1.

General knowledge

How to scale

1. Vertical vs horizontal

2. Add load balancer

4. Adding cache

5. CDN load balancer (serves images closer to user

6. Using async message broker

Defer long-running or non-critical tasks to background queues or message brokers. This ensures your main application remains responsive to users.

**Example:** Slack uses asynchronous communication for messaging. When a message is sent, the sender's interface doesn't freeze; it continues to be responsive while the message is processed and delivered in the background.

**What about acid transcations?**

Notice this is different from the regular consistency CAP theorem,

Essentially, ACID transactions guarantee that database operations are executed correctly, and that’s the key part here

What are some advantage and disadvantages?

**Advantages**

Isolation– ACID transactions ensure that each transaction is independent of the others. It also contributes to preserving data integrity by preventing interference between concurrent transactions.

Data integrity– Even if a transaction fails, ACID transactions guarantee that the database will remain in a consistent state. It contributes to data reliability and integrity.

**What are some solutinos to addressing the issues In a distributed system here ?**

* Two-phase commit: This protocol ensures that all nodes in a distributed system agree to commit a transaction before it is committed, ensuring data consistency and agreement on the transaction’s outcome.
* Replication: In a distributed system, replication involves keeping multiple copies of the same data on various nodes, reducing network latency and increasing availability.
* Sharding: This process involves dividing data across multiple nodes in a distributed system, improving performance and scalability but increasing the complexity of maintaining data consistency.

**And then here we have the code**

**Disadvantages**

Deadlocks– Multiple transactions waiting for each other to release resources can cause deadlocks. Deadlocks can be difficult to resolve and have an impact on database reading and retrieval performance.

**What’s the difference between high level diagram and low level diagram?**

High level

Very genearl system design questions here

1. System architecture
2. Database design
3. Brief description of systems, services, platforms, and relationships among modules.

When do we use sql vs nosql? Then If a lot of relations use sql, follows ACID properties

**Nosql**

1. If data changes structure all the time, use no sql then

2. SQL databases are better for multi-row transactions, while NoSQL is better for unstructured data like documents or JSON. SQL databases are also commonly used for legacy systems built around a relational structure.

3. This scaled very quickly in that sense here

**System design**

This part of the article is based off of the youtube video where the guy talks about the designing facebook

How do we ensure the data communication between 2 servers?

**A single server at the start here**

The example here is w

1. We are building a server facebook for students all across the world

With s single server

If you lose data, that’s it one strike and then it’s gone here.

If a user creates a profile, single copy of data will be stored on a server here.

**How do we reduce the latency here with this exapmle**

1. Add more database servers, 2 servers (in different cities, 1 server for oxford student and 1 server for the Havard)
2. No comm between seach server and there is no data sharing so that’s not good here

**How do we solve communication between different servers?**

1. So when you have multi copies of data

A hand writing on a whiteboard

Description automatically generated

So you have 2 copies of data in each data server, and then you want to update the data in US and the one ni Europe as well.

**Updating informations in both cities?**

1. When update student C in US, how to update C in Europe as well right away? (In bank system) this can take a while
2. You can use TCP here and
3. What happens if network msg fails?

maybe the Europe server is down, what do you do then? Well, you will never get to know unless you get an acknowledgement, right? So the European server should send back an acknowledgement. That's again, quite simple. If you get an acknowledgement, you know, that the update has gone through and you're sorted,

**What if no ack,**

You can retry until the update happens here, infinite.

Usually what happens next

1. If you have server A, server B.

A drawing of a diagram

Description automatically generated

**How does a server B know request for server A has gone through here?**

To solve this problem

1. Make A the leader (the only one with write), so only A can do the update to write
2. B gets replicas of the data, B is slave node, now B will read the data here

**Why only have 1 master?**

Using the 2 phase commit protocol

1. 1 leaderA, multi followers here, and then B, C and D.

A person writing on a whiteboard

Description automatically generated

Send prepare first and then commit request

you send a prepare request to your followers, your followers give you an acknowledgement that, Hey, yes, I got the prepare request. And when you get these acknowledgements, you ask your followers to commit.

When lsave gets prepare (which starts begin)

If the leader fails, then the follower will do a rollback, no commit

**System design how to ensure atomicity between transactions in datasbase?**

Well we have 2 options here, either synchronous or asynchrounous.

**What’s atomic transaction?**

1. *In synchronous replication, the leader node waits for all of the follower nodes to acknowledge receipt of the data change before reporting success to the client.*
2. Can be slow as said for the user so not so good in that regard

**Aysnc transaciton**

1. *In async situation, the leader node does not wait for the follower nodes to acknowledge receipt of the data change before reporting success to the client.*

*This means that the follower nodes may not have the most up-to-date copy of the data, and*

***How to compensate this?***

In practice, many systems use a combination of synchronous and asynchronous replication. For example, a database might use synchronous replication to replicate data to a small number of critical replicas, and asynchronous replication to replicate data to other replicas. This configuration is sometimes also called Semi-Synchronous.

**How do we handle node failures as said?**

**Follower Failure**

Followers are relatively easy to handle in the event of a failure. When a follower fails, it can simply reconnect to the leader and resume replicating data. This process is known as *catch-up recovery*.

**Leader Failure**

Leader failure is more challenging to handle, as the leader is responsible for processing all write requests. When the leader fails, one of the followers must be promoted to the role of leader. This process is known as *failover*.

Implementation of replication logs:

**What’s replica log?**

Replication logs are a key part of leader-based replication, a method of keeping multiple copies of a database in sync. The leader node is responsible for writing all changes to the database, and the follower nodes replicate those changes from the leader.

How does master propagate cahnge to the follower

**What’s the logical way of logging?**

1. Using logical (role-based log replica?
2. Logical (row-based) log replication is a more flexible type of replication than WAL shipping. In logical (row-based) log replication,
3. the leader logs every change to the database at the granularity of a row. The log record contains enough information to identify the row that was changed and the new values of all columns in the row.
4. The leader then sends the log record to the followers, and the followers apply the change to their databases.

And then here we have more code here

1. And then

**Q: How can we achieve strong consistency between the master and slave databases?**

A: WAL (Write-Ahead-Logging), a well-known and important technology in database, is needed for this purpose-. When the system updates the operation write logs (Oracle Redo Log and MySQL Binlog among others) or commits a transaction, one should first ensure the flushing of the logs generated by the transaction to the disk. This would ensure no data loss occurs.

The method for achieving the strong data consistency between the master and slave databases is also simple:

* When you commit a transaction, it initiates two log writing operations: the first one to write the log to the ephemeral disk, and the other to synchronize the log to the slave database and ensure it is saved to the disk
* The master database will return the response to the application only upon the successful return of the two operations, at which point the transaction commitment is successful.

And then here you have the following:

Q: How to ensure a high performance while ensuring the strong synchronization?

Diagram of a diagram of a computer

Description automatically generated

When app init a req to commit a transaction, must sync the transaciton log to slave db here to ensure robust sync of data and store the log to the disk

System desgin design reddit:

How to handle faliure in distributed system?

How to build fault tolerant system in db?

**What is fault tolerance**

Fault tolerance refers to the ability of a system (computer, network, cloud cluster, etc.) to continue operating without interruption when one or more of its components fail.

So in terms of the hardware and the software below will then happen:

* **Hardware systems** that are backed up by identical or equivalent systems. For example, a server can be made fault tolerant by using an identical server running in parallel, with all operations mirrored to the backup server.
* **Software systems** that are backed up by other software instances. For example, a database with customer information can be continuously replicated to another machine. If the primary database goes down, operations can be automatically redirected to the second database.

How to make sure your system is highly concurrent?