attempted to make up for that in this post.*

In today's technical landscape, we are witnessing a strong and increasing desire to scale systems *out* when additional resources (compute, storage, etc.) are needed to successfully complete workloads in a reasonable time frame. This is accomplished through adding additional commodity hardware to a system to handle the increased load. As a result of this scaling strategy, an additional penalty of complexity is incurred in the system. This is where the CAP theorem comes into play.

The CAP Theorem states that, in a distributed system (a collection of interconnected nodes that share data.), you can only have two out of the following three guarantees across a write/read pair: Consistency, Availability, and Partition Tolerance - one of them must be sacrificed. However, as you will see below, you don't have as many options here as you might think.

- **Consistency** - A read is guaranteed to return the most recent write for a given client.
- **Availability** - A non-failing node will return a reasonable response within a reasonable amount

of time (no error or timeout).

- **Partition Tolerance** - The system will continue to function when network partitions occur.

Before moving further, we need to set one thing straight. Object Oriented Programming != Network Programming! There are assumptions that we take for granted when building applications that share memory, which break down as soon as nodes are split across space and time.

One such *fallacy of distributed computing* (http://en.wikipedia.org/wiki/Fallacies_of_Distributed_Computing) is that networks are reliable. They aren't. Networks and parts of networks go down frequently and unexpectedly. Network failures *happen to your system* and you don't get to choose when they occur.

Given that networks aren't completely reliable, you must tolerate partitions in a distributed system, period. Fortunately, though, you get to choose what to do when a partition does occur. According to the CAP theorem, this means we are left with two options: Consistency and Availability.

- **CP** - Consistency/Partition Tolerance - Wait for a response from the partitioned node which could result in a timeout error. The system can also choose to return an error, depending on the scenario you desire. Choose Consistency over Availability when your business requirements dictate atomic reads and writes.

(http://robertgreiner.com/uploads/images/2014/CAP-CP-full.png)
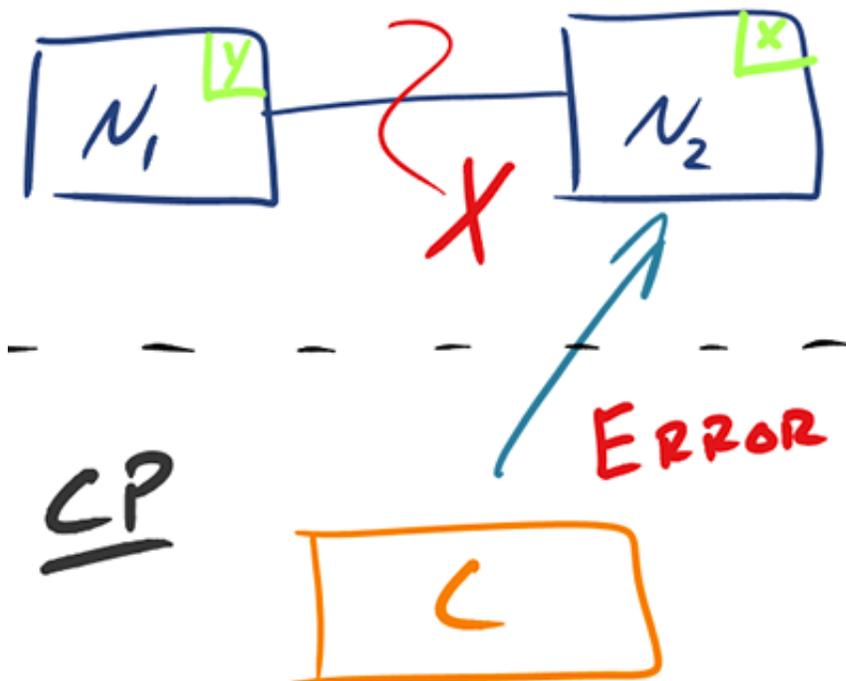
- **AP** - Availability/Partition Tolerance - Return the most recent version of the data you have, which could be stale. This system state will also accept writes that can be processed later when the

partition is resolved. Choose Availability over Consistency when your business requirements allow for some flexibility around when the data in the system synchronizes. Availability is also a compelling option when the system needs to continue to function in spite of external errors (shopping carts, etc.)

(http://robertgreiner.com/uploads/images/2014/CAP-AP-full.png)
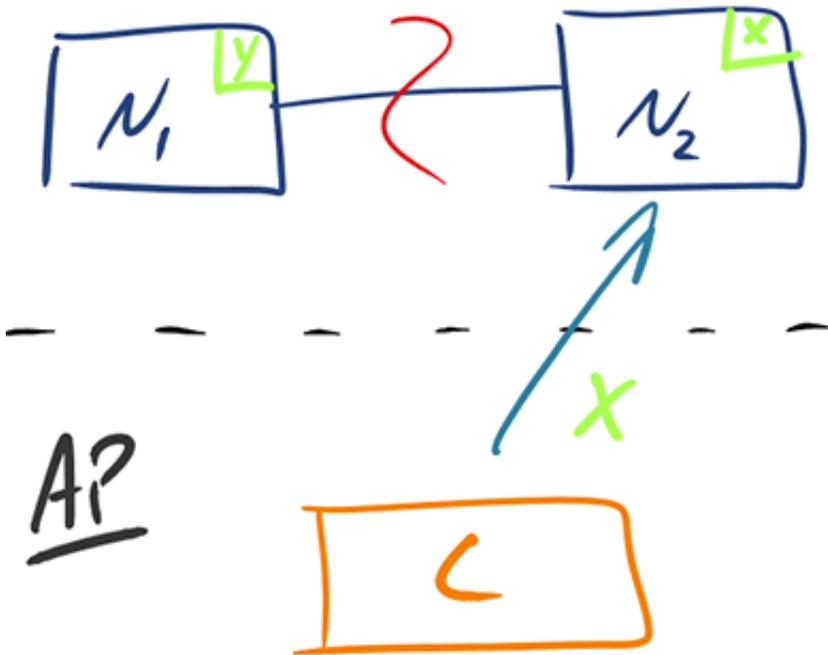
The decision between Consistency and Availability is a *software trade off*. You can choose what to do in the face of a network partition - the control is in your hands. Network outages, both temporary and permanent, are a fact of life and occur whether you want them to or not - this exists outside of your software.

Building distributed systems provide many advantages, but also adds complexity. Understanding the trade-offs available to you in the face of network errors, and choosing the right path is vital to the success of your application. Failing to get this right from the beginning could doom your application to failure before your first deployment.

Popular Posts

- The Platinum Rule (http://robertgreiner.com/2014/04/the-platinum-rule/)
- Remote (http://robertgreiner.com/2013/11/remote-office-not-required-review)
- Sync to Paper (http://robertgreiner.com/2013/03/sync-to-paper)
- 3 Books Every Software Developer Should Have Already Read (http://robertgreiner.com/2013/09/software-developer-book-recommendations)

- Work Yourself Out of a Job (http://robertgreiner.com/2012/09/work-yourself-out-of-a-job/)

Recent Posts

- Alternative to Arguing (/2016/04/alternative-to-arguing/)
- How To Drastically Reduce Your PowerPoint File Size (/2016/01/how-to-reduce-powerpoint-file-size/)
- Wooden Blocks (/2016/01/people-arent-wooden-blocks/)
- DiSC (/2016/01/DiSC/)
- Don't Be This Type of Leader (/2015/11/leadership-worst-practices/)

What do you think?

18 Comments   **Robert Greiner**

🔴 1   **Login** ⌄
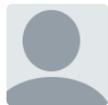
♥ **Recommend** 9   ➦ **Share**

Sort by Oldest ⌄

👤 | Join the discussion…

👤 **Chris** · a year ago
Nice and succinct
⌃ | ⌄ · Reply · Share ›

👤 **Mike** · a year ago
See https://www.linkedin.com/pulse...
⌃ | ⌄ · Reply · Share ›

Robert Mod ➜ Mike · a year ago
Thanks, Mike. I appreciate the heads-up. I reported the post and it looks like LinkedIn removed it.
⌃ | ⌄ · Reply · Share ›

👤 **dee** ➜ Robert · a year ago
they didn't i just saw it
⌃ | ⌄ · Reply · Share ›

Robert Mod ➜ dee · a year ago
Ah, thanks for letting me know.

He actually added attribution at the bottom, so I'm fine with it.

1 ∧ | ∨ • Reply • Share ›

Chandra Divi · a year ago

Thanks, the concept was explained well.

∧ | ∨ • Reply • Share ›

Stefano M. · a year ago

The concept is very clearly explained! The picture explaining it is a masterpiece.
Just a point: with the concept of "quorum" (the majority of nodes still see each other, e.g. 2 out of three) I think that a system can survive partition still guaranteeing consistency and availability (of the quorum nodes).

∧ | ∨ • Reply • Share ›

Robert Mod → Stefano M. · a year ago

Thanks for the comment. Good point as well.

∧ | ∨ • Reply • Share ›

Umesh → Stefano M. · 6 months ago

But this is eventual consistency, isn't it?

∧ | ∨ • Reply • Share ›

Stefano M. → Umesh · 6 months ago

No. Quorum enables strong consistency without reconciliation

∧ | ∨ • Reply • Share ›

Alex Worden · 10 months ago

Brilliant description Robert! It's very rare to find someone with deep understanding that can also explain this clearly. How did you make the whiteboard-like diagrams?

∧ | ∨ • Reply • Share ›

Robert Mod → Alex Worden · 10 months ago

Alex - thanks for the kind words. This means a lot to me.

I drew those diagrams in One Note on my Surface Pro 2. It actually worked out really well and I've done something similar for work a few times sense.

∧ | ∨ • Reply • Share ›

Umesh Wankhede · 6 months ago

I don't think many in the field can explain this clearly. Even wikipedia doesn't have this clarity. And illustrations are just marvellous!

∧ | ∨ • Reply • Share ›

Robert Mod → Umesh Wankhede · 6 months ago

Thanks, Umesh. I apprecaite it. Glad I could help.

Michael · 5 months ago

This is the best explanation of CAP theorem.
Thank you very much!

Shivendra · 3 months ago

Nice, clear post :)

Aishwarya Parasuram · 2 months ago

I've never heard of the CAP theorem before (kind of new to Distributed Systems) but this explained it so well :) Thanks!

Николай Лаптев · 2 months ago

Everything was fine and logically explained step by step until you started describing CP and AP. For persons new to this field (I suppose they are a target audience) it's not clear how you jump from missing Availability to waiting for a response from the partitioned node etc. I suppose other parts of the article after CP and AP took much less time to write :-)