

O'REILLY®

Designing Data-Intensive Applications

THE BIG IDEAS BEHIND RELIABLE,
SOLID, AND MAINTAINABLE SYSTEMS



SECOND EDITION

SYSTEM DESIGN INTERVIEW



AN INSIDER'S GUIDE

Alex Xu

TIPS for System Design Interview

Friday, 31 December 2021 3:09 PM

Start with the Expectations, i.e. functional, nonfunctional requirements

① Don't go into details prematurely

↳ But, when it comes to DB design, ask if they want you to draw ER diagram right then (ideally, we also want to do that)

② Don't have a said Architecture 100%
i.e. not caring about requirements and designing
as you've already read the question.

③ Keep it simple, stupid (KISS)

④ Don't make statements without justification.

↳ e.g.: saying you'll use a NoSQL DB without any relevant statements to back the design.

⑤ Be aware of current technologies

↳ e.g.: say you'll use a CDN, it might be nice to mention a real CDN tech. like Akamai
(or) CloudFront

(or)

For a message queue, say you'll use maybe
RabbitMQ / Kafka etc..

- ⑥ Don't do capacity estimation without any reason
↳ Maybe speak out with interviewer when you notice a particular service needs to know the capacity when you're 100% sure.
- ⑦ mention n/w protocols b/w communication of the systems in architecture !!! And Mention pros & cons of using one over other (ex: HTTP vs FTP etc...)
- ⑧ Speak out internals of system (ex:- why or NOSQL DB like Cassandra over MySQL. what does Cassandra bring to the architecture other DB's cannot)
- ⑨ TRADE OFFS !!!

Terms & Analogy

Saturday, 25 December 2021 3:07 PM

① Scaling → Horizontal
→ Vertical

② Load Balancing

③ Backups (Avoid single point of failure)

④ Microservices Architecture (Isolate Responsibilities)

⑤ Distributed System (Partitioning)

⑥ Decoupling

⑦ Logging & Metrics

⑧ Caching

⑨ API (Application programmable interface)

⑩ CAP Theorem (Consistency, Availability, Partition Tolerance)

⑪ CDN (Content Delivery Network)

⑫ DNS (Domain name system)

⑬ Ports (ex. HTTP - 80, HTTPS - 443)

⑭ Network protocols (IP, HTTP, TCP)

Snippets

Friday, 14 January 2022 5:23 PM

Read heavy?

↳ Throw a Cache

Write heavy?

↳ Push write reqs to a message queue.

Search Data?

↳ Put a search index

More users? Need to scale database?

↳ Need ACID properties?

↳ Shard the RDBMS Database

↳ ACID Properties not mandatory?

↳ use NoSQL Database

High throughput?

↳ Scale application servers & put load balancer with
consistent hashing

Network Protocols

Wednesday, 12 January 2022

10:39 PM

IP
TCP
HTTP

Just a set of rules to interact b/w 2 things.

IP (Internet Protocol)

2^{16} bytes ≈ 0.06 MB

2 main things in IP packet are

① Header (src, target, total size of packet, IP version etc.)
↓
very small in general

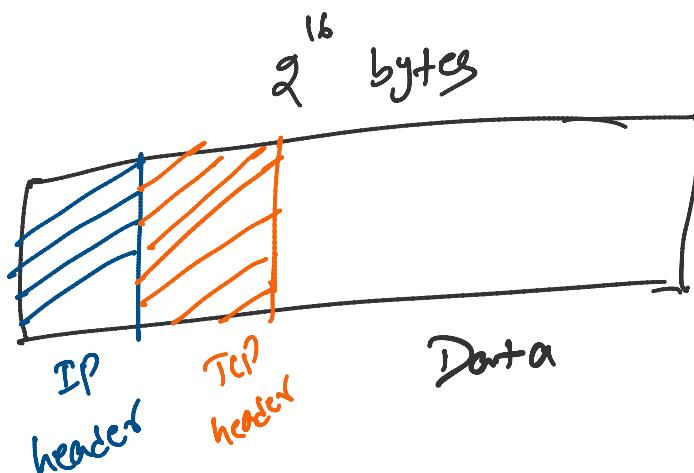
IPv4 / IPv6

② Data

TCP (Transmission Control Protocol)

↳ Built on top of Internet Protocol

→ sends IP packets in order, reliable, error free way



Hyper Text Transfer Protocol

HTTP (Hyper Text Transfer Protocol)

↳ Built on top of TCP

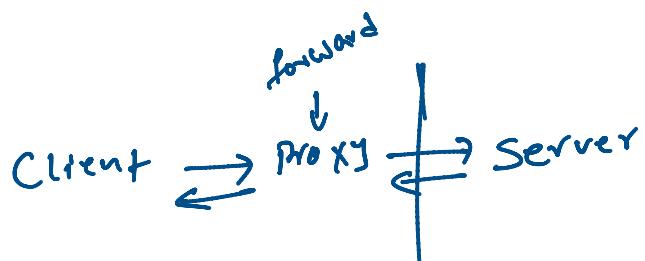
↳ This abstraction is called request-response paradigm.

Proxy (on Behalf)

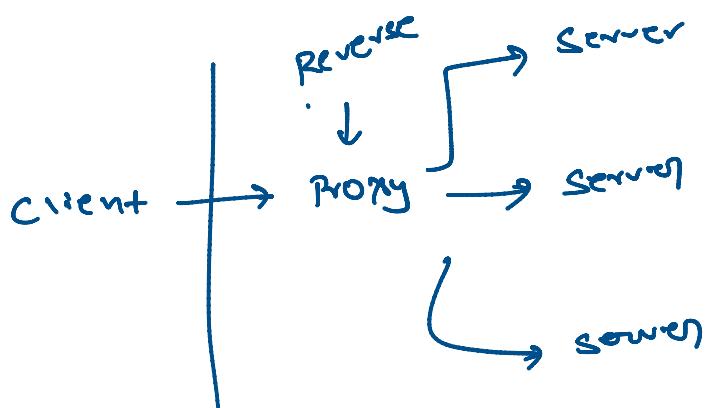
Friday, 31 December 2021 8:07 PM

Forward proxy → Server doesn't know which Client it is connecting from (VPN)

Reverse proxy → Client doesn't know which Server it is connecting to (balancing)



Forward proxy :- proxy sits b/w Client & Server and
disguises client IP address, block malicious
traffic from clients etc.



Reverse proxy :- It's basically a server side proxy.
↳ can be used for traffic control, Load Balancing,
caching response from servers etc...



CAP Theorem

Sunday, 2 January 2022 12:04 AM

- ↳ consistency
- ↳ Availability
- ↳ Partition Tolerance

Any N/W Shared System (distributed system) cannot have all 3 properties (nearly impossible). You need to forfeit one of them

Consistency :- A read happening after a write should always return the latest value of the recently done write operation.

Availability :- Every available node in system should respond in a non-error format to any read request, without the guarantee of returning the latest write.

Partition Tolerance :- System will be responding to all read & write requests even if the communication channel (or middleware) b/w nodes is broken (or partitioned)

→ Partition Tolerance mostly happens through N/W failures. In a distributed system, this is almost unavoidable. (so, we cannot give VP on this property)

→ we can design a system with either

① Partition Tolerance + Availability (AP)

(or)

② Partition Tolerance + Consistency (CP)

Degrees of Consistency & Availability

→ So we know that we cannot provide 100% consistency and 100% availability with partition tolerance in practice.

→ But we CAN provide some extent of consistency and availability!
 (or)
 degree

↳ This absolutely depends on usecase if we can achieve it or not!

e.g.) In ATM machine, no way we can be inconsistent, it always has 100% consistent atleast. (or) Booking Tickets System.

"In ATM machine, no way we can do --
have to be 100% consistent alright. (or) Booking Tickets System.

2) Let's say, this time we're talking about No. of views
for a youtube video. Here, you can forfeit the 100%
consistency (The nodes will eventually reconcile & update
anyways).

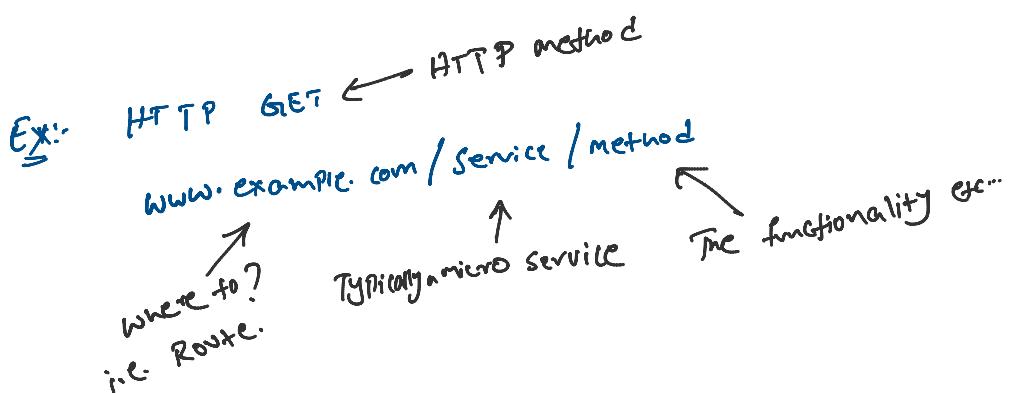
API (It's only a contract!!!)

Tuesday, 28 December 2021 7:44 PM

- ↳ A documented way in which external consumers know how to interact with your code (they don't know how CCP works, they only know how to interact with it).

Things to Remember when Designing API

- ① Naming is important
- ② Don't take additional parameters unless absolutely necessary. (to optimise the call.)
- ③ Don't give more info. than caller needs
- ④ Error handling → Response codes (HTTP) → Respond with correct HTTP codes.
(Don't overdo it)
Response messages → Don't give out sensitive info here!!



Content Delivery N/W

Wednesday, 29 December 2021

8:47 PM

↳ example Akamai, S3

CDN does: (Primarily focuses on nearly fast data delivery by having multiple data centers geographically distributed !!)

- ① Host the data close to users
- ② Follow Regulations
- ③ Provides some interface to update/remove? data from the data centers?
↳ The CDN basically?
- ④ can configure how much time to store some data in this CDN.

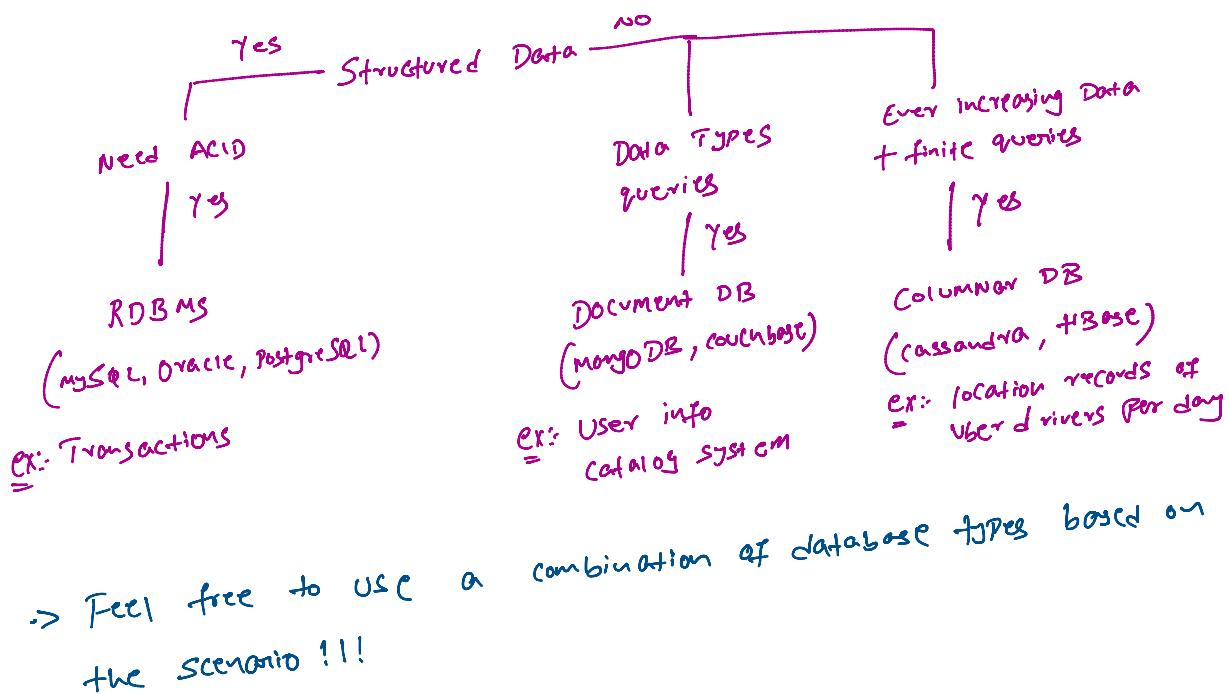
Cloudfront

- ↳ super cheap
- ↳ reliable & easy to use

SQL vs NoSQL

Monday, 24 January 2022 1:35 AM

- ① Always think of caching to reduce latency.
 - ↳ the server API can cache
 - ↳ Redis, Memcached (key value stores) → non relational DBs
- ② Storing Images, Videos? These are blob storages & not queried upon.
 - ↳ So use Amazon S3 to store
 - ↳ use a CDN in front of S3 to distribute same data geographically! (faster way)
- ③ Text searching feature (ex: Amazon Product etc Search / Netflix show title search etc..)
 - ↳ Elasticsearch / Solr



① Relational Databases

- ↳ tables & rows with a fixed schema
- ↳ ACID properties
- ↳ use case → Entirely structured data / need acid properties
 - ex: Employee Table, Transactions etc..
- ↳ vertical scaling is easy (add more RAM, CPU etc.)
- ↳ horizontal scaling is tough here (i.e. sharding)

... databases (Horizontal Partitioning inbuilt mostly)
... map.

↳ Built for scale

- ② Non relational Databases (Horizontal Partitioning inbuilt mostly)
- ↳ Key-value store (Redis, Memcached)
 - ↳ fast & provide quick access & they are in memory
 - ↳ used in caching solutions
 - ↳ Document Based (MongoDB)
 - ↳ used when not sure about structure of data & would evolve over time (i.e. No fixed schema, and can be changed easily)
 - ↳ supports heavy reads & writes, provides sharding capabilities
 - ↳ They have collections (like tables) and documents (like rows)
 - ↳ usecase → Product details of Amazon (since we want all data of one product at one place, no joins etc..)
 - ↳ Cons
 - ↳ NO ACID properties though we can handle it through application code, main loss is strong consistency.
 - ↳ Joins are tough to achieve.
 - ↳ Column DB (Cassandra, HBase, Sybase)
 - ↳ sort of mid way of relational & document DBs
 - ↳ Doesn't support ACID
 - ↳ usecase → streaming data / event data for analytics
 - ↳ Distributed databases in general
 - ↳ Search Databases (ElasticSearch, Solr)
 - ↳ Full text Search
 - ↳ Fuzzy Search
 - ↳ Keep in mind, this is NOT PRIMARY DATABASE!

For Reference

Tuesday, 28 December 2021 9:46 PM

The screenshot shows a Google search result for "Numbers Everyone Should Know". The results list various system performance metrics with their corresponding times in nanoseconds (ns). A handwritten note "a day." is written above the first result, and a large arrow points from the note to the "Main memory reference" row. Another handwritten note "# don't →" is written next to the arrow. At the bottom of the screenshot, there is a red annotation with a bracket and an arrow pointing to the value "400 MB/sec".

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
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

How to Avoid Cascading failures

Friday, 31 December 2021 1:47 PM

Problems

- ① Cascading failures → Rate limiting
- ② Going viral → Pre scale
- ③ Predictable load increase → Auto scale
- ④ Bulk job scheduling → Batch processing
- ⑤ Popular posts → Approximate statistics.

Database Sharding

Sunday, 26 December 2021 3:59 PM

Problem :- How to query huge data in DB?

① Optimise queries (but lot of data, even with optimised queries, it's not good enough)

② Index on table (fine, but get lots of data!!!)

③ Sharding

→ Horizontal Partitioning

Takes a key and partitions the data to different servers
(this key is generally an attribute of data we're storing)

→ Practice of separating one table's rows into multiple different tables, known as Partitions → Sharding

Database Servers!!

Problems

① Joins across multiple shards.

② Fixed no. of shards

↳ USE Hierarchical Sharding

i.e. Let's say even after sharding the data in one shard is huge.

. . . in this shard we can again . . . in multiple

one shard :-
Now, in this shard we can again
shard / Partition the big shard into multiple
shards.

② What happens if a shard goes down

↳ use master/slave architecture

↳ master is the most updated one &
slaves keep reading from master for
updates

↳ All writes go to master, reads are
distributed to slaves

↳ If master fails, one slave becomes
master.

Distributed Caching

Tuesday, 28 December 2021

6:22 PM

- Querying DB for commonly used data?
↳ use cache! (In memory cache)

Helps in saving few calls

- Avoid computations again & again
↳ instead of computing it every time from querying DB, query once & store in cache.
- This also avoids load on DB.

Caches generally run on SSD (Solid State Drive)
which are expensive!!!

- ① When to load data into cache
- ② When to evict/remove data from cache.

} Cache policy!

- Cache policy determines how cache performance is.
↳ ① LRU (Least Recently Used) → most used!!!
② LFU (Least Frequently Used) → not much used

Thrashing

↳ constantly loading & removing data from cache without ever using the results!

Problem

→ How can data be consistent in caches?
i.e. server 2 updates info in DB, which is
cached in server 1 cache. Now what ???

① Write Back Cache

↳ when updating data, hit cache (if this data exists,
then update it here in cache, then update DB)

This method will have
some data inconsistency
if multiple clients write
to same

→ This update can be delayed if
data updated in cache is not of high
importance.

→ Then all updates on cache can
at once be sent as a batch to
DB for update.

② Write Through Cache

↳ First hit the DB & then update in cache. (Synchronous)

↳ Reliable & better (But slow as update needs to
happen in both places)

Load Balancing

↳ let's say we have "n" servers.
 and we need to balance load properly across
 those "n" servers. i.e. Load Balancing.
 → **consistent Hashing** let's us do load balancing efficiently.

Example: $h(\text{request}) / \text{no. of servers}$

↓

hash function is uniformly random
 i.e. takes request id & gives out some
 random no./string/something

AND every server will have uniform load.

* Requests $\rightarrow \text{load} = k/n \rightarrow \text{load factor} = 1/n$

↓
 on each server

(General Old fashioned hashing) doesn't work.

Problem?

After adding new servers (Scalability),
 the requests are routed again

- i.e. In real world, a request id is generally uniform. And when we hash this, we always get same output.
- If this req is previously sent to server "A" because of addition of new servers, same req would go to some other server "B". (And this is a problem with simple hashing)
- Why? → Bcz, there could be cached data we want to reuse.
 And we want to send a same req. with same id to same server.

Solution (use Consistent Hashing)

Total M
 ... in $[0, M-1]$ range

Solution (use Consistent Hashing)

① Let's say our hash function outputs in $[0, M-1]$ range

② Servers themselves have IDs also, we hash these ID's also and take remainder with M.

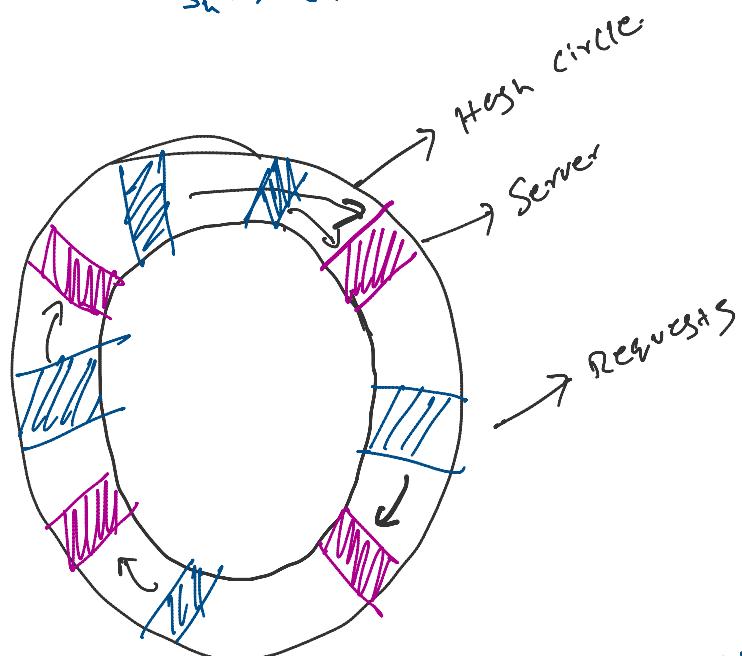
$$\text{i.e. } S_0 \rightarrow h(0) \% M$$

$$S_1 \rightarrow h(1) \% M$$

$$S_2 \rightarrow h(2) \% M$$

$$S_3 \rightarrow h(3) \% M$$

$$S_n \rightarrow h(n) \% M$$



⇒ The request is served by the immediate server on the clockwise direction.

⇒ If a new server is added, it goes into the hash ring.

⇒ If a server goes down, all requests that would go to this server will now be served by the next server on clockwise direction.

Problem

⇒ If there are 5 requests that have Server "A" on clockwise direction. If this server "A" goes down, then what happens?

- If the 5 requests go to next server on clockwise direction. If this server "A" goes down, then all these 5 requests go to next server on clockwise direction, lets say server "B". Due to this server "B" load increased a lot!!

SOLUTION:-

use multiple hash functions for servers

$$\text{I.C. } \begin{aligned} s_0 &\rightarrow h_1(0) & h_2(0) \rightarrow \dots \\ s_1 &\rightarrow h_1(1) & h_2(0) \rightarrow \dots \end{aligned}$$

- If we have "K" hash functions, we can have K points for each server.
- Likelihood of one server getting lot of load is now very less.
- If a server is removed, then remove the K points of that server.

Message Queue

Sunday, 26 December 2021 3:34 PM

- Takes tasks, persists the tasks, assign tasks to right servers,
waits for task completion, if taking long time/server dead
assigns to another server

All of these are done by a
"Message Queue"

Ex:- RabbitMQ, JMS (Java messaging service)

Horizontal vs Vertical Scaling

Saturday, 25 December 2021 2:46 PM

→ Host services on cloud (AWS/GCP/Azure)



Can be your product/service/anything.

Problem → How to handle more requests
i.e. Scalability

① Buy bigger machine → **vertical scaling**

② Buy more machines → **horizontal scaling**

Trade-offs

Horizontal	vertical
① Need load balancing	① N/A
② Resilient	② Single point failure
③ n/w calls (RPC) between servers	③ Inter Process communication
④ Data consistency issue	④ Consistent
⑤ Scales well	⑤ Hardware limit

Solution

Horizontal scaling but each server is powerful enough