System Design

- Vedant Dhiman

Schedule

- CAP theorem
- Load Balancer
  - Why?
  - How?
  - When?
  - Benefits
  - Issues?
- Web Protocols
  - TCP/IP
  - OSI
  - UDP
  - HTTP
  - HTTPS
  - Web Sockets
- Practice – System Design Problems

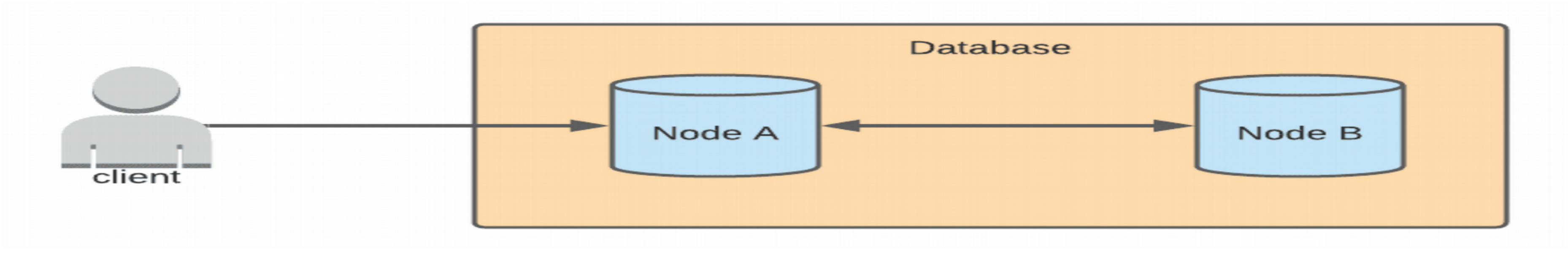
CAP Theorem

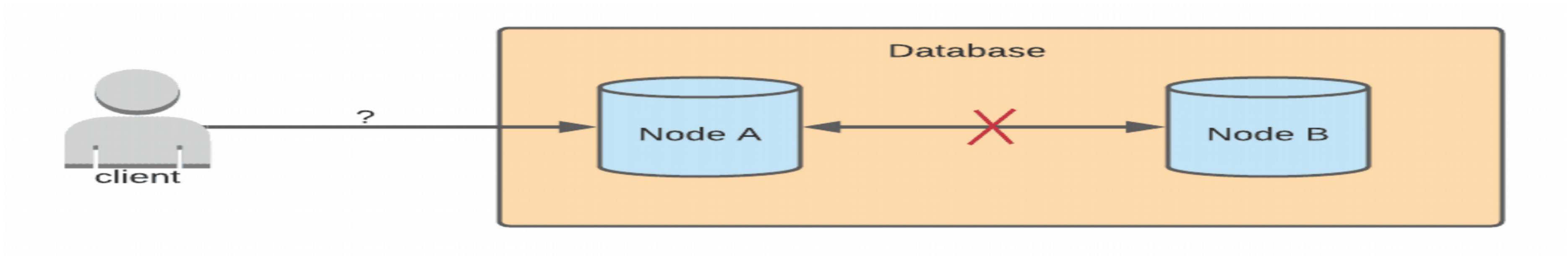
- Consistency
- Availability
- Partition Tolerance

CAP Theorem

- CAP stands for ‘Consistency’, ’Availability’ and ‘Partition Tolerance’
- Nodes in a distributed database should be running even when there is a network partition
- Network partition is a (temporary) network failure between nodes

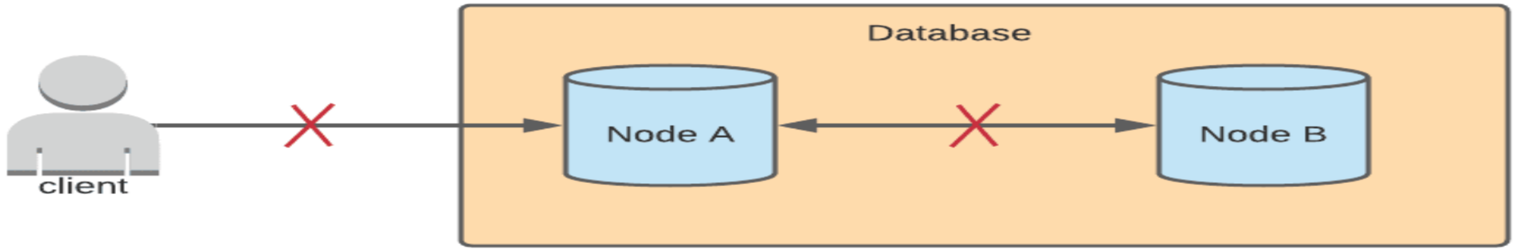
Example – Network Partition





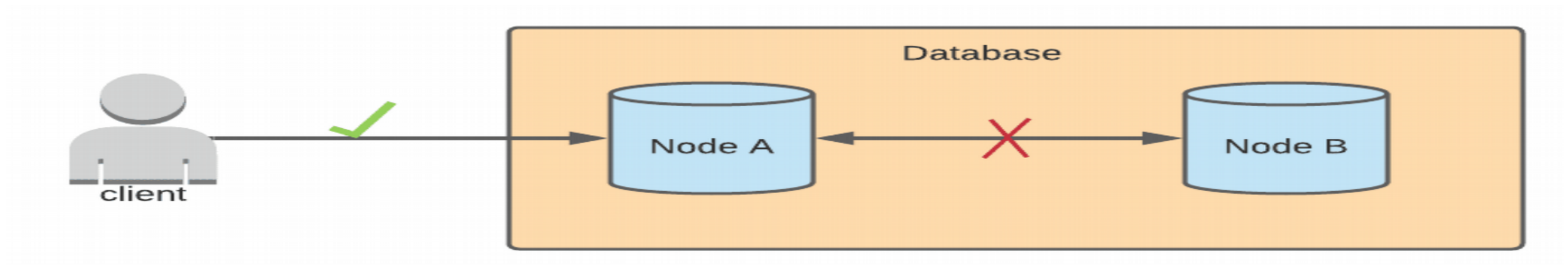
Consistency

- When a write is sent to a database, all read requests sent to any node should return that updated data.
- Node A and Node B both will reject any write requests sent to them.
- The state of the data on all the nodes will be same if we reject write requests
- Read requests are available as don’t update the state of the data.



Availability

- It’s OK to have inconsistent data across the nodes, where one node may contain stale data and the other the most updated data
- We prioritize nodes to successfully complete request sent to them
- Node A will receive the request first, and after some time, Node B will be updated as well.



Consistency vs Availability – Whom to prioritize ?

- Consistency Use-Cases:
  - Banking Systems
  - Booking/Reservation Systems
  - Payments Systems
- Availability Use-Cases:
  - Social Media News Feed – Facebook, Instagram, TikTok
  - Video Streaming/Image Services
  - Other read heavy use-cases

Consistency vs Availability – Which DB to pick ?

- Prioritize Consistency:
  - SQL Databases
  - MongoDB
  - Redis
  - Google BigTable
  - HBase
- Prioritize Availability:
  - Cassandra
  - AWS DynamoDB
  - CouchDB

Question

- You’re building Amazon’s product listing app, where shoppers can browse a catalogue of products and purchase them if they are in-stock.
- Should the distributed database you choose to store product information prioritize consistency or availability?

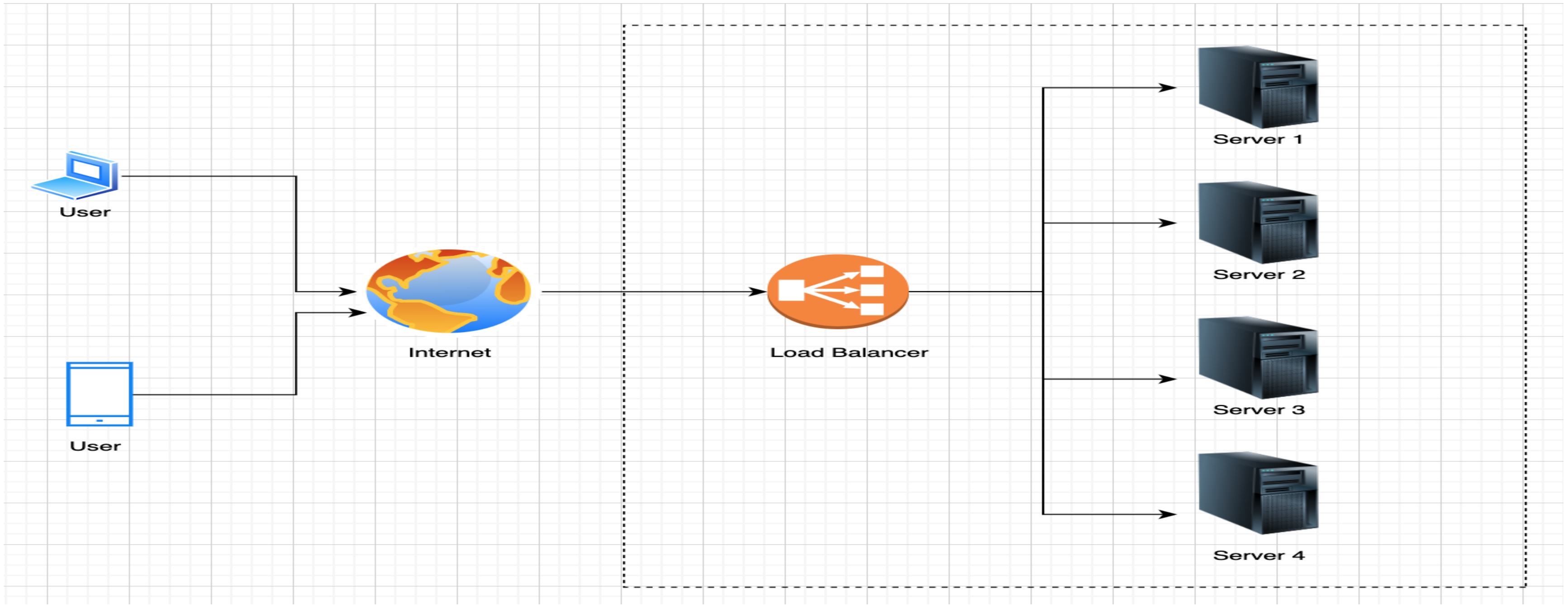
Load Balancer

How to distribute Web Traffic

What is Load balancer?

- A Load Balancer is a type of web server that distributes incoming web traffic across multiple web servers
- Load Balancer allows applications to:
  - Scale up or down with demand
  - Achieve higher availability
  - Efficiently utilize server capacity

Load Balancer – Simple Example



Why we need Load Balancers ?

- System Capacity: Memory (RAM), Processor (CPU), Network Connections
- System Capacity can be increased by:
  - Increasing the RAM or CPU on the server
  - Efficiently using available resources
  - Multithreading
- At a certain point of increase in traffic, increasing system capacity is either not possible or it is not cost effective
- Horizontal scaling: The concept of adding more servers to the system
- Load Balancer helps us decide which request goes to which server in a horizontally scaled server

How load balancers work ?

- Load Balancer distributes incoming traffic to:
  - Maximize system capacity utilization
  - Minimize the queueing time
- Load Balancing Strategies:
  - Round Robin: Servers are assigned in a cyclic sequence. This strategy ensures next server assigned is guaranteed to be the least recently used server
  - Least Connections: Assigns the server current handling the least number of requests
  - Consistent Hashing: Similar to database sharding, the server can be assigned consistently based on the IP address or URL

Load balancers – Best Practices

- Don’t implement your own load balancer
- Industry-standard reverse proxy such as HAProxy or Nginx or Envoy
- Use SSL Termination and health checks for load balancer
- Use cloud providers out-of-the-box load balancers such as Amazon’s Elastic Load Balancer (ELB)

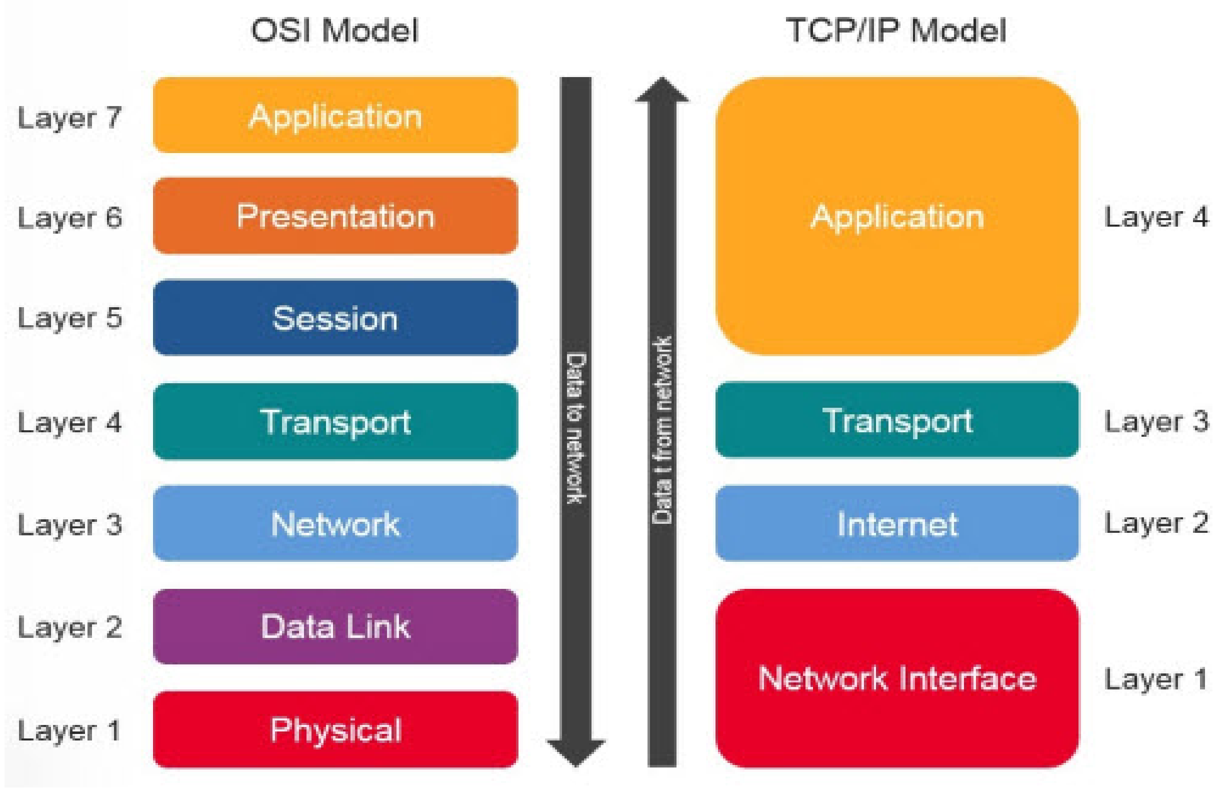
When to use Load Balancer

- Use it when the system will benefit from increased scale or redundancy
- Load balancer sits between external traffic and application servers
- In microservices architecture, load balancer is used in front of each internal service so that each service can scale independently
- Load balancer cannot solve many scaling problems in the design such as:
    - Poor database performance
    - Algorithmic complexity
    - Unreliable third-party APIs
- To handle the above-mentioned performance issues, it’s better to process tasks asynchronously such as a job queue
- Load balancing is different from rate limiting
- Rate limiting: When traffic is intentionally throttled or dropped in order to prevent abuse by a particular user or organization

Load Balancer - Issues

- **Issues:**
- **Bottlenecks:**
  - As scale increases, load balancers can themselves become a bottle or single point of failure
  - Use multiple load balancers to handle the above use-case
  - DNS round robin can be used to handle traffic across different load balancers
- **User sessions:** Same user’s request can be handled by different backends. This could be problematic for applications that rely on session data that isn’t shared across servers
- **Longer deploys:** Deploying new server versions can take longer and require more machines since the load balancer needs to roll over traffic to the new servers and drain requests from the old machines

OSI vs TCP/IP


OSI Model
TCP/IP Model
Layer 7
Application
Layer 6
Presentation
Layer 5
Session
Layer 4
Transport
Layer 3
Network
Layer 2
Data Link
Layer 1
Physical
Data to network
Data t from network
Application
Layer 4
Transport
Layer 3
Internet
Layer 2
Network Interface
Layer 1

TCP/IP Model

The Network Access Layer / Link Layer

represents a local network of machines i.e. hardware

The Internet Layer

describes the much larger network of devices interconnected by IP address

Example: IPv4 and IPv6

The Transport Layer

includes protocols for sending and receiving data via packets

Example: TCP and UDP

The Application Layer

describes how data is sent to and from users over the internet

Example: HTTP, HTTPS and Web Sockets

TCP/IP

- TCP – Transmission Control Protocol / IP – Internet Protocol
- It is one of the most commonly used protocol suite
- TCP is used where accuracy is more important than the speed of delivery
- TCP enforces the “rules of the road” like a traffic cop
- TCP is connection-oriented
    - Server must be listening for connection request from client
    - Client and server must be connected before any data is sent
- TCP is stateful i.e. context is embedded in the TCP segment
    - It can detect errors in transmission
    - It can request re-transmission of error packets

UDP

- UDP is connectionless, making it faster than TCP
- UDP has none of the error-handling capabilities of TCP
- UDP is mainly used for streaming application such as Skype, Zoom, Google Meet, WhatsApp Calls
- UDP is used when users accept occasional delays in exchange for real time service

HTTP and HTTPS

- HTTP defines:
- Request Methods – GET, POST, PUT, etc – the same methods RESTful APIs use
- URL
- Port Numbers
- HTTPS is a secure version of HTTP – Secured by TLS

TLS Handshake Procedure

- HTTPS works on top of Transport Layer Security (TLS)
- TLS is used to:
  - Encrypt communications in the transport layer
  - Preventing unauthorized parties from listening in on communications
- TLS handshake is process of initiating a secure session through TLS
  - Clients requests for a secure connection with a server usually on port 443 – secured for TLS
  - Client and server agree to a cipher suite (ciphers and hash function)
  - The server submits a digital certificate which serves as proof of identity. Digital certificates are issued by 3rd party Certificate Authorities (CAs) and effectively vouch for the server.
  - Once the certificate is accepted, client generates a session key to encrypt any information transmitted during the session

WebSocket vs HTTP

- WebSocket is a newer communications protocol designed to solve the issues with HTTP
- Issues with HTTP:
  - Strictly unidirectional – Client must always request data from server
  - Only one HTTP request can be sent per session
- Modern applications require longer session and/or continuous updates from the server
- Long-polling does solve the problem by keeping the connections open for longer, however it is very resource intensive

WebSocket vs HTTP

- WebSocket is a newer communications protocol designed to solve the issues with HTTP
- Issues with HTTP:
  - Strictly unidirectional – Client must always request data from server
  - Only one HTTP request can be sent per session
- Modern applications require longer session and/or continuous updates from the server
- Long-polling does solve the problem by keeping the connections open for longer, however it is very resource intensive

WebSockets

- WebSocket works similar to HTTP, but with some improvements and tradeoffs
- Servers can send data to clients without receiving a request first
- Multiple messages can be passed back and forth over the same connection
- Fully compatible with HTTP
- Much less computationally demanding than long-polling
- WebSockets common use-cases require real time data
- Common use-cases
    - Messaging services such as WhatsApp, Facebook Messenger, Telegram, Google Chat
    - Gaming apps: Call Of Duty Mobile, etc.
    - Trading Platforms: Zerodha, etc.

WebSockets - Disadvantages

- Common disadvantages of WebSockets over HTTP are as follows:
  - WebSocket has no built-in, standardized API semantics like HTTP's status codes or request methods.
  - Keeping communications open between each client and server is more resource-intensive and adds complexity
  - It's less widespread, so development can take longer.

When to bring web protocols in interview

- Prefer bringing up the specific web protocols only when the interviews as follow-up questions.
- Tips:
- At the transport layer, choose:
- TCP if you are more concerned with the data
- UDP if fast transmission is needed
- At the application layer, choose:
- HTTPS over HTTP for security reasons
- Choose WebSockets when you have to maintain open client-server communications
- If you are design a service with APIs, choose
- HTTPS over WebSocket to use HTTPS’s standardized request methods and status codes
- Specially if we are designing RESTful APIs

Additional Reading Material

- How WebSockets work at scale - Slack Engineering Blog
- Migration from HTTP to HTTPs - Google Security Blog
- Envoy vs NGINX vs HAProxy - Blog Link
- Scaling Microservices with Envoy - Blog Link

Design Problem

Design Facebook Messenger