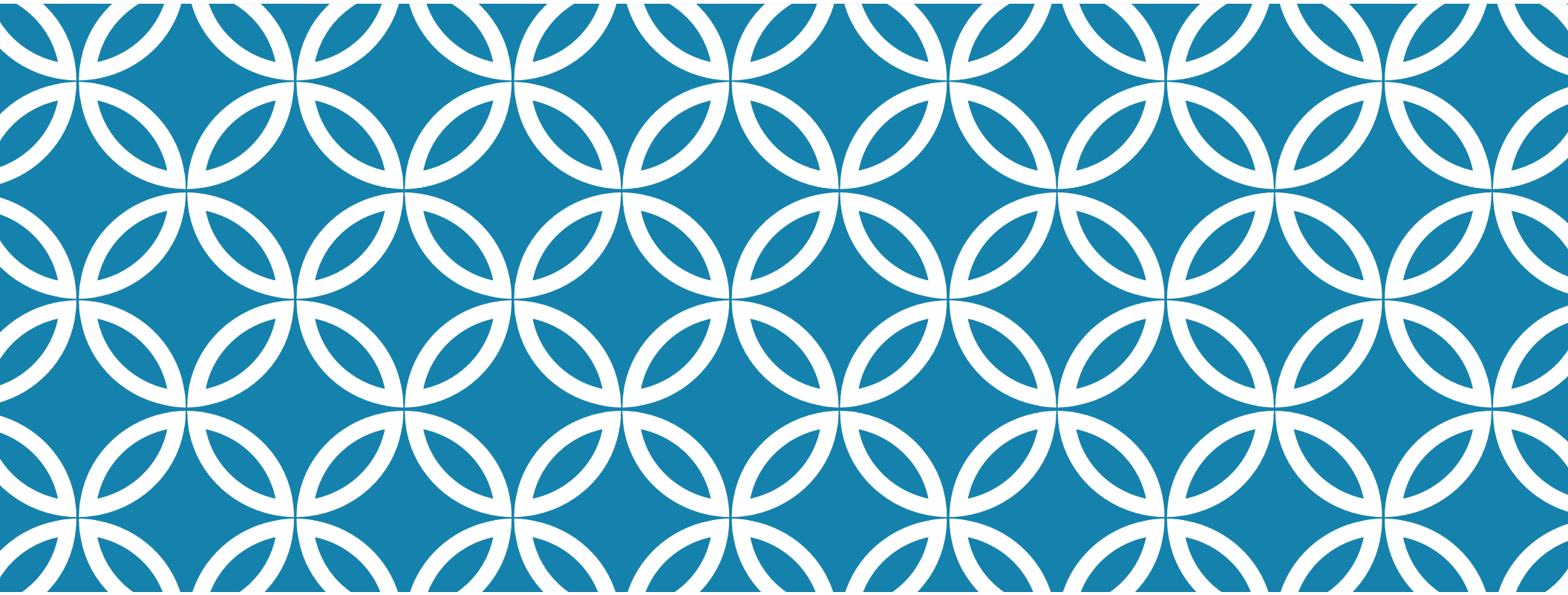


SYSTEM DESIGN INTERVIEW

SYSTEM DESIGN INTERVIEW PREP

Often intimidating, could be as vague as "**designing a well-known product X?". The questions are ambiguous and seem unreasonably broad.** How could anyone design a popular product in an hour that has taken hundreds if not thousands of engineers to build?

Good news: no one expects you to. Real-world system design is extremely complicated.

If no one expects you to design a real-world system in an hour, **what is the benefit of a system design interview?**

The system design interview simulates real-life problem solving where two co-workers collaborate on an ambiguous problem and come up with a solution that meets their goals.

The problem is open-ended, and there is no perfect answer.

The final design is less important compared to the work you put in the design process.

This allows you to demonstrate your design skill, defend your design choices, and respond to feedback in a constructive manner.

WHAT IS THE INTERVIEWER LOOKING FOR?

An effective system design interview gives strong signals about a person's ability to collaborate, to work under pressure, and to **resolve ambiguity constructively**.

The ability **to ask good questions** is also an essential skill, and many interviewers specifically look for this skill.

A good interviewer also looks for red flags.

Over-engineering is a real disease of many engineers as they delight in design purity and ignore tradeoffs. They are often unaware of the **compounding costs of over-engineered systems,** and many companies pay a high price for that ignorance. You certainly do not want to demonstrate this tendency in a system design interview. Other red flags include **narrow mindedness, stubbornness**, etc.

DO

- Always ask for clarification. Do not assume your assumption is correct.
- Understand the requirements of the problem.
- There is neither the right answer nor the best answer. A solution designed to solve the problems of a young startup is different from that of an established company with millions of users. Make sure you understand the requirements.
- Let the interviewer know what you are thinking. Communicate with your interview.
- Suggest multiple approaches if possible.
- Once you agree with your interviewer on the blueprint, go into details on each component. Design the most critical components first.
- Bounce ideas off the interviewer. A good interviewer works with you as a teammate.
- Never give up.

DON'T

- Don't be unprepared for typical interview questions.
- Don’t jump into a solution without clarifying the requirements and assumptions.
- Don’t go into too much detail on a single component in the beginning. Give the high-level design first then drills down.
- If you get stuck, don't hesitate to ask for hints.
- Again, communicate. Don't think in silence.
- Don’t think your interview is done once you give the design. You are not done until your interviewer says you are done. Ask for feedback early and often.

INTERVIEW FORMAT

System design interview questions are usually very broad, and 45 minutes or an hour is not enough to cover the entire design. Time management is essential.

How much time should you spend on each step?

The following is a very rough guide on distributing your time in a 45- minute interview session. Please remember this is a rough estimate, and the actual time distribution depends on the scope of the problem and the requirements from the interviewer.

Step 1 Understand the problem and establish design scope: 3 - 10 minutes

Step 2 Propose high-level design and get buy-in: 10 - 15 minutes

Step 3 Design deep dive: 10 - 25 minutes

Step 4 Wrap: 3 - 5 minutes

STEP1: UNDERSTAND THE PROBLEM AND DESIGN SCOPE

"Why did the tiger roar?"

A hand shot up in the back of the class.

"Yes, Jimmy?", the teacher responded.

"Because he was HUNGRY".

"Very good Jimmy."

Throughout his childhood, Jimmy has always been the first to answer questions in the class.

Whenever the teacher asks a question, there is always a kid in the classroom who loves to take a crack at the question, no matter if he knows the answer or not. That is Jimmy. Jimmy is an ace student. He takes pride in knowing all the answers fast. In exams, he is usually the first person to finish the questions. He is a teacher's top choice for any academic competition.

DON'T be like Jimmy.

THINK...ASK QUESTIONS

Do not jump right in to give a solution. Slow down.

Think deeply and ask questions to clarify requirements and assumptions. This is extremely important.

One of the most important skills as an engineer is to ask the right questions, make the proper assumptions, and gather all the information needed to build a system. **So, do not be afraid to ask questions.**

When you ask a question, the **interviewer either answers your question directly or asks you to make your assumptions.** If the latter happens, write down your assumptions on the whiteboard or paper. You might need them later.

What kind of questions to ask? Ask questions to understand the exact requirements. Here is a list of questions to help you get started:

- What specific features are we going to build?
- How many users does the product have?
- How fast does the company anticipate to scale up? What are the anticipated scales in 3 months, 6 months, and a year?
- What is the company's technology stack? What existing services you might leverage to simplify the design?

DESIGN A NEWS FEED

If you are asked to design a news feed system, you want to ask questions that help you clarify the requirements.

“News feed is the constantly updating list of stories in the middle of your home page. News Feed includes status updates, photos, videos, links, app activity, and likes from people, pages, and groups that you follow on Facebook” . This is a popular interview question. Similar questions commonly asked are: design Facebook news feed, Instagram feed, Twitter timeline, etc

The conversation between you and the interviewer might look like this:

What kind of questions to ask?

NEWS FEED DESIGN

Candidate: Is this a mobile app? Or a web app? Or both?

Interviewer: Both.

Candidate: What are the most important features for the product?

Interviewer: Ability to make a post and see friends' news feed.

Candidate: Is the news feed sorted in reverse chronological order or a particular order? The particular order means each post is given a different weight. For instance, posts from your close friends are more important than posts from a group.

Interviewer: To keep things simple, let us assume the feed is sorted by reverse chronological order.

Candidate: How many friends can a user have?

Interviewer: 5000

Candidate: What is the traffic volume?

Interviewer: 10 million daily active users (DAU)

Candidate: Can feed contain images, videos, or just text?

Interviewer: It can contain media files, including both images and videos.

STEP 2 - PROPOSE HIGH-LEVEL DESIGN AND GET BUY-IN

• Come up with an initial blueprint for the design. Ask for feedback. Treat your interviewer as a teammate and work together. Many good interviewers love to talk and get involved.

• Draw box diagrams with key components on the whiteboard or paper. This might include clients (mobile/web), APIs, web servers, data stores, cache, CDN, message queue, etc.

• Do back-of-the-envelope calculations to evaluate if your blueprint fits the scale constraints. Think out loud. Communicate with your interviewer if back-of-the-envelope is necessary before diving into it.

If possible, go through a few concrete use cases. This will help you frame the high-level design. It is also likely that the use cases would help you discover edge cases you have not yet considered.

Should we include API endpoints and database schema here?

TWO FLOWS

Feed publishing: when a user publishes a post, corresponding data is written into cache/database, and the post will be populated into friends' news feed.

Feed publishing API

To publish a post, a HTTP POST request will be sent to the server. The API is shown below:

POST /v1/me/feed

Params:

- content: content is the text of the post.
- auth_token: it is used to authenticate API requests.

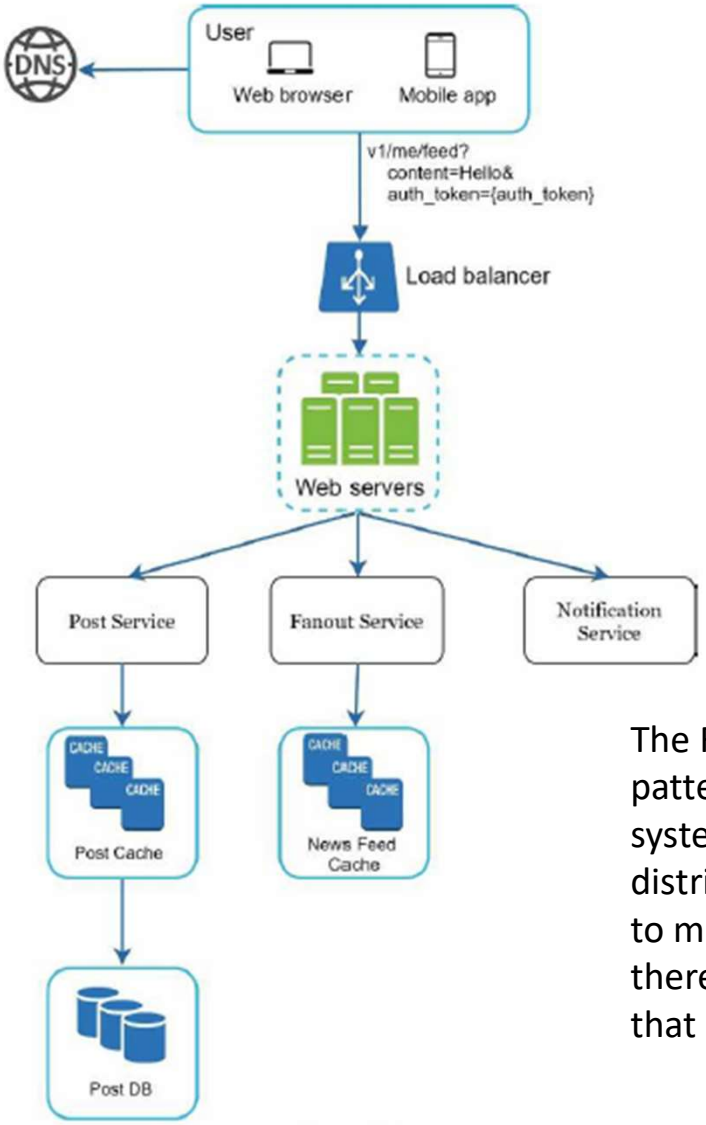
Newsfeed building: the news feed is built by aggregating friends' posts in a reverse chronological order.

The API to retrieve news feed is shown below:

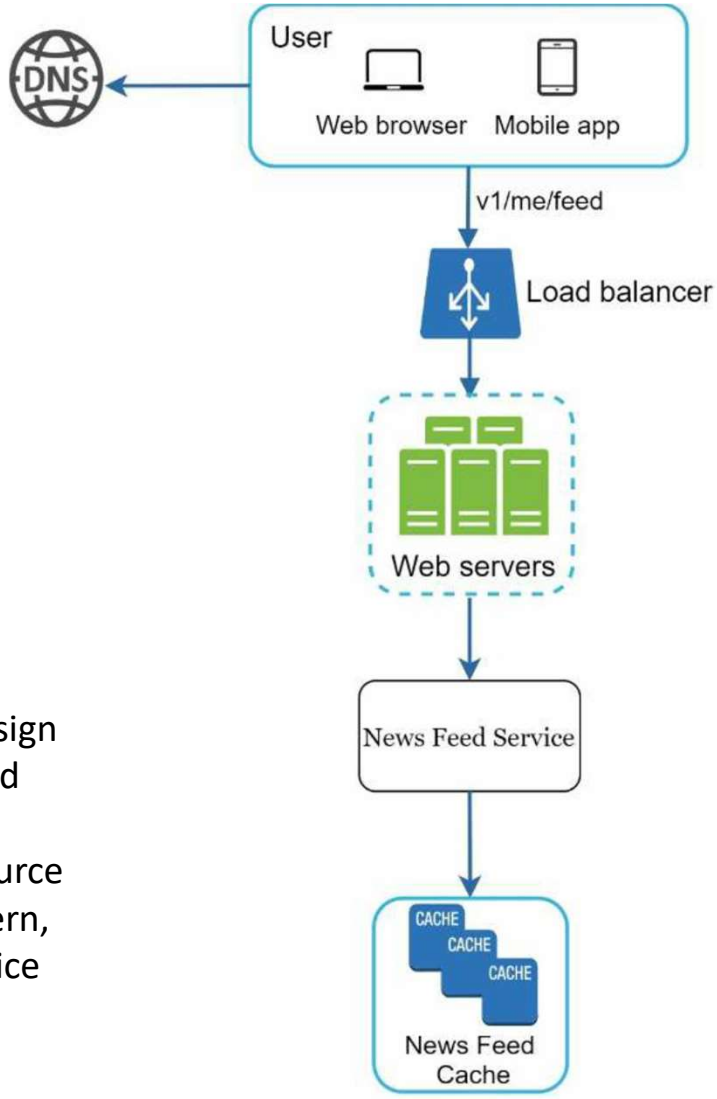
GET /v1/me/feed

Params:

- auth_token: it is used to authenticate API requests.



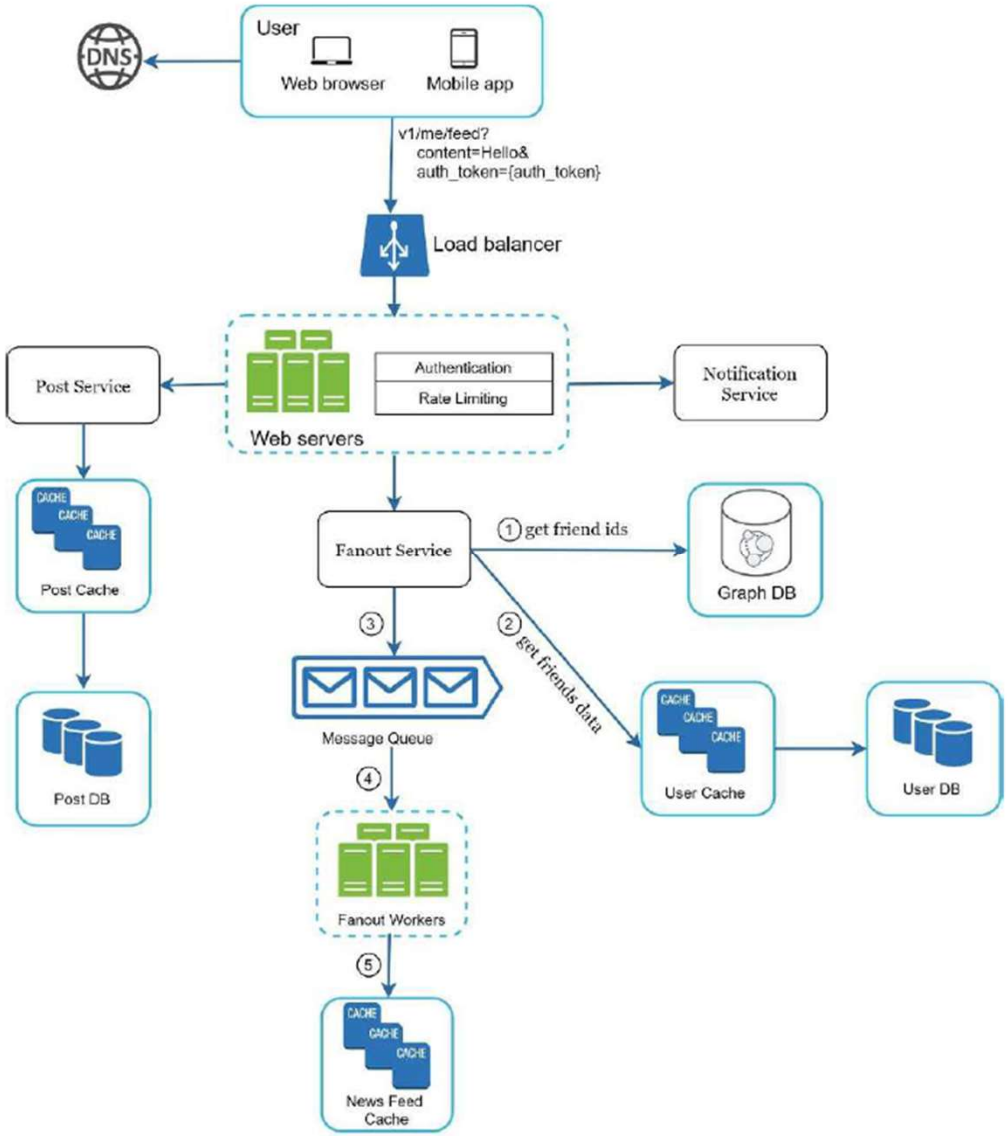
The Fan-out Pattern is a software design pattern commonly used in distributed systems to efficiently manage and distribute messages from a single source to multiple destinations. In this pattern, there is a central component or service that acts as the source of messages.

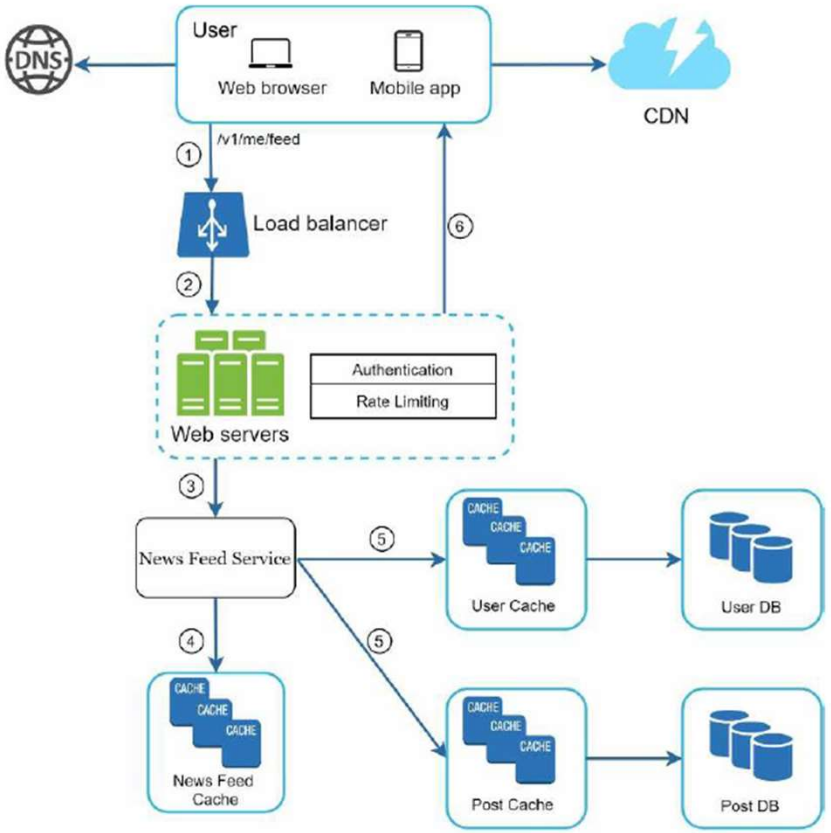


DESIGN DEEP DIVE

At this step, you and your interviewer should have already achieved the following objectives:

- Agreed on the overall goals and feature scope
- Sketched out a high-level blueprint for the overall design
- Obtained feedback from your interviewer on the high-level design
- Had some initial ideas about areas to focus on in deep dive based on her feedback


DNS
User
Web browser
Mobile app
v1/me/feed?
content=Hello&
auth_token={auth_token}
Load balancer
Post Service
Authentication
Rate Limiting
Notification Service
Web servers
CACHE
Post Cache
Fanout Service
1 get friend ids
Graph DB
3
2 get friends data
Post DB
Message Queue
User Cache
User DB
4
Fanout Workers
5
News Feed Cache


DNS
User
Web browser
Mobile app
CDN
1
/v1/me/feed
Load balancer
6
2
Authentication
Rate Limiting
Web servers
3
News Feed Service
5
User Cache
User DB
4
5
News Feed Cache
Post Cache
Post DB

WRAP-UP

- The interviewer might want you to identify the system bottlenecks and discuss potential improvements. Never say your design is perfect and nothing can be improved. There is always something to improve upon. This is a great opportunity to show your critical thinking and leave a good final impression.
- It could be useful to give the interviewer a recap of your design. This is particularly important if you suggested a few solutions. Refreshing your interviewer's memory can be helpful after a long session.
- Error cases (server failure, network loss, etc.) are interesting to talk about.
- Operation issues are worth mentioning. How do you monitor metrics and error logs?
- How to roll out the system?
- How to handle the next scale curve is also an interesting topic. For example, if your current design supports 1 million users, what changes do you need to make to support 10 million users?
- Propose other refinements you need if you had more time.

BACK OF THE ENVELOPE ESTIMATES

In a system design interview, sometimes you are asked to estimate system capacity or performance requirements using a back-of-the-envelope estimation.

According to Jeff Dean, Google Senior Fellow, “back-of-the-envelope calculations are estimates you create using a combination of thought experiments and common performance numbers to get a good feel for which designs will meet your requirements”.

You need to have a good sense of scalability basics to effectively carry out back-of-theenvelope estimation. The following concepts should be well understood: power of two, latency numbers every programmer should know, and availability numbers.

Power of two

Although data volume can become enormous when dealing with distributed systems, calculation all boils down to the basics. To obtain correct calculations, it is critical to know the data volume unit using the power of 2. A byte is a sequence of 8 bits. An ASCII character uses one byte of memory (8 bits). Below is a table explaining the data volume unit (Table 2-1).

Power	Approximate value	Full name	Short name
10	1 Thousand	1 Kilobyte	1 KB
20	1 Million	1 Megabyte	1 MB
30	1 Billion	1 Gigabyte	1 GB
40	1 Trillion	1 Terabyte	1 TB
50	1 Quadrillion	1 Petabyte	1 PB

Operation name	Time
L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns = 10 µs
Send 2K bytes over 1 Gbps network	20,000 ns = 20 µs
Read 1 MB sequentially from memory	250,000 ns = 250 µs
Round trip within the same datacenter	500,000 ns = 500 µs
Disk seek	10,000,000 ns = 10 ms
Read 1 MB sequentially from the network	10,000,000 ns = 10 ms
Read 1 MB sequentially from disk	30,000,000 ns = 30 ms
Send packet CA (California) ->Netherlands->CA	150,000,000 ns = 150 ms

By analyzing the numbers in Figure 2-1, we get the following conclusions:

- Memory is fast but the disk is slow.
- Avoid disk seeks if possible.
- Simple compression algorithms are fast.
- Compress data before sending it over the internet if possible.
- Data centers are usually in different regions, and it takes time to send data between them.

Availability numbers

High availability is the ability of a system to be continuously operational for a desirably long period of time. High availability is measured as a percentage, with 100% means a service that has 0 downtime. Most services fall between 99% and 100%.

A service level agreement (SLA) is a commonly used term for service providers. This is an agreement between you (the service provider) and your customer, and this agreement formally defines the level of uptime your service will deliver. Cloud providers Amazon [4], Google [5] and Microsoft [6] set their SLAs at 99.9% or above. Uptime is traditionally measured in nines. The more the nines, the better. As shown in Table 2-3, the number of nines correlate to the expected system downtime.

Availability %	Downtime per day	Downtime per year
99%	14.40 minutes	3.65 days
99.9%	1.44 minutes	8.77 hours
99.99%	8.64 seconds	52.60 minutes
99.999%	864.00 milliseconds	5.26 minutes
99.9999%	86.40 milliseconds	31.56 seconds

EXAMPLE: ESTIMATE TWITTER QPS AND STORAGE REQUIREMENTS

Please note the following numbers are for this exercise only as they are not real numbers from Twitter.

Assumptions:

- 300 million monthly active users.
- 50% of users use Twitter daily.
- Users post 2 tweets per day on average.
- 10% of tweets contain media.
- Data is stored for 5 years.

Estimations:

Query per second (QPS) estimate:

- Daily active users (DAU) = 300 million * 50% = 150 million
- Tweets QPS = 150 million * 2 tweets / 24 hour / 3600 seconds = ~3500
- Peek QPS = 2 * QPS = ~7000

We will only estimate media storage here.

- Average tweet size:
- tweet_id 64 bytes
- text 140 bytes
- media 1 MB
- Media storage: 150 million * 2 * 10% * 1 MB = 30 TB per day
- 5-year media storage: 30 TB * 365 * 5 = ~55 PB

TIPS

Back-of-the-envelope estimation is all about the process. Solving the problem is more important than obtaining results. Interviewers may test your problem-solving skills. Here are a few tips to follow:

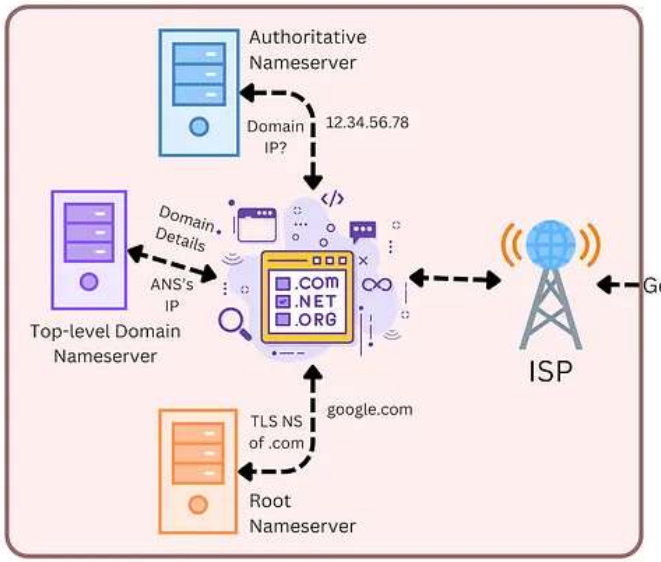
• Rounding and Approximation. It is difficult to perform complicated math operations during the interview. For example, what is the result of “99987 / 9.1”? There is no need to spend valuable time to solve complicated math problems. Precision is not expected. Use round numbers and approximation to your advantage. The division question can be simplified as follows: “100,000 / 10”.

• Write down your assumptions. It is a good idea to write down your assumptions to be referenced later.

• Label your units. When you write down “5”, does it mean 5 KB or 5 MB? You might confuse yourself with this. Write down the units because “5 MB” helps to remove ambiguity.

• Commonly asked back-of-the-envelope estimations: QPS, peak QPS, storage, cache, number of servers, etc. You can practice these calculations when preparing for an interview. Practice makes perfect.

System Design Blueprint: The Ultimate Guide



Metrics | Security | Options / Algo

M /@zonito

- Gzip, deflate
- Request ID
- Idempotent Key
- Many more...

Request

Response

- 4XX, 5XX, 2XX
- Pagination
- Expiry Headers
- Mime
- Cookie
- many more...

- Response Time
- Failed Status Codes
- Error Codes / Message

LOAD BALANCING

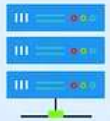
- Input Validation
- Authorization / Authentication
- Rate Limiting / Throttling
- Whitelist / Blacklisting
- Flow Control
- Authentication (OAuth 2.0)
- Request Header Validations
- TLS Termination
- Request Deduplication
- Metering / Usage data collection
- Request Dispatching

API Gateway

- Access Logs
- Status Codes
- Number of Requests
- Active Connections

Load Balancer
#MultiPrimary

- Round Robin
- Weighted Round Robin
- Least Connections
- Hash / Server Stickiness
- Random



Frontend Servers
#InMemoryConnections
For Live Streaming / Chat

- RTMP
- WebRTC
- Websocket
- SSE (Server Side Events)
- HTTP Short Polling
- HTTP Long Polling
- Webhook
- Stream API

User	Connection
user_1	ada_inst_obj

- Bandwidth
- Topics

Dispatch Messages to other FE Servers

CDN / Edge Servers
#Global #Regional #Static
Static / Stream with ABR

- Cache Miss Outs
- Active Resources
- Total Resources

- Push Hot Resource
- Pull Rare Resource
- Hybrid

Serves Static Content

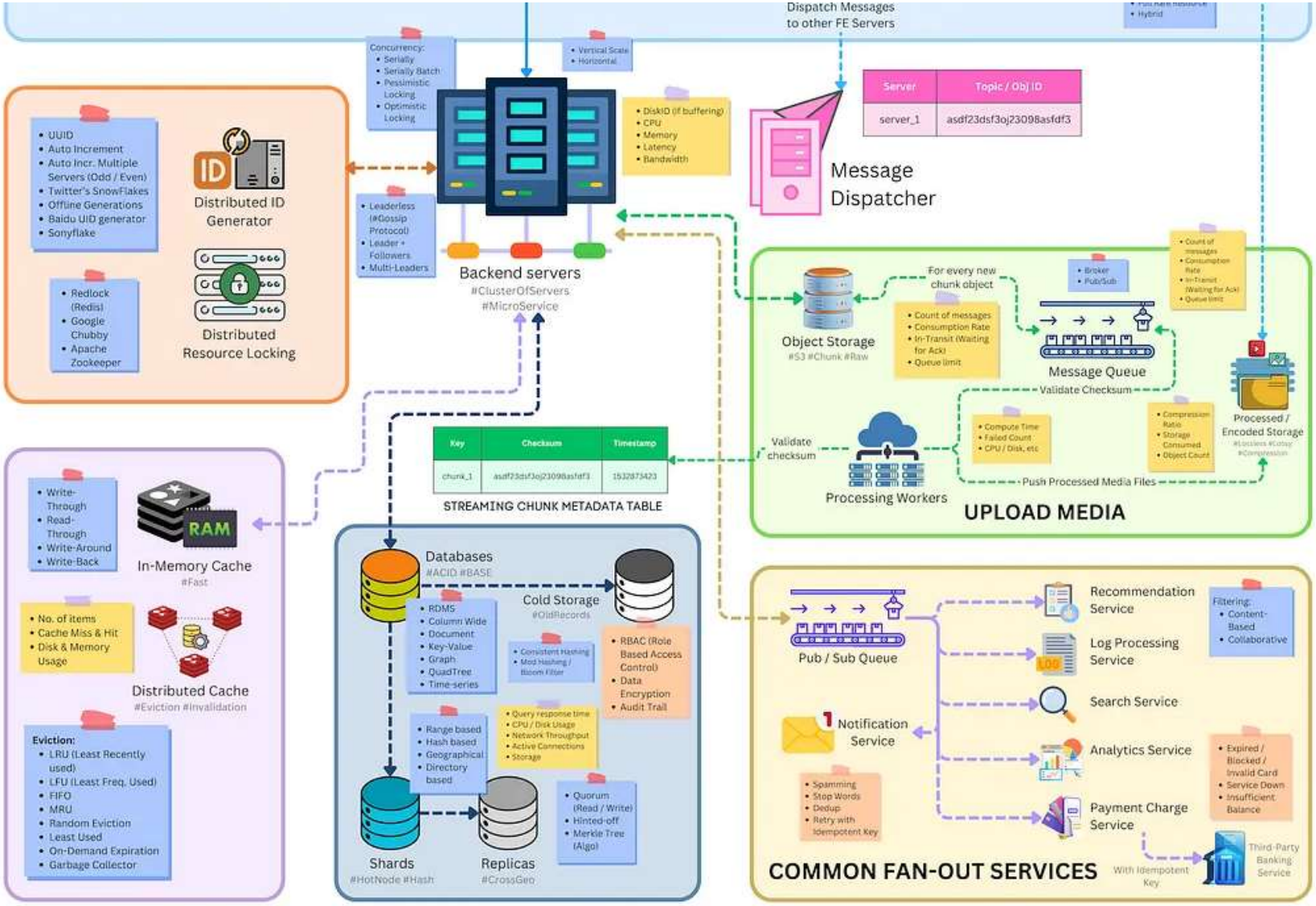
- Concurrency:
- Serially
- Serially (Batch
- Pessimistic Locking
- Optimistic Locking

- Vertical Scale
- Horizontal

- DiskIO (if buffering)
- CPU
- Memory

Server	Topic / Obj ID
server_1	asdf23dsf3oj23098asfdf3

- UUID


Dispatch Messages to other FE Servers
Concurrency:
• Serially
• Serially Batch
• Pessimistic Locking
• Optimistic Locking
• Vertical Scale
• Horizontal
• DiskIO (if buffering)
• CPU
• Memory
• Latency
• Bandwidth
Server
Topic / Obj ID
server_1
asdf23dsf3oj23098asfdf3
Message Dispatcher
• UUID
• Auto Increment
• Auto Incr. Multiple Servers (Odd / Even)
• Twitter's SnowFlakes
• Offline Generations
• Baidu UID generator
• Sonyflake
Distributed ID Generator
• Redlock (Redis)
• Google Chubby
• Apache Zookeeper
Distributed Resource Locking
• Leaderless (#Gossip Protocol)
• Leader + Followers
• Multi-Leaders
Backend servers
#ClusterOfServers
#MicroService
Object Storage
#S3 #Chunk #Raw
For every new chunk object
• Count of messages
• Consumption Rate
• In-Transit (Waiting for Ack)
• Queue limit
• Broker
• Pub/Sub
Message Queue
Validate Checksum
Processed / Encoded Storage
Validate checksum
Processing Workers
• Compute Time
• Failed Count
• CPU / Disk, etc
• Compression Ratio
• Storage Consumed
• Object Count
Push Processed Media Files
UPLOAD MEDIA
Key
Checksum
Timestamp
chunk_1
asdf23dsf3oj23098asfdf3
1532873423
STREAMING CHUNK METADATA TABLE
• Write-Through
• Read-Through
• Write-Around
• Write-Back
RAM
In-Memory Cache
#Fast
• No. of items
• Cache Miss & Hit
• Disk & Memory Usage
Distributed Cache
#Eviction #Invalidation
Eviction:
• LRU (Least Recently used)
• LFU (Least Freq. Used)
• FIFO
• MRU
• Random Eviction
• Least Used
• On-Demand Expiration
• Garbage Collector
Databases
#ACID #BASE
Cold Storage
#OldRecords
• RDMS
• Column Wide
• Document
• Key-Value
• Graph
• QuadTree
• Time-series
• Consistent Hashing
• Mod Hashing / Bloom Filter
• RBAC (Role Based Access Control)
• Data Encryption
• Audit Trail
• Range based
• Hash based
• Geographical
• Directory based
• Query response time
• CPU / Disk Usage
• Network Throughput
• Active Connections
• Storage
• Quorum (Read / Write)
• Hinted-off
• Merkle Tree (Algo)
Shards
#HotNode #Hash
Replicas
#CrossGeo
Pub / Sub Queue
Recommendation Service
Log Processing Service
Search Service
Notification Service
Analytics Service
Payment Charge Service
Filtering:
• Content-Based
• Collaborative
• Spamming
• Stop Words
• Dedup
• Retry with Idempotent Key
• Expired / Blocked / Invalid Card
• Service Down
• Insufficient Balance
With Idempotent Key
Third-Party Banking Service
COMMON FAN-OUT SERVICES