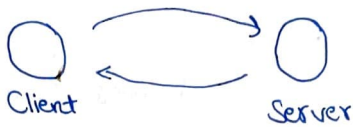
SYSTEM DESIGN

- **Client - Server model:**



⇒ Vaguely defining, a client is someone who requests for data, and server is someone who listens for requests and replies with data.

⇒ Let client be a browser window and server be a company website.

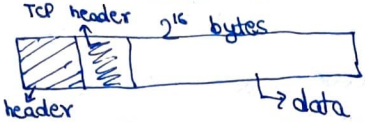
- ✱ Browser will first make a DNS query, to find out the public IP address of server.
- ✱ Server will be having a public IP address that is provided by its cloud provider. Eg. Google cloud ~~provider~~ platform.
- ✱ Once the server's IP address is known, client sends HTTP request, that will also contain the source's IP address.
- ✱ Server listens on specific ports (80 for HTTP, 443 for HTTPS), and then returns the corresponding response to the sender's IP address.

- **Network protocols:**

⇒ A protocol is agreed upon set of rules for interaction b/w two parties.

⇒ IP : Internet Protocol

★ An IP packet is fundamental unit for the data that is sent over the internet.



IP packet (stored in bytes)

⇒ TCP : Transmission Control Protocol

★ It is built on top of IP and maintains the relative ordering of the IP packets, in a reliable and error-free way.

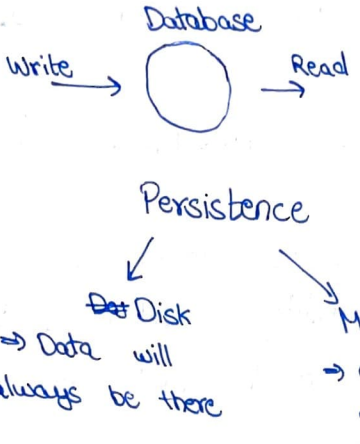
★ TCP connections are established using handshake mechanism.

⇒ HTTP : Hyper Text Transfer Protocol

★ It is built on top of TCP and provides higher level of abstraction using request response paradigm.

★ Requests and responses can be viewed as objects.

- **Storage:**



⇒ There are different type of database services available based on uptime, distribution and structure of data, consistency etc.

- **Latency and throughput:**

⇒ Latency: How long does it take the data to traverse through the system.

⇒ Throughput: The amount of work that can be done by the network in a given amount of time.

⇒ Just increasing throughput won't increase performance. For eg., increasing number of servers can be more feasible to avoid bottleneck.

⇒ Latency and throughput are not necessarily correlated.

- Availability:

⇒ It can be measured as a system's uptime in a given year.

☆ One measure is nines (99% = two nines, 99.9% = three nines)

⇒ Services have SLA (service level agreement) that determines the availability standards. SLO (service level objective) need to be met in order to fulfil the agreement.

⇒ In order to achieve high availability, we should avoid single points of failures. This can be done by adding redundancy. (adding multiple components)

- Caching:

⇒ Used to improve latency of system.

⇒ Caching can be done on client side, so that a network request to the server is not made again and again for the same query.

⇒ Caching can be done on server side, so that computations aren't performed again and again, and stored results can be used.

⇒ For writing operation, possible approach can be to write to cache version of data, and sync. can be performed on regular intervals b/w cache memory and database.

- Proxies:

⇒ Forward proxies act on behalf of client, clients first send request to proxy, which communicates with server and replies client with response.

⇒ Reverse proxies act on behalf of server, thus client interacts with proxy only thinking of it as server, which can help server to avoid certain type of clients.

- Load balancer:

⇒ As the number of clients and requests increase, server response and processing can become slow. To solve this, we can either vertically scale (increase power of server) or horizontally scale (increase number of servers) the system.

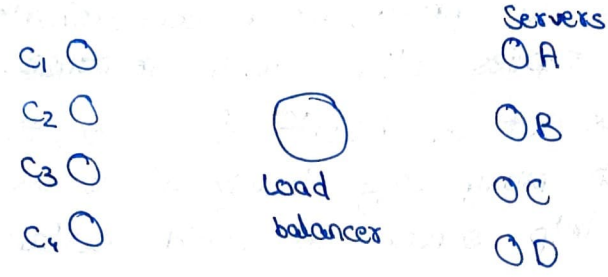
⇒ When we horizontally scale, load balancers do the work of evenly distributing the workloads among resources.

⇒ Whenever a new server is added/removed, it is registered with load balancer to avoid inconsistency.

⇒ Load balancing can be done in a randomised way, round robin way (supplying requests one by one to each different server each time), weighted round robin way (assigning weights/priorities to servers) ~~etc.~~, IP based hashing (creating hash key of client's IP and assigning server), path based selection (particular servers for particular website paths) etc.

• Hashing:

⇒ Scenario:



1) C_1 sends request to load balancer, gets routed to server A. Server A checks in memory cache, it is empty so performs computations, stores result and returns response.

2) C_4 again sends request to load balancer, this time gets routed to server D. Thus, unnecessary computations will be performed again.

This could've been avoided by IP based hashing.

⇒ Problem: When a new server is added / removed, our hashing function will change (mod based). This will lead to losing all the cache information. Solutions include consistent hashing (assigning positions on an ~~ass~~ abstract circle) or rendezvous hashing (calculate server rankings for each client and map client to that server. Score will be calculated by some operations).

- **Key Value Stores:**

⇒ While most systems rely on relational databases for storage, if a non relational database is to be chosen, key value store can be an option.

⇒ Key value stores are mappings between keys and values. This can enable faster retrieval of data.

- **Specialized Storage Paradigms:**

1) Blob store :-

⇒ Blob stands for Binary Large Object. It refers to some arbitrary piece of unstructured data.

⇒ Blob store is optimised to store and retrieve large amounts of blobs. Eg. Google cloud storage, Amazon S3.

2) Time Series Database :-

⇒ Used to store time dependent data (eg. events). Use cases include monitoring, IOT systems etc. Eg. Influx DB, Prometheus

3) Graph Database :-

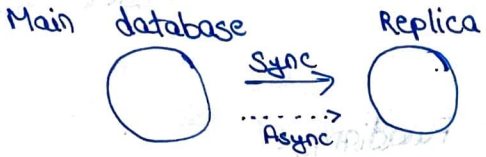
⇒ The concept of relationship is implemented using graphs. Eg. Neo4j

4) Spatial Database :-

⇒ Used to store spatial data. They have spatial index for faster operations. Data structure that can be used : Quadtree

- Replication and Sharding:

⇒ If we work with a single main database, it is possible that it fails and we are not able to read/write to it. To avoid this, we can maintain a replica that will serve the purpose in case of failures.



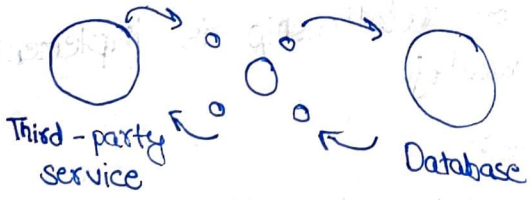
⇒ Replica will need to be in sync. all times. This increases the cost of writing operation.

⇒ Async. can be considered when the writing in one of the databases doesn't require immediate reflection in other databases.

⇒ When the data is very large, data can be splitted on certain criteria and placed in corresponding shards. This can help increase throughput.

- Leader Election:

⇒ Scenario:-



Let's suppose that the database contains data about customer subscriptions. Third party service is used to process the payments. They cannot be allowed

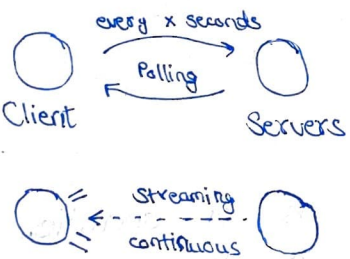
to communicate directly as data is sensitive. So, a server is used in the middle for communication. To avoid single point of failure, multiple servers use a leader election algorithm to decide the leader that will communicate.

⇒ Tools like Zookeeper and Etcd allow to implement leader election in an easy way. Consensus algorithms like Paxos and Raft are used for leader election.

- **Polling and Streaming:**

⇒ Polling refers to client issuing request every x seconds.

⇒ In streaming, instead of client issuing requests repeatedly, server will be pushing data to client.



- **Rate Limiting:**

⇒ It is limiting the amount of operations in a given time. It is helpful to prevent attacks such as DOS.

⇒ It's not always the best option as it cannot prevent DDOS attacks.

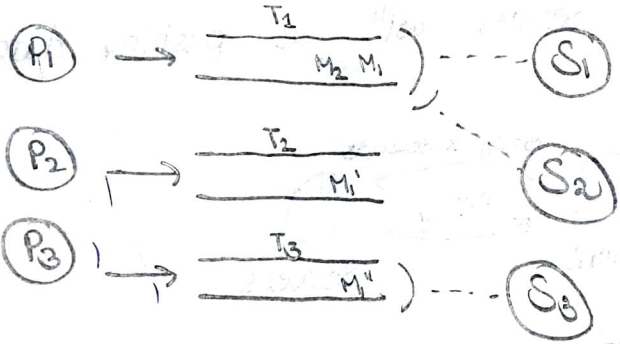
- Logging and Monitoring:

⇒ In order to debug issues that client may face, logging is used to keep updated information and errors thrown.

⇒ Monitoring is making sure that system with important metrics is available and visible so that important stats and system health can be regularly checked.

- Publish / Subscribe Pattern:

⇒ The paradigm consists of four entities, namely publishers (act as servers), topics (act as channels), subscribers (act as clients) and messages (act as data).

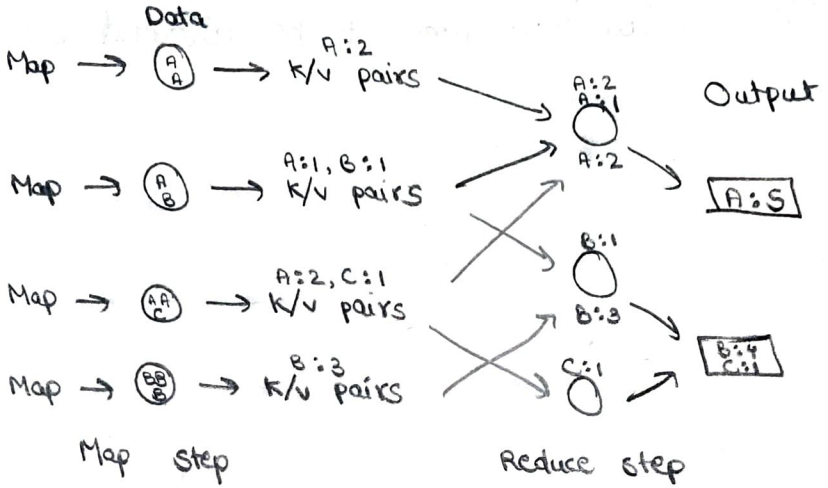


⇒ Idempotent operation is an operation that has the same output despite of how many times its performed.

⇒ The messages will be stored in persistent storage and thus this paradigm guarantees at least once the delivery of messages to the subscribers.

• MapReduce:

⇒ Allows to process huge amount of data that is distributed in a lot of different systems efficiently, quickly and in a fault tolerant way.



⇒ Map and reduce steps should be idempotent.

• Security and HTTPS:

⇒ We cannot assume that a communication over HTTP is secure, as it is prone to man in the middle attacks.

⇒ HTTPS uses encryption to solve the problem. Symmetric encryption uses single key to encrypt and decrypt while asymmetric encryption uses 2 keys (public & private).

⇒ HTTPS uses TLS handshake (Transport level security) and SSL certificate.

- API Design:

⇒ Thorough thought process is required as it is really difficult to change things once other products start relying on API.

⇒ Discussions related to entities involved, endpoint definitions, functionalities to be allowed etc.