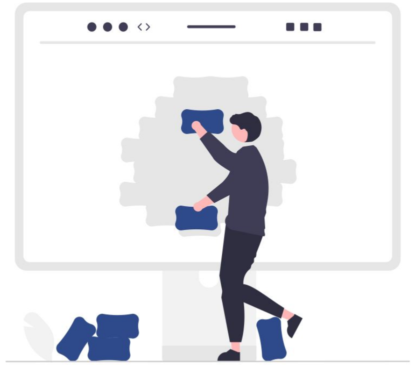
Episode 2

System Design for Noobs

July 23, 2022



Core topics covered in this slide:

1. Trade-offs
   a. Performance vs scalability
   b. Latency vs throughput
   c. Availability vs consistency
2. DB Replication
3. Reverse Proxy

FAQ Recap

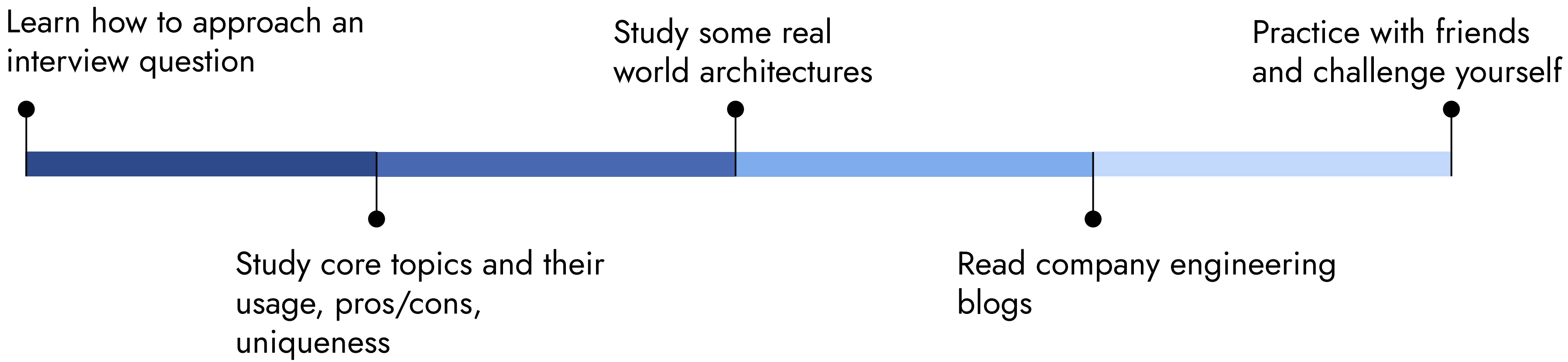
Do I need to know everything?

Absolutely f*cking **NOT!**

System Design questions mainly depends on some variables:

- Your experience
- Your technical background
- Applied company
- Applied position
- Luck!

The Plan


Learn how to approach an interview question
Study core topics and their usage, pros/cons, uniqueness
Study some real world architectures
Read company engineering blogs
Practice with friends and challenge yourself

The Approach

Step 1: Outline use cases, constraints, and assumptions

- Who is going to use it?
- How are they going to use it?
- How many users are there?
- What does the system do?
- What are the inputs and outputs of the system?
- How much data do we expect to handle?
- How many requests per second do we expect?
- What is the expected read to write ratio?

Step 2: Create a HLD (High Level Design)

- Draw the **core components** and their connections
- Question and justify your selections
- Specify the read/write scenarios

Step 3: Design the core components

No fixed formula.

Example:

- API design
- Database schemas
- Encryption

Step 4: Scale the system

- Load balancers
- Horizontal/Vertical scaling
- Caching
- Reverse proxy?
- DB replication
- Discuss trade-offs

A very good template:

https://leetcode.com/discuss/career/229177/My-System-Design-Template

Trade-offs

- Performance vs Scalability
- Latency vs Throughput
- Availability vs Consistency

Remember, **EVERYTHING IS A TRADE-OFF** in system design.

Performance vs Scalability

Serve less but serve well

or, serve all but less efficiently?

- The issue is **performance**, if your system is slow for a single user.
- The issue is **scalability**, your system is fast for a single user but slow under heavy load.

Please read this ***sherrra*** slide, you'll learn everything:

https://www.slideshare.net/jboner/scalability-availability-stability-patterns/

A bit long and vague, but you're smart enough.

Latency vs Throughput

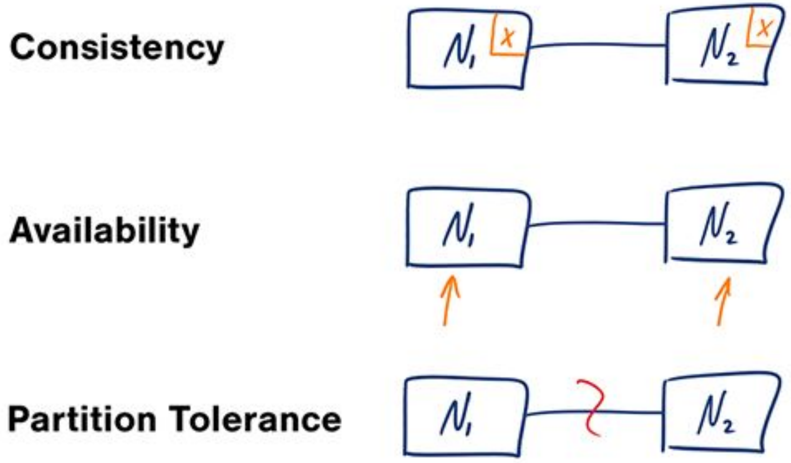
Latency is the time to perform some action or to produce some result.

Throughput is the number of such actions or results per unit of time.

We want to ensure **maximum throughput** with **acceptable latency**.

Availability vs Consistency

Let’s recap the CAP theorem.



https://robertgreiner.com/cap-theorem-revisited/

CP or AP?

CP - consistency and partition tolerance

Waiting for a response from the partitioned node might result in a timeout error.

➔ CP is a good choice if your business needs require atomic reads and writes.

AP - availability and partition tolerance

Responses return the most readily available version of the data available on any node, which might not be the latest. Writes might take some time to propagate when the partition is resolved.

➔ AP is a good choice if the business needs allow for eventual consistency or when the system needs to continue working despite external errors.

Consistency Patterns

Weak

After a write, reads may or may not see it.

Seen in real time use cases such as VoIP, video chat, and real-time multiplayer games.

Eventual

After a write, reads will eventually see it (typically within milliseconds). Data is replicated **asynchronously**.

Seen in systems such as DNS and email. Eventual consistency works well in highly available systems.

Strong

After a write, reads will see it. Data is replicated **synchronously**.

Seen in file systems and RDBMSes. Strong consistency works well in systems that need transactions.

Ensuring Availability

Active-Passive Fail-over

Heartbeats are sent between the active and the passive server on standby. If the heartbeat is interrupted, the passive server takes over the active's IP address and resumes service.

Active-Active Fail-over

Both servers are managing traffic, spreading the load between them.

If the servers are public-facing, the DNS would need to know about the public IPs of both servers. If the servers are internal-facing, application logic would need to know about both servers.

Ensuring Availability

Active-Passive Fail-over

Heartbeats are sent between the active and the passive server on standby. If the heartbeat is interrupted, the passive server takes over the active's IP address and resumes service.

Active-Active Fail-over

Both servers are managing traffic, spreading the load between them.

If the servers are public-facing, the DNS would need to know about the public IPs of both servers. If the servers are internal-facing, application logic would need to know about both servers.

Ensuring Availability

DB Replication

Scaling distributed database systems to maximize availability.

DB Replication

Active Replication (Push)

In active replication each client request is processed by all the servers. So, all of the distributed data storage gets updated **actively**.

Passive Replication (Pull)

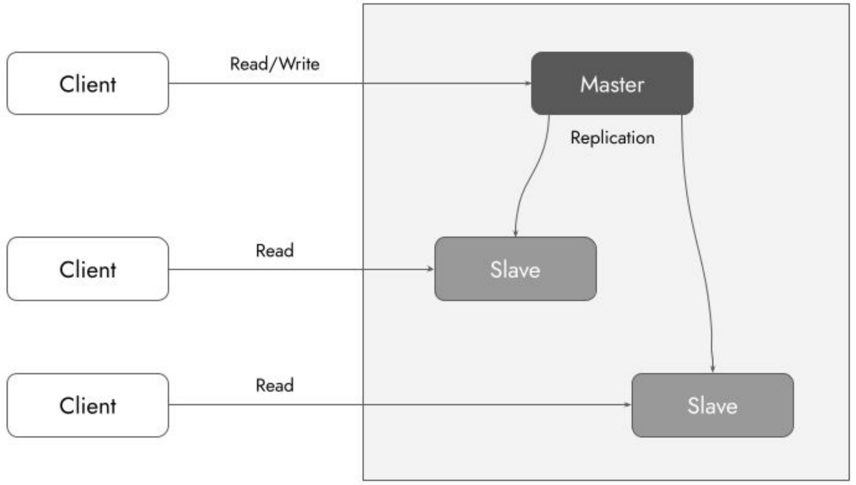
In passive replication there is only one server (called primary) that processes client requests. After processing a request, the primary data storage updates the others **passively**.

Passive Replication Methods

- Master-Slave replication
- Master-Master replication
- Tree replication
- Buddy replication

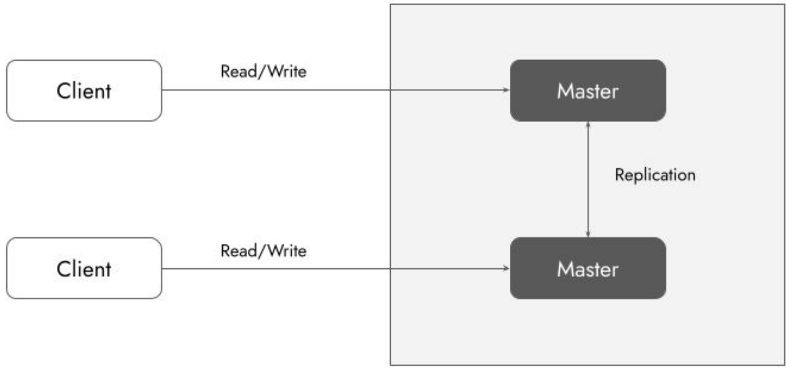
Master-Slave Replication

- Master (primary storage) gets the latest data.
- Slaves (secondary storages) replicate the data internally.



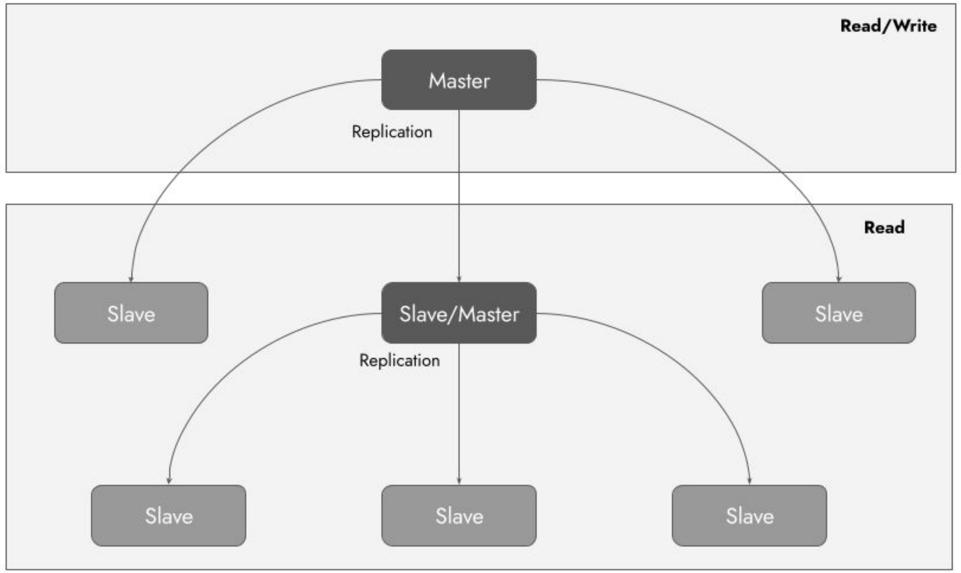
Master-Master Replication

- Every master storage can get latest data.
- They replicate each other to stay updated.



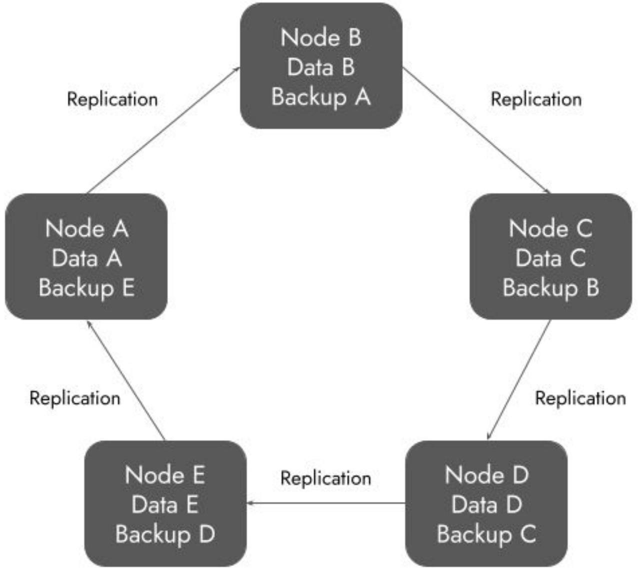
Tree Replication

- Master (tree root) gets the latest data.
- Child masters and their slaves replicate the data internally.



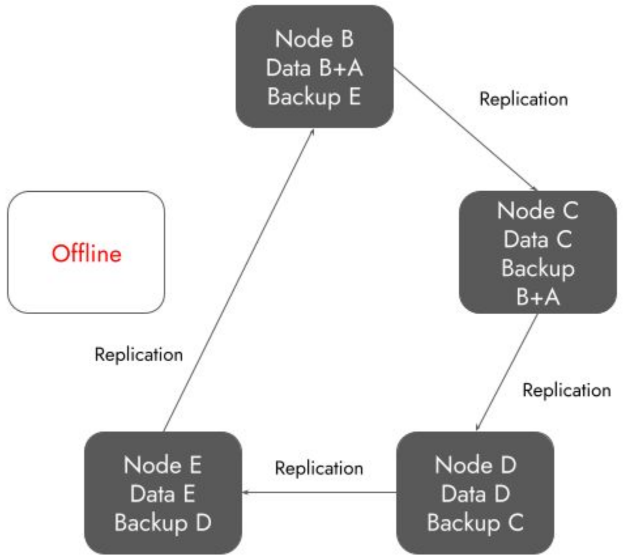
Buddy Replication

- Each node stores the backup of the previous node on the chain.
- The backup is performed using replication.



Buddy Replication Fail-over

- Store extra backup.
- Update the replication chain.



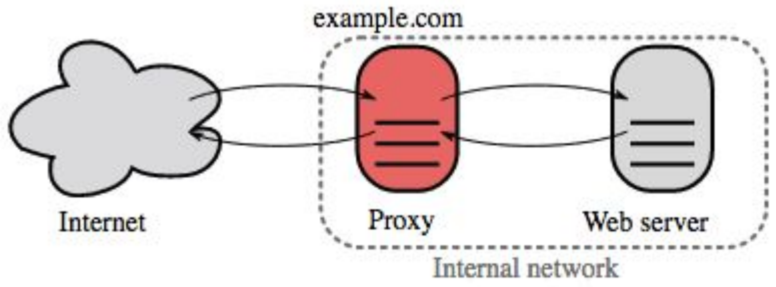
Replication Cons

- There is a potential for loss of data if the master fails before any newly written data can be replicated to other nodes.
- Writes are replayed to the read replicas. If there are a lot of writes, the read replicas can get bogged down with replaying writes and can't do as many reads.
- The more read slaves, the more you have to replicate, which leads to greater replication lag.
- On some systems, writing to the master can spawn multiple threads to write in parallel, whereas read replicas only support writing sequentially with a single thread.
- Replication adds more hardware and additional complexity.

Reverse Proxy

A reverse proxy is a web server that centralizes internal services and provides **unified interfaces** to the public.

Requests from clients are forwarded to a server that can fulfill it before the reverse proxy returns the server's response to the client.



Why Reverse Proxy?

- **Increased security**: Hide information about backend servers, blacklist IPs, limit number of connections per client.
- **Increased scalability and flexibility**: Clients only see the reverse proxy IP, allowing you to scale servers or change their configuration.
- **SSL termination**: Decrypt incoming requests and encrypt server responses so backend servers do not have to perform these potentially expensive operations.
- **Compression**: Compress server responses.
- **Caching**: Return the response for cached requests.
- **Direct static content serving**:
    - HTML/CSS/JS
    - Photos
    - Videos

Reverse Proxy Cons

- Introducing a reverse proxy results in increased complexity.
- A single reverse proxy is a single point of failure, configuring multiple reverse proxies (a failover) further increases complexity.

Reverse Proxy vs. Load Balancer?

- Deploying a load balancer is useful when you have multiple servers. Often, load balancers route traffic to a set of servers serving the same function.
- Reverse proxies can be useful even with just one web server or application server, opening up the benefits described in the previous section.
- Solutions such as NGINX and HAProxy can support both layer 7 reverse proxying and load balancing.

Further study:

- https://www.nginx.com/resources/glossary/reverse-proxy-vs-load-balancer/
- https://www.nginx.com/blog/inside-nginx-how-we-designed-for-performance-scale/
- http://www.haproxy.org/download/1.2/doc/architecture.txt

Thank you.

Do you have any questions?

azmainadel47@gmail.com
Direct/WhatsApp: +880 1684 723252