SYSTEM DESIGN – II

Latency & Throughput

Latency is the time required to perform some action or to produce some result. Latency is measured in units of time -- hours, minutes, seconds, nanoseconds or clock periods.

Throughput is the number of such actions executed or results produced per unit of time. This is measured in units of whatever is being produced (cars, motorcycles, I/O samples, memory words, iterations) per unit of time. The term "memory bandwidth" is sometimes used to specify the throughput of memory systems.

In our industry based on the latency many services are dependent. People will be using AWS services. If AWS has some issues in their services, then client will be at loss because of the delay from the AWS. Netflix servers are AWS servers.

SLA – Service Level Agreement – this has details about the money

SLO – Service Level Objective – paying money back if services are not meet

If the services are not met, then the company has to pay back its client for its loss.

The availability factor is very much important in such factors.

Availability (Importance):

Sometimes AttainU is done and we are not able to login. But that would not impact our life. But if the software managing planes goes down, then it would be hard for the pilot to land the flight and many other issues occurs.

Ex: Airlines – if the services are not available then it leads to big issues and causalities.

In **Banking System** – services managing transactions should be highly available. It cannot be highly available, but needs to be highly consistent.

In **multiple servers**, having a load balancer, the requests would be redirected to the servers. In the database servers we store details. In Database servers, we have horizontal partitioning.

There are two types of Systems:

1) **State-less** –

A stateless process or application can be understood in isolation. There is no stored knowledge of or reference to past transactions. Each transaction is made as if from scratch for the first time. Stateless applications provide one service or function and use content delivery network (CDN), web, or print servers to process these short-term requests.

An example of a stateless transaction would be doing a search online to answer a question you've thought of. You type your question into a search engine and hit enter. If your transaction is interrupted or closed accidentally, you just start a new one. Think of stateless transactions as a vending machine: a single request and a response.

Ex: a = 2, b = 3, a + b = 5

2) **State-full** –

Stateful applications and processes, however, are those that can be returned to again and again, like online banking or email. They're performed with the context of previous transactions and the current transaction may be affected by what happened during previous transactions. For these reasons, stateful apps use the same servers each time they process a request from a user.

If a stateful transaction is interrupted, the context and history have been stored so you can more or less pick up where you left off. Stateful apps track things like window location, setting preferences, and recent activity. You can think of stateful transactions as an ongoing periodic conversation with the same person.

The majority of applications we use day to day are stateful, but as technology advances, microservices and containers make it easier to build and deploy applications in the cloud.

Ex: Facebook

Suppose we have a load balancer with 3 servers. In the first computer we have stored A user details. Now, the next time she come again; he visits to server 3. But his details are present in server 1. So, we have a consistency issue. This is a stateful service because it is storing the information of A.

We can have a job which will run every 5 mis to replicate the server 1 information to the other servers. When this is happening, in the load balancer we put a mapping that

A data is put in S1 that every time A comes to load balancer, then he is redirected to S1.

In a distributed system, S1 can fail and the request will be failed as the system will not be available. This is a single point failure, so we have a copy of S1 as S1'. This avoids the request being failed by A. The probability of failure has come down. These servers are located in separate locations across the world, so it is not possible for all the servers to go down at the same time.

Master Slave Architecture:

S1 has 3 copies. In these three servers, S1 is master and remaining two are slave. Whenever data is given or entered, it is recorded into the master node and all the read request will be handled by the slave.

Content Delivery Network (CDM):

Small start-ups will have a smaller number of servers. Ex: AttainU. If we are in US and are trying to access the website from there, all the files on the website will be downloaded from India and this will cause **latency**. So,

AWS → cloud front

Cloudwise → CDN

Akamai → CDN

They are offering CDN. This happens; all the files downloaded from AttainU servers in India, if they have purchased a CDN, then CDN is placed between you and client. For the request sent, the CDN will check if the it has the image and then, if the request is not found, then the request is sent to AttainU and the image is now shared to us at the same time, all the servers across the globe are updated with the image.

The next time if another person requests the same, then the CDN server near to him, will answer the request. ***This decreases the delay which in-turn reduces latency***. CDN acts as a cache and every time a file is requested which does not exit in the CDN, it will be updated.

Taking advantage of CDN architecture#

As mentioned, reduced latency resulting in a faster loading website for users is one of the main benefits of using a CDN. However, there are also other important advantages to using a CDN's infrastructure for content delivery. For example:

- **Downtime Protection and Reliability**: CDNs take a large load off the origin server allowing for easy scalability in case you receive a sudden spike in traffic. CDN architecture is designed to handle very large amounts of bandwidth usage by providing ample resources and networking capabilities. This allows for more users to access the website while minimizing the risk that the origin server gets overloaded. CDNs also help increase reliability as if one POP goes down, traffic gets rerouted to a different POP - increasing redundancy.
- **Increased Security:** With the number of servers a CDN has to offer, a website that is using a CDN also benefits from increased security from things such as DDoS attacks. Many CDN's, including KeyCDN, have infrastructure built in to help mitigate the risk of attacks. CDNs may also provide additional security features such as the ability to install SSL for a secure connection between the CDN servers and the end user and the ability to create Secure Tokens.
- **Improved SEO:** A cross benefit from a faster loading site which results in a better user experience is that it also helps improve SEO. As Google uses site speed as a ranking factor in their algorithm, having a speedy site will also help move you up in the SERPs. Reliable and modern CDN architecture allows for increased speeds by implementing improvements such as HTTP/2 and SSD-optimized servers for even faster content delivery.

Latency Number: *($\mu s = 10^{-6}$ s)*

If we have 1 MB of data and we want to get it from RAM, it will take 100 – 200 **µs**. The same data from SS will take 1000 **µs**.

1 MB	RAM	100 – 200 **µs**
1 MB	SSD	1000 **µs**
1 MB	1 GB	10000 **µs**
1 MB	HDD	20000 **µs**

If we have a packet (a small chunk of data) and this is to move from California to Netherland and then back to California it would take 150000 **µs**. There are some databases in market that store data in RAM instead of HDD, as it is very fast.

Database ➔ RAM → Radis, Memsql (costly because RAM is limited of size)

➔HDD | SSD → Postgres, MongoDB, Cassandra etc.,

Now we have few servers, A1, A2, A3, A4, A5.

It is such that,

A1 = a, b, c

A2 = d, e

A3 = f

A4 = g

A5 = h, i, j

If A1 server dies, then we need to migrate the data to other servers. If A2 server is copying, the data so it would not be available. A3, A4 & A5 are copying the information for A1 so, even they are also not available. This raises an issue. If a server fails to migrate its data to other servers, the it is an issue.

The load balancer will have a mapping regarding the data and so we need to **shuffle the data**. We have to update the load balancer mapping. This will be in the $O(n)$ operation.

Sharding → breaking or separating data among different computers. If we have 2 PT of data, then we have to separate the data into different servers. The solution is **Consistent Hashing**

Hashing

Hashing is the process of mapping one piece of data — typically an arbitrary size object to another piece of data of fixed size, typically an integer, known as hash code or simply hash. A function is usually used for mapping objects to hash code known as a hash function.

Consistent Hashing

Consistent Hashing is a distributed hashing scheme that operates independently of the number of servers or objects in a distributed hash table by assigning them a position on an abstract circle, or hash ring. This allows servers and objects to scale without affecting the overall system.

We will have 4 separate hash. Supposed we have 1000 GB of space. The ring has 0 – 1000 with S1 at 0, S2 at 250, S3 at 500 and S4 at 750.

Now S0 server will handle all the users with ID from 750 to 0.

S1 will handle the data between 0 – 250

S2 will handle the data between 250 – 500

S3 will handle the date between 500 – 750

If S0 fails, all the details will be sent to S2 only so there is no shuffling and S2 will have the data of S1 servers. This is called **Consisting Hashing.**

If all the servers are down it would be called, **Cascading Failures.**

So instead of having four servers, managing these ranges, we spread S1 across the data. So, S1, S2 and S3 are present across the range. The data is spread across the servers. This increases the range of all the servers. By doing this cascading failure can be overcome. This is **important when servers are dynamically added**.