CAP Theorem

Bernard Lee Kok Bang

Introduction

- The CAP theorem is a tool used to make system designers aware of the trade-offs while designing networked shared-data systems.

- CAP has influenced the design of many distributed data systems.

- It made designers aware of a wide range of tradeoffs to consider while designing distributed data systems.

- Over the years, the CAP theorem has been a **widely misunderstood tool** used to categorize databases

Introduction

- The CAP theorem applies to distributed systems that store state.

- Eric Brewer, at the 2000 Symposium on Principles of Distributed Computing (PODC), conjectured that in any networked shared-data system there is a fundamental trade-off between consistency, availability, and partition tolerance.

- In 2002, Seth Gilbert and Nancy Lynch of MIT published a formal proof of Brewer's conjecture.

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

What is the CAP Theorem?

- The CAP theorem is an idea outlining different outcomes to show the limitations of the average system.

- This theorem, also known as Brewer's theorem, basically says that a distributed computer system cannot provide *consistency*, *availability* and *partition tolerance*, all at optimal levels.

- *i.e.* The CAP theorem states that a distributed system cannot simultaneously be consistent, available, and partition tolerant.

The three properties of CAP

- The theorem states that **networked shared-data systems** can only guarantee/strongly support two of the following **three** properties:
  - **Consistency** -
    - A guarantee that every node in a distributed cluster returns the same, most recent, successful write.
    - Consistency refers to every client having the same view of the data.
    - Consistency in CAP (used to prove the theorem) refers to linearizability or sequential consistency, a very strong form of consistency.

  - **Availability** -
    - Every non-failing node returns a response for all read and write requests in a reasonable amount of time.
    - The key word here is every.
    - To be available, every node on (either side of a network partition) must be able to respond in a reasonable amount of time.

The three properties of CAP

- **Partition Tolerant** - The system continues to function and upholds its consistency guarantees in spite of network partitions. Network partitions are a fact of life. Distributed systems guaranteeing partition tolerance can gracefully recover from partitions once the partition heals.

The Cap Theorem

- https://tinyurl.com/bde68d77
- https://tinyurl.com/d9ww563t

System categories –CAP theorem

- The CAP theorem categorizes systems into three categories:
  - **CP (Consistent and Partition Tolerant)** –
    - At first glance, the CP category is confusing, i.e., a system that is consistent and partition tolerant but never available.
    - CP is referring to a category of systems where availability is sacrificed only in the case of a network partition.
  - **CA (Consistent and Available)** –
    - CA systems are consistent and available systems in the absence of any network partition. Often a single node's DB servers are categorized as CA systems.
    - Single node DB servers do not need to deal with partition tolerance and are thus considered CA systems.
    - The only hole in this theory is that single node DB systems are not a network of shared data systems and thus do not fall under the **purview** of CAP.
  - **AP (Available and Partition Tolerant)** –
    - These are systems that are available and partition tolerant but cannot guarantee consistency.

due to network partition

Summary

- 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.

Analogy for CAP theorem

It is most probable that an IT professional driving (well… is there undrive?) in Bangalore traffic, one or the other time will think, let me quit this job and start a restaurant.

Let's start the story. Srinivas was fed up (it usually happens to 85% of people just after appraisals) with IT job and eventually quit his job to start a restaurant. After careful examination, he started delivery by a phone call. He hired few delivery boys whom he got at very cheaper rates after many food delivery start ups vanished in thin air.

Day5: Srinivas chose to operate the call himself while sitting at billing counter. Morning it was lull period. But from 7 pm onwards he started getting many calls. Whatever the order he gets, he writes on a paper, gives it to kitchen and …boom… it is cooked (well, not every time) and delivered to the customer. Around 8:30 pm he saw one customer walking to him. He is gasping for breath and apparently looks angry (may be hungry inside). "I have been calling for last 30 minutes. Your phone is always engaged. I had to walk for 20 mins to come here to place the order. I am not happy."

Idea time: Srinivas apparently was not happy and shaken to the core. When he was in IT services field, he was every day told by his bosses that "customer is God and you can't make God upset". Being a God fearing person he sacrificed many things in life including going to temple. After some disturbed sleep and thinking time, he got a brilliant idea. "Let me hire one more operator who can take the calls. If one line is engaged, another person will pick up." It took a week to onboard new person while he dealt with fuming customers for a week. This is improving "**Availability".**
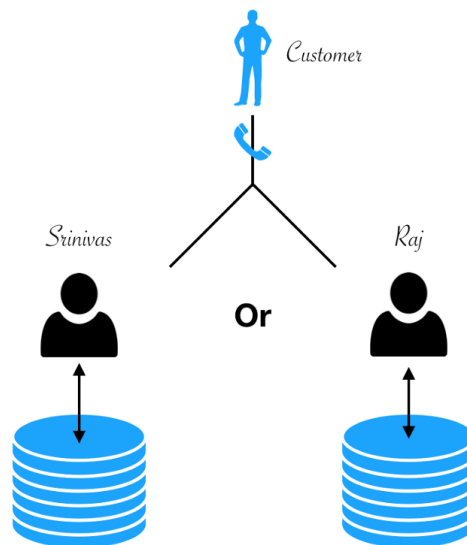

Improve Availability

Day 15: Now new employee Raj is on-boarded and Srinivas is delighted. Customer's waiting time in the call drastically reduced. If one line is engaged, calls are automatically transferred to second line. Between Srinivas and Raj things are working well. They were able to take orders and process them.

Day 27: At 8:00 pm Srinivas got a call from a customer. "I placed an order 45 minutes back. What is the status?" Srinivas took his phone number and name and tried looking at his order list. He doesn't have it. He looked at Raj who is next to him. Raj is busy in taking other orders. He can't disturb him. Srinivas apologised and asked the customer to wait for 2 mins. Customer is already unhappy and making him wait made him furious. He said "Cancel my order" and disconnected the phone. God fearing Srinivas is again distressed.

Idea time: Srinivas thought bit more about it. He also realized this kind of situation will come to Raj as well. After some thinking time…….Eureka— he found a solution. Next day he agreed with Raj to exchange order details as soon as they take orders. For example order number 223 was taken by Srinivas. He will have the original order and pass the copy of the order details to Raj. Similarly order number 224 was taken by Raj and he will pass the copy to Srinivas. Now they both have all the order details. Later if a customer asks the status, they can answer without keeping the customer in waiting. This is having "**Consistency**"
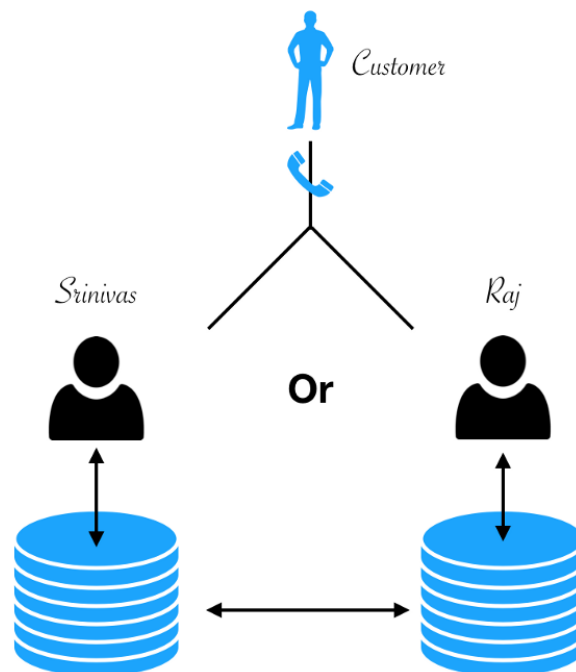
Available and Consistent

Day 283: Everything is going on well so far. The business increased multifold. Now he has 3 people taking orders and he built 1 kitchen. Both Srinivas and Raj are not doing this work any more. New team is Suma, Ramesh and Supriya. They are young, vibrant and nonchalant. As for the previous process, each of them updates other two on the orders.
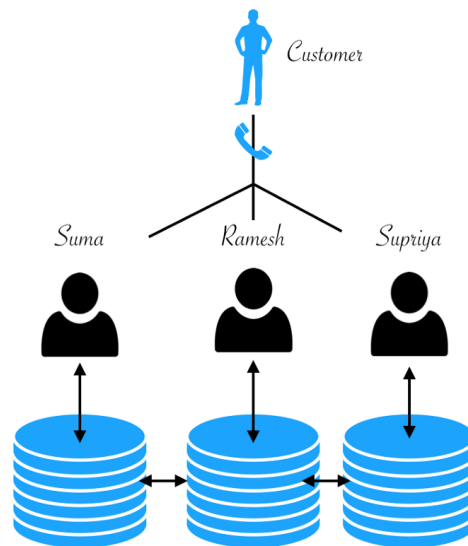
All is well

Day 289: All well and good until one fine day. Like in a bollywood movie, Supriya fell in love with Ramesh and Ramesh fell in love with Suma. And things started becoming complicated and Supriya started feeling like a loser. Things became worse with time. Both Ramesh and Suma stopped communicating the order details to Supriya and Supriya also did same. This led to broken communication. Now pretty much things went back to day 1. There is no "Partition tolerance" The only way service can be made available and consistent is by getting rid of either Ramesh and Suma or Supriya or making them work together. Or otherwise you can make system "Available" but with inconsistent data.
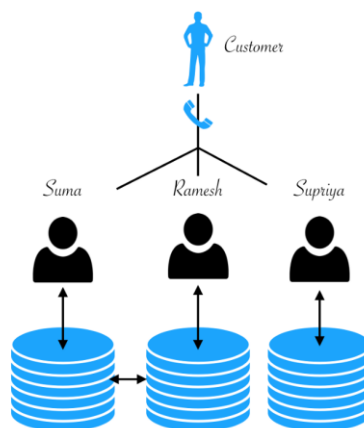
No partition tolerance because of broken communication

Lets come back to reality of our IT world.

CAP stands for Consistency, Availability and Pratition Tolerance.

- Consistency (C): All nodes see the same data at the same time. What you write you get to read.
- Availability (A): A guarantee that every request receives a response about whether it was successful or failed. Whether you want to read or write you will get some response back.
- Partition tolerance (P): The system continues to operate despite arbitrary message loss or failure of part of the system. Irrespective of communication cut down among the nodes, system still works.

*Often CAP theorem is misunderstood. It is not any 2 out of 3. Key point here is **P** is not visible to your customer. It is Technology solution to enable C and A. Customer can only experience C and A.*

P is driven by wires, electricity, software and hardware and none of us have any control and often P may not be met. If P is existing, there is no challenge with A and C (except for latency issues). The problem comes when P is not met. Now we have two choices to make.

AP: When there is no partition tolerance, system is available but with inconsistent data.
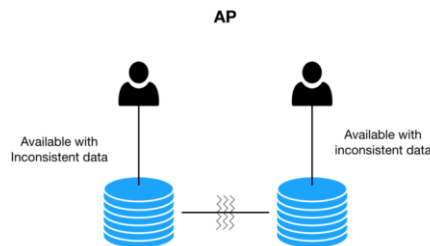
CP: When there is no partition tolerance, system is not fully available. But the data is consistent.
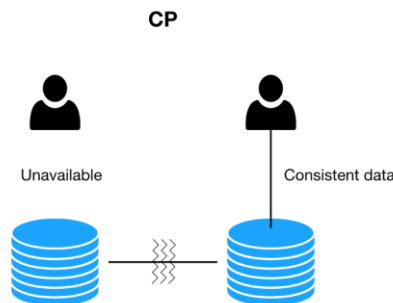
Partition Tolerance

- ability of different nodes or servers in a system to *continue operating* and *serving their purpose* even if there are network failures or partitions.

- even if some parts of the system cannot communicate with each other temporarily, they can *still function independently* and provide their services.

- e.g., Imagine you have a group of friends who love playing games together, and you decide to have a game night at one of your houses. However, due to some unforeseen circumstances, your house experiences a power outage, and you lose the ability to communicate with your friends.

  - partition tolerance is like the ability of your group to adapt and continue playing games despite the power outage and the temporary loss of communication between you and your friends.

Prioritizing Consistency and Partition Tolerance

- Using *Shopee* as an example

- *Consistency*
    - whenever a customer adds an item to their shopping cart, all subsequent requests to view the cart should reflect that addition.

- *Partition tolerance*
    - even if there are temporary disruptions in the network, the website should continue functioning, and customers should be able to access and modify their shopping carts.

- *ensure that the data in each customer's shopping cart is always consistent and available, even during network failures.*

Prioritizing Availability and Partition Tolerance

- Using *WhatsApp* as an example
- *Availability*
  - WhatsApp is always be up and running
  - WhatsApp would still allow you to send messages to your contacts and receive their messages
  - It would store these messages locally on your device until the network connection is restored.
  - might experience a delay in receiving messages from other contacts who are connected to a different server.
- *Partition tolerance*
  - able to handle network failures or communication breakdowns between different servers
  - WhatsApp application can continue functioning even if there are temporary disruptions in the network
- *ensure that users can always send and receive messages, regardless of network issues.*