## **CAP Theorem**

CAP theorem is a fundamental theorem in a distributed system that states any distributed system can have **at most two of the three properties** of

**Consistency** This states that the data has to remain consistent after the execution of an operation in the database. For example, post database updating all queries should retrieve the same result.

**Availability** The databases cannot have downtime and should be available and responsive always

**Partition Tolerance** The database system should be functioning despite the communication becoming unstable

CAP theorem states that a system in Distributed systems cannot simultaneously be Consistent, Available, and Partition tolerant.

## **Horizontal vs Vertical Scaling**

When you have a single node(host/server) and 300 users using your app, everything might work flawlessly. When you hit 600 users, the node you have might not be able to keep up with the incoming number of requests with the hardware it has. At this point, you need to scale and you have two options.

You can improve the hardware on the same node by doubling the memory or CPU, or whatever you need so that you can accommodate 600 users with a single machine. This is what we call **scaling up** or **vertical scaling**. This solves the immediate problem but you cannot keep adding more memory or CPU to the same machine forever. At some point, you’ll hit the max hardware a machine can support.

Another approach is putting new nodes that serve the incoming requests/tasks together, which is called **scaling out** or **horizontal scaling**. While this brings a lot of challenges, it allows you to scale almost infinitely, hence the recommended approach for most use cases. In addition to that, doubling the number of nodes is cheaper than doubling the hardware in a single node.

**Horizontal Scaling** refers to the addition of more computing machines to the network that shares the processing and memory workload across a distributed network of devices. In simple words, more instances of servers are added to the existing pool and the traffic load is distributed across these devices in an efficient manner.

**Vertical Scaling** refers to the concept of upgrading the resource capacity such as increasing RAM, adding efficient processors etc of a single machine or switching to a new machine with more capacity. The capability of the server can be enhanced without the need for code manipulation.

## **Load Balancing**

Load balancing refers to the concept of distributing incoming traffic efficiently across a group of various backend servers. These servers are called server pools. Modern-day websites are designed to serve millions of requests from clients and return the responses in a fast and reliable manner. In order to serve these requests, the addition of more servers is required. In such a scenario, it is essential to distribute request traffic efficiently across each server so that they do not face undue loads. Load balancer acts as a traffic police cop facing the requests and routes them across the available servers in a way that not a single server is overwhelmed which could possibly degrade the application performance.

## **Latency, Throughput and Availability (Performance Metrices)**

**Latency**: This is the time taken in milliseconds for delivering a single message.

**Throughput**: This is the amount of data successfully transmitted through a system in a given amount of time. It is measured in bits per second.

**Availability**: This determines the amount of time a system is available to respond to requests. It is calculated: System Uptime / (System Uptime + Downtime).

## **What is Sharding (Horizontal Partitioning)?**

Sharding is a process of splitting the large logical dataset into multiple databases. It also refers to horizontal partitioning of data as it will be stored on multiple machines. By doing so, a sharded database becomes capable of handling more requests than a single large machine. Consider an example - in the following image, assume that we have around 1TB of data present in the database, when we perform sharding, we divide the large 1TB data into smaller chunks of 256GB into partitions called shards.

Sharding helps to scale databases by helping to handle the increased load by providing increased throughput, storage capacity and ensuring high availability.

It allows you to spread reads/writes evenly across nodes by partitioning the data into multiple nodes. Each shard/partition will own some part of the data (i.e. some rows of a table).

## **How is NoSQL database different from SQL databases?**

## **Questions**

Design URL Shortening service like TinyURL

Companies :