**SYSTEM DESIGN**

DNS is a decentralized service that maps domain names to IP addresses

To map domain names to Ip addresses, we use A Records, A-address records

Your OS caches the records so that it does not have to make the DNS Query again

grapQL – developed by facebook, 2015 – make a single query to fetch all resources rather than single requests for different resources like REST

gRPC – Saves data as protocol bufferes rather than json which is more efficient cos data is serialized as binary but it is less human readable

**Websockets** -

Polling – making occasional/periodic requests to the server to check if there is a new message.

Why Use A Database Instead of just txt files ?? With dbs, we can more efficiently store data using data structures as btree AND fast retrieval of data using queries since data is organized into rows and tables

DBs follow the acid principle –

A – Atomicity – Every transaction is all or nothing

C – Consistency – Foreign keys and other keys constrains are enforced

I – Isolation – Transactions run independently

D – Durability – Data is stored on disk and hence not lost if a system is shut down

NoSQL dbs dropped consistency(keys) all together because it makes it difficult for a database to scale.

Sharding – Scaling dbs horizontally with a shard key

Replication – Leader, Leader, Leader, follower

CAP Theorem –

PAC Theorem

Message Queues have dedicated databases

LARGE SCALE DISTRIBUTED SYSTEMS

Eg Google Map

Stores , retrieves and updates a lot of dat frequently, service is not hosted on one server but is spread out between different servers which could be in different regions of the world.

**Publisher – Subscriber Module**

Extensibility – How easy it is to modify/Redesign your solution or code. Solution should be as less feature or requirement coupled or specific as possible.

Stateless Data Transfer – Server says, I don’t know who you are or what you want, describe everything in the request headers – HTTP is stateless

WebRTC protocol – peer to peer protocol

HDFS, S3– File Storage System,

Vimeo - Video hosting system

RTMP – Real time media protocol – Loses no data but WebRTC can lose some data

What the Hell is Map Reduce ???

MPEG-DASH – A video streaming protocol

HTTP- DASH – Dynamic Adaptive Streaming Over Https

Use Case Diagram

Sequence Diagram

\*\*\* Clock drift – Difference in clock time between servers running on a distributed system

Benefits of DISTRIBUTED SYSTEMS

More reliable, reduced latency, increased performance, cost effective

\*\*\*Availability Calculation – 5.26 minutes downtime per year, that is 99.999% availability

Availability Vs Reliability –

UNIX Timestamps – 4 bytes

Bandwidth – Amount of data you can push through the network

Load balancers – Improve reliability and scalability as threy balance incoming traffic to multiple servers

Nginx, HAProxy – software based load balancers , they can run on any commodity server

F5, Citrix – Hardware based Load balancers, can handle lots of traffic because they are specifically designed to be load balancers. But can experience vendor locking

Your webserver routes traffic to your application eg Express, Django, Wordpress etc

Load balancer Routing routes –

Round Robin – Demerit( All requests are not same)

Least Connection -

Least Response time – Route traffic to the server that responds faster

IP Hash – Shopping cart, making sure all requests of a particular type are directed to a particular server eg shopping cart

LAYER 4 VS LAYER 7 LOAD BALANCER –

Layer 4 does not have access to request information , it is faster and not cpu intensive

Layer 7 does have access to request headers and authentication and can terminate requests and redirect requests. It is more CPU intensive

**Redundant Load Balancer Setup**

Active AND Passive Load balancer

**CACHING**

Simply involves using an in memory storage between the application and the database, since difference in time spent fetching information between the disk and RAM is from (50-200X faster) – confirm this.   
Can serve same amount of traffic with fewer resources – since it serves more requests within shorter time

Distributed cache

TTL – Time to live – time set before a cache is refreshed

LRU – Least Recently used – When cache is full, remove data that hasn’t been accessed or used recently

LFU – Least Frequently used – When cache is full, remove the data that is least requested or used

**CACHING STRATEGIES**

Cache Aside -

Read Through -

Write through -

Write back –

**DATABASE SCALING**

**How ???**

Denormalization – Writing redundant data to reduce joins - > Risks inconsistent data, slower writes, boosts read performance

**Caching**

**Indexing** – more storage space, slower writes and updates, faster reads or lookups

Horizontal Sharding – Distributing data across different databases

Upside – Faster reads, since not one server handles all read requests

Downside – All data will not have same read or write requests, so one shard may end up having more requests than the others, ie uneven distribution of traffic to shards

Vertical Partitioning – Splitting schema such that you are not fetching unrequested data but only swiftly fetching frequently requested or used data from your schema.

Eg corporate general info on one table, corporate branding info on another

**YOGITA SHARMA – SYSTEM DESIGN COURSE**

System – A loosely used term used to describe an architecture or collection of softwares or techonology connected to and communicating with/ each other to serve a certain set of users for a certain set of requirements eg image sharing systems like Instagram, Netflix, whatsapp, hotstar

Design – a process of understanding the needs of users and selecting components, modules and software technologies and how they will be combined to serve those requirements

Components – building blocks of systems

Logical components – Data, database, applications, cache, message queues, infra, communication

Tangible Entities – Text, images,

Thick clients – Data processing happeneing on the client eg outlook, image editing softwares

Thin clients – eg youtube,

3 tier – Client – logic – data

N tier – caching, load balancer, logic all introduced

Proxy – On behalf

Forward proxy – max ip address

Reverse proxy – useful for load balancing, many proxy servers between the main proxy server AND the application server

Proxies are important elements for security, privacy and traffic management

Horizontal scaling is difficult with relational dbs

Where data grows adding colums and changing the schema becomes difficult,

Not suitable for storing data in which you do not know the structure before hand

Column dbs support a large number of writes – Cassandra,

NON RELATIONAL – support a lot of reads and writes,

If you need all the data in one document without joins –

Updates become complex with document dbs – nested, null or inconsistent data is very possible

Elastic search, solar – search dbs – these are non relational dbs – data stored is not usually the primary data store, but maybe most frequently read data which is refreshed regularly

Cache hit , cache miss

Cache invalidation

Cache eviction – FIFO, least frequently used, least recently used

**Caching strategies/patterns**

Cache Aside strategy – Cache never talk to db but only to application code. Application checks cache for data, if available in cache, it reads from there, if not, it fetches from db and updates the cache

Read through strategy – Application only talks to Cache and cache talks to db. Perfect for a read heavy scenario. Downside is there will be a lot of cache miss, mitigated by first pre loading the cache if it is known before hand what data will be requested

Write through strategy – Asides reading from cache, Application also writes to cache

Write Around – App writes to db but reads from cache

Write back – All write requests are kept in cache, batched and written to db after some time – can handle db failures for write heavy application. Cache going down is disastrous

REST – Representational State Transfer

Representational state talks about data – as data changes on the backend, it is updated when the client requests it too. State of data is been transferred too.

REST Guidelines –

Client to server

Data is cacheable

Layered – client only knows about the next layer

Stateless – Server and client are not saving state in any way – Authorization is a special case. State of rest Api request is not saved by server or client.

Uniform interface –

Code on demand : optional –

\*\*Throttling and Api rate limiting.