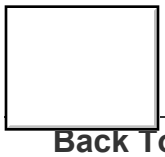

Join

Back To Course Home

Grokking Modern System Design Interview for Engineers & Managers

0% completed

Search Course

System Design Interviews

Introduction

Abstractions

Non-functional System Characteristics

Back-of-the-envelope Calculations

Building Blocks

Domain Name System

Load Balancers

Databases

Key-value Store

☐

Content Delivery Network (CDN)

☐

Sequencer

☐

Distributed Monitoring

☐

Monitor Server-side Errors

☐

Monitor Client-side Errors

☐

Distributed Cache

☐

Distributed Messaging Queue

☐

Pub-sub

☐

Rate Limiter

☐

Blob Store

☐

Distributed Search

☐

Distributed Logging

☐

Distributed Task Scheduler

- [ ]

Sharded Counters

- [ ]

Concluding the Building Blocks Discussion

- [ ]

Design YouTube

- [ ]

Design Quora

- [ ]

Design Google Maps

- [ ]

Design a Proximity Service / Yelp

- [ ]

Design Uber

- [ ]

Design Twitter

- [ ]

Design Newsfeed System

- [ ]

Design Instagram

- [ ]

Design a URL Shortening Service / TinyURL

- [ ]

Design a Web Crawler

Design WhatsApp

Design Typeahead Suggestion

- System Design: The Typeahead Suggestion System
- Requirements of the Typeahead Suggestion System’s Design
- High-level Design of the Typeahead Suggestion System
- Data Structure for Storing Prefixes
- Detailed Design of the Typeahead Suggestion System
- Evaluation of the Typeahead Suggestion System’s Design
- Quiz on the Typeahead Suggestion System’s Design

Design a Collaborative Document Editing Service / Google Docs

Spectacular Failures

Concluding Remarks

Course Certificate

Mark Course as Completed

Detailed Design of the Typeahead Suggestion System

Learn about the detailed design of the typeahead suggestion system.

We'll cover the following

- Detailed design
  - Suggestion service
  - Assembler

Detailed design#

Let’s go over the flow and interaction of the components shown in the illustration below. Our design is divided into two main parts:

- A suggestion service
- An assembler

Suggestion service#

At the same time that a user types a query in the search box, the `getSuggestions(prefix)` API calls hit the suggestions services. The top ten popular queries are returned from the distributed cache, Redis.

Assembler#

In the previous lesson, we discussed how tries are built, partitioned, and stored in the database. However, the creation and updation of a trie shouldn't come in the critical path of a user's query. We shouldn't update the trie in real time for the following reasons:

- There could be millions of users entering queries every second. During such phases with large amounts of incoming traffic, updating the trie in real time on every query can slow down our **suggestion service**.
- We have to provide top suggestions that might not frequently change after the creation or updation of the trie. So, it's less important to update the trie frequently.

In light of the reasons given above, we have a separate service called an assembler that's responsible for creating and updating tries after a certain configurable amount of time. The assembler consists of the following different services:

- **Collection service:** Whenever a user types, this service collects the log that consists of phrases, time, and other metadata and dumps it in a database that's processed later. Since the size of this data is huge, the **Hadoop Distributed File System (HDFS)** is considered a suitable storage system for storing this raw data.

  An example of the raw data from the collection service is shown in the following table. We record the time so that the system knows when to update the frequency of a certain phrase.

Raw Data Collected by the Collection Service

Phrases	Date and Time (DD-MM-YYYY HH:MM:SS)
UNIVERSAL	23-03-2022 10:16:18
UNIVERSITY	23-03-2022 10:20:11
UNIQUE	23-03-2022 10:21:10
UNIQUE	23-03-2022 10:22:24
UNIVERSITY	23-03-2022 10:25:09

- **Aggregator:** The raw data collected by the **collection service** is usually not in a consolidated shape. We need to consolidate the raw data to process it further and to create or update the tries. An aggregator retrieves the data from the HDFS and distributes it to different workers. Generally, the MapReducer is responsible for aggregating the frequency of the prefixes over a given interval of time, and the frequency is updated periodically in the associated Cassandra database. **Cassandra** is suitable for this purpose because it can store large amounts of data in a tabular format.

  The following table shows the processed and consolidated data within a particular period. This table is updated regularly by the aggregator and is stored in a hash table in a database like Cassandra. For simplicity, we assume that our data is case insensitive.

Useful Information Extracted from the Raw Data

Phrases	Frequency	Time Interval
UNIVERSAL	1400	1st 15 minutes
UNIVERSITY	1340	1st 15 minutes
UNIQUE	1200	1st 15 minutes

- **Trie builder:** This service is responsible for creating or updating tries. It stores these new and updated tries on their respective shards in the trie database via ZooKeeper. Tries are stored in persistent storage in a file so that we can rebuild our trie easily if necessary. NoSQL document databases such as MongoDB are suitable for storing these tries. This storage of a trie is needed when a machine restarts.

  The trie is updated from the aggregated data in the Cassandra database. The existing snapshot of a trie is updated with all the new terms and their corresponding frequencies. Otherwise, a new trie is created using the data in the Cassandra database.

  Once a trie is created or updated, the system makes it available for the suggestion service.

Point to Ponder

Question

Should we collect data and build a trie per user, or should it be shared among all users?

Show Answer

Next

Evaluation of the Typeahead Suggest…

Mark as Completed

Report an Issue