Photo by Clay Banks on Unsplash

✦ Member-only story

System Design Fundamentals: Data Replication and Partitioning

Mariam Jaludi · Follow
Published in Level Up Coding
5 min read · Apr 26, 2022

Listen Share ••• More

This post is part of my system design series. You can find the other topics covered here:

1. Distributed Systems
2. Load Balancers

The performance of a system is usually dependent on how well its database performs. If a system's database is down, the system as a whole is very likely to be unavailable. Similarly, if a system's database has high latency, the system itself will have high latency.

In order to improve a database's availability and reliability, we can use ***data replication*** and ***data partitioning***.

What is Data Replication?

Imagine you have a system with a single main database, and that database has gone down. If this were to happen, you can no longer read or write data to your database. All the data would be lost.

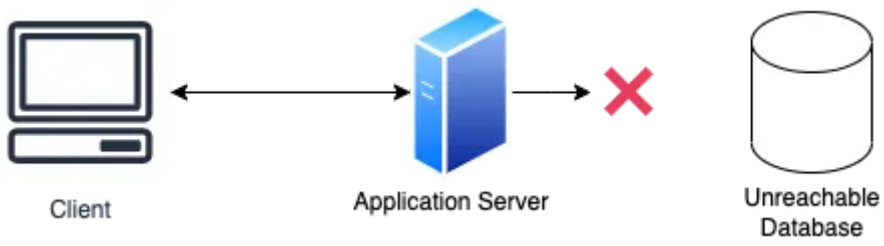


Fig. 1: System with a single database that has gone down.

Replication allows us to overcome this problem by having a backup replica of our database on standby for when issues like this occur.

The main database handles read and write requests from the application server and will write to the backup database to keep it up to date. If the main database is down, the application server will switch to the back up database to keep the system available. The back up will then become the main database and when the other database is running again, it will replicate all its data to it.

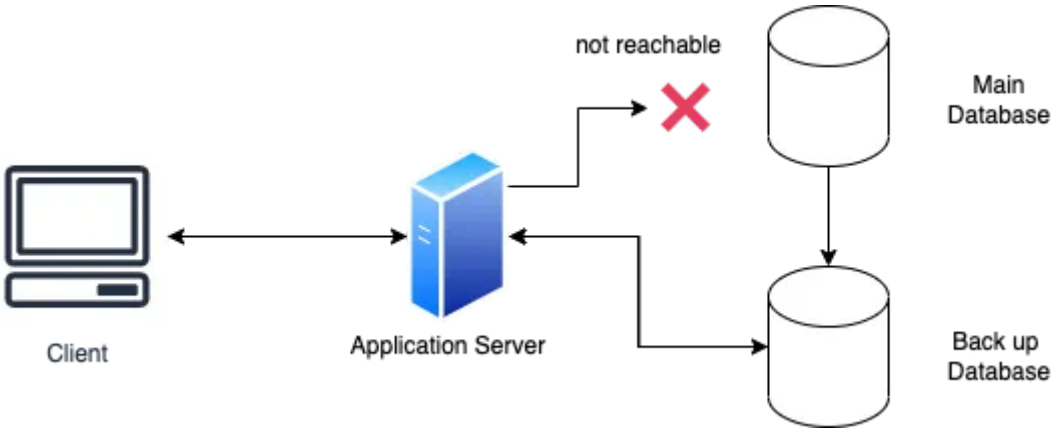


Fig. 2: System with a replicated database for better availability

Another term for this is redundancy. Redundancy is the duplication of critical components or functions of a system with the intention of increasing the reliability of the system, usually in the form of a backup or fail-safe, or to improve actual system performance.

How Are Replica Databases Kept Up To Date?

The primary database gets written to, which then is passed to the replica databases. Each replica outputs a message stating that it has received the update successfully.

It is necessary for data to be completely in sync at all times between databases if you want your database to take over from the main database in case of failure. When the main database is written to, it should update backup databases synchronously and only complete the write request when all replicas have returned successful responses.

If it is not critical to have all the data in all databases to be completely in sync, you can use asynchronous writes.

An example of this could be users posting photos on Instagram. Instagram might have databases for different regions. If a user in California posts a photo, it will be written into the regional database first, but waiting for it to write to all other regional databases might add too much latency. Additionally, it's likely that other users who would want to access that post are located in the same region. A solution would be to update the other regions asynchronously. This way the post will be available immediately to other users in California, while once it replicates to other regions, another user in Korea could access the post too.

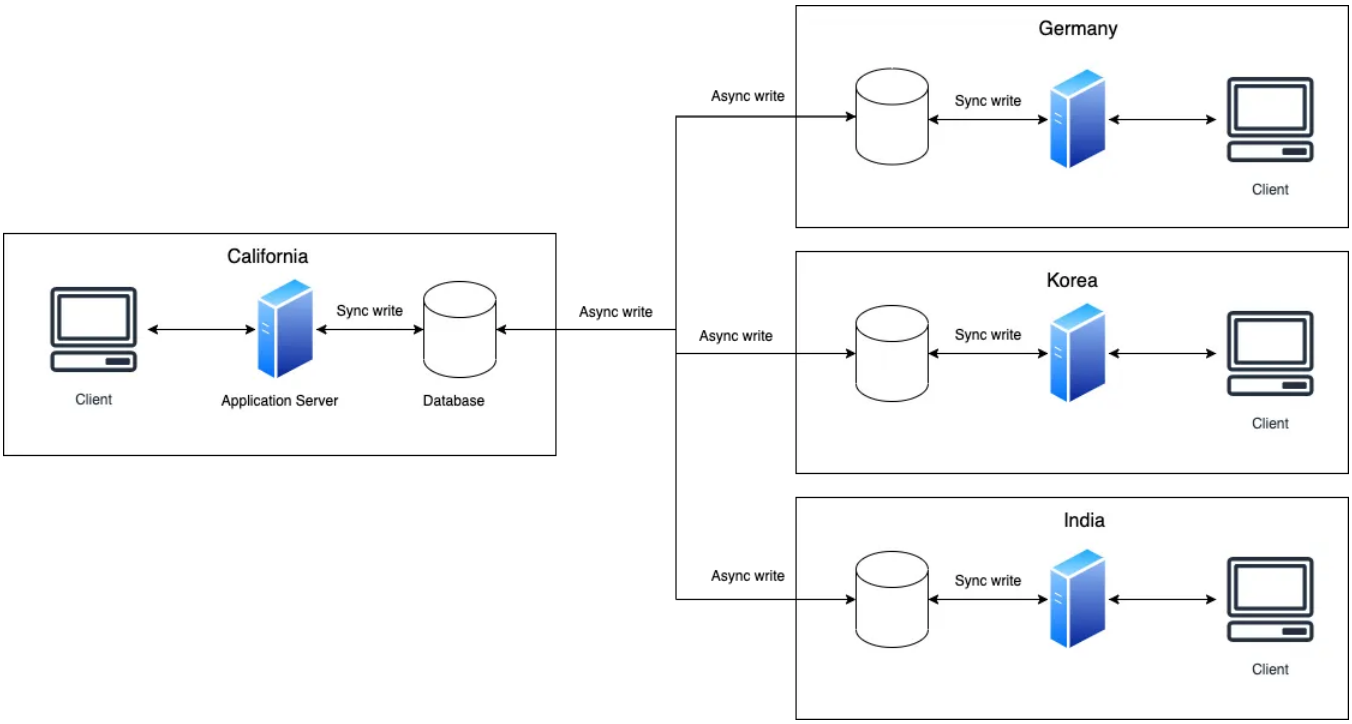


Fig. 3: Databases updating asynchronously for eventual consistency

This type of asynchronous updating is called eventual consistency.

What is Data Partitioning?

Data partitioning is a method of splitting a large database into many smaller parts. This is useful when you have a database that is overloaded with read and write requests and you would like to scale to improve throughput.

The data is split into multiple databases to improve the manageability, performance, availability, and load balancing of an application.

Horizontal Partitioning

Horizontal partitioning, also known as *Data Sharding*, splits a database by rows into separate databases.

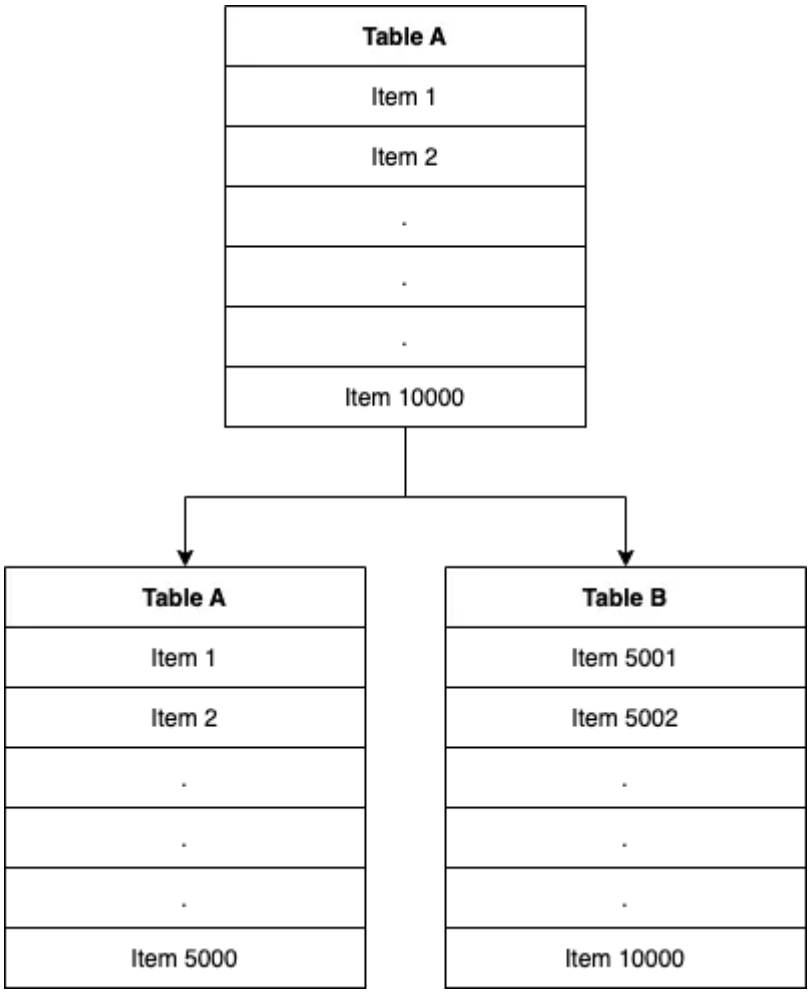


Fig. 4: Table A is split horizontally into two tables. Table A holds items 1–5000 and Table B holds items 5001–10000

In figure 4, Imagine we have a database with one table, Table A, and it has 10000 rows. We can partition this table horizontally by splitting the data between two tables (also known as shards). One table holds the first 1–5000 items and the second table holds 5001–10000 items.

One problem with horizontal partitioning is that if the partitioning strategy isn't chosen carefully, you could end up with unbalanced databases (i.e. one of the databases holds significantly more data than the others). An example is if we wanted to partition a database into shards based on usernames, like figure 5 below.

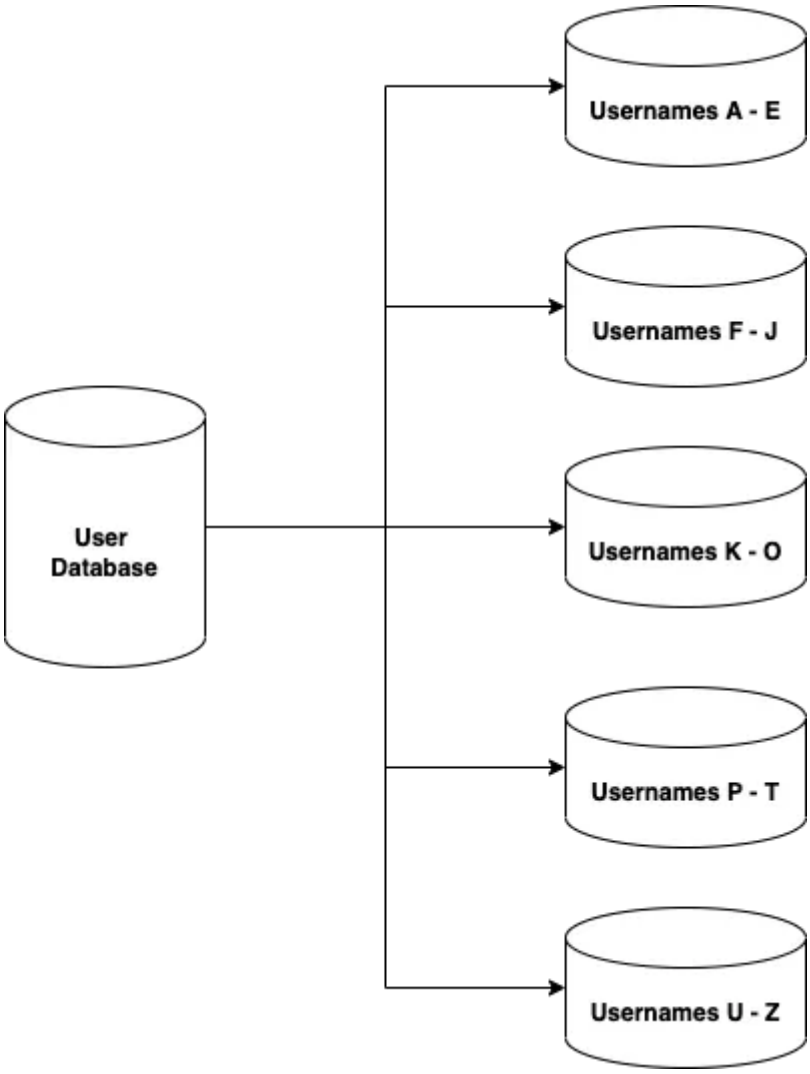


Fig. 5: User database partitioned by username

You may end up with the database that stores usernames U — Z holding a lot less than the other shards if less users begin their usernames with these letters.

What Partitioning Strategies Can We Use?

1. **Key or Hash-based partitioning**: We can use a hash function to determine which piece of data goes to which shard. By using consistent hashing, we can ensure that the data is evenly distributed and are easily able to add additional shards as needed.

2. **List Partitioning**: each partition is assigned a list of values. When we want to insert a new record, we determine which partition based on which list it belongs under. The example in figure 5 is using list partitioning.

3. **Mix of Hash and List Partitioning**: You can apply both strategies together to partition your databases. Ex. hash the data and then use ranges of hashes as the list partitions.

Vertical Partitioning

With this method, data is split by columns rather than row. For example, if we had a database storing user information in a user table like figure 6:

User Table

id	userName	firstName	lastName	address	profilePhoto	friendsIds

Fig. 6: A user table

We can partition this table by setting up a separate database to hold profile photos and another for friends lists.

User Table

id	userName	firstName	lastName	address

Profile Photos Table

id	userName	profilePhoto

Friends List Table

id	userName	friendsIds

Fig. 7: User data has been partitioned into separate tables

With vertical partitioning, if a partition becomes too large, you can also apply horizontal partitioning to that partition to scale further.

I hope you've enjoyed reading this article and now know how to distribute data in a distributed system

Understanding Database Sharding | DigitalOcean

Any application or website that sees significant growth will eventually need to scale in order to accommodate increases...

www.digitalocean.com

System Design - Database Replication | Synchronous vs Asynchronous

What is Database Sharding?

Software Engineering | System Design Interview | System Design | Software Development | Software Architecture

Follow

Written by Mariam Jaludi

557 Followers · Writer for Level Up Coding

Currently software engineering @Prime Video. Artist on the side. https://www.linkedin.com/in/mariam-jaludi/

More from Mariam Jaludi and Level Up Coding

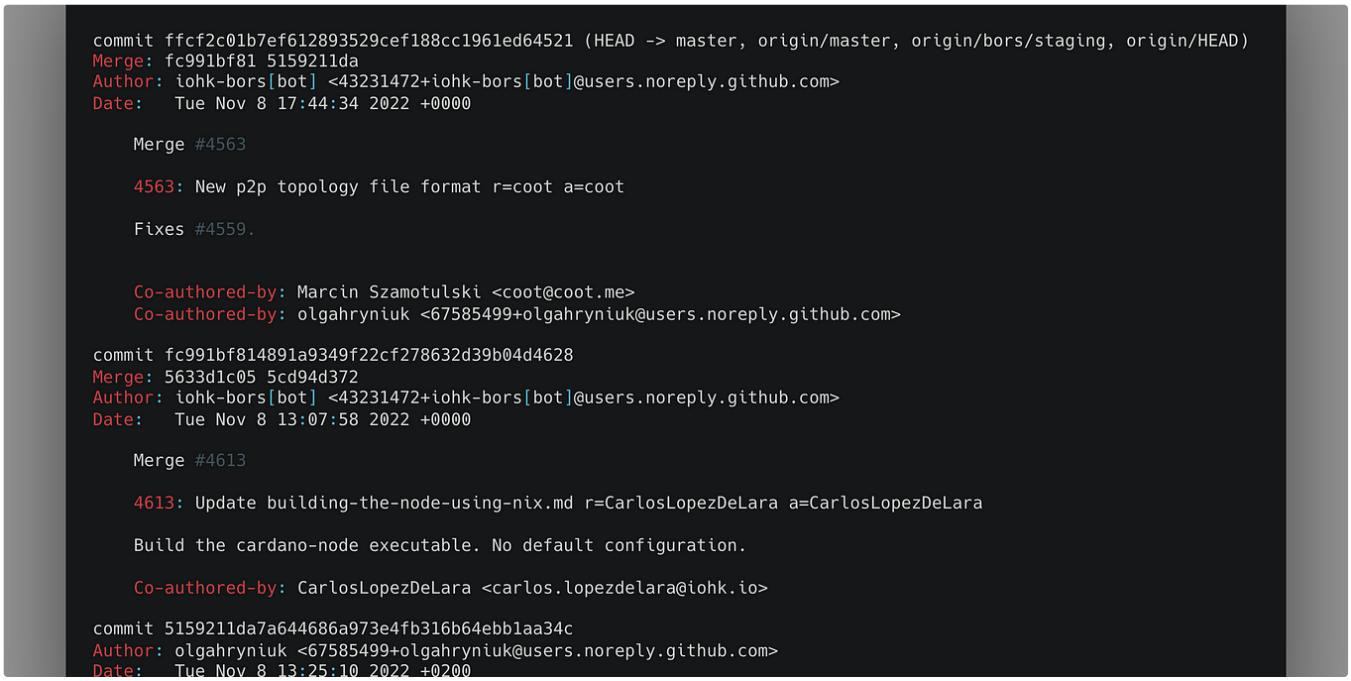
Mariam Jaludi in The Startup

Data Structures: Heaps

This post is the fifth in a series on data structures. The topics covered in this series are 6 major data structures that will come up in...

8 min read · Sep 15, 2019

556 7



Jacob Bennett in Level Up Coding

Use Git like a senior engineer

Git is a powerful tool that feels great to use when you know how to use it.

✨ · 4 min read · Nov 15, 2022

6.8K 69

Arslan Mirza in Level Up Coding

The Dark Side of Software Engineering

Why I Regret My Career Choice

· 10 min read · Jun 13

741 15

Mariam Jaludi in The Startup

Data Structures: Graphs

This post is the sixth in a series on data structures. The topics covered in this series are 6 major data structures that will come up in...

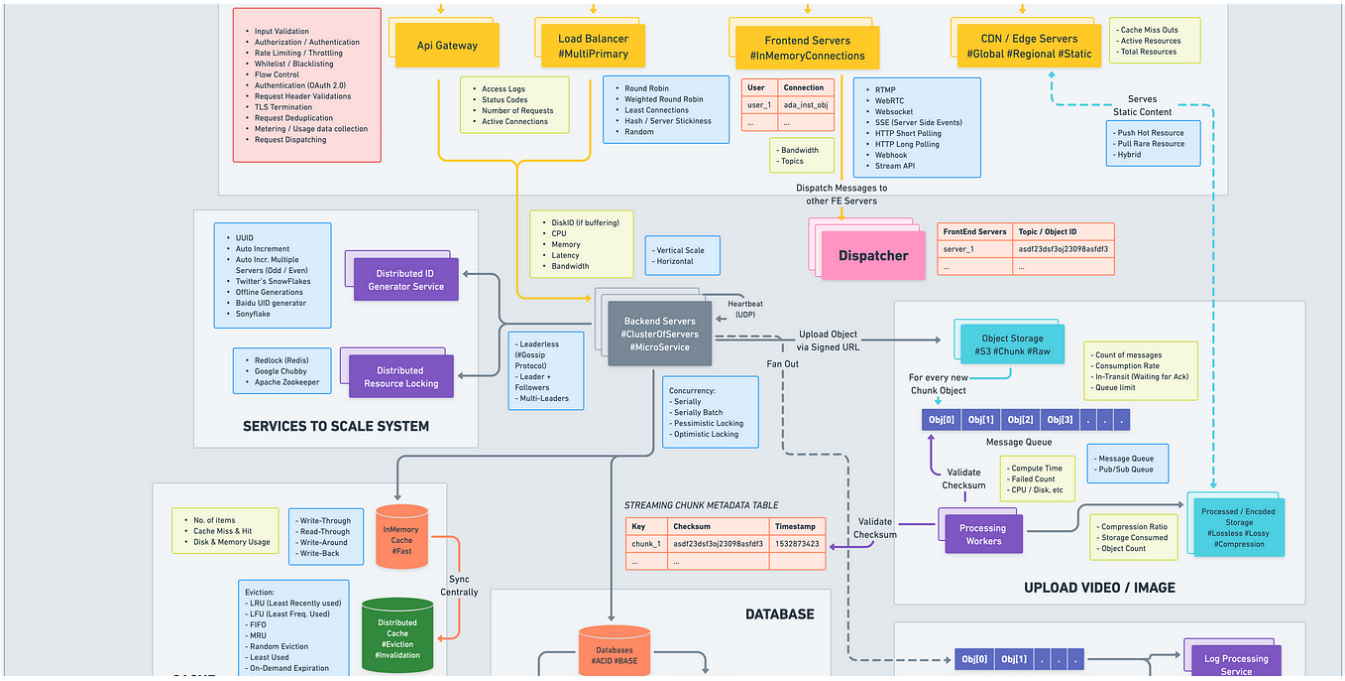
6 min read · Sep 23, 2019

384 1

See all from Mariam Jaludi

See all from Level Up Coding

Recommended from Medium



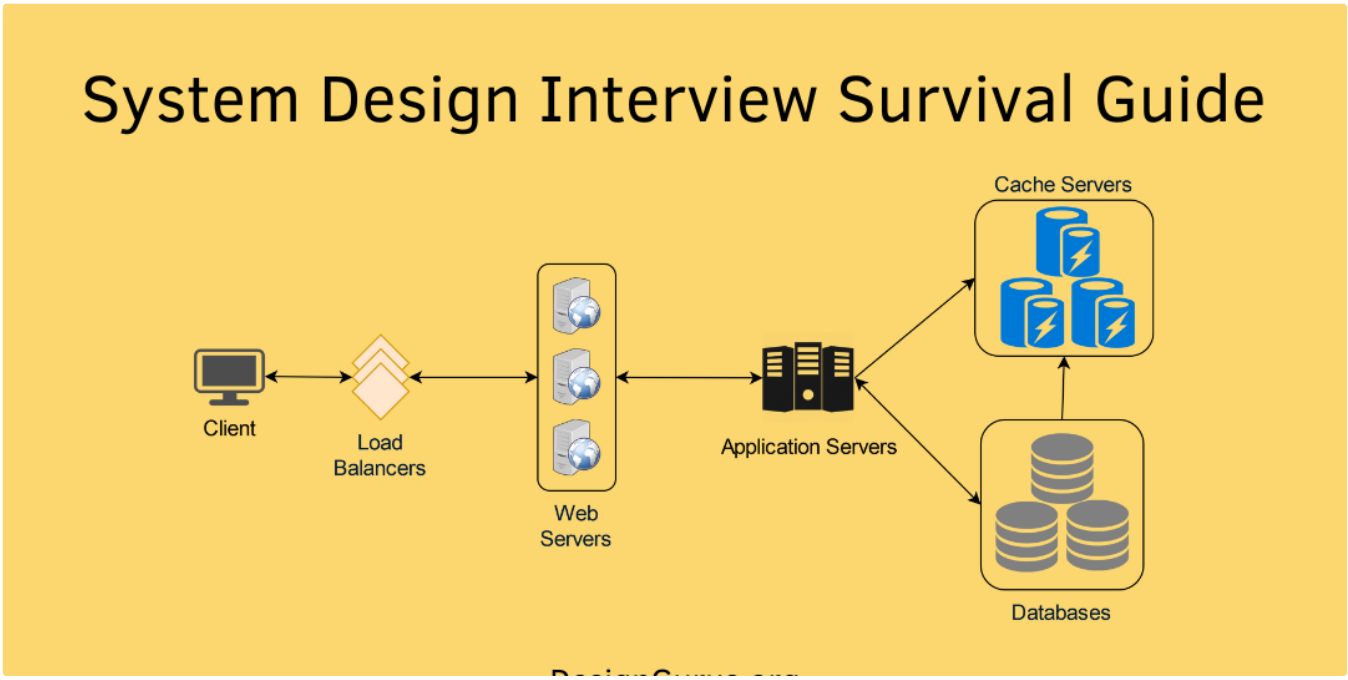
Love Sharma in Dev Genius

System Design Blueprint: The Ultimate Guide

Developing a robust, scalable, and efficient system can be daunting. However, understanding the key concepts and components can make the...

✦ · 9 min read · Apr 20

4.8K 37



Arslan Ahmad in Level Up Coding

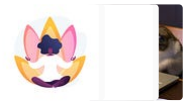
System Design Interview Survival Guide (2023): Preparation Strategies and Practical Tips

System Design Interview Preparation: Mastering the Art of System Design.

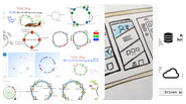
· 14 min read · Jan 19

1.8K 8

Lists



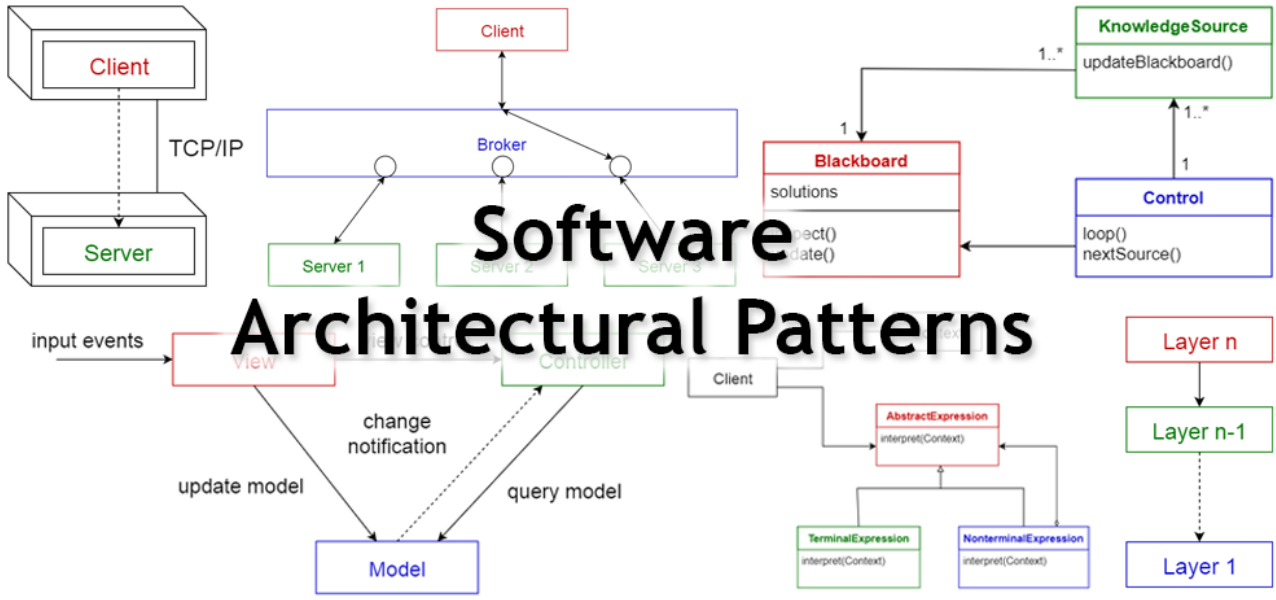
Stories to Help You Grow as a Software Developer
19 stories · 156 saves



General Coding Knowledge
20 stories · 41 saves

Leadership
31 stories · 67 saves

It's never too late or early to start something
10 stories · 15 saves



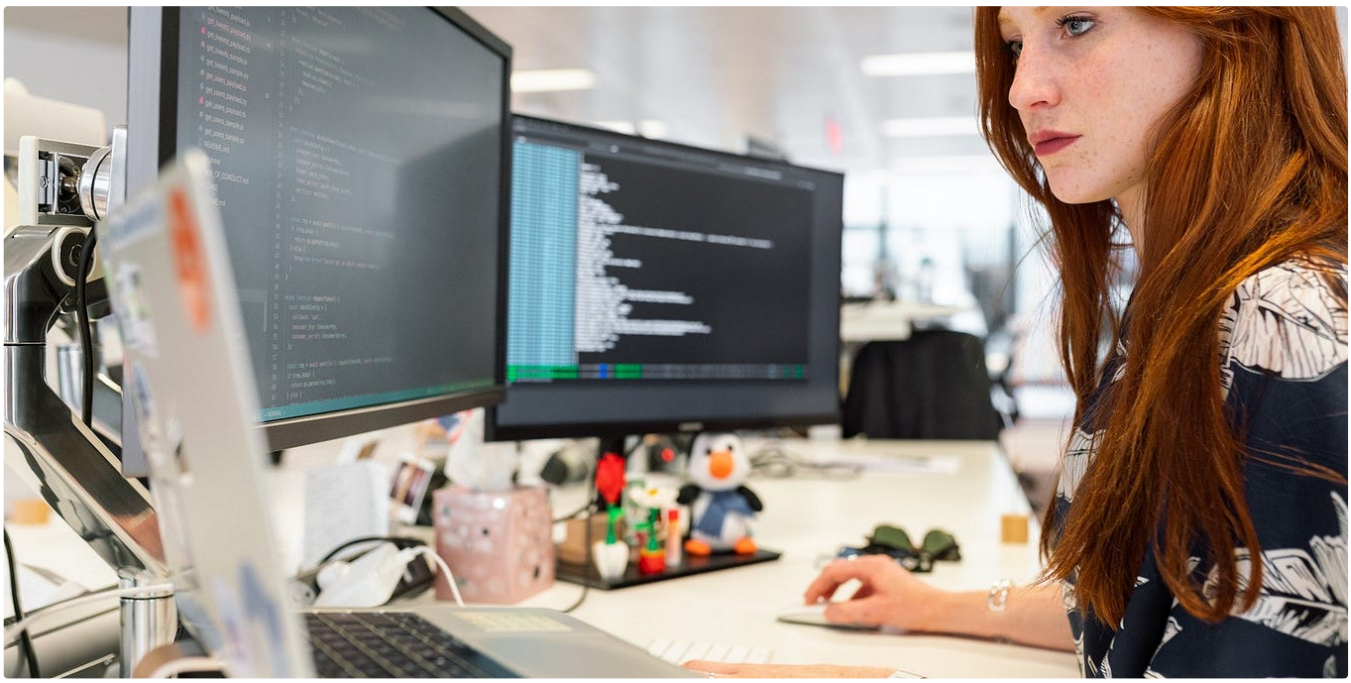
Vijini Mallawaarachchi in Towards Data Science

10 Common Software Architectural Patterns in a nutshell

Ever wondered how large enterprise scale systems are designed? Before major software development starts, we have to choose a suitable...

· 5 min read · Sep 4, 2017

38K 125



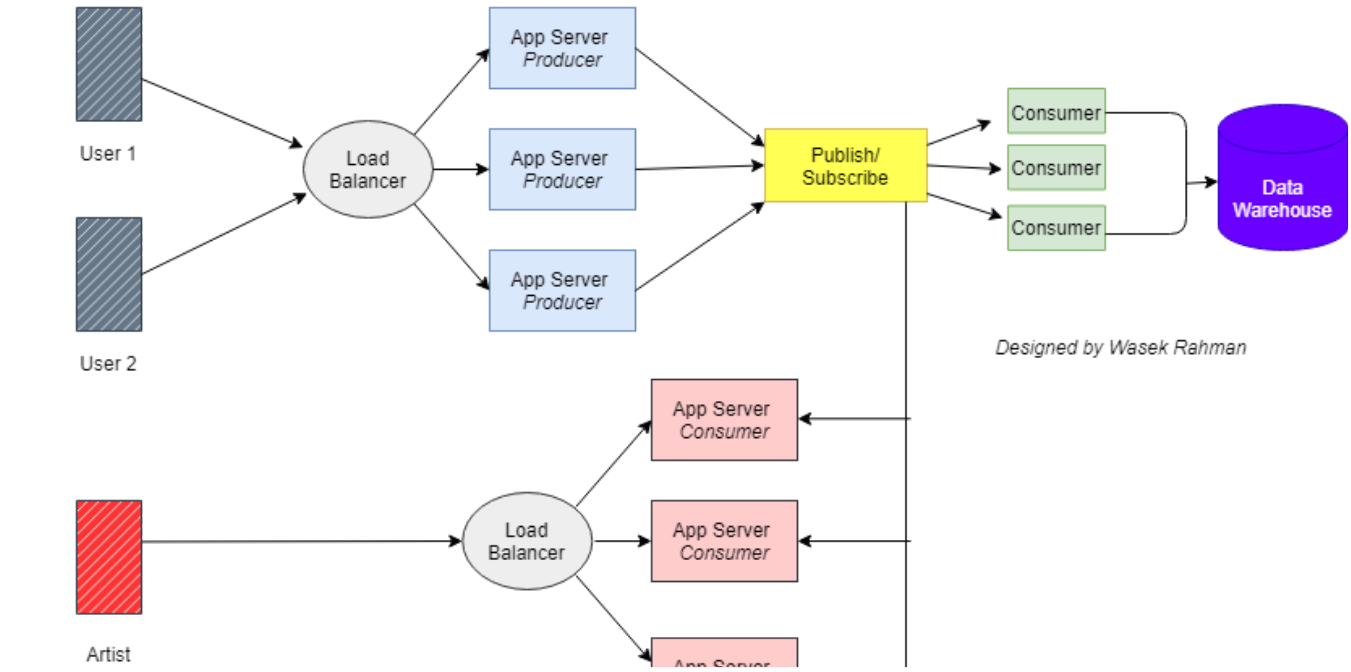
The Coding Diaries in The Coding Diaries

Why Experienced Programmers Fail Coding Interviews

A friend of mine recently joined a FAANG company as an engineering manager, and found themselves in the position of recruiting for...

✨ · 5 min read · Nov 2, 2022

4.5K 103



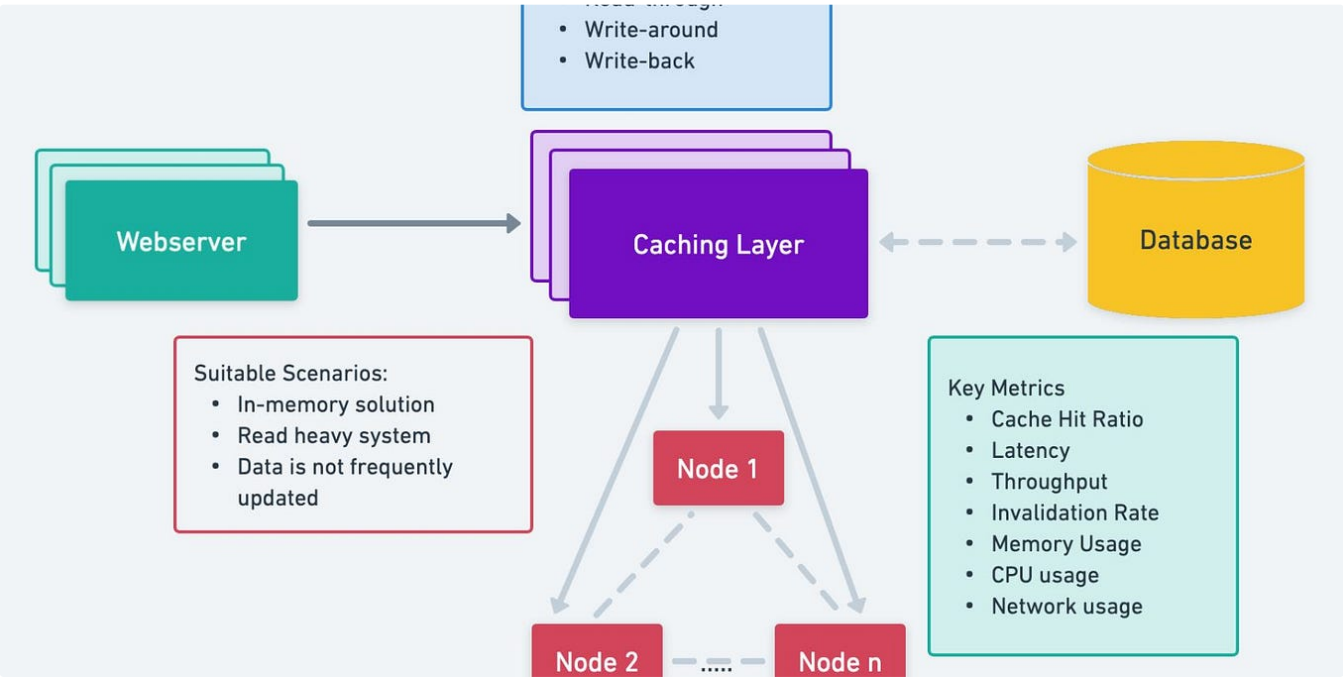
Wasek Rahman in Bootcamp

System Design Interview Prep: Spotify Real-Time Player

One of the most critical sections of interviewing at a top company or senior role is the System Design section where you would have to come...

✨ · 4 min read · Feb 18

112 1



Love Sharma in Dev Genius

A Comprehensive Guide to Distributed Caching

An essential website requires a web server to receive requests and a database to write or read data. However, this simple setup will only...

· 13 min read · Feb 9

513 4