Лена Анюшева
t.me/lenka_ne_work

System Design Interview Prep

Кроме технической части, интервьюер оценивает

- **Technical communication** -> ответ должен быть понятным, мотивированным, стоит реагировать на подсказки интервьюера
- **Care about user** -> когда собираешь Functional и non-functional requirements, полезно поставить себя на место пользователя

Make life easier for the interviewer -- write down everything!

Structure of the Answer

Functional Requirements

Что система должна делать, на какие реквесты как отвечать. Это хорошее место, чтобы показать, что ты think about user. Пример:

- **putData**
  - Restrictions on data: schema, length
  - What to do with duplicate requests?
- **getData**
  - Pagination? Infinite scroll?
- **user auth**?
- Do we want to provide **analytics**?

Non-Functional Requirements

- **High Availability** -> no single point of failure
- **Scalability** -> cause the may be spikes in load
- **Low latency**, which requests are more important?
- **Read to write ratio**? Ex. 1:100
- What **load** do we expect? Ex. 500M put requests/day
- When does data **expire**? Ex. 10 years

API

Есть варианты: REST, RPC, SOAP. REST is widely used and adopted, good for async communication and unreliable network.

- www.site.com/bookTicket?{} PUT
    - body: {userId: ... , ticketID},
    - Response:
        - 200OK,
        - 500ISR,
        - not authorized
- www.site.com/ticket?{} GET
    - body: {userId: ... },
    - Response:
        - 200OK, list of tickets, ticket has date, id, eventId
        - 500ISR,
        - not authorized

Data Modeling

How to choose DB?

- SQL (например Postgres, MySQL)
    - Support for transactions
    - Support for data schema
    - Relations between entities
- NoSQL (например MongoDB)
    - Easier to scale

Обычно можно выбрать SQL базу данных и написать для нее data schema для основных таблиц. Укажи, где **primary index**, где **secondary index**.

Пример:

Events

- eventId <- primary key
- Date
- performer <- secondary index
- imageUrl

Tickets

- ticketId <- primary key - 5 byte
- eventId <- secondary key - 5 byte

Back of the Envelope Estimates

Нагрузку из NFR пересчитать в формате количества реквестов в секунду. Пример:

- Write requests: 500M requests / day = 500M requests / 24 / 3600 = 5700 r/sec
- Read requests: 5700 r/sec * 100 = 570.000 r/sec

Сколько данных надо будет хранить. Посчитать, сколько места на диске нужно, чтобы хранить данные, которые накопятся за N лет. Пример:

- 500M requests / day = 500M requests * 365 * 10 = 1.825 B new records / 10 years
- 1.825 B new records * 2.5 Kb / record = 4.5 PB / 10 years

Architecture

Самая базовая архитектура выглядит так:.

Client -> **API Gateway** -> **Load Balancer** -> **Replicated Service** -> **Replicated DB**

В зависимости от задачи можно добавлять:

- Cache
- Separate read and write path
- Additional DB
- Message Queue (Kafka, RabbitMQ)
- Stream processor (Flink)
- Data Lake, ETL process
- Batch job (for analytics, machine learning, etc)

Общие трюки, которые могут пригодиться

В зависимости от задачи

Как рассчитать минимальную длину unique string

Пригодится например для задачи TinyURL

- [a-zA_Z00-9] 62 символов, которые будем использовать
- Пусть n -- длина unique string, тогда
- 62 ^ n всего различных значений unique string
- Если нужно покрывать m различных значений, то
- n >= log(m) / log(62) -- минимальная необходимая длина строки

Geo-hashing

Каждую локацию кодируем с помощью строки, Чем длиннее строка, тем точнее локация. Например

- “AABA” -- город,
- “AABACB” -- район внутри города,
- “AABACBFFF” -- конкретный дом

Sequencer

Задача: генерировать последовательно следующее число в распределенной системе.

Проблемы:

- Любая машина может упасть, поэтому нельзя выбрать master, который будет генерировать
- Из-за задержек в network данные могут приходить в другом порядке.

Как решается: алгоритмом консенсуса, например PAXOS или RAFT.

Где прочитать больше: Designing Data Intensive Applications, Chapter 9: Consistency and Consensus

Больше трюков к конкретным задачам можно найти в System Design Interview – An insider's guide - Volume 1

Deep-dive topics

- Плохая новость! Интервьюер может спросить или не спросить любую из этих тем на любом вопросе.
- Хорошая новость! Если хорошо знаешь какую-то из этих тем, можно любой вопрос увести в любую из этих тем

Replication

Для чего нужна репликация?

- Geo-replication -> reduce latency
- Continue to work when some part goes down -> increase availability
- Scale the number of machines to serve requests -> increase read throughput

Синхронная vs. асинхронная репликация.

- Если репликация асинхронная, то возникают проблемы eventual consistency.
- Если синхронная, то выше latency и бывают cascading errors

Модели рекпликации:

- Single-leader
- Multi-leader
- Leaderless

Где прочитать больше: Designing Data Intensive Applications, Chapter 5: Replication

Partitioning

Разделяем dataset по ключу на несколько частей, потому что весь dataset не помещается на одной машине

Где прочитать больше: Designing Data Intensive Applications, Chapter 6: Partitioning

DB Index

В основе B-Tree или LSM-tree.

Где прочитать больше: Designing Data Intensive Applications, Chapter 3: Storage and Retrieval

Consensus Algorithm

Используется для total ordering, применяется для leader election для single-leader replication или Sequencer.

Где прочитать больше: Designing Data Intensive Applications, Chapter 9: Consistency and Consensus

Capacity estimates

Как много реквестов может держать один сервер? Как много данных реалистично хранить на одной машине?

Где прочитать больше: погуглить бенчмарки конкретных баз данных. Бенчмарки постоянно обновляются, потому что железо меняется. Примеры:

- Бенчмарк Postgres от AWS
    - Postgres on r5.4xlarge
    - 500gb disk per machine
    - ~5k RPS
- Рекомендация по нагрузке на Kafka от Confluent
    - 15,000 RPS для стандартного кластера

Monitoring + Analytics

Обычно добавляется OLAP в качестве DataLake и ETL процесс.