CAP or no CAP? Understanding when the CAP theorem applies and what it means.

June 21, 2022 · 16 min read

Alex DeBrie
Founder, DeBrie Advisory

The CAP theorem might be the most misunderstood idea in computer science. If you're looking to understand the CAP theorem through a series of examples, you're in the right place.

I recently wrote a post on the various notions of 'consistency' in databases and distributed systems. Part of that discussion focused on consistency as the 'C' in the CAP theorem and the implications of the theorem. In conversations on Twitter and in my DMs, it's clear there's still some confusion about CAP, when it applies, and what its implications are.

In this post, we'll try to clear up that confusion.

I'm an old man, but I have two brothers-in-law in their early 20s. Because of this, I have access to Gen Z slang that is out of reach of most Millenials like me. For example, I learned from them the expression "no cap", which, according to Dictionary.com, means "'no lie' or 'for real,' often used to emphasize someone is not exaggerating about something hard to believe."

I'm going to co-opt this term with a game I call "CAP or no CAP?" where we walk through various distributed system failure scenarios and discuss if the CAP theorem applies and what its implications are. Additionally, even if the

CAP theorem does not apply, we'll talk about any key lessons you should learn from the scenario.

By the end of this post, you'll understand the CAP theorem better than your friends -- no cap!

Background

I want to start with a little background on the CAP theorem, what it implies, and what it doesn't imply. This won't be super deep, so be sure to check out other resources on the CAP theorem if you're interested.

First, the CAP theorem is about replication. It only addresses the behavior of a distributed system in which multiple nodes share the same piece of data. This could be a relational database with replicas, a MongoDB replica set, or any number of similar database configurations.

Second, the CAP theorem is about failure. Even more, it's about a specific type of failure -- a network failure between the nodes that eliminates the ability to communicate updates.

This network failure is called a 'partition' (the "P" in CAP). In this failure scenario, the CAP theorem states that you have to choose between two properties:

- *Availability*, defined as each non-failing node being able to successfully respond to a request; or
- *Consistency*, defined as each read receiving the most recent write for that piece of data.

Because of the way these terms are defined, it is basically impossible for you to beat the CAP theorem or for it to be disproven. If you have a non-failing node that is unable to communicate with other nodes over the network, it cannot guarantee a successful response with the most up-to-date information (barring ESP or some other phenomenon).

CAP's absolutes are useful, but limited

The definitional absolutes of CAP are useful. They help to frame the outer edges of the debate and show the key tradeoffs. Further, they do it in a way that doesn't require math. Math is great, and we can use it to explore the exact implementation we want to make, but a high-level conceptual model like CAP makes it easy to grok the contours of the debate and makes it accessible to more folks.

That said, these absolutes can make it so CAP doesn't directly apply to your situation. I think of three places where CAP is less applicable than you might think.

First, the specific type of failure (network partition) is far from the only type of failure, and it can be a more rare type of failure than others.

Second, the 'Consistency' in CAP seems to require a strong, linearizable consistency. Application developers may choose a weaker version of consistency for many reasons, including in times where there isn't a network partition. I discussed this a bit in my other post with the concept of PACELC, and Werner Vogels has a classic post on the concept of eventual consistency.

Finally, definition of "Availability" in CAP differs from the standard usage of "availability" when talking about services. The "big-A Availability" in CAP talks about any non-failing node (even if isolated) responding successfully, while the "little-a availability" is generally a percentage of successful responses from a system over time.

Marc Brooker has done some really great write ups on this final concept, including on the differences on Availability and availability as well as the concepts of harvest and yield for availability.

Most of the questions I get around CAP seem to be architectures and strategies to improve little-a availability in the event of failures and during times that don't fall under CAP. Because this confusion is so common, I'll discuss the impacts on little-a availability in the scenarios below.

CAP or no CAP Scenarios

With the background out of the way, let's get into our game of "CAP or no CAP?"

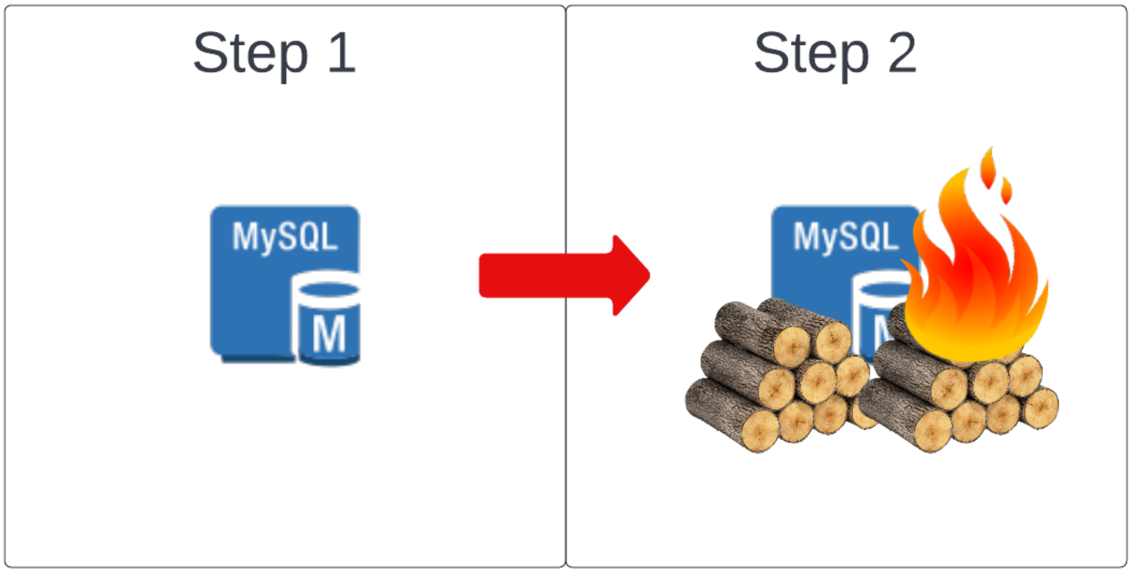
I'll show a series of system scenarios and failure modes and discuss them along three axes:

- Does the CAP theorem apply?
- If yes, which choice did the system make -- AP or CP?
- What is the impact of this failure on little-a availability?

Let's start the game!

Scenario 1: Single-node RDBMS

We'll start off with an easy one. Your system uses a single-node instance of a relational database. You configured a cron job to periodically delete logs, but you configure it for the wrong directory. Your disk fills up with logs, and your database goes down.



So many logs.

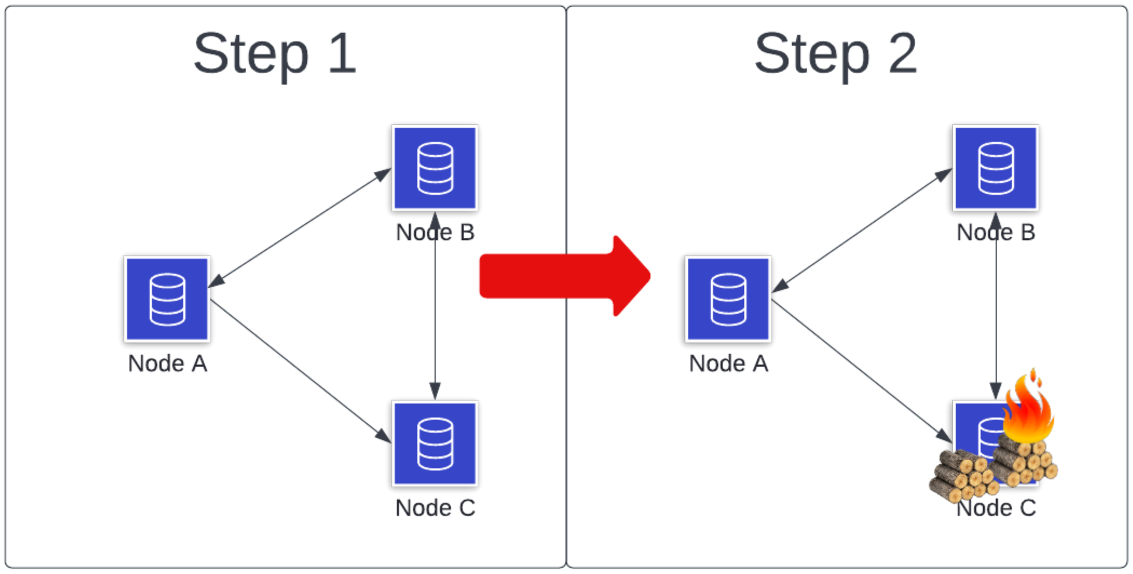
Let's run through our analysis:

- **CAP or no CAP?** *No CAP!* Remember the CAP theorem is about replication, but this is a single-node system. CAP doesn't apply.
- **AP or CP?** Not applicable -- not in CAP-world
- **Impact on little-a availability?** 💩 Your single database instance went down. You're not going to have a good time.

Ok, so CAP requires multiple nodes sharing the same data. Let's update our example accordingly.

Scenario 2: Three-node system, one node fails

In this scenario, we'll use a similar setup but add some replication. We now have a three-node cluster. However, our cron job is still configured for the wrong directory. One of our three nodes goes down as it runs out of disk space.



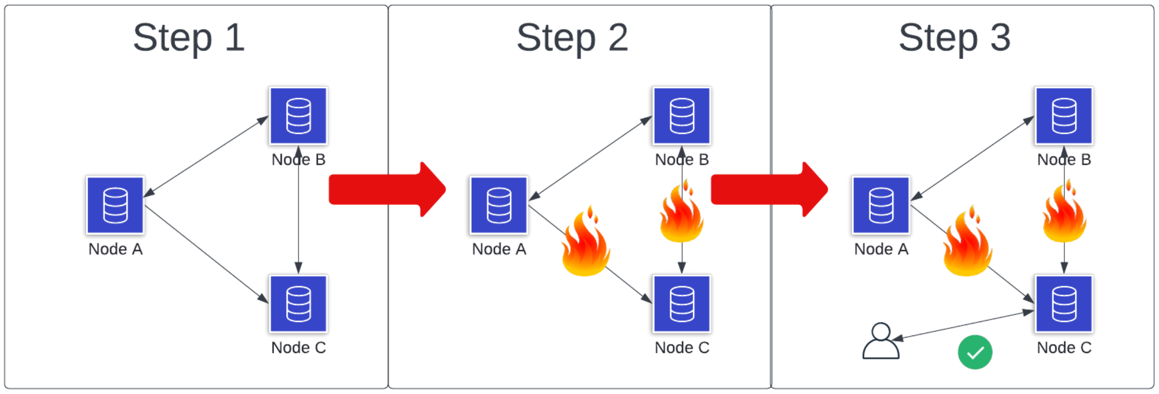
Are we in CAP land now? Let's do the analysis:

- **CAP or no CAP?** *No CAP!* We have replication now, but remember that CAP is about a specific type of failure -- network partitions. Our network is fine, but one of our nodes died.
- **AP or CP?** Not applicable -- not in CAP-world
- **Impact on little-a availability?** This will have higher availability than Scenario 1. A single node went down, but now we have two others to pick up the load. Note that this could reduce availability from our ideal, three-node state, depending on the amount of excess capacity we had in our nodes.

Scenario 3: Three-node system, network partition, still responding

Ok, let's fix our last problem. Same three-node setup as before, but now one of our nodes cannot talk to the other nodes due to a network partition.

However, the isolated node is still operating and can receive requests from clients. When it does, our isolated node gives them whatever data it has locally.



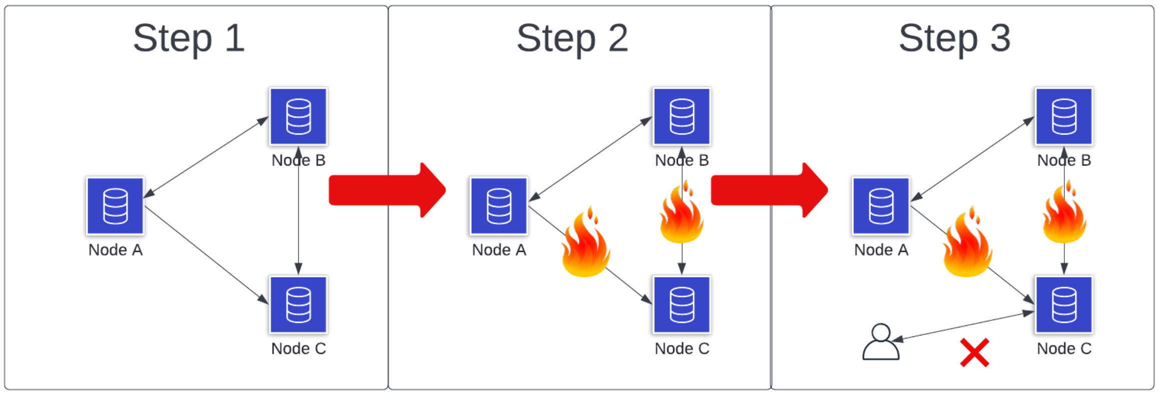
How does this stack up?

- **CAP or no CAP?** *CAP!* This is a replicated system with a network partition. We are right in the sweet spot contemplated by CAP.
- **AP or CP?** AP. The system chose to give a successful response from the isolated node (choosing 'Availability'), even if it potentially does not have the latest write from other nodes (sacrificing 'Consistency').
- **Impact on little-a availability?** Equal or higher availability than Scenario 2, as all three nodes are available for responses. If Scenario 2 had to fail some requests because the two nodes were overloaded, Scenario 3 would increase availability by keeping additional capacity online.

Scenario 4: Three-node system, network partition, not responding

Let's do the same one as before but make a different choice.

Again, we have our three-node setup, and a network partition occurs. This time, we do not allow the isolated node to respond to client requests.



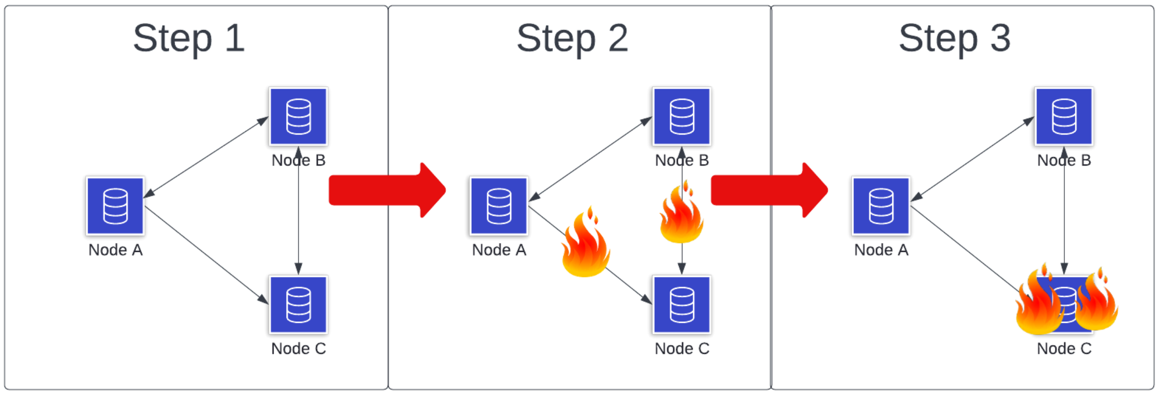
How does this change things?

- **CAP or no CAP?** *CAP!* Same thing as last scenario -- replication with a network partition.
- **AP or CP?** CP. This time, we're deciding we don't want to give up Consistency by giving a potentially out-of-date response, so we're sacrificing big-A Availability instead.
- **Impact on little-a availability?** Technically, I think this would be lower than Scenario 2 or 3. Client requests that reach our isolated node will receive an error, which reduces our availability numbers.

This last point is interesting, so let's tweak the scenario slightly...

Scenario 5: Self-immolation

Let's start with the same scenario as above. In this one, once our isolated node realizes it is cut off from the rest of the cluster, it shuts itself down to avoid taking any client requests.



You could get a similar effect from a service discovery system that realizes via health checks that the node is down. The key point is that the isolated node is no longer allowed to receive requests.

What's the impact on our categories?

- **CAP or no CAP?** *No CAP.* CAP is only about network partitions, and the definition of Availability refers to *non-failing nodes*. If our node has failed itself, it moves out of CAP-land.
- **AP or CP?** N/A.
- **Impact on little-a availability?** Better than Scenario 4, as requests aren't getting to an isolated node but failing due to the system's choices. This will be similar to Scenario 2.

Like in Scenario 2, the key point is that you're only truly in CAP-land when there's a network partition.

We've done most of the basic scenarios. Time to get a little bit wacky.

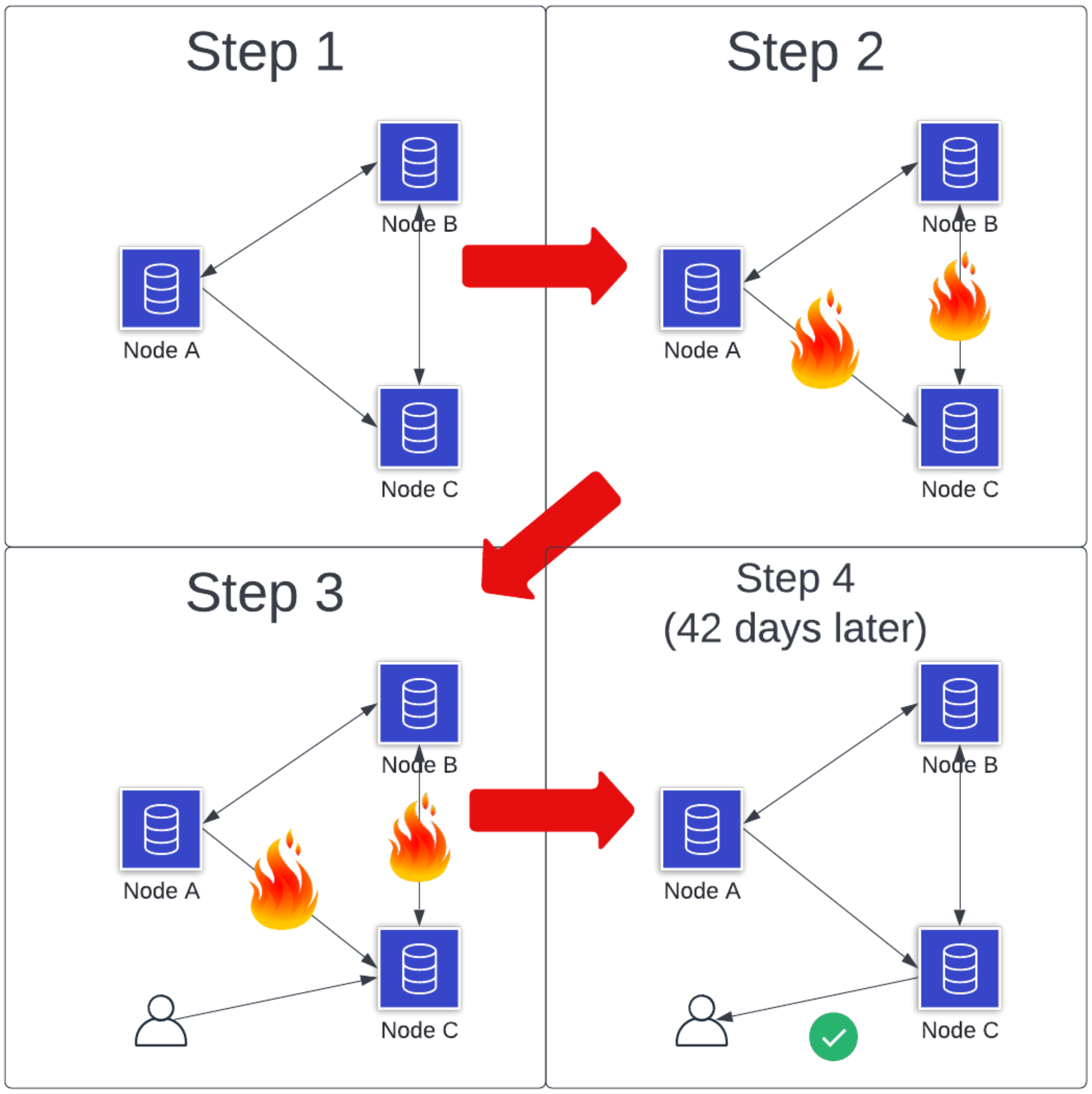
Scenario 6: The Long Goodbye

Let's return to our network partition example. We have a three-node cluster, and there's a network partition to isolate one node. That node receives a read request on a piece of data. Rather than responding immediately, the node waits.

And waits.

And waits.

Forty-two days later, the network partition is fixed. The node receives all updates that happened during the partition. It then responds to the client with a consistent view of the requested data.



What is going on here?

It's hard to answer this one! The Availability aspect of CAP doesn't have a time component as part of the request. While most clients would prefer a speedier response, I suppose we *could* wait until the partition is resolved.

You could arguably classify this in two ways:

- *CAP doesn't apply.* CAP is only talking about network partitions, and we didn't have a request that completed during the partition. Thus, no CAP.
- *We've defeated CAP!* Coda Hale has a classic post on how you can't sacrifice partition tolerance in CAP. That is, you can't say "I choose CA, because we won't have partitions." But in this case, we do sacrifice partition tolerance by just waiting until it goes away (or maybe we have clients that are extremely tolerant of partitions? I don't know. This is weird.).

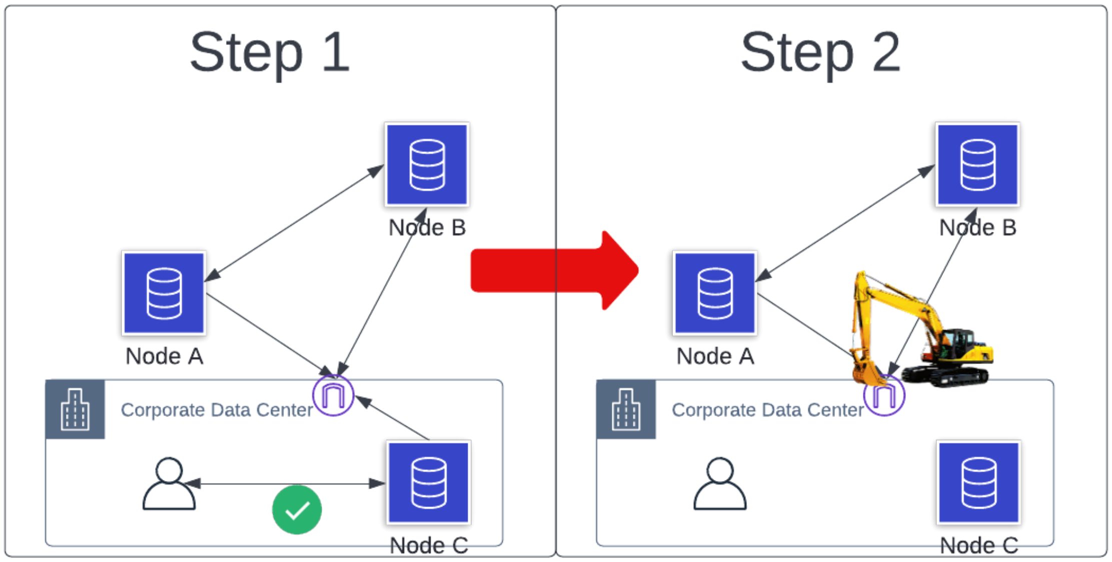
My sense is that the second option is technically correct. I like to frame CAP as "assuming a partition..." to make the tradeoffs clear, but I don't think it's necessarily required.

But regardless of the technical correctness, *surely this choice is insane*. Our clients want a response quickly, and request timeouts are likely on the order of seconds rather than fortnights. This practice would reduce the little-a availability that we care about, even if we're technically not violating the big-A availability of CAP.

Scenario 7: If a tree falls in the forest ...

In our next scenario, we'll again imagine a network partition for one of our nodes. Perhaps a backhoe cuts a cable to a datacenter holding our node.

Importantly, the only clients that would talk to this node are also in the datacenter. Because these clients are also having issues, our isolated node never receives a request and thus never has to choose between Consistency and Availability.



What happens here? If a node is isolated in a datacenter but no one is around to request it, does it make a dent in CAP?

- **CAP or no CAP?** *CAP*. We have a network partition in a replicated system.
- **AP or CP?** Irrelevant to this case. It's likely our node has made a choice as to what it would do (and either choice is valid), but we didn't actually specify here.
- **Impact on little-a availability?** None! Our system should be available as it once was. Now, *the system that calls us* is probably less available (or something up the line is), but technically our system is fine.

The key takeaway here is one that Eric Brewer, the original creator of the CAP theorem, makes in a 2017 whitepaper on Google Spanner and the CAP theorem.

One of the points Eric makes in the paper is that some of your network outages can also take down your clients (a term he calls "fate-sharing"). In that situation, your system isn't losing availability like it otherwise would.

The broader point I take from this is that you should weigh the benefit of additional availability work against the additional cost. This is like the "us-east-1" problem -- if you run your application in us-east-1, you are succeptible to availability hits if an outage hits that region. However, an outage in us-east-1 will also be a world-impacting event where *lots* of sites will be down. If your site isn't critical in those scenarios and can sort of fade into the background of the dumpster fire, it may not be worth adding the complexity of multi-region to increase your availability.

Eric Brewer's paper on Google Spanner and the CAP theorem brings me to the next scenario ...

Scenario 8: Consistency and high availability? 🤯

In the aforementioned paper on Google Spanner and the CAP theorem, Eric notes that Spanner claims to be consistent and highly available. But what gives? We already learned that this is impossible! And Eric of all people should know that, given he created the CAP theorem in the first place!

The nuance here goes back to the difference between big-A Availability (as in CAP) and little-a availability (as in percentage of successful requests).

When Spanner claims to be consistent, it is saying it will be a CP system in the event of a network partition. However, in claiming to be highly available, it is saying that an outage, of *any sort*, not just from the network, is rare.

Part of that explanation is due to some fancy TrueTime technology that Google uses, and part of it is just continual improvement in practices to reduce outages. I discussed similar advances and techniques in Amazon Aurora and DynamoDB to reduce recovery time and limit outages.

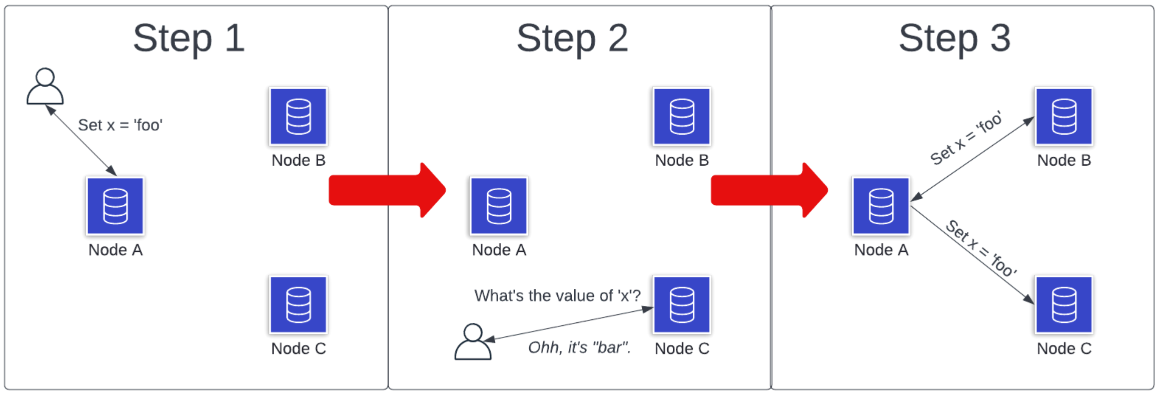
In these situations, we haven't disproved CAP. It's still making a choice. It's just saying that the tradeoff is increasingly unlikely, and that it might be better to focus more on other things because of CAP's narrow implications.

Scenario 9: Eventual consistency

For the last scenario, let's change gears a bit. Since the release of the Dynamo Paper in 2007, we've seen an increase in NoSQL databases like DynamoDB, MongoDB, Cassandra, and others.

Many of these NoSQL databases offer eventually consistent reads, meaning that a read from one node may not reflect all updates to that data item from other nodes.

Let's explore this with our typical three-node cluster. A write operation comes to one node, and the write is acknowledged and returned to the client. Subsequently, a client requests the same data item from another node *before it has been replicated from the first node*. Accordingly, the client receives a stale response from the second node.



In our image above, the client tells Node A to set the value of 'x' to "foo". Then, a second client asks Node C for the value of 'x' and receives the stale value of 'bar'. Finally, Node A replicates the new value of 'x' to Nodes B and C.

How does our CAP analysis hold up here?

- **CAP or no CAP?** *No CAP*. There's no network partition here, so we're not in CAP-land.
- **AP or CP?** N/A because we're not in CAP.
- **Impact on little-a availability?** Probably higher! By loosening the constraints around replication, we can respond quicker and be more resilient to node failures.

The eventual consistency example is an interesting one, and one that shows the general limitations of the CAP theorem. It's so focused to a particular situation that it doesn't address other, more common tradeoffs you need to consider.

This is where PACELC can be helpful for framing this decision, as it helps you to consider tradeoffs in the non-failure scenario.

Conclusion

In this post, we took a deeper look at the CAP theorem. First, we learned the CAP theorem's key contours plus some issues with the CAP theorem. Then, we looked at nine different situations to see whether the CAP theorem applied, what choice it made, and the implications on little-a availability.

I hope this was helpful! My main takeaway is not that the CAP theorem is irrelevant but that you should appreciate how it frames the debate while still understanding its limitations. Further, sophisticated patterns to reduce the probability or impact of various outages puts you in the Scenario 8 bucket. This may mean you're not strictly talking about the CAP theorem, but you are able to improve the little-a availability that's more important to your users.

If you have any questions or corrections for this post, please leave a note below or email me directly!

Tags: Distributed Systems Databases

ALSO ON **ALEXDEBRIE**

Using Custom Resources to …
6 years ago · 5 comments
This CloudFormation custom resources tutorial walks you though when …

Building a developer community
5 years ago · 4 comments
In this post, learn which strategies work and which ones don't when you're …

A Detailed Overview of AWS API Gateway
6 years ago · 31 comments
Look inside the black box AWS API Gateway to understand authorization,

What do you think?

5 Responses

Upvote Funny Love Surprised Angry Sad

0 Comments 1 Login ▾

G Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

♡ Share Best Newest Oldest