System Design Fundamentals

Topics → SQL, Server, Cache, Polling, Load Balancer, Leader Election, Peer-to-Peer, Availability, System, Proxies, Nginx, Client, MapReduce, HTTP, DB, Hashing, Replication

∅ Problem Solving Availability (V/S) Design Advanced Engineering Systems

— (Robust Functional, Scalable Systems)

What are Design Fundamentals?

Type of System? Functionality? Characteristics? } ~60-120 min discussions in interview questions

Correctness : Subjective
Pattern : Vague (Prob. Solver's jobs to ask for the requisite specification)

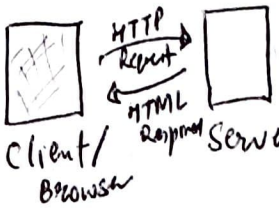
Categories of Design Fundamentals

1. Foundational / Underlying Knowledge
2. Key System Characteristics :
   ↳ Availability, Latency, Throughput
3. Deeper System Components
   ↳ Proxies, Load Balancer, Leader Election, etc.
4. Implementing Technologies
   ↳ HTTP, Proxy, DB, Hashing.

DNS System → Describes the entities & protocols involved in the translation from Domain Name → IP address

System Ports (2^{10}) 0-1023

Client Server Model



HTTP → Protocol to communicate to Server (Default : 80)
HTTPS : Default 443

Client → Machine speaks to Server (send/requests data)

Server → Set of Machine responding to the client.
↳ Ports → Listening Area for external clients to comm.

DNS Query → Special request to pre-determined server to extract a particular website's IP address.
(Linux, "dig" command does DNS query)

N.B.
1. The client/browser first extracts IP addres of a particular website using DNS Query/Lookup.
2. Using the DNS Query, the client then communicates w/ the requested server using HTTP Protocol

Lab

Terminal-1 (Server)
>> nc localhost 8081
"1 → hello"
"2 → hi"

Port

Terminal-2 (Client)
>> nc -l 8081 ← listening
"1 → hello"
"2 → --"
Receiver Server

IPv4 : a.b.c.d ($a \in [0, 255]$)

Special values
↳ 127.0.0.1 : Localhost
↳ 192.168.x.y : Private N/w (Used by wifi)

Ports → (0, 65535) ↳ 2^{16} total
80 → HTTP
443 → HTTPS
22 : Secure Shell
53 : DNS Lookup

Network Protocols ⭐

↓

IP, TCP, HTTP

Protocol → Agreed upon set of rules for communication on the internet.

↳ {Kinds of Messages}, {Transmission Medium}, {format/ordering}, {Response format}

IP → Internet Protocol

(modern internet runs on IP,

IP Packets → fundamental communication units (or building blocks) for internet communication

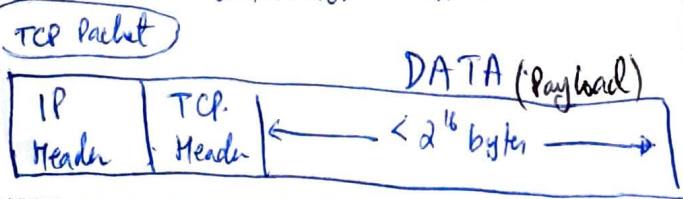
IP Packet.

- (in Bytes)
  - Source IP
  - Dest IP
- [Total Packet Size
  IP Version (4/6)]

- Data size of 2^{16} bytes is often too small for modern internet communication, necessitating multiple IP packets being exchanged for a single communication intent.
- Ordering / Structure of IP packets being sent from one machine & received at another machine is not garanteed → this TCP comes into picture.

TCP →
- Built on top of IP Packet
- Ordering, Structure, Error-free reception of IP packet at destination machine.

TCP Packet



N.B.

Before sending TCP packets from Machine 1 → Machine 2, the TCP first sets up the connection b/w host & destination using HANDSHAKE scheme (Special TCP interaction for establishing connection for the TCP packets to be sent)

Advantage

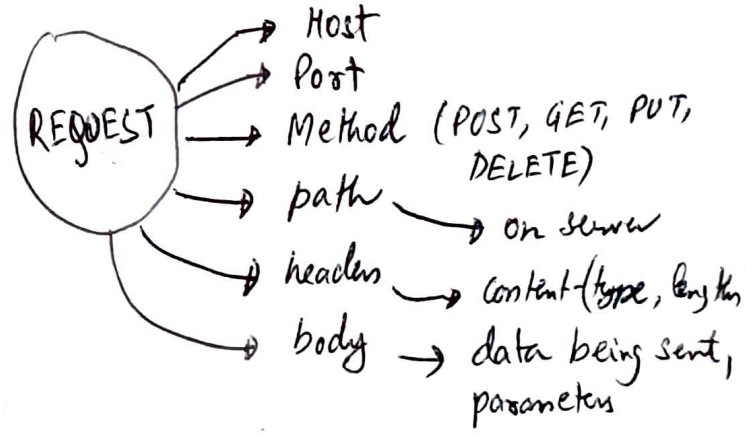
↳ More powerful, more functional wrapper on top of IP.

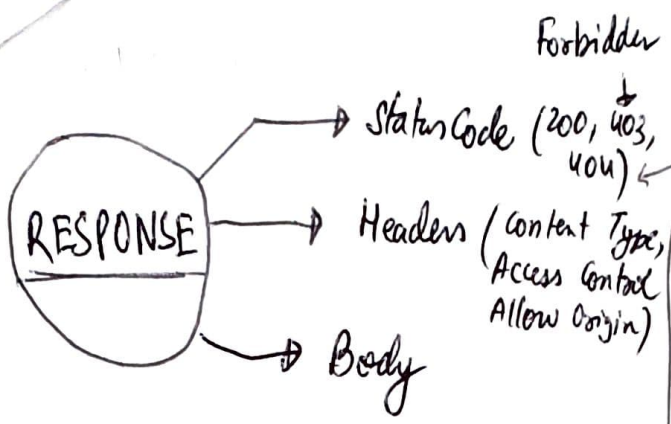
DisAdvantage

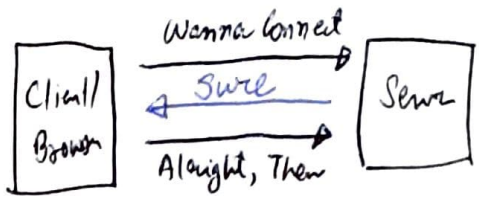
↳ Lacks really robust framework for SDE to define meaningful, easy-to-use channels for client ↔ server communications.

HTTP

1) Hyper Text Transfer Protocol
2) Higher-Level Abstraction above TCP & IP
3) Introduces, Request-Response Paradigm w/ support for flexible business rules (logic)
4) Power, Rigor provided by the Request, Response methods. (ex: POST, GET, PUT, DELETE, etc.)



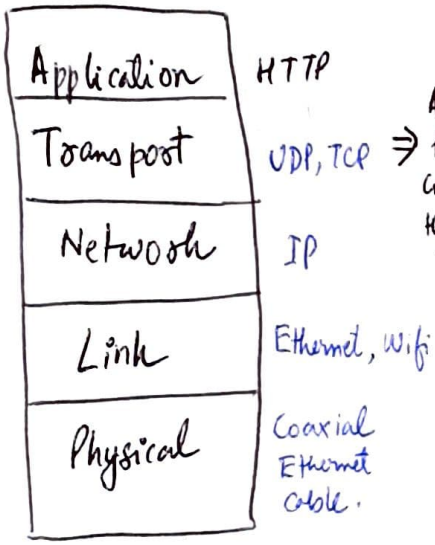




3-Way Handshake

UDP v/s TCP Transport Protocol

Every N/w protocol follows the following 5 layers



⇒ Allows multiple apps to use one N/w connections simultaneously through ports (2^{16})

UDP Protocol (User Datagram Protocol)

- Lightweight, Connectionless choice
- ✓ Smaller packet sizes (UDP Header → 8 bytes, TCP Header → 20 bytes)
- ✓ Doesn't require connection to send data
- ✓ More control over when data is sent.
- ✗ Primitive Error Detection (16-bit checksum)
- ✗ Doesn't compensate for lost packets
- ✗ No Ordering, Congestion Control)

TCP (Transmission Control Protocol)

- Reliable, connection-based choices.
- (3-way Handshake) Connection.

Features

1. Connection, Message via 3-way Handshake
2. Retransmission (for lost packets)
3. In-Order Delivery (as the segments are numbered)
4. Congestion Control & Error Detection (checksum)
   - ↳ Delays transmission when the N/w is congested.

DisAdvantages

1. Bigger Header (20-byte v/s 8 byte for TCP v/s UDP)
2. As TCP has Congestion control, data doesn't get sent out immediately (side-effect)
3. Bigger Overheads → Packet ReTransmission, Acknowledgement

Message v/s Streams

1. UDP is message-Oriented
   - ↳ Data is sent in distinct chunks (ex: Mail)
2. TCP is stream-oriented
   - Data is sent in chunks
   - Continous data flow
   - ex: Audio Conversation

CheckSum

TCP : IPv4, IPv6 → Mandatory Checksum

UDP : IPv6 → Mandatory CheckSum

Better : UDP v/s TCP

1. Text Communication : TCP (Ordering, Assured Packet Delivery)
2. File-Transfer, SSH → TCP
3. Multimedia Streaming → UDP (Less Overhead, No congestion control)
3. Small Transactions → UDP (ex: DNS Lookup) → No need to create/close connection.
4. Bandwidth intensive apps that tolerate packet loss → UDP

Exploratory Seminar

Building a High Performance Networking Protocol for MicroServices. ~ RSocket

Features of RSocket

1. OS Layer 5/6 Communication protocol
2. Application level flow-control (built-in)
3. Binary Encoded & ASynchronous Message passing.
4. Reactive Streams Semantics.

What's needed for high-performance MicroService Networking

Features of High-Performance MicroServices

1. Temporal & Spacial Decoupling
2. Binary
3. Application-Flow Control

1. Loose Coupling

(A) Spacial DeCoupling ⇒ Sender should not directly call the destination

(B) Temporal DeCoupling ⇒ Execution time & processing a message shouldn't affect the original caller.

that are no longer needed.

RSocket is loosely coupled using asynchronous message-passing

---

System Design

Storage

- Database : in-memory / persistent, Read / Query / Write
- Persistent Storage (Disk) : HDD / SSD,
  - Frequent Access
  - Infrequent Access
- Specialized protocols on top of HTTP / TCP.

Relational Databases (RDBMS)

- ACID Transactions, Strong & Eventual Consistency,
- Strong schema guarantee.
- Strong Consistency
  - Consistency of ACID transactions
- Eventual Consistency
  - Reads may be stale. However, a guarantee that future reads (after some lag) will reflect recent updates to DB.
- Non-RDBMS do not impose strong structure on the data stored in it.
- RDBMS impose strong structure on the data to support SQL. (Not all RDBMS do not support SQL)
- Complex Queries support SQL much more than noSQL → hierarchical Queries, transaction, trigger, cursor.
- ACID Transactions
  - Atomicity : transaction should all succeed / all fail.
  - Consistency : No stale state
  - Isolation : multiple transactions should have the same effect on transaction as a single transaction
  - Durability : committed Transactions are safe in Disk

Indexes

↳ Clustered, Non-Clustered

- Auxillary DS to support faster querying.
- Slower write/update/delete. (Space ↑)

Transactions

```
» BEGIN TRANSACTION;
  Update <> Set <> = <> - K where <>=<>;
    "
    :
  COMMIT;

» CREATE INDEX <> ON TBL-NAME(COL-);
```

Key-Value Stores

What is key-value store?

- Flexible NoSQL DBs that's often used for caching & dynamic configurations.
- Ex: DynamoDB, Etcd, Redis, Zookeeper

Etcd

- Strongly consistent & highly available key-value store.
- Used for implementing leader election in a system.

Redis

- Used for implementing rate limit.
- In-memory key-value store.
- Used as fast, best-effort caching solution
- Occassionally supports options for persistent storage.

Zookeeper

- Strongly Consistent, Highly available key-value store.
- Used: Leader Election, Store Imp. Configurations

Key-Value Stores

- Commonly used NoSQL paradigms, the key-value store bases its data model on the associative array data types.
- Fast, flexible storage machine resembling a hash table →
- Data Arrangements

Key	Value
foo	good
bar	SysExp, 1, [A1, Ge, ..]
baz	, too, 3, {'_':'_'}

- Lookup : O(1)
- Advantages
  - Low latency, high throughput
  - Plays on the strength of the caching.

Specialized Storage Paradigms

↳ Blob Store, Time Series DB, Graph DB, Spatial DB (Quad Tree)...

Blob Stores — Binary Large Object

- ~ Arbitary piece of unstructured data
  - Efficient storage, analytics on big-data scale of unstructured data.
  - Implementing Blob Store solutions is pretty complicated.

Blob Store solutions appear to mimic traditional key-value stores, however unlike key-value stores which are optimized for latency & throughput, Blob Stores are optimized for Analytics on Massive unstructured data.

Time Series DB

ex: InfluxDB, Prometheus

- For real-time monitoring, Prometheus DB is very popular.

Graph DB

ex: Neo4J.

- Built on the top of graph model, i.e relationship & associations lends well to the storage data getting stored.

ex:

```
CREATE (facebook : Company
                {name : 'FB'})
CREATE (clement : Candidate {name : 'Clem'})
```

Vars

Assoc

```
CREATE (alen) - [: INTERVIEWED {
    Score : 'passed'}] → (clement)

CREATE (molly) - [: APPLIED {status : 'rejected'}]
        → (facebook)
```

Sample Query

↳ Find the interviewers who interviewed & failed Clement & who also applied to & got rejected by Facebook.

```
MATCH (interviewer : Interviewer) - [:
INTERVIEWED {score : 'failed'}] →
(: Candidate {name : 'Clement'})
WHERE (interviewer) - [: Applied {status : 'rejected'}]
→ [: Company {name : FB}) RETURN interviewer.name;
```

- For certain data sources, storing data in graph DB aids performance.

Spatial DB

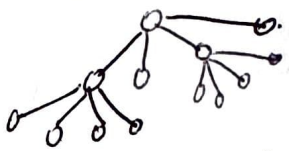
↳ Optimized for storing spatial data, i.e anything to do with geometric space (ex: lat, long)

ex: Quad Tree

- Spatial Index : QuadTree, KDTree

- Quad Tree

Every node has either 0/4 children nodes (exactly) - nothing in between.



} 0/4 children Node

- QuadTree makes it efficient to run queries in $O(\log 4N)$ time complexity

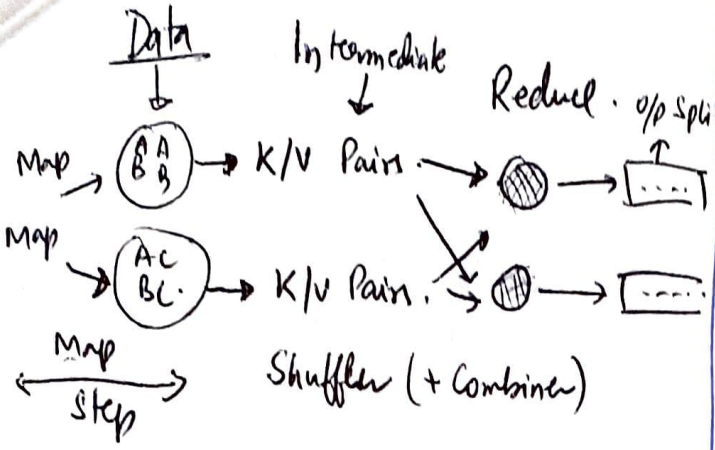
Map Reduce

Idempotent Operation ⇒ Same ultimate outcome regardless of how many time it's performed.

(Ex) Pub/Sub messaging system have to be Idempotent, since such systems allows for multiple times consumption of the same message

>> Update col Set col = col + 1 ✗ Not Idempotent

>> Update col Set col = True ✓ Idempotent, as col value stays True regardless of N time running the same command.



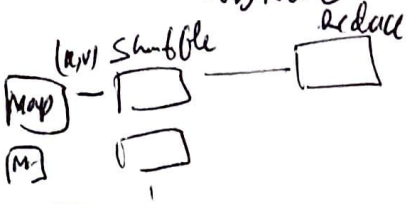
Task ⇒ Processing distributed datasets is not a trivial task → fault-tolerance, network partition, redundancy.

- Framework supporting big-data scale distributed compute in a fault-tolerant manner.
- Central control plan administering data

Shuffler = Master Node & Data Node

Task Node

Job Node



Shuffle Stage ⇒ Ensures all the key-value pair for a given key are on the same worker node(s) to avoid complex program design/ inconsistency.

Latency & Throughput

Latency ⇒ Time taken for an operation to complete.

RAM < SSD < Over Network < HDD < Inter-Continental Round Trip

0.25ms 1ms 10ms 20ms 150ms

Throughput ⇒ No. of operations a system can handle smoothly per unit time.

↳ Throughput of a server is handled in requests per second (RPS/QPS)

- Latency for a cascaded pipeline may refer to ~~total~~ sum of latencies of individual blocks.

(***) ⇒ Latency & Throughput are not correlated.

↳ A system w/ really faster data transfer (low latency) may have very low throughput as well.

So, High Throughput ≠ Low Latency (or, vice-versa)

Availability

Server ⇒ May act as client (for a DB) & server (end-user) at the same time.

Node/Instance/Host ⇒ Virtual/physical machine running a process (program on a machine).

Key Terms

(1) Availability : Server Up-time

↳ Odds of a particular server being up & running at any point in time (described in %).

A server with 99% availability is operational 99% of the time (described as having 2 nines of availability)

High Availability (HA)

↳ System having high levels of availability → typically 5 nines/more.

99% (2 9s) → 87.7 hrs
99.9% (three 9s) → 8.8 hrs
99.99% (four 9s) → 52.6 m
99.999% (five 9s) → 5.3 m

Expected downtime (per year)

Reliability

Process of replicating system subset(s) to make them individually or the entire system more reliable.

SLA (Service Level Agreement)

↳ Composed of one/more SLOs (Service Level Objective)

SLOs are gaurantee on system's HA.

SLAs are collection of SLOs gaurantee larger system HA.

Availability is also the System's fault tolerance
↳ DB, Server, Congestion

SLA|SLO ↔ Nines ↔ Redundancy

- System Availability is expressed as no. of nines of uptime → two nines/three nines/...
- HA : 5 or more nines.
- Implied/Explicit gaurantee - level through SLAs (one/more SLOs)
- Availability (HA) may require toadeoffs b/w latency & throughput.
- Single points of failures are overcome using Redundancy at server-level/loadbalancer-level

- Passive Redundancy ⇒ Standby nodes ensure system availability even when a few components fail.

- Active Redundancy ⇒ When the fail-over machines are the only machines handling requests → In case of failure, a leader election takes place to elect the next leader which takes the place of a failed-over machine.

Caching

- Stores previously computed results for fast retrieval. Typically used to cache Network requests as well as results of computationally long operations.
- Cache hit ⇒ When requested data is found in cache.
- Cache Miss ⇒ When requested data should have been found in cache, but it isn't → Poor design choice/ Negative consequence of a system failure.
- Cache Eviction Policy
  Policies governing eviction/removal from cache
  - LRU (Least Recently Used)
  - FIFO (First-In First Out)
  - LFU (least-frequently used)
- CDN (Content Delivery Network)
  - Third-party services acting like cache for one's server. As CDN servers are global, they may outperform one's own server when catering to requests originating from another region.

  Ex → Cloudflare, Google Cloud CDN

Top down DP is one good example of caching to bring down the system latency by avoiding re-computations of computationally heavy tasks.

- Can be done at the client level, DB level, Server level & Hardware level
- At client level, it avoids making a lot of same requests → LRU-cache
- Caching can be done for the query or parameters or both.
- Caching can also be done for the data which needs to be written ↓
  - Write through cache (saves the data both in the server-side cache & DB (cache). That way, we avoid reading that post from DB again.
  - Write-back Cache

Overhead of waiting to DB blocks at the same time

~~Keep data in both caches & DB,~~
For new data to be appended to the cache, it's first written to cache & asynchronously or in background, it's written to DB (avoid computational overhead of writing to DB at the same time as in write-through cache).

- Caches may become stale/inconsistent if not properly updated/synced with DB (Ex: updated comment on YouTube)
  - In that case, more complex architecture on top of cache needs to be implemented

Cache Eviction Policies

- We don't have infinite cache space!
- Policies around stale data & cache freshness
- Effective Cache Eviction Policies

Policies

(A) LRU
(B) FIFO/LIFO
(C) LFU (Least Frequently Used)
(D) Random (Benchmarks)

Hashing

hash(x) = <fixed-bit integer> ← less

hash(x_1) = hash(x_2) = Collision

A good function attempts to minimize collision to promote uniformity maximization.

Types of Hashing

(A) Consistent Hashing

A type of hashing that minimizes the no. of keys that needs to be remapped when a hash table is resized.

- Used by: Load Balancers
  - To distribute traffic to servers
  - Attempts to minimize the no. of requests that gets remapped to diff. servers when new servers are added/existing servers are brought down.

(B) Rendezvous Hashing

- Also called, highest random weight hashing
- Allows minimal redistribution of mappings when servers are brought down.

SHA

- "Secure Hash Algorithm"
- Collection of cryptographic hash function used in industry
- Popular choice : SHA-3 / SHA-5, MD-5

Strategies
- Random
- Round-Robin (A→B→C→D)
- Least-busy Server

One prob w/ load balancer routing strategy is that the cached request values on individual servers may be under-utilized if a given request is sent to diff. server everytime.

Ex: Say R_1 is already cached on S_1, however, if R_1 is redirected to S_2/S_3, the computation needs to be performed from scratch → Highly Inefficient Process.

Thus is where hashing comes into play ⇩

Server Allocation Scheme = Hash(Client-id / C_1/C_2) % (Available servers)

~ Hash(C_4) % 3 ∈ {S_1, S_2, S_3}

corresponding respective client

- Ensures the requests end up on the same server

- Good Hashing function maximizes uniformity ⇒ ie, the requests are evenly distributed (No bias).

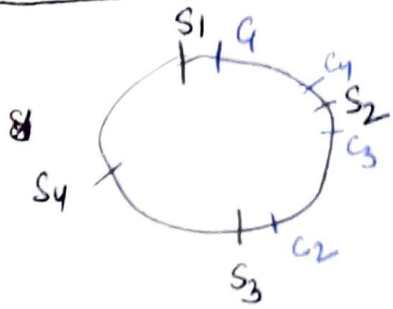
Q) What happens when an existing server dies/ new servers are added?

N.B. ⇒ We can't keep hashing, as the mod of the hash result may be inconsistent, leading to Cache Miss.

How to solve the problem?

⇒ Using Consistent / Rendezvous Hashing

Consistent Hashing



Servers = {S_1, S_2, S_3, S_4}

- Position the clients (C_i) & Servers (S_i) in a circular fashion.
- Client is being served by a server closest to it (in a clockwise/ counter-clockwise fashion)

Clock Wise Fashion ⇒ $C_1 : S_2$, $C_2 : S_3$, $C_3 : S_3$, $C_4 : S_2$

Counter-Clockwise fashion ⇒ $C_1 : S_1$, $C_2 : S_2$, $C_3 : S_2$, $C_4 : S_1$

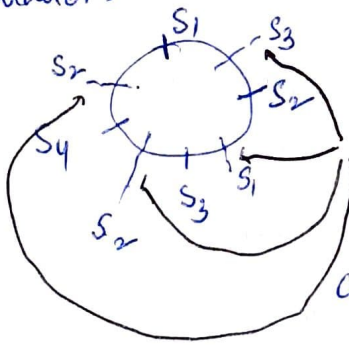
When a server dies, only a small no. of clients are redistributed (When S_3 dies, only C_2 is affected, it now gets redirected to S_4) ←

- This is how Consistent Hashing is so powerful ⇒ it maintains high level of consistency b/w the hashes & their target servers.
  ↳ Minimal target server redistribution when server dies / new servers are added.

- Disadvantages:
  - Less evenly distribution of client request (skew based on client positioning)

    ⇩ SOLN:

    Each server will have multiple locations in the circular arrangement

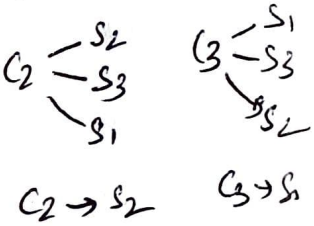


  - If a few servers are more powerful than other servers. And, we want powerful servers to get more requests. Passing the powerful server to more hash functions to determine its ~~same~~ more position in the arrangement. i.e, increasing virtual positions of the powerful servers.

Rendezvous Hashing

For an incoming client request, it ranks the available server (using pre-determined logic) & then assigns the highest ranked server to the client.

Ranking → [C1 ≤ S1, S2, S5]



C2 → S2 C3 → S1

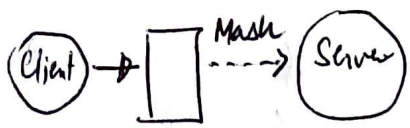
Final → C1 → S1

Allocation

- When S2 dies, C2 — S3 : C1 → S3,
  C2 — S1
  the next highest ranked server (S3) is picked up by the client.
- Ensures minimal target server redistribution in case of fault tolerance. (Better than Consistent hashing — in terms of minimizing redistribution)

Proxies

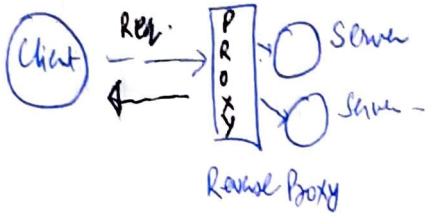
① Forward Proxy



Proxy Server (Forward

Proxies (or, Forward Proxy) are additional server sitting b/w Client - Server, typically used to mask client's identity (IP Address).

Reverse Proxy



Additional proxy sitting at the server end.

Used For ⇒ Logging, Load Balancing, Caching

Nginx (engine X)

Popular webserver used as reverse proxy / load balancer.

Note

1. Forward Proxy ⇒ mask client IP with its own IP. It may expose the client IP in some form to the server IP (if the need be).
2. Client does DNS query for the Reverse proxy IP & not the actual server IP.
3. Forward Proxies are also called Tunneling / Gateway IP.
4. Other Advantages
   - Improves performance & lessens N/w traffic (Or, Shields from malicious servers).
   - Improves N/w traffic and even supports IP filtering (to ensure policies are met using URL rewrite).
   - May improve performance using IIS compression (if enabled), - by reducing size of requested information.

IIS → Internet Information Service

Load Balancers

- Server-Selection Strategy

  Method of allocating a client's request to a server from the pool of available servers.

  Popular Algorithms
  - FIFO/LIFO, Round-Robin, Least-Busy Server, Random selection, Performance-based selection (choosing based on performance metrics : Fastest Response Time, Least Amount of Traffic), IP based Routing, Weighted Round Robin (maximizes caching)

- Hot Spot

  Server allocation may be skewed - happens when sharding key / hashing functions are suboptimal, → thus, creating a hot-spot

- Load Balancers also enable systems to have higher throughput by reducing congestion on a single system(s).

- Types : Softwares & Hardwares
  ↳ More Power, Flexibility, Scaling

- Working :

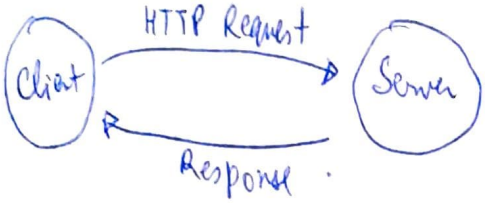
  Load balancer registers all active servers, including new server additions / removal of servers from the systems. ex: zookeeper

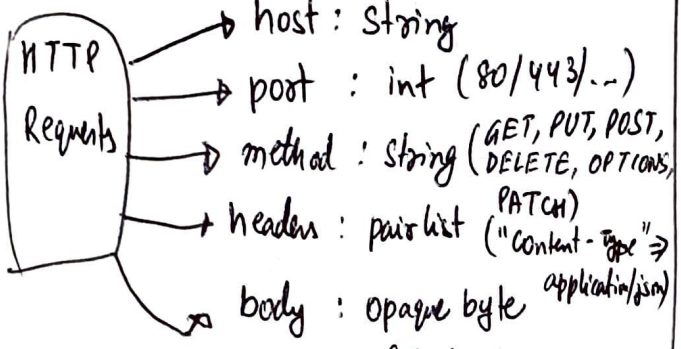
- Load Balancers can assign / redirect requests using path-based mechanisms or in conjugation with CDN

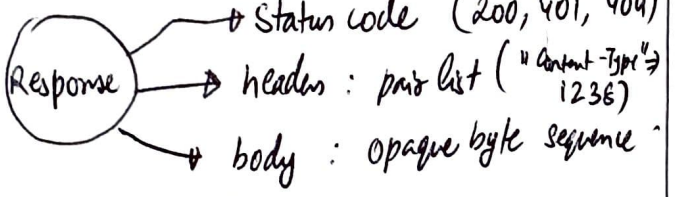
- System of load balancers can prevent load balancers from congestion or scale horizontally to handle the incoming requests.

Security and HTTPS

IP Packet → IP Header + Payload







Footprint

HTTP	Encryption	HTTPS
↓	↓	↓
MITM Attack →	Symmetric (1 key)	TLS (Handshake)
	Asymmetric (2 keys) →	SSL (Certificate)

HTTP → Communication Protocol. Abstraction on top of TCP.

↳ Communication b/w client & server using HTTP can have security vulnerability (some 3rd party actor → known as, Man In the Middle Attack (MITM))

Soln

HTTPS ⇒ Secured HTTP using Encryption Scheme.

Encryption

- Encrypting the data being exchanged b/w the client & server architecture.
- Family of Encryption ⇒ Symmetric & Asymmetric

Symmetric Encryption

- Requires merely 1 key.
- Makes use of Symmetric key algorithm.
- Symmetric key algorithms are type of cryptographic hash function that relies on a single key for both encryption & decryption
- Uses "AES" (Advanced Encryption Standard) specification to describe a symmetric key algorithm (ex: AES 256, AES → 128, 192)
- Symmetric Encryptions relies on a common key to be shared b/w client & server ⇒ Very fast encryption, decryption

  ↳ Disadvantage

  Requires key-sharing b/w the parties that may be vulnerable to theft.

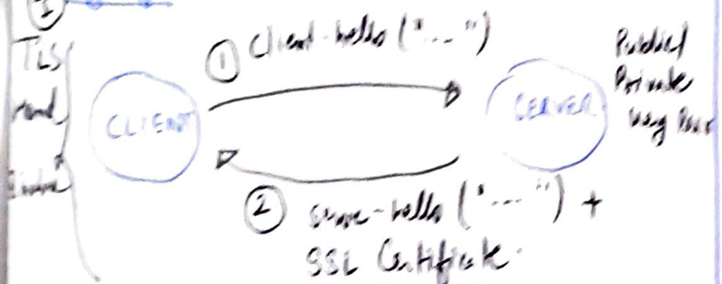
- Asymmetric Encryption
  - Relies on pair of keys for encryption & decryption (Public key for encryption & private key for decryption)
  - The public-private keys are bound mathematically s.t. if a message is encrypted using a public key, it can only be decrypted using a private key.
  - The public & private keys are generated together.
  - Disadvantages
    - Slower than symmetric encryption

HTTPS

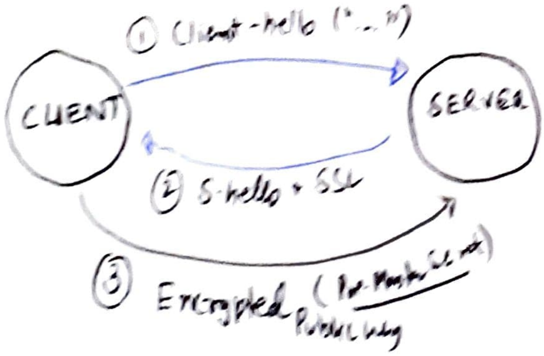
- Secured communication channel b/w client & server. Abstraction on top of HTTP & runs on top of TLS protocol (handshake)
- TLS (Transport Layer Security) → Security Protocol
- SSL (Secure Sockets Layer) → Security Protocol
  - ↳ Predecessor of TLS

— TLS (handshake) establishes a connection b/w client & server ⇒ using a random string of bytes

I



II Client generates the pre-master secret & encrypts it with the received public key (from server) & sends it to server.



III Both client & server generate symmetric encryption keys (session keys ~ around 4)

↳ Both of them possess:
C-hello, S-hello, Pre-Master Secret, Public key

The symmetric encryption key generated for session are used for the total length of the session (total of 4 keys are generated)

IV Confirmation of successful TLS connection

At this point, both server & client send one final message confirming the successful establishment of the successful TLS handshake

V Role of SSL Certificate

- SSL Certificates are issued by trusted third-party (CA - Certificate Authority)
- It imposes trust on the public-key shared by the Server (step 2)
  - ↳ Digitally signed certificate legitimizing the public key ownership to the intended server (sometimes, owned by server)
  - Signed by the private-key of CA.
  - SSL Certificates are crucial defense against the man-in-the-middle (MITM) attacks.

Symmetric Encryption Keys (SES) generated in Step 3 are using :-

1) Client-hello ⇒ Str. of random bytes
2) Server-hello ⇒ "
3) Premaster secret ⇒ Encrypted (Str. of Random Bytes) (Public key)

- SES Keys (~4) are used for both encryption & decryption during the full session length.

Advanced Distributed Systems Concept.

Replication & Sharding

Replication

- Act of duplicating data from one DB server to another. Can improve system redundancy & fault tolerance.
- May also be used to move data (or, requests) closer to the client to reduce access latency.

Sharding (Data Partitioning)

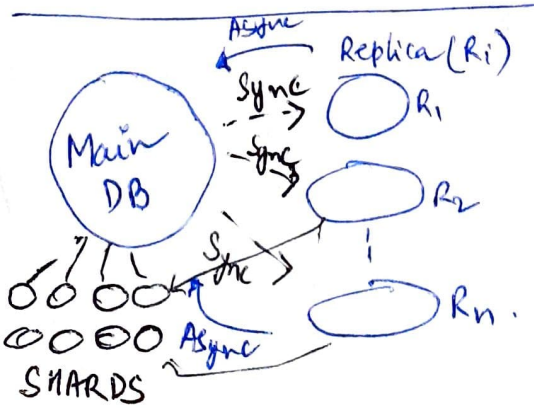
Act of splitting the DB into multiple parts (called shards) to increase DB throughput.

Popular Sharding Strategies include ⇒ (Consistent Hashing)

(A) Sharding based on the hash of the column
(B) Sharding based on clients region
(C) " based on data-types (user data in one shard, payment data in another shard)

Hot Spot

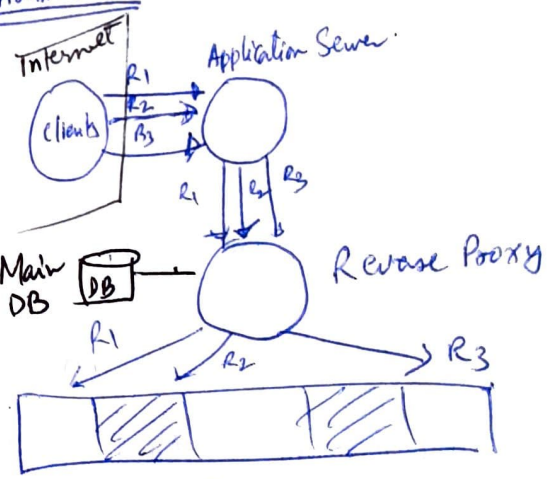
When sharding keys / hashing functions are suboptimal, the workload may be naturally skewed ⇒ Some servers receive more traffic than the rest → creating a "hot-spot".



Role of Replica

1. Standby of primary DB (as a secondary)
2. Update to replica may happen using sync/async.
3. Any failed transaction on ~~Master~~ Master should be well reflected on Replica. (Achieved using Sync operation)
   ↓
   Both DBs are consistent with each other
4. To avoid higher overheads, Async update to replica may be well suited → at the cost of inconsistency b/w Main & Shared DBs.
5. When DBs are scaled horizontally (Shards), each main DB has its own replica DBs.

Architecture



- Application servers can as well perform as a logic engine to redistribute traffic to the shards, however using a Reverse Proxy helps in efficient scaling & distribution.
- Reverse Proxy (acts on the behalf of Main DB)

Peer-To-Peer Networks *

Client-Server Model

Paradigms by which modern systems are designed on the internet.

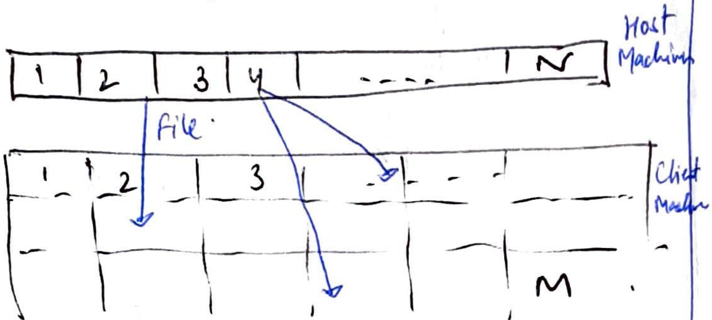
Peer-to-Peer Network

- Popularly used in file-distribution Systems.
- Sets of machines dividing the task among themselves to complete the workload faster or make the execution feasible.

Use case

Say, we want a webserver (CICD) pipeline to move files across machines. Requirements ⇒

- 40 Gbps (5GBps) network throughput
- 5 GB files. (1 GigaByte = 8 Giga Bits)



With 5 GBps network throughput, the avg. time required to transfer 5GB file to 1000 machines = $\left(\frac{5GB}{5GBps}\right) \times 1000 = 1000s \approx 17$ mins (too long!)

↓ with Scheduler interval of 15 mins, we miss SLA

Soln. 1 ⇒ Multiple Host Machine

Instead of single machine transferring files, we can have a cluster of 10 machines.

total times required = $\frac{17 \text{ mins}}{10 \text{ machines}} \approx 1.7$ mins (not great!)

↳ Plus, space wastage of copying these files to 10 machines.

Soln. 2 ⇒ Sharding

On a cluster of host machines, we only keep a fraction of the original 5GB file, thereby saving space → Host Machine Congestion, Split Ordering, Consistency, Fault Tolerance.

Best Soln ⇒ Peer-to-Peer Network (Peer → Cluster of Machines)

↳ The machines are peers which work together towards a common goal.

Arrangement:

A Single large machine acts as a host, & sends the split/chunk of the original 5GB file to each of the client machine (Peer-to-Peer Network), which build up/communicate their split to all the other machine.

Thay way, a single machine receives 5MB from the host, & the missing chunks from all the other machines in the network.

Eventually, each machine will have all the chunks to create the final 5GB file.

Total time required = 0.001s (5MB from host) + 999 (0.001s)

$\frac{5MB}{5GBPS} = \frac{1}{1000}$

≈ 1s

Algorithms Used

(I) Peer Discovery ⇒ Central Tracker Machine / Gossip N/W

(II) Peer Selection ⇒ Gossip N/W Protocol

↓ every peer has mapping of what chunks other peers have (Distributed Hash Table)

Peer to Peer Network applications are ubiquitous & widely used in deployment engineering.

Gossip Protocol

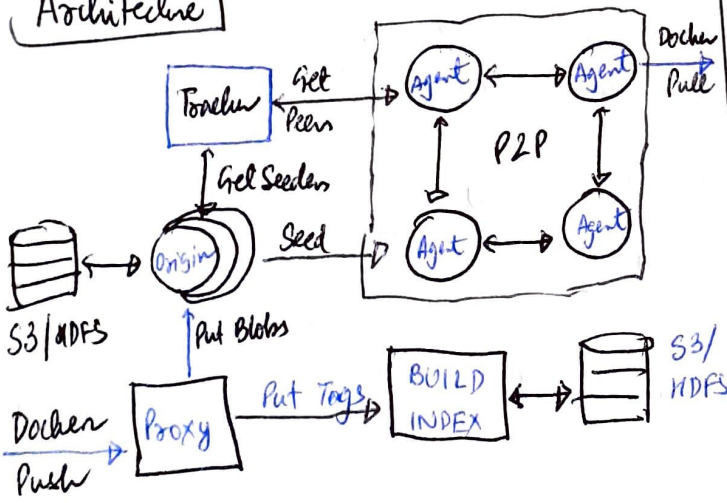
When the machines on a peer-to-peer network communicate with other machines to spread information through a system without any central teacher node (uncoordinated manner)

Example of Peer-to-Peer Network

Uber's Kraken

- P2P powered Docker registry focussing on scalability, availability.
- Used for distributing billions of blobs per day
- Designed for docker image management, replication & distribution in a hybrid cloud

Architecture



Configuration

① JSON (JavaScript Object Notation)

{ "version": 1.0, "name": "AlgoExpert Conf" }

↳ APIs, Configuration

② YAML (Yet Another Markup Language)

- Used in Configurations

```
version: 1.0
name: Algoexpert Configuration
```
'yaml'

③ Key-Value Store (Flexible NoSQL DB)

- Etcd, Redis, Zookeeper, DynamoDB
- Usage: Caching, Dynamic configuration

Configuration

- Set of parameters / constants critical to syst
- Static (shipped w/ system's application code)
- Dynamic (lives in the environment, the syste is deployed in)

config.json

```
{ "apiKey": "_",
  "__": false,
  "languages": [
    "cpp",
    "java",
    "python"
  ],
  "version": {
    "num": 0,
    "date": "2020-10-11"
  }
}
```

config.yaml

```
apiKey: ____
bodcastAll: false
Supported langs:
- cpp
- java
- python
version:
  num: 0
  releaseDate: "2020-10-11"
```

Static Config.: Constants bundled w/ the main application code

Dynamic Config.: Constants set in the environment / run-time (more power, flexibility)

∅ Dynamic configuration necessitates complex service tools to ensure that the change in configuration file doesn't break the entire feature/system. Also, it requires extensive testing to capture all combinations & its impact on the final sub systems.

Polling and Streaming

Sockets ⇒ Allows two-way communications b/w long-lived different processes on the same/different communication machines.

A Unix socket is used as client-server application framework to process client request as a stream.

FTP, SMTP, POP3 makes use of socket to establish connection b/w the client & server & facilitate two-way data exchange.

Polling

Fetching data / initating requests at a specified interval.

Streaming

Fetching continous updates from an open connection b/w two streams / processes.

∅ Streaming may add a lot of computational overhead in exchange for reduced length of staleness in the data.

∅ Also called Pushing / Listeners. As the server constantly pushes updates to the client through the open connection / long-lived connection.

∅ Both Polling & Streaming necessitates necessary tradeoff. None is inherently better than the other.

Rate limiting

Act of limiting the no. of requests to/from a system. Often used to limit the no. of incoming requests to prevent DoS attacks.

Mechanisms

1) Can be enforced at the IP-address level, at the user-account level, or at the region level.
2) Can be implemented in tiers: Network requests limit to time intervals (1s, 1min, 1hr)

DoS (Denial of Service) Attack

Malicious user/service attempting to bring down or damage a system. Common technique is by flooding the webserver w/ traffic.

DDoS (Distributed Denial of Service) Attack

when the originating traffic flooding a webserver comes from multiple sources.

- Such attacks are much harder to defend against

Redis (in-memory key-value store) can be popularly used for implementing rate limiting.

- Rate limiting can be used to safeguard the system SLA (throughput).
- DDoS attacks are difficult given the inseparability b/w "real" user & "malicious" user.
- Rate Limiting on distributed systems is achieved using a central Redis engine that all the applications query to.

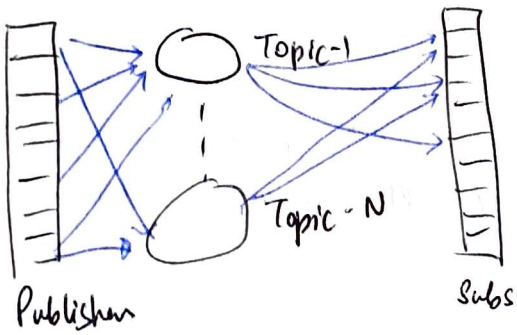
In industry standards, the rate limiting are coded w/ complex business rules (3 per user req. per sec, 'x' req per team in IS, etc.)

Publish/Subscribe Pattern

Pub/Sub ~ (Produce/Consume, Push/Pull, Send/Receive, Throw/Catch)

Pub/Sub Model

Publish/Subscribe pattern is a popular messaging model composed of publishers & subscribers.



A publisher can write to multiple topics, while the consumer can consume from multiple topics.

Salient Features

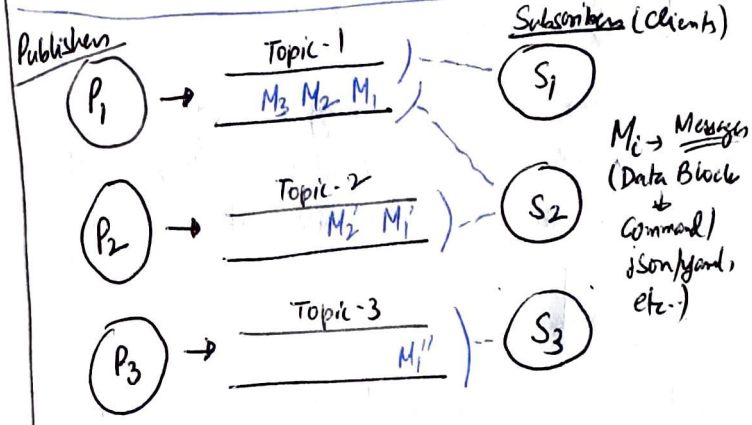
1) Powerful gaurantee on:
   (A) At least once delivery
   (B) Persistence
   (C) Message Ordering
   (D) Replayability of messages

2) Pub/Sub systems are Idempotent ⇒ since a single message can be consumed multiple times.

Pub/Sub Systems

(1) Apache Kafka ⇒ Distributed messaging system created by LinkedIn ⇒ Streaming paradigm

(2) Google Cloud Pub/Sub ⇒ Gaurantees at least once message delivery & supports "rewinding" in order to reprocess messages.



Diff b/w Pub-Sub & Streaming

1) PubSub offers a distributed, reliant streaming paradigm offering fault-tolerance & separation b/w storage & compute.

2) Further, Pub/Sub pattern abstracts server-client communication with topics acting as intermediaries.

3) Zookeeper nodes are used for maintaining indexes of read topics in the message queue.

4) Exactly once delivery is impossible to achieve in the distributed systems → Feasible (Atleast once)

5) Atleast once also adhere to the Idempotent operations ⇒ multiple read operations won't change the read-status/messages.

6) Pub/Sub systems gaurantee ordering within the messaging queue (partition) ⇒ Replayability/Time-Order Partition.

7) Why multiple topics?
   Support strong schema adherence & scalable to heterogenous systems.

Topics ⇒ offer separation of duty (schema gaurantee)

8) Sharding across messaging queues may also offer advanced auto-scaling solutions.

9) End to End encryption can be efficiently achieved at the messaging-queue end.

API Design

ACL ⇒ Access Control List

- Defines the permissioning model - which users are allowed to perform which operations
- ACLs control read, write, edit level controls

Pagination

- APIs designed to limit the response content, accompanied with an identifier / token for the client to request more contents begining at the current offset.

CRUD Operations

- Bedrock of any client-server model.

- API Design is a (Sibling) of Systems Design.

Very popularly Interview questions.

N.B.

- Web based products & services are heavily backed by efficient API design & implementation
- API offers neat access pattern for backend functionalities.
- Design choices with APIs reflect efficient engineering & impact the backend system performance.

Limit

① Limiting the length of the response code

API Design for Interviews

① Clarifying questions?

② Understanding the boundedness of the API design

③ Formulate the API outline → ~~resources / attributes~~ resources/entities, properties/attributes, response structure, System design concepts baked into the API.

④ API design template

API Definition
↳ Entities = Charge, Card, Customer

Endpoint Definitions

For each (entity):
Define the entities,
(Pagination Support)
Filtering Support
Limit Support.

POST /v1/charges
GET /v1/charges/:id
POST /v1/charges/:id/capture
GET /v1/capture

Swagger ⇒ API definition language (json/yaml)

Response Code (Enumerate Statuses)

Paths ⇒

Exercises

① API Design Question
② API documentation → ex: Swagger Description Language.
③ OAuth.

Additional API Skill

① Learn OAuth (v2)
② Filtering, Limit & Pagination Supports

Logging & Monitoring

Logging

- Persistence of the audit trails of events occurring in the system.
- Typically, programs output logs messages to the STDOUT or STDERR pipes, that automatically gets aggregated into Centralized Logging Solutions.

Monitoring

Establishing visibility into a system's key metrics, API, performance, bottlenecks.

Alerting

Polling | Listener based communication on events of interest → {Multiple Sources} : {Multiple Sinks}

Logging

- Capturing the trace of the program flow & being able to query the logs using some DB.
- Google Stackdriver ⇒ Centralized Logging Solutions
- Scalability, low Computational Overheads.

Monitoring ⇒ Metrics on top of logs to learn System behavior & API conformity.

↳ Time series log DB offer more robust log monitoring solutions.

Alerting Solns ⇒ Create Sinks for subscribed log-based events.

Leader Election

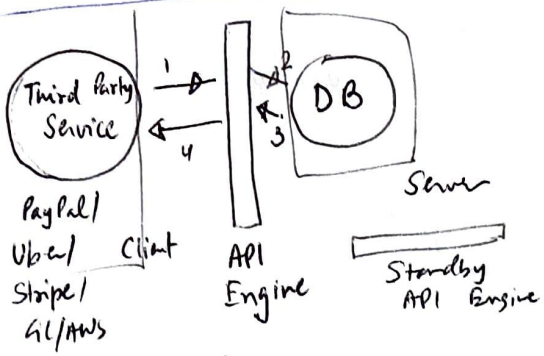
- Process governing the selection of leader nodes from the worker nodes that in turn support the primary operations that the nodes are collectively trying to achieve.
- All the worker nodes at all point in time are aware of their leader node.

Consensus Algorithm

- Algorithm governing agreement of multiple nodes on a common decision point, like leader selection, etc.
- Paxos & Raft are popular consensus algorithm

Etcd ⇒ Popularly used to implement leader-election in a system.
Strongly Consistent, Highly Available (HA) key-value store.

ZooKeeper ⇒ Perform leader election, store important configurations. Strongly Consistent, HA key-value store.

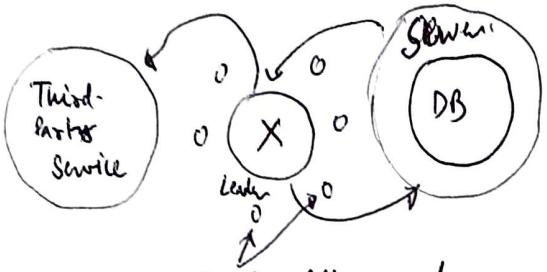


(1) Fault-Tolerance w/ API engine is achieved using redundancy ⇒ cluster of API engines responding to user requests.

(2) One big problem is which API engine node in a cluster should respond to user.
This is where leader election comes into picture.

Each follower node at any point in time are always aware of their leader.

③ So, one leader node that takes care of the business logic ⇒ The other nodes are in standby mode which may become leader when the leader goes down, etc.



Q How does leader election takes places?

Ans ⇒ Using consensus algorithm (Paxos, Raft) or a service implemented on top of them (Zookeeper, Etcd)

Etcd ⇒ Strongly consistent, highly available
↳ Implements the Raft algorithm

⇓

Supports leader election

- Leader constantly sends the heartbeat to the zookeeper/Etc with a lease acknowledgement Criterion (business rules, specified intervals, etc)

⇓

If the zookeeper/Etcd node sees the leader violating the leader lease (business rule) fresh election takes place to elect new leaders.

↳ Following which, new leader w/ new lease is created.

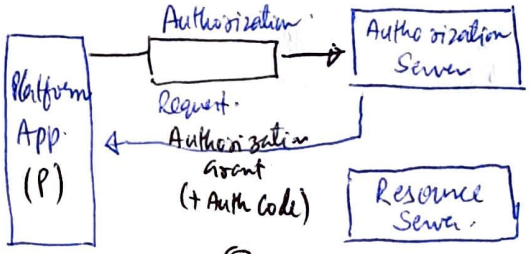
OAuth 2.0

- OAuth 2.0 is a technology to authorize clients in a client-server paradigm → easy, simplified
- Roles in OAuth 2.0 (Simplified)
  - User
  - Application
  - API → Authentication Server
    → Resource Server

For example, when an user logs onto Spotify, the application is Spotify while the api is FB/Google (or, whatever authentication medium is chosen).

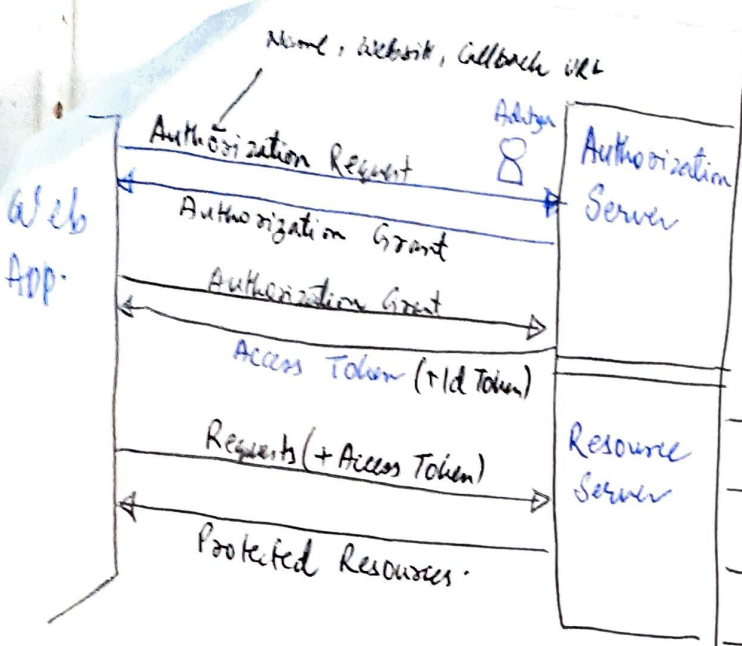
Workflow of OAuth 2.0

Facebook Auth Engine.



Aditya (User) ⇒ wants to login to "P" using credentials from fb.

- When an user grants access to an API ("P") to authenticate, actual passwords are never shared. The Authentication engine shares only the Authorization Code (Auth Code) b/w the authorization server & the platform application ("P")
- The Authorization Server, then grants "P" the access token (Acc Token) that is used to request resources from the Auth Engine's Resource Serv.



N.B.

① OAuth 2.0 acts as an authorization framework

② Actual authentication is performed by OpenID connect (which through ID tokens passed alongside the Access Token).

Grant Types in OAuth 2.0

① Authorization Code Grant
Popularly used w/ webservers

② Implicit Grant

③ Password Grant

④ Client Credentials Grant.

Memory Leaks

For programs without a robust garbage collector, not actively cleaning up previously unused memory can be detrimental to the system.

Type	32 bit	64 bit	Extra
int/long	14	30	+2 · (No. of digits)
float	16	24	+2 · (Length)
Str/complex	28	52	
tuple	24	64	+4 · length → 32 bit +8 · length → 64 bit

>> sys.getsizeof(obj) # size in bytes
↳ calls __sizeof__()

Object Interning Property

General Rule ⇒ Objects are allocated only on assignment; Variables just point to object (they don't hold memory).

Object Interning Property ⇒ 1) Frequently used objects are pre-allocated & shared, instead of costly new alloc.
2) Mainly due to Performance Optimization

String Interning ⇒ 1) Method of storing only one copy of each distinct String value, which must be immutable.
2) May improve performance on dictionary lookup (key comparison w/ a pointer over a string compare).

Mutable Containers Memory Allocation Strategy

List, Sets, Dictionaries allocate more memory blocks than their single instantiation, & further reserve more blocks when the need be. Shrinks when overallocation threshold is reached.

Advantages :

① Reduced expensive function calls with realloc(), memcpy() → everytime a new object is added to them ⇒ Free Append

Ex: List

→ Overallocation for list growth (by append)
↳ (4, 8, 16, 25, 35, 46, ...)x

→ For large lists, less than 12.5%

→ cheap operations → end of list
Expensive : beginning / middle (requires memory copy or shift)

→ List Allocation Size
32 bits → 32 + (4 · length)
64 bits → 64 + (4 · length)

→ Shrink Strategy
When less than 1/2 of the list is allocated.

Dictionaries / Sets

Overallocation when 2/3 capacity is reached
Shrinks large if allocated memory is much larger than keys in it.

Garbage Collector, Reference Counts, Cycles

→ Objects are garbage collected when reference count goes to 0.

→ Also uses cycle-detection() to understand objects that are no longer needed.

Reference-Count Increment

- Object Creation
- Additional Aliases
- Passed to a Function

Reference-Count Decrement

- Alias is destroyed / reassigned
- Local reference goes out of scope.

Tools

Psutil, memory-profiler, Objgraph, Melia, Hea
gdb-heap

Psutil ⇒ Memory %, Memory Info. → RSS, VMS

Memory Profiler →
- Line by Line Profiler
- Memory usage monitoring
- Debugger breakpoints.