Contents

`

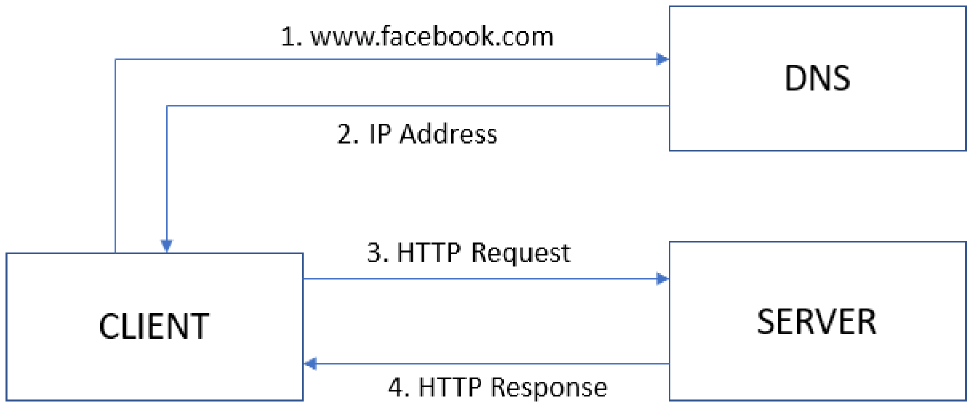
1. INTRODUCTION

Building scalable, production ready applications is both art and science. Science in that it requires knowledge of many topics in computer engineering; art, in that it demands an eye for making smart design choices and piecing together the right technologies.

To build a modern-day system, good understanding of below concepts is required.

- Basic Fundamentals like Client-Server model, Network Protocols and many more.
- Key characteristics of a system like Latency, Throughput, Availability and many more.
- Key components of a system like Load Balancing, Proxies, Caching, Rate Limiting, Leader Election and many more.
- Technologies used to build the system with key characteristics and key components. Those technologies include Redis, ZooKeeper, Amazon S3, GCP and many more.

2. CLIENT – SERVER MODEL



- Client is a computer or machine that speaks or requests data from server.
- Server is a computer or machine that listens to the client and speaks back or responds to the client.
- Client does not know what the server represents, rather the client requests for information or data.
- Client makes DNS query to find out what is the IP address of the server. DNS query is a special request to the predefined set of servers to find the IP address of a server.
- IP Address is a unique address of a computer connected to public internet.
- **H**yper**T**ext **T**ransfer **P**rotocol is a way to send data to the server so that it can understand.
- Server listens to request from client on specific ports. Usually, HTTP requests are served on port 80 and HTTP(S) requests are served on port 443.

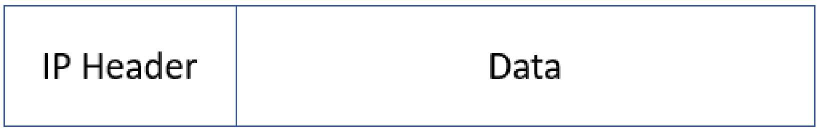
3. NETWORK PROTOCOLS

- Low level network concepts are essential to understand how machines in a system communicate with one another.
- In simple terms, protocol is agreed upon set of rules for an interaction between two parties.

- In context of networking, computers interact with one another following network protocols.

3.1 INTERNET PROTOCOL

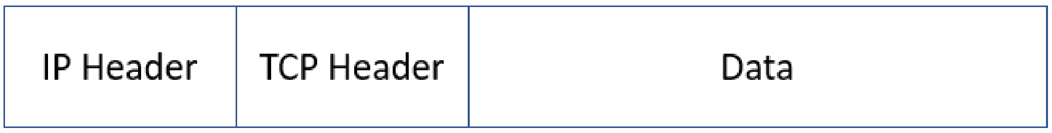
- **I**nternet **P**rotocol defines how almost all the machines should communicate with one another in the modern internet world. When two machine interacts, data is transferred in the form of IP packets.



- The size of the IP header would be 20 bytes to 60 bytes and it contains useful information like source IP address, destination IP address, size of data and version of IP address.
- The size of the data is 2^16 bytes.
- The disadvantage of IP is that while transferring multiple packets, there is no guarantee on sending all the packets and order of packets.

3.2 TRANSMISSION CONTROL PROTOCOL

- **T**ransmission **C**ontrol **P**rotocol is built on top of Internet Protocol. TCP guarantees the ordering of packets and no loss of packets, at least information on packet loss is known.
- Core idea behind TCP is client establishes a connection with server through a handshake process where one computer interacts with another computer.



- Handshake process helps in identifying connection timeout or server informs the client by sending a message that server wishes to end the connection.
- The TCP header contains information like ordering of packets.

3.3 HYPERTEXT TRANSFER PROTOCOL

- **H**yper**T**ext **T**ransfer **P**rotocol is built on top of Transmission Control Protocol. It is a request/response paradigm where one computer requests and another computer respond.
- Requests and responses are usually objects and follow the below schema
    - Request objects contains host, port, HTTP method, headers and body.
    - Response objects contains status code, headers and body.
- HTTP methods are GET, POST, PUT, DELETE, PATCH and OPTIONS.
- Headers are example like Content-Type: application/json.
- Status code are example like 200, 404 and so on.

4. STORAGE

- The systems require some form of storage and this is where databases come into the picture.

- Databases are programs that use either disk or memory to record data and query data.
- Thinking database is just a server, it retains data even in case of power outage. Even though it is true, in some cases it is not.
- Disk usually refers to either **H**ard **D**isk **D**rive or **S**olid **S**tate **D**rive persists data even in case of any database crash and is slower. Disk is also referred as non-volatile storage.
- SSD is faster than HDD and is more expensive.
- HDD is typically used where data is rarely accessed/updated but stored for long time.
- SSD is typically used where data is frequently accessed/updated.
- Memory usually referred as **R**andom **A**ccess **M**emory does not persist data in case of any database crash or server down and is faster.

5. LATENCY AND THROUGHPUT

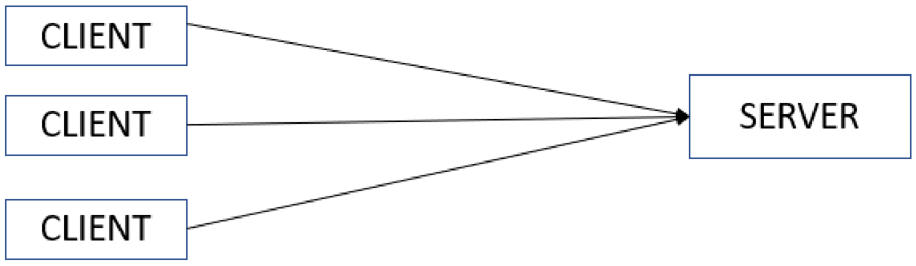
5.1 LATENCY

- Latency defines how long does it takes for data to traverse from one point to other point (client -> server -> client).

    Reading 1 MB from memory: 0.25 milliseconds
    Reading 1 MB from SSD: 1 millisecond
    Transfer 1 MB over network: 10 milliseconds
    Reading 1 MB from HDD: 20 milliseconds
    Inter-Continental Round Trip: 150 milliseconds

5.2 THROUGHPUT

- Throughput defines number of operations that system can handle properly per time unit.
- Usually, throughput is measured as gigabits per seconds or kilobits per seconds.



- Based on the above picture, the throughput of the server is measured as how many requests can be handled in a given amount of time.

6. AVAILABILITY

- Availability defines how resistant is a system is to failures. The availability is usually measured in percentages.
- The server that has 99% of availability will be operational 99% of the time. It is described as two nines of availability.
- The server is said to be highly available when it is 99.999% (5 nines) operational.

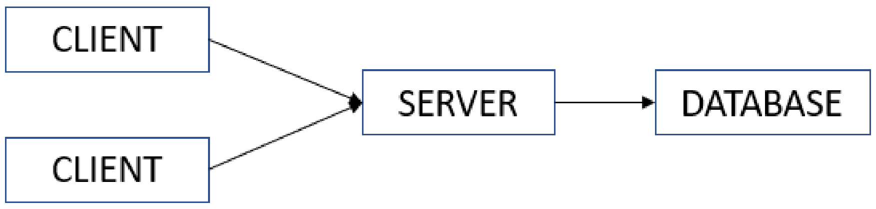
Availability	Downtime/Year	Downtime/Quarter	Downtime/Month	Downtime/Week	Downtime/Day
90% (one nine)	36.53 days	9.13 days	73.05 hours	16.80 hours	2.40 hours
99% (two nines)	3.65 days	21.9 hours	7.31 hours	1.68 hours	14.40 minutes
99.9%(three nines)	8.77 hours	2.19 hours	43.83 minutes	10.08 minutes	1.44 minutes
99.99% (four nines)	52.60 minutes	13.15 minutes	4.38 minutes	1.01 minutes	8.64 seconds
99.9999999 (nine nines)	31.56 milliseconds	7.89 milliseconds	2.63 milliseconds	604.80 microseconds	86.40 microseconds

Source: https://en.wikipedia.org/wiki/High_availability

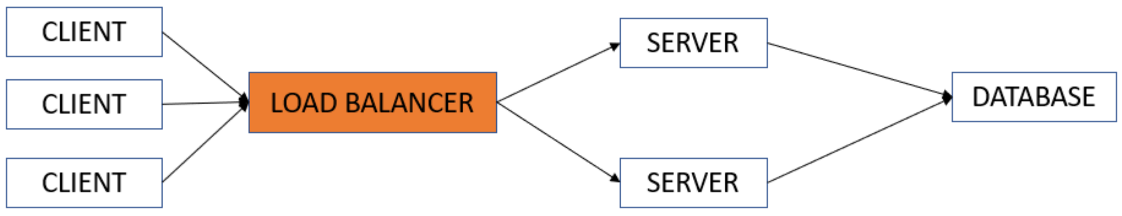
- **S**ervice **L**evel **A**greement is a collection of guarantees given to a customer by service provider. It typically makes guarantees on system availability and SLA is made up of one or more **S**ervice **L**evel **O**bjective.

6.1 REDUNDANCY

- Redundancy is duplicating critical parts of system to avoid single point of failure. Single point of failure is if one part of system dies, entire system will stop working.
- In the below picture, if the server is down, the entire system stops working thus server is single point of failure.



- Since the server is a single point of failure, the server can be duplicated by adding one or more servers.



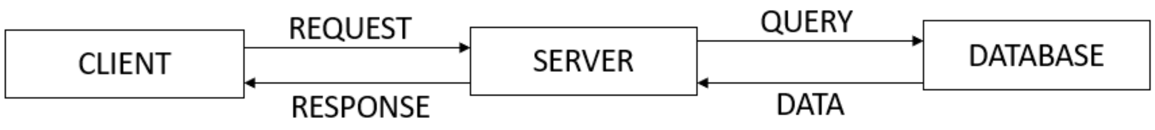
- In the previous diagram, the load balancer becomes the single point of failure and thus it can be duplicated by adding one or more load balancers. This type of duplicating critical components of a system which might become single point of failure is known as active redundancy.
- In active redundancy if one machine that handles traffic fails, the other machines will be able to continue smoothly without bringing the system down.
- There is another type redundancy called passive redundancy in which there will be a group of machines but only one will be handling the traffic and when that one machine fails,

somehow any one active machine from the group will know to reconfigure and takeover the responsibilities.

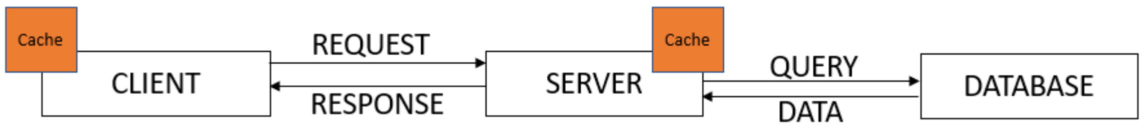
- The best example of passive redundancy is Leader Election and we would see it in action in later part of this document.

7. CACHING

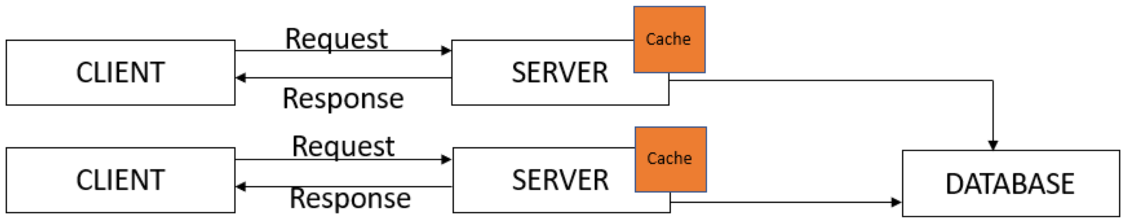
- In a typical system like below, the client makes network request to a server and server in turn requests for data from the database. The database responds back to the server and server in turn responds back to the client.



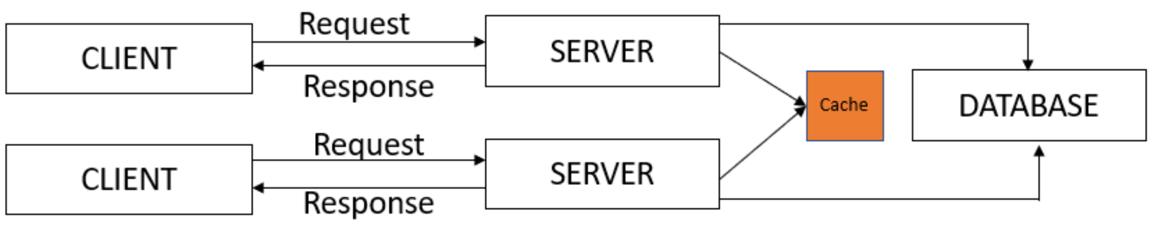
- In the above system, caching can be introduced either at server level or at the client level to cache the result of network requests to increase the speed of the system.
- Another instance where caching would be useful is to cache the result of computationally complex problems at the server level thus to avoid redoing the computations again.



- Write Through Cache is a mechanism where the data is recorded/updated in both cache and database.
- Write Back Cache is a mechanism where the data is updated only in cache and the system asynchronously updates the database in certain intervals. There is a possibility of losing data if the cache crashes.
- Cache mechanism when not designed properly, the server might respond back to the client with stale data. Let's consider we are designing comments section of Facebook.



- Assume both clients C1 and C2 reads the initial version of comments for a post from the database and the same is cached at both the servers S1 and S2.
- Now let's say client C1 respond to one of the comments and the same is updated back to the cache at server S1 and not updated in the main source of truth which is database.
- Now the cache at server S2 is having a stale data and so clients reading the comments from server S2 will get an older version of comments which is a very bad design.
- This situation can be avoided using a centralized cache like Redis.



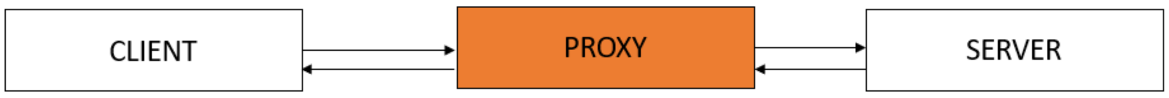
- Cache hit is when requested data is found in cache.
- Cache miss is when request data is not found in cache.
- Cache eviction is policy by which data is removed from cache and popular ones are **L**east **R**ecently **U**sed, **L**east **F**requently **U**sed, **F**irst **I**n **F**irst **O**ut.

8. PROXIES

- Proxies are nothing but servers which acts as a gateway between the clients those are making requests and servers those are responding back to client.

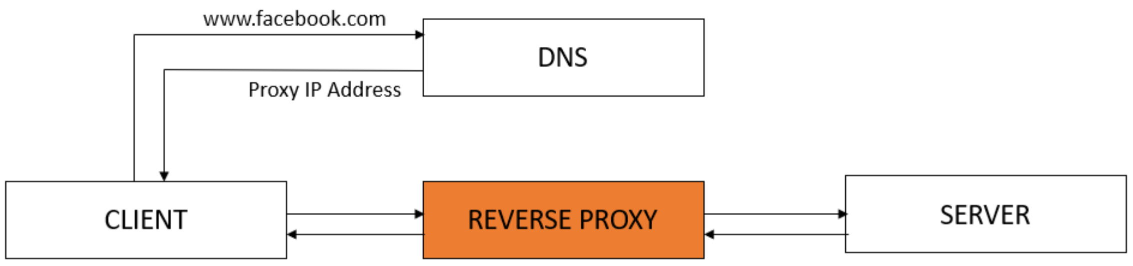
8.1 FORWARD PROXY

- A server which sits between the client and server and acts on behalf of clients.
- Hides the identity of client
- The source IP address is going to be a proxy IP address which is read by the server.
- Simply called as Proxy.



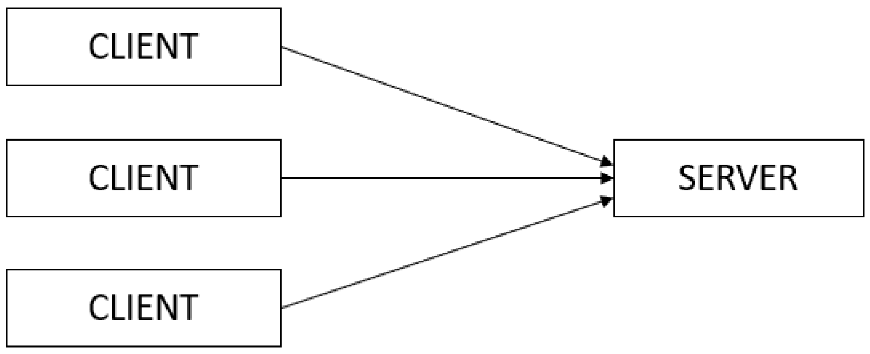
8.2 REVERSE PROXY

- A server which sits between the client and server and acts on behalf of servers.
- Requests from clients are received by reverse proxies and client does not know that requests are received by reverse proxies.
- The DNS returns back the IP address of reverse proxy when the client requests data from actual server.
- Used to filter out requests, logging and caching.
- Load balancers are usually reverse proxy.

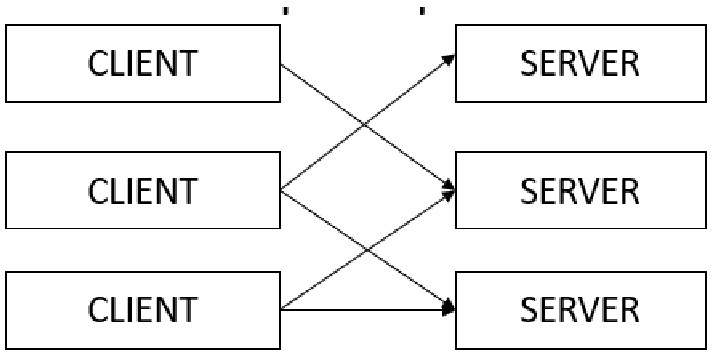


9. LOAD BALANCERS

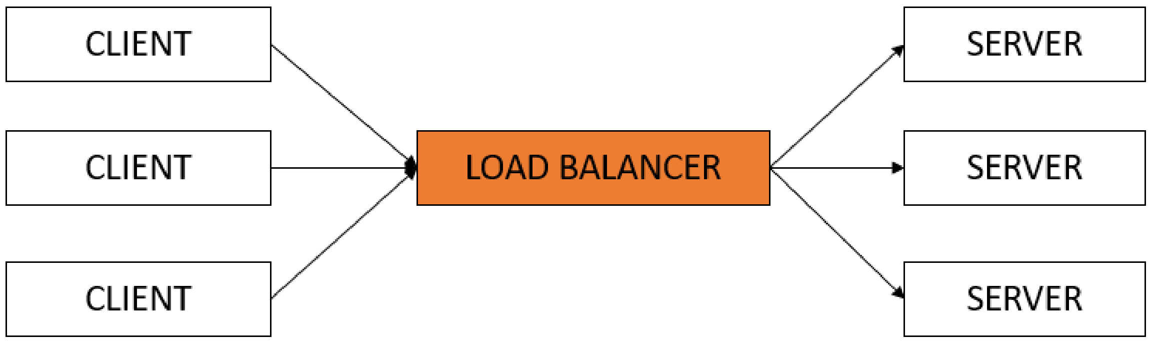
- Let us take a simple example like below,



- In the above case, the server might become overloaded due to handling multiple requests. To avoid this, vertical scaling (adding more power) is one option but this is also only to certain extent.
- Finally, the horizontal scaling is only option where multiple servers are placed to handle traffic from multiple requests.



- Even though multiple servers are placed, how can it be made sure that the load is evenly distributed or machine with more power is handling more requests and so on?
- This is where Load Balancers comes into the picture which sits between the client and server and acts as a reverse proxy.

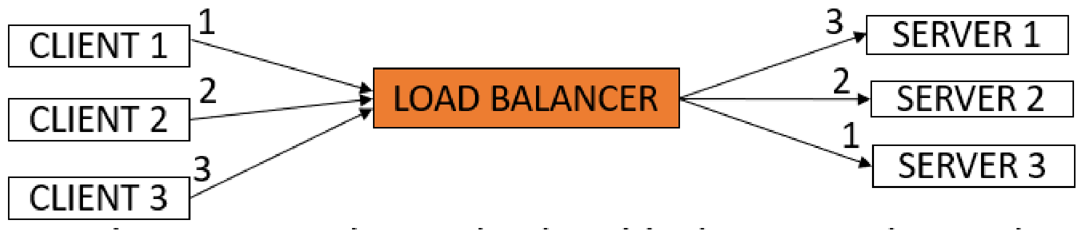


- How come load balancers knows the identity of server is, physical machines register or deregister with the load balancer. Example: Netflix Eureka.
- Nginx is a very popular load balancer.

- There are several server selection strategies by which load balancers selects the server and route the traffic to.

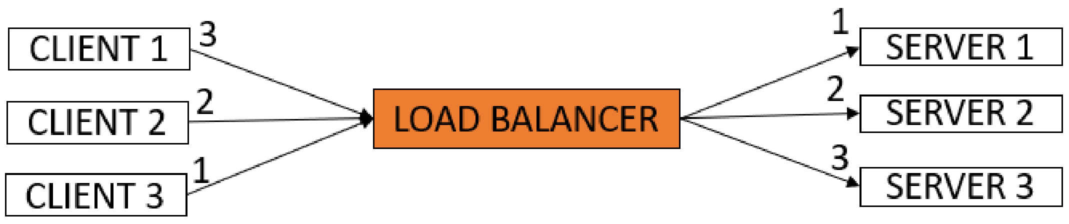
9.1 RANDOM SELECTION

- Random selection is where the load balancer selects the server to route the traffic to in a random order.



9.2 ROUND ROBIN SELECTION

- Round Robin selection is where the load balancer selects the server in a round robin manner where each server gets a turn to handle the traffic.



9.3 WEIGHTED ROUND ROBIN SELECTION

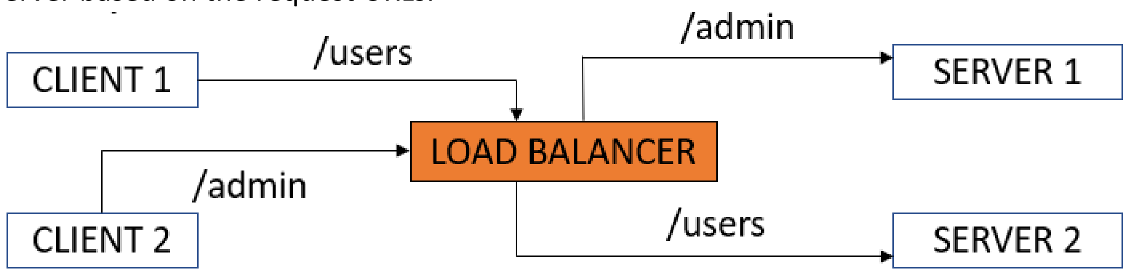
- Servers can have weightage based on traffic handling capacity. In Weighted Round Robin, load balancers will route more requests to servers having more weightage when compared to servers having less or no weightage.

9.4 PERFORMANCE BASED SELECTION

- Performance based server selection will allow the load balancer to route the traffic to the machines that is performing high. This is identified by doing health check at any given time.

9.5 PATH BASED SELECTION

- Path based server selection is where the load balancer routes the traffic to particular server based on the request URLs.



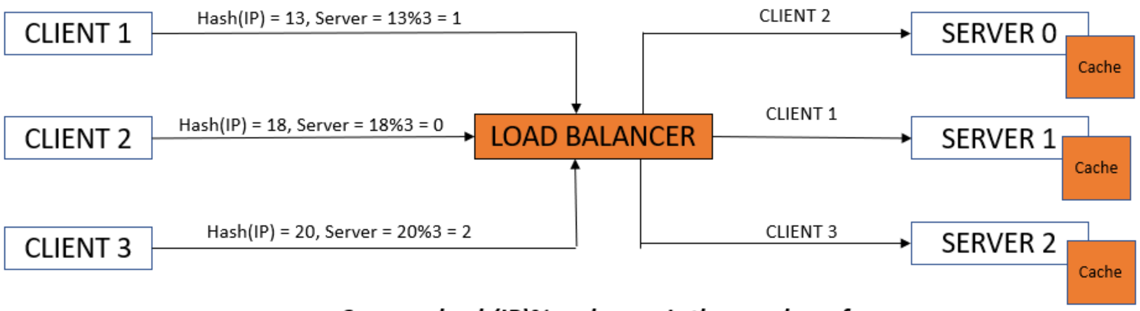
9.6 IP BASED HASING SELECTION

- IP based server selection is where the IP address of a client that is making request is hashed with the help of hash function and the below formula is applied. Hashing is covered in detail in the next section.

Server = hash(IP)%n   where n is the number of servers.

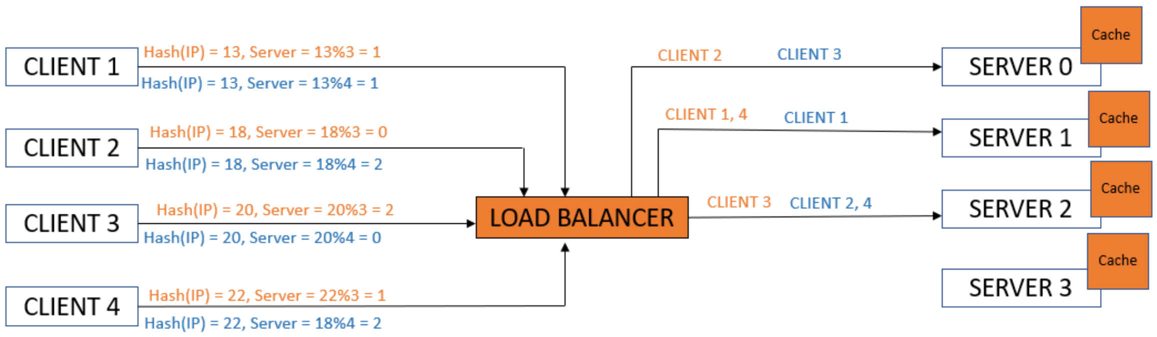
10. HASHING

- Hashing is the process of converting any arbitrary data into a fixed size data typically an integer. Hash Table is an example!



Server = hash(IP)%n where n is the number of servers.

- The above diagram represents IP based hashing. In this case, the client requests are always routed to same server.
- Since the requests are routed to same server, a particular client can always make use of cache thus increasing cache hits.
- This type of simple hashing with help of IP address or whatever leads us to another problem. What happens if a server dies or a new server is added?
- The moment a server is dead or new server is added, the logic to find the server which can handle the traffic has to be modified due to change in value of ‘n’, where n is number of servers.
- In the below diagram, with three servers, the mapping is C1, C4 -> S1, C2 -> S0, C3 -> S2 and with four servers, the mapping is C1 -> S1, C2, C4 -> S2, C3 -> S0.

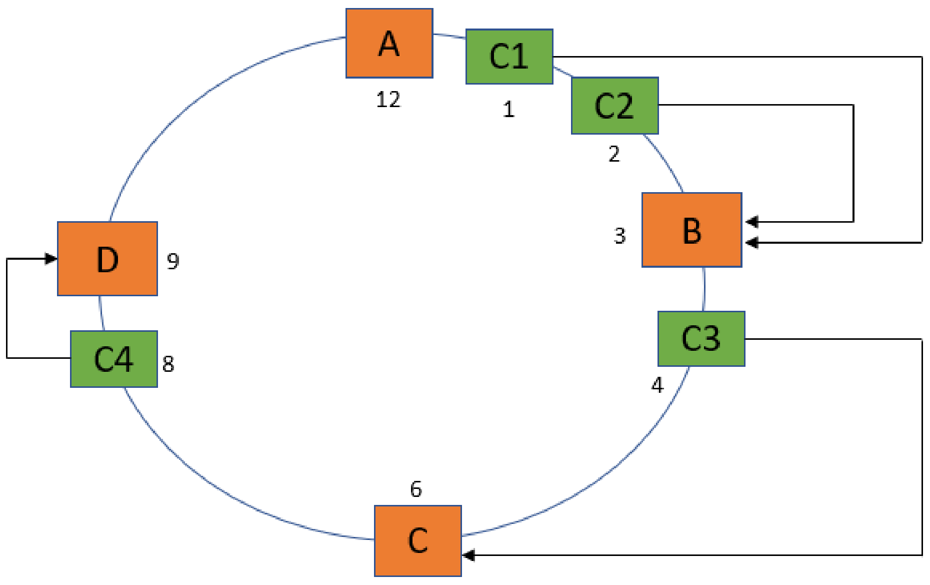


CLIENT 1
Hash(IP) = 13, Server = 13%3 = 1
Hash(IP) = 13, Server = 13%4 = 1
CLIENT 2
Hash(IP) = 18, Server = 18%3 = 0
Hash(IP) = 18, Server = 18%4 = 2
CLIENT 3
Hash(IP) = 20, Server = 20%3 = 2
Hash(IP) = 20, Server = 20%4 = 0
CLIENT 4
Hash(IP) = 22, Server = 22%3 = 1
Hash(IP) = 22, Server = 18%4 = 2
LOAD BALANCER
CLIENT 2
CLIENT 3
CLIENT 1, 4
CLIENT 1
CLIENT 3
CLIENT 2, 4
SERVER 0
Cache
SERVER 1
Cache
SERVER 2
Cache
SERVER 3
Cache

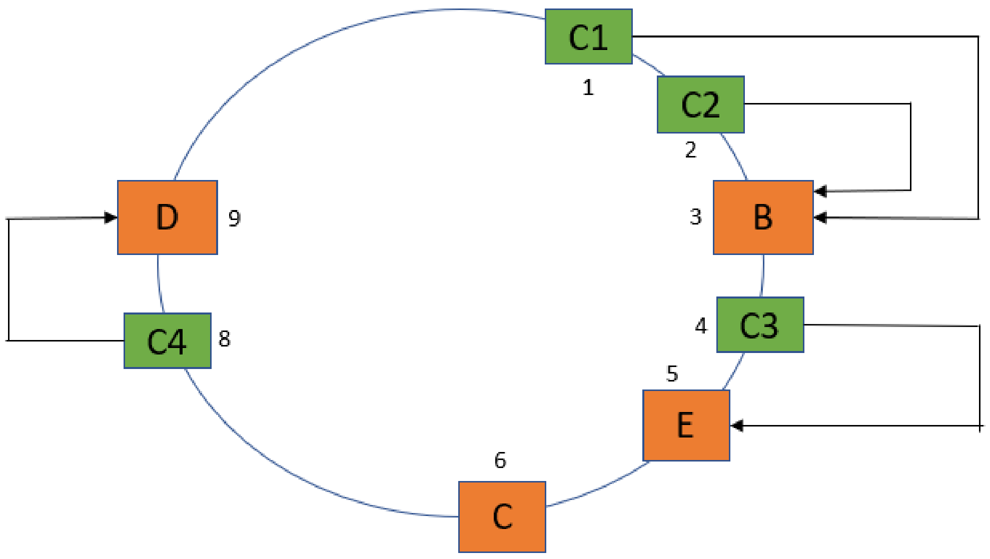
- As you can see, the moment a server is added or removed, all the mappings between server and client changes and this would lead to lot of cache miss and this is really bad in a system design.
- Consistent Hashing and Rendezvous Hashing are the strategies used to fix this problem.

10.1 CONSISTENT HASHING

- Consistent Hashing is a technique that helps in minimizing the number of keys that needs to be remapped when hash table gets resized.
- For a change, let’s consider we place servers and clients in the below circle using a hash function. The result of hash function which is a fixed size integer and the same would be a point in the circle. Let’s say our servers are A, B, C and D and clients are C1, C2, C3 and C4.



- In the above diagram, the clients are mapped to servers in a clock-wise direction for example and so that C1, C2 -> B, C3 -> C, C4 -> D. But how come this would solve the problem of remapping client and servers whenever a new server is added or existing server is down?
- Let’s say the server A is removed from the circle. What happens? Nothing will happen and the mapping would still be retained as follows C1, C2 -> B, C3 -> C, C4 -> D. This is because server A is not handling any client request
- Let’s take another scenario where a new server called E is introduced is between servers B and C.



- As you can see, all the mappings between client and servers are retained and only requests from C3 would be routed to E instead of C as we are moving in clock wise direction.
- In addition, when a server seems to be more powerful, it can go through multiple hash functions and can be positioned at many places in the circle thus handling more requests.

10.2 RENDEZVOUS HASHING

- Rendezvous hashing also known as highest random weight hashing which allows minimal redistribution of mappings when server goes down.
- Let's take there are clients C1, C2, C3, C4 and servers S1, S2, S3. The Rendezvous hashing follows the below algorithm.
  - ✓ Hash all possible client-server combinations with a random hash function.
  - ✓ Assign the client to the server with the largest hash value.
  - ✓ Maintain the first-choice invariant while adding and removing servers.
- Let's start assigning hash values for each client server combinations by applying a random function.

CLIENT	SERVER	HASH
C1	**S1**	**1421**
	S2	4
	S3	123
C2	S1	14
	S2	4
	S3	**72**

CLIENT	SERVER	HASH
C3	S1	2
	S2	**123**
	S3	72
C4	S1	12
	S2	41
	S3	**72**

- Now let's sort the servers based on hash value and apply first choice invariant.

CLIENT	SERVERS		
C1	**S1**	S3	S2
C2	**S3**	S1	S2
C3	**S2**	S3	S1
C4	**S3**	S2	S1

- Based on the above table, the mappings are C1 -> S1, C2, C4 -> S3, C3 ->S2.
- Now let’s remove server S3 and maintain first choice invariant.

CLIENT	SERVERS		
C1	**S1**	S3	S2
C2	**S3**	**S1**	S2
C3	**S2**	S3	S1
C4	**S3**	**S2**	S1

- By doing so, only the mapping for C2 is changed to S1 and C4 is changed to S2 and the mappings for C1 and C3 is retained.

11. REPLICATION AND SHARDING

- The system’s performance is often decided on how the database of the system performs. Optimizing the database improves the overall performance of the system.
- Having a database server as a single point of failure is a very bad design and it is very important to introduce redundancy for database servers as well.
- Replication is the process of duplicating the data from one database to other databases. This means it is possible to have n number of replicas for a main database.
- When a main database is having one or more replica, it is important to update those replicas as well whenever there is a request to write a piece of information in a main database.
- There are two ways to update the replica
- Synchronous
- Asynchronous

11.1 SYNCHRONOUS REPLICATION

- Whenever there is a write operation to main database, the replica is also updated immediately thus making the update synchronous.
- Synchronous updates are useful when the replica needs to take the control over main database in case of any failures. So, it is important to have the replica up to date with the main database.
- Systems having synchronous updates for replicas might experience high latency.

11.2 ASYNCHRONOUS REPLICATION

- Whenever there is a write operation to the main database, the replica is updated with the data only after specific interval of time, every one hour for example. This is asynchronous update.
- Asynchronous updates are useful in cases where it is okay to have such delays. Let us look into an example.
- Let us assume that there are two database servers with the main database located in USA and the replica being located in India.
- The main database would be updated immediately whenever the users from USA posts information and so that the users from USA would be able to view the posts immediately without any delay.
- Since the replica located in India would be updated after a specific interval, the users from India would be able to see the posts only after a delay.

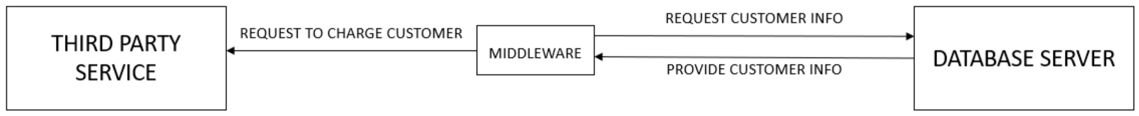
11.3 SHARDING

- Let us take another example where having n number of replica is also having a problem. Now we can assume to increase of the throughput of a system we are having 5 replica.
- Now all the 5 replicas would be having the same set of data and would be updated either synchronously or asynchronously. Is this an optimal way to do it?
- Instead of storing the same set of data across all replica, data can be partitioned and stored those data into Shards. It is called as Data Partitioning or Sharding.
- How data can be partitioned? There are three strategies,
- Shard based on client's region. Let us take a payment database, the payments from North America can be stored into one shard and payments from Asia can be stored into one shard and so and so forth.
- Shard based on type of data being stored. User data get stored into one shard and payments data gets stored into one shard and so on and so forth.
- Shard based on hash value of a column and only in case of relational databases.
- Picking up the correct Sharding strategy is important or else any one of the shards might become a hotspot. Hotspot is a component of a system which becomes overloaded due to lot of traffic.
- Let's assume that we have payments data and two shards and we are partitioning the data based on customer name. Customer name with starting letters A, S and K are stored into one shard and customer name with starting letters X, Q, Z are stored into one shard.
- It is obvious that shard which stores customer with their names starting with A, S and K will become overloaded.
- Partitioning the data based on a hash function that gives uniformity is one reasonable way to store equal amount of data in given shards.

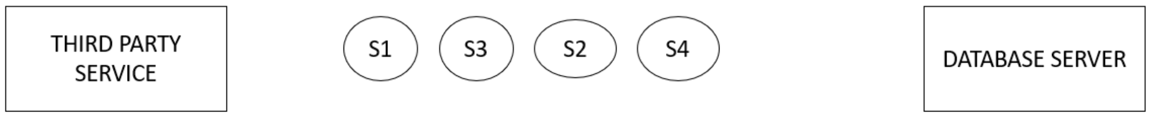
12. LEADER ELECTION

- Let us assume that we have a product that allows the customers to subscribe for the product on a recurring basis. Typical example is Amazon Prime or Netflix where customers are subscribed to the product on a monthly basis or annual basis.

- The product will have a database server which would store information like whether the customer is currently subscribed or not, date at which the subscription needs to be renewed, amount that needs to be charged, customer bank account or credit/debit card details and so on.
- There would be a third-party service which would actually debit funds from the customer accounts but how come the third-party service knows when and which customer account needs to be debited?
- Typically, there would be another service which sits between third party service and database server that would read customer information from database periodically and requests the third-party service to debit the customer account.



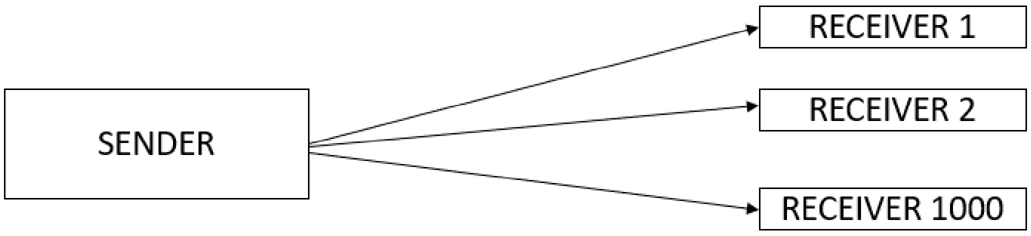
- But this system would impose a problem, what would happen the middle service dies? To avoid this redundancy would be introduced by placing four servers S1, S2, S3 and S4.



- Now there is another problem, in this system, the server in the middle requests the third-party service to charge the customers and this request should not be duplicated or should not be triggered more than once for a customer at a given point of time. But how this is ensured?
- Here comes Leader Election (Passive Redundancy) comes into the picture. It is the process in which all nodes in a cluster elects one node as a leader and that one leader node is responsible to do the task and re-elects another leader when the existing one dies.
- Paxos and Raft are mathematically complex algorithms used to have multiple nodes agree upon certain data value, the data value being the leader node among group of nodes.
- ETCD and Zookeeper are highly available and strongly consistent key-value store is used to implement Leader Election system and those two technologies in turn utilised Raft algorithm.

13. PEER – TO – PEER NETWORKS

- Let us assume that we are designing a system where one machine should transfer 5 GB file to 1000 of machines for every 30 minutes. One use case would be to transfer the video footage of security cameras from one machine to multiple other machines.
- Let us assume the network throughput of the machine that sends data is 5 GBPS or 40GbPS. With this throughput it would take 1000 seconds (~ 17 minutes) to transfer a 5GB file to 1000 machines.



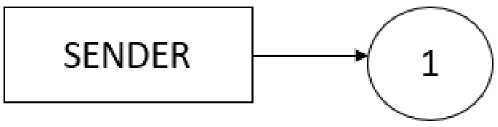
- Since this transfer is going to be on a recurring basis, 17 minutes for each transfer is really a bottleneck. How come this problem can be avoided?
- May be, by introducing redundancy, let's say there are 10 senders which would transfer the large file to 1000 machines.
- Again, there is a problem, first of all the speed of the data transfer operation is still not great which is ~1.7 minutes (17 minutes / 10 senders).
- Another problem is, do we need to really duplicate the large 5GB file across all those 10 senders? Necessarily not and also what happens if there are going to be 10,000 receivers in the future? Again, there is going to be a bottleneck.
- What about Sharding? Even though if the data is partitioned into multiple senders, again each of those 1000 machines have to contact each sender to receive the corresponding part and again there is a bottleneck of 17-minute operation.
- Here comes Peer – To – Peer networks into the play. A collection of machines referred to as peers that divide a workload between themselves to complete the workload faster than would otherwise be possible. It is often used in file distribution systems.
- To understand how Peer – To – Peer networks works internally, let us create smaller chunks of data that need to be transferred. Create 10 500 MB chunks from 5 GB file and now it would take 0.1 second to transfer one 500 MB chunk and 1 second to transfer the whole 5 GB file to a single machine. Note: It need not be necessarily 10 500 MB chunks, can be 1000 5MB chunks as well.

5 GB or 5000MB = 1 second, 500 MB = 1/10 = 0.1 second

- Now let us see how these chunks are transferred both in naïve solution and in Peer – To – Peer networks.

13.1 NAÏVE SOLUTION

- Let us assume, one sender machine wanted to send 10 500 MB file to one peer machine.

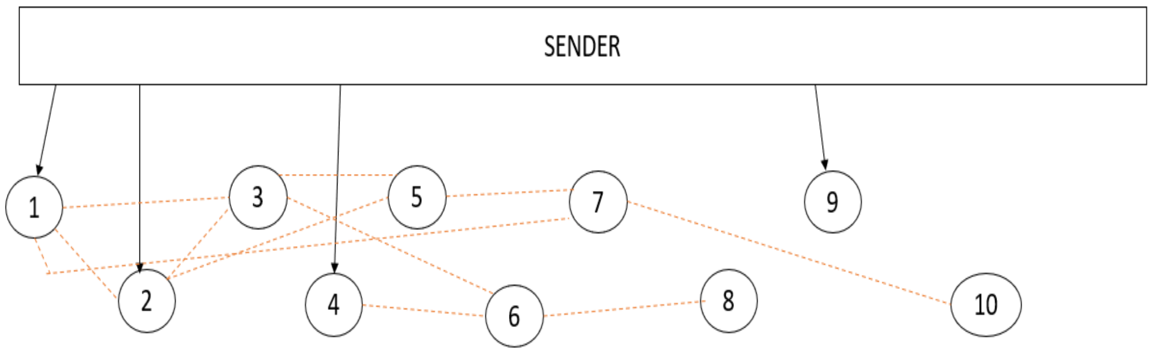


- In the 0.1st second, 500 MB is transferred to peer 1.
- In the 0.2nd second, 500 MB is transferred to peer 1. In total 1000 MB is transferred.
- In the 0.3rd second, 500 MB is transferred to peer 1. In total 1500 MB is transferred.
- In the 0.4th second, 500 MB is transferred to peer 1. In total 2000 MB is transferred.
- In the 0.5th second, 500 MB is transferred to peer 1. In total 2500 MB is transferred.
- In the 0.6th second, 500 MB is transferred to peer 1. In total 3000 MB is transferred.
- In the 0.7th second, 500 MB is transferred to peer 1. In total 3500 MB is transferred.

- In the 0.8th second, 500 MB is transferred to peer 1. In total 4000 MB is transferred.
- In the 0.9th second, 500 MB is transferred to peer 1. In total 4500 MB is transferred.
- In the 0.10th second, 500 MB is transferred to peer 1. In total 5000 MB is transferred.
- Finally in naïve solution, it took 1 second to transfer 5GB/5000MB to a single peer.

13.2 PEER – TO – PEER NETWORK SOLUTION

- Now let us assume, one sender machine wanted to send 10 500 MB file to 10 peer machines.



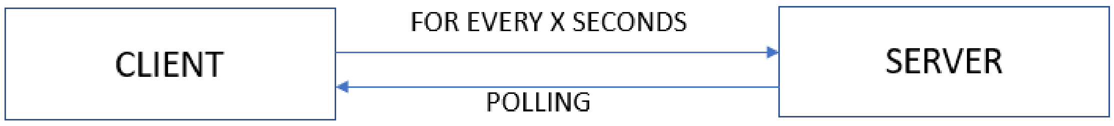
- In the 0.1st second, 500 MB is transferred to peer 1.
- In the 0.2nd second, 500 MB is transferred to peer 2. In the meantime, the peer 1 opens a connection and talk to peer 3 to send the chunk it has. So, in total 1500 MB is transferred.
- In the 0.3rd second, 500 MB is transferred to peer 4. In the meantime, the peers 1, 2 and 3 would transfer the chunk to other peers as well. In total 3500 MB is transferred.
  - 1st peer sent the chunk to 3rd peer and 7th peer.
  - 2nd peer sent the chunk to 5th peer.
  - 3rd peer sent the chunk to 6th peer.
- In the 0.4th second, 500 MB is transferred to peer 9. In the meantime, peers those have chunks would transfer to other peers. In total 7500 MB is transferred.
- As you can see, at each second, the data transfer is growing exponentially. Exactly at 1 second, it is possible to transfer 5GB/5000MB data to all 10 receivers. This is the power of peer-to-peer networks.
- Peer – To – Peer networks to function properly, the peers in the network should somehow know to which peer it can talk to next. There are two ways to achieve it.
- Have a centralized database or tracker which tracks which chunk the peer has and which chunk a peer needs and the peers in network can make use of tracker to know to which peer to talk and get the chunk.
- Instead of having a centralized database, the peers somehow configure themselves and talk to each other and get the data it needs. This is called as Gossip protocol or epidemic protocol.

14. POLLING AND STREAMING

- Let us take a use case where the client requires a piece of data that change regularly, for example: temperature monitoring system that would change frequently.

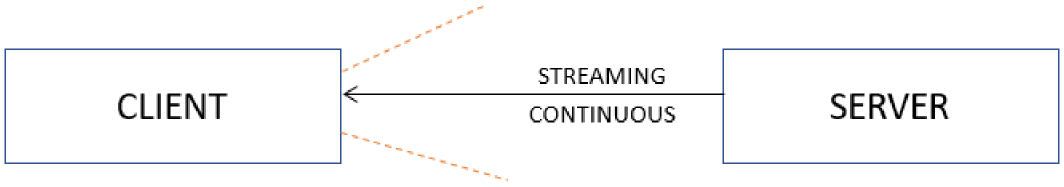
14.1 POLLING

- Polling is the process in which client is going to issue a request for data on a recurring basis. This would give users a bad experience since it is not be instantaneous. When polling is implemented in a messaging app, the users have to wait for x seconds to receive the message from the other end. Reducing the interval will be instantaneous but becomes over load.



14.2 STREAMING

- Streaming is the process where client opens a long-lived connection with the server, basically a socket.
- Socket is a file, a computer can write to and read from to communicate with another computer.
- In a long-lived connection, server will push the data to the client and client is streaming the data. Clients will not request for data rather listens.
- Depending on the use case, polling or streaming might be superior then one another.



15. CONFIGURATION

- Configuration is very simple but very important to understand. It is set of constants that the system is going to use which will configured elsewhere in a JSON/YML/PROPERTIES format.

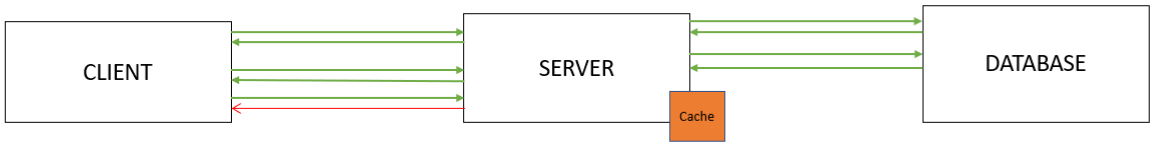
- Static configuration is typically bundled with application code. Advantages of having static configuration is safer as it goes throw code review and disadvantage is it is slower as the application code needs to be deployed whenever there is a change in constant.

- Dynamic configuration is typically separated with application code. Advantage is changes would get effect immediately and disadvantage is not safer as there is no code review.

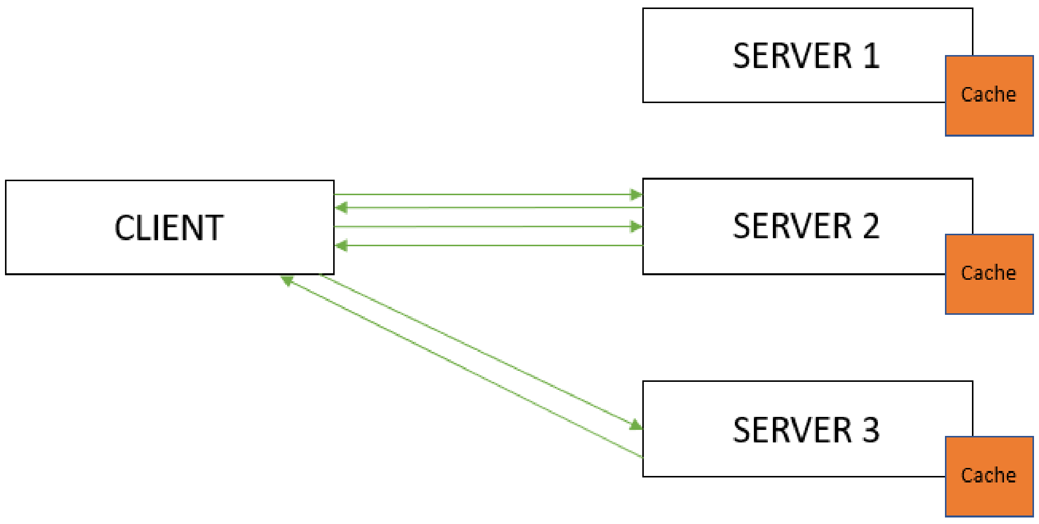
16. RATE LIMITING

- Rate limiting is act of limiting the number of requests sent to or from a system.
- Rate limiting is used to limit the number of incoming requests in order to prevent **D**enial **o**f **S**ervice attacks.

- Rate limiting can be enforced at IP address level, user account level or at region level for example.
- Rate limiting can also be implemented in tiers, for example, a type of request that could be limited to 1 per second, 5 per 10 seconds and 10 per minute.
- Let us say there is a system that rate limits only 2 requests per 10 second for each client at user account level. The third request is responded with an error.



- The in-memory cache at the server level can store the source IP address of the incoming requests and can be be used to identify the user requesting and to return an error in case of more than 2 requests are received from same user within 10 seconds.
- In the previous example, there is only one server with an in-memory cache to identify the requests and to return an error if required. But what happens in distributed system where there are n number of servers and clients can request to any server?
- In the below example, if a client issues a third request to a different server within 10 seconds, that server would not know whether the client had already issued two requests to another server. This breaks rate limiting. The better solution is to use Redis, an in-memory centralized cache from which any server gets information.



- **D**enial **o**f **S**ervice attack is an attack in which a malicious user will try to bring down or damage the system in order to render it unavailable for other users.
- **D**istributed **D**enial **o**f **S**ervice attack is an attack where the traffic flooding the target system is generated from many different resources which makes it harder to defend.

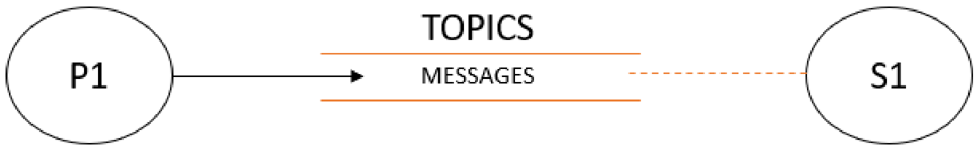
17. LOGGING AND MONITORING

- Logging is the process of collecting and storing logs which are useful information about events in your system.
- Monitoring is the process of having visibility into a system's key metrics, monitoring is typically implemented by collecting important events in a system and aggregating them in a human readable format.

- Alerting is the process through which system admin gets notified when critical system issues occur. Alerts can be set by defining specific thresholds on monitoring charts, past which alerts are sent to a communication channel like Slack.

18. PUBLISHER AND SUBSCRIBER PATTERN

- Let us assume that we have a simple an application where the client opens a long-lived connection with the server, the server pushes the data to the client, client listens to the server for data and receives the data as soon as server pushes the data. This whole process is called Streaming.
- For a simple application, this looks fine, what would happen if there is a network partition in large scale distributed systems? What would happen to those data? Will they get lost? It is really bad in case of data is lost and is very expensive.
- So, it is very important rule to have a persistent storage in case of distributed systems and this is where pub/sub pattern comes into the picture.
- There are 4 important players in pub/sub pattern.
- Publisher or server which publishes the data to the topic.
- Topic or channels which stores the specific type of message or data.
- Subscribers or clients subscribe or listening to the topics to receive the message or data.
- Message or data is some form of data that is published by the publisher on the topic and subscribers listening to the topic would receive the message.
- It is very important to note that in this pattern, the publisher and subscriber don't know each other and both of them will communicate to the topics by publishing and subscribing.



- In a pub/sub system, the topics which acts as an intermediary between the publishers' subscribers are persistent storage which would effectively store the messages.
- The topics would guarantee at least once delivery of the messages to the subscribers listening to the topics.
- The pub/sub system guarantees at least once delivery and not exactly once delivery, this means a message can be sent to a subscriber multiple time.
- Let's say the publisher publishes a message M1 to the topic, the subscriber listening to that topic receives the message but doesn't send an ACK to the message, making it to resemble that the message was not consumed by the subscriber.
- So, the next time the subscriber gets the connection back, the same message M1 is sent to subscriber again.
- Since there is a possibility that a subscriber can receive the message multiple times, operations performed through pub/sub pattern should be idempotent.
- Idempotent operations are the ones that has same outcomes regardless of how many times it is performed. For example: HTTP GET method is idempotent which when called N times, clients will get same results or there will not be any changes in the server or

database. HTTP POST is not idempotent which when called N times, created N number of records in the database.

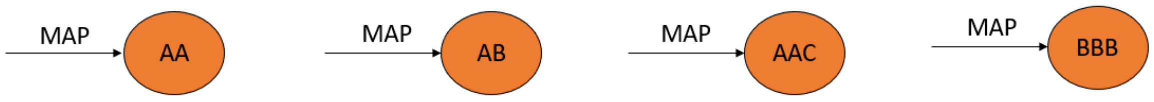
- The pub/sub system also guarantees the ordering of messages. The messages that first flow in to the topic will be the one that flow out of the topic first. This is like queue.
- Replay or rewinding to the previous message is also possible in a pub/sub system.
- Some of the popular pub/sub tools out there are
  - ✓ Apache Kafka messaging system designed by LinkedIn, very useful when using the streaming paradigm.
  - ✓ Highly scalable Pub/Sub message service created by Google, guarantees at least once delivery of messages and supports rewind in order to reprocess messages.

19. MAP REDUCE

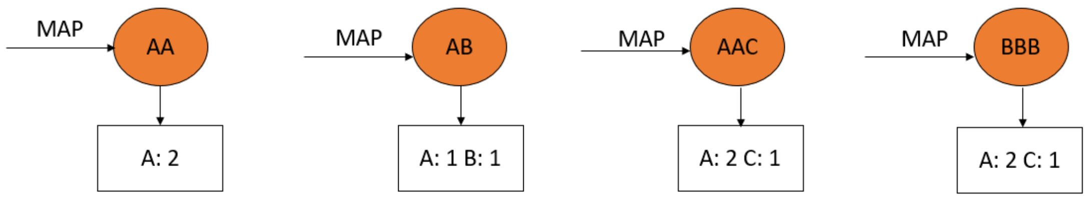
- In early 2000's, Google Engineers faced a challenge while processing large data sets from hundreds of thousands of machines of a distributed file system. The challenges were in distributed setting and mostly impacted the efficiency, latency and fault tolerance.
- To solve the above issue, in 2004, Google Engineers submitted a white paper on MapReduce to process large data sets effectively, quickly in a distributed setting and in fault tolerant manner.
- The MapReduce framework consists of three steps,
  - ✓ The Map step which runs a map function on the various chunks of the dataset and transforms those chunks into intermediate key-value pairs.
  - ✓ The Shuffle step which reorganizes the intermediate key-value pairs such that pairs of the same key are routed to same machine in the final step.
  - ✓ The Reduce step which runs a reduce function on the newly shuffled key-value pairs and transforms them into meaningful data.
- There are some important things to be taken care before looking at an example of MapReduce job.
  - ✓ The large data sets from hundreds of thousands of machines of a distributed file system are split into chunks.
  - ✓ The distributed file system has a Central Control Plane that is aware of everything going on in the MapReduce job or process. This means the Central Control Plane knows where the chunk data resides, how to communicate the worker machines that applies the Map function, how to communicate the worker machines that applies the Reduce function, knows where the output from final step is going to live.
  - ✓ The Map function is applied to the data wherever it resides, there is no data movement.
  - ✓ The key-value pairs data structure of the data is especially important. While reducing the data values from multiple chunks of same dataset, some form of commonality is expected.
  - ✓ Handle faults or failures, reperform Map and Reduce operation, thus both of those operations should be idempotent. This means that the output of Map and Reduce operation should be unique regardless of how many times, it is performed.
- When dealing with MapReduce operation, Engineers need to worry only about the Map and Reduce functions and also about inputs and outputs. The other concerns like

parallelization of operations, fault tolerance is taken care by the MapReduce internal implementation.

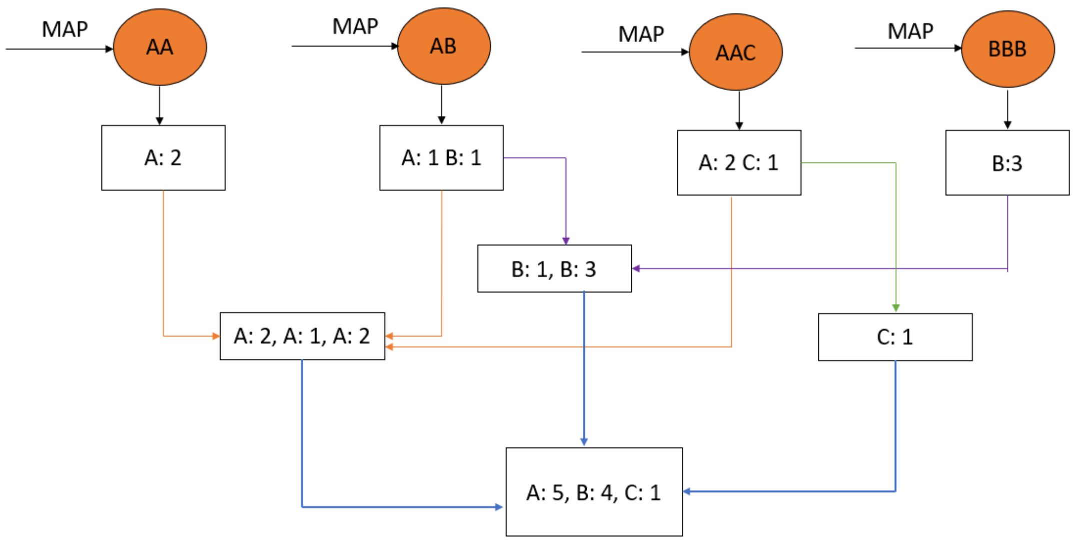
- The canonical example of MapReduce is counting number of words in a large text file. The large text file could be an aggregation of Wikipedia. For simplicity, let's take an example of counting number of letters.



- Initially, there are multiple chunks namely AA, AB, AAC, BBB from a large data set. In this step, the Map step is applied to those chunks and result of map function is going to be stored in the key-value data pairs and it looks like below.



- The key-value data pairs store the key as letter and value as number of occurrences of a particular letter in the data set. The key-value data structure can be update once after the Map function completed processing the chunk or the map function can emit the data to key-value pairs after processing each letter for example. By doing the later, the key value pair for third chunk might look like → A: 1, A:1, C:1. How and when to store the data in key-value pairs is depend on the use case.
- Now, in the Shuffle step, from the key-value pairs of each chunk, unique keys are taken and assigned to reduce worker machines. After that reducer worker machine aggregates the count and writes the final output.



- A Distributed File System is an abstraction over a cluster of machines that allows them to act like one large file system.

- The two most popular implementations of a DFS are the **G**oogle **F**ile **S**ystem and the **H**adoop **D**istributed **F**ile **S**ystem.
- Hadoop is popular open-source framework that supports MapReduce jobs and many other kinds of data processing pipelines. Its central component is **H**adoop **D**istributed **F**ile **S**ystem on top of which other technologies have been developed.

20. SECURITY AND HTTPS

- Nowadays modern-day systems communicate using **H**yper **T**ext **T**ransfer **P**rotocol and it is assumed that any message that is exchanged between the client and server are meant to be private.

20.1 MAN IN THE MIDDLE ATTACK

- Typically, it is not private when the client and server exchange messages using HTTP and any malicious actor can hijack the communication channel and intercept the messages and alter the messages. This is really bad in a system design.
- This type is attack is usually called **M**an **I**n **T**he **M**iddle attack. By establishing a secure connection between the client and server, it is possible to prevent the consequences of MITM attack.

20.2 ENCRYPTION AND DECRYPTION

- Encryption is the process of obfuscating the messages or data that is exchanged between client and server. Client can send the data to the server after encryption using any popular encryption techniques.
- Decryption is the process of converting the obfuscated message into a human readable format or original format.

20.3 SYMMETRIC ENCRYPTION

- Symmetric Encryption is a type of encryption which is basically relied on one single key. Client encrypts the message using a key and the server decrypts the message using same key.
- The Symmetric Encryption is typically very faster than its counterpart Asymmetric Encryption but it has a problem of sharing the keys publicly to encrypt and decrypt. This again leads to a problem where a malicious actor can get the keys by some means of hijacking.
- The most widely used symmetric key algorithms are part of **A**dvanced **E**ncryption **S**tandard.

20.4 ASYMMETRIC ENCRYPTION

- Asymmetric Encryption also known as public key encryption which relies on two keys namely public key and private key. The server can share the public key publicly but the message can be decrypted only by the private key which only the server has access.

20.5 HYPERTEXT TRANSFER PROTOCOL SECURE

- **H**yper **T**ext **T**ransfer **P**rotocol **S**ecure is an extension of HTTP and it is used for secure communication. It requires the server to have a trusted **S**ecure **S**ockets **L**ayer certificate and uses the **T**ransport **L**ayer **S**ecurity, a security protocol built on top of TCP, to encrypt the data exchanged between client and server.

20.6 TRANSPORT SECURITY LAYER

- The **T**ransport **L**ayer **S**ecurity is a protocol over which HTTP runs in order to provide a secure communication.

20.7 SECURITY SOCKETS LAYER

- SSL certificate is typically granted to the server by **C**ertificate **A**uthority which contains the server's public key and is shared to the client in the TLS handshake process.
- An SSL certificate effectively confirms that the public key belongs to the server claiming it belongs to them. It is crucial part in defending MITM attacks.

20.8 CERTIFICATE AUTHORITY

- **C**ertificate **A**uthority is used to digitally sign the SSL certificates that is used in HTTPS connections.

20.9 TLS HANDSHAKE PROCESS

- TLS Handshake undergoes below process,
  - ✓ The client sends client hello (CHELLO) to the server.
  - ✓ The server responds back with server hello (SHELLO) and also the SSL certificate which contains the public key. This public key can be used by the client to encrypt messages.
  - ✓ The client verifies the SSL certificate using the public key of CA. It has to be noted that SSL certificate itself encrypted using the private key of CA. On successful verification, the client generates the premaster secret and encrypt the secret using the public key in SSL certificate.
  - ✓ The server decrypts the premaster secret using the private key.
  - ✓ Both the client and server would use the CHELLO, SHELLO, Premaster Secret to generate a **S**ymmetric **E**ncryption key or session keys which would be used to have a secure communication for rest of the session.
  - ✓ Once the session is over, the key is invalidated and no more used.