Architecture and System Design

Hari Sridhar

June 27, 2021

Contents

Abstract

Document serves as a collection of notes compiled on system design and system paradigms.

Keywords : System Design, Architecture, Interview, etc.,

1 Systems Design

- Single Points of Failure [SPOFs] are akin to Tarjan's Articulation Points algorithm for graph networks
- Throw money and machines are problems : add replica servers, then replicate that network across regions .
- Utilize already existing distributed, decentralized systems such as DNS
- Throw in a load balancing when dealing with routing to components
- Throw in uniform hashing with load server when dealing with user requests and needing to uniquely identify them
- We do not care if the client fails. We are evil like this!
- Minimize networking hopping and network calls; function calls always faster
- Unique Indices, then cols, then share the database!
- SRP, KISS, and Decoupling - they are all related
- Master-Slave architectures for copying and persistence, especially with databases
- Vertical scaling first : then horizontal scaling
- Peak-preprocessing and overnight cron jobs
- Figure out if our IO operations can be non-blocking, or if must remain blocking.
- Caching is both a memory optimization and a network optimization : avoid RPC or HTTP calls issued over the network.
- Monoliths vs Microservices
- Copying over servers resolves not just replication/SPOF, but also capacity handling - can avoid overloaded capacity.
- Hash (userID, request IDs)
- Break down information : images, files, messages, text - each takes up different amounts of data size to upload/download over networks
- Speed of operations, in decreasing order : read -¿ process -¿ write
- Master-slave : write to Master, have slaves read
- Computer memory processes in terms of seconds or nanoseconds -¿ establish as they baseline

- Message queues optimized by not just maintaining a list of orders -¿ but also their priorities
- Distinguish LDD from HLD [Low-Level Design versus High-Level Design]
- Utilize heartbeat mechanisms in load balancers : poll to check alive status of servers from the Heart Service. Also expand heartbeat mechanism via a 2-way street.
- Service Discovery problem : persist in snapshot to obtain (IP, Port Numbers).
- Be wary of cron jobs causing a backlog of network requests and network responses.
- Exert caution in event of index updating - ¿ may need to rehash and reindex entirety of database tables
- Resolutions and Imagine-Video quality are key : support is across multiple devices and multiple connection types!
- FPS - Frames Per Second : Video is just multiple images in the hiding.
- Pizza sharding
- Event logging all systems into one common area : better than grepping IDs across log files!
- Databases for shared communication across servers remains an antipattern
- MQ Queues preferred in large-scale systems!
- No Database can be optimized for both read and writes -¿ choose one only!
- Reads-and-writes, at same time, entails locking!
- Cloud computing boils down to virtualization and load-balancing among servers!
- Capacity Planning = old-school style approach.
- Four parts - memory, storage, disk, processing.
- The point of cloud computing : hey, I'm a small business and I can't but enough hardware, plus I might need to scale. Lets use a large-cap companies boatload amount of hardware and data centers - e.g. AMZ/Google - to horizontall scale instead! instead!
- Cloud computing is rent : small-busineses avoid high down payments / initial investment costs or maintenance costs for their equipment!

- Java makes us architecture agnosting : VMs and containers makes us OS and hardware agnostic.

- Virtualization = resource management and hardware circumnavigation.

- Containers = lighter weight VMs : just specify your requirements, without even needing the OS

- Publisher-Subscriber model : Used by twitter/Instagram, and remains basis for event-driven architectures.

- Gateways are intermediaries - they "massage" incoming messages or requests, BUT, do not do much to the actually requests themselves! Rest of service architecture does this.

- Gateways needed when interacting with multiple components (e..g logging facilities, transaction processing, invoice facilities - waiti this is TGW here!)

- Message brokers decoupling responsibilities and help with replay and the conitnuous persistency of messages across systems. Also highly scalable - can easily add servers.

- Beware non-atomic transactions : transactions across services which affect state can break your system badly!

- Financial transactions with deductions can break publisher-subscriber model.

- Idempotency rule : operation applicable multiple times without changing state each time.

- A messaging chat application can never substitute in for an e-mail or messaging application, as messaging applications naturally lend support not only to hierarchical information storage and retrieval, but also to more means of information retrival and persist ency of messages.

- Can easily organize e-mail in a hierarchical order with folders based on attributes such as subject lines. They lend support to more metadata too.

- True, one can search messages in messenger via timestamps or attributes, BUT, they are not as tag gable with as many options as e-mails (say, 20) . For example, FB messenger application supports at maximum 6 attributes per message, and they are emojis! They are not tuples of (color, category) nor support optional encryption / decryption too per message.

- E-mail message systems maintain their place in the ecosystem of applications and a messaging chat application can never supplant them; only accompany them.

- Utilize batch processing to reduce network calls.

- When in doubt, aim for hybrid solutions.
- Database disks will be slower than cache line hits! OH - not optimized like most machines anyways.
- Types of IO : (Cache,Memory)-Disk-Database. DBs entail network calls / RPC.
- Redis is too a global distributed cache as AWS is too the servers.
- Networked systems which read-write data across do entail difficulties with data consistency (e.g. caches - database write-through write-back).
- When performing updates across networks : with (n-1) connections and a probability of failing, lets hope that updates go through as expected.
- In many system design, interviewers want to see your database schema, ERs, and means of optimizing database queries.
- Any in-memory component (e.g. caching) remains constrained by memory size limits; out-of-memory components are not as constrained, but entail network calls.
- Reduce number of instantiated objects ; less memory and less objects to debug.
- Buffers ensure more safety with IO operations.
- Minimize information sent in HTTP (request, response) payloads, to reduce network delays from (downloading, uploading) bytes of information.
- APIs = the public methods of exposed libraries (on each server-side application too)
- Invalid client inputs and do take responsibility on server-side too.
- A HTTP /GET, with parameter passing, can be more efficient than a HTTP /POST, with a response payload attached! Less work over the computer network for sure! Also more cacheable.
- Browsers = the app which supports HTTP endpoints (APIs hosted on that !)
- Handle large requests - pagination or fragmented APIs
- Conversions between HTTP/public network protocols versus Internal Protocols on private networks.
- Tradeoff : Persistency versus a cache
- How to handle loads? Use service degradation?

- When working with user profiles [uname/pwd], throw in an Authentication server, and an authentication server cache. You must make the network call and process that shit before doing anything else in your application or intranet!
- Worst case : capitulate to the thundering herd of user requests and throw a *404 File Not Found* Error
- Check if you can or can not afford the network calls.
- When in doubt, prefer general deployment to parallel deployment.
- Cascading Failure problems engenders many solutions.
- Handling the tiny deltas to system-wide updates
- We can always fudge around the metadata : not as much accuracy needed there!
- Analyze the RPS - Request Per Second - metric.
- Minimize number of I/O calls and use batch processing to maximize bandwidth.
- For each request, ACK the response
- When in doubt, slow uploads/writes are better for users than slow downloads/reads.
- Underlying their operations, databases execute mergesort operations for chunks of records (sorted by a unique index)
- Sorted string table and compaction expedites database read-writes
- Image storage tends to be in BLOBs (Binary Large Objects)
- Securing filesystems wih ACLs is easier than securing a database!
- Focus on feature development, and start in this sequence : (front end -¿ back end servers -¿ databases).
- Main purpose of gateway : 1. Interact with clients 2. Authenticate clients requests each time (talk to profile processes) 3. Route to other servers!
- Gateway is a decoupler for both services rendered and for network protocols used.
- Client-Server Protocols [e.g. HTTP] do not support chat applications. You need P2P Protocols [e.g. XMPP] or utilize message queues to push/pull information.
- Gateways are useful when interacting with multiple clients. Not needed as much with one client though!.

- Keep session information and use TCP to maintain connections amongst users.
- utilize tokens for safer Authentication
- Load balancing (consistent hashing) needs only UID information to perform hashing!
- Understand which services need to interact with one another, and which do not!
- Can reply to a comment (but ask if chained-replies are supported)
- When working with records, incorporate both timestamp and unique IDs. Timestamps assist with chronological sorting and debugging too.
- Sometimes, starting out with the ER [Entity-Relationship] diagrams and SQL schemas helps further narrow down the system design.
- System design also encompasses database tables designs / ER diagrams! If you need to grab "X" information based on "Y" field (e.g. comments for a post, likes for a post), then set up separate tables for each (comments/likes table)
- Bring in multiple servers for horizontal scalability : incorporate load balancer, and then snapshots [SS] on Gateway side, for efficient routing.
- Use a load balancer with SS [SnapShot] technique when horizontally-scaling server-side. Utilize SS to perform routing in memory on Gateway application, instead of performing a network call each time to the load balancer.
- Load balancer maintains state, and continuouslly polls and updates snapshots.
- Notable concept : maintain snapshots/images in a main application, and have other network components, which maintain the state accurately, continuously update those snapshots/images every 5-10 seconds or so!
- Joining will never scale (happened at Intel during ETL database aggregation !)
- Desire stateless servers, to avoid storing in the memory or the task of a server, in event of an unexpected crash. Can quickly bring back up processes on servers without worry. Hence, the preference for external disks arises to maintain state.
- Server-side notification is more efficient than client-side polling.
- Use rate limiting/throttling, via batch processing, to prevent server-side crashes in the event of a sudden surge!

- Push-pull model on notifications : push -¿ traffic surge : pull -¿ normal conditions . Push is more seamless and real-time.

- User Feed Services will work off of fixed-size lists/queues (queues - possibly with priority - preferred for real time cases)

- Internal protocols used : faster than external protocls (e.g. HTTP - no need for security or big header fields)

- Two types of copy - shallow copy vs. deep copy : synchronous copy vs. asynchronous copy.

- Network communication can fail in a distributed system (e.g. a router or a modem goes bonk). Can prevent consistency in transactions for databases.

- Can rollback a transaction before committing a transaction.

- Distributed consensus : a means for multiple nodes to commit transactions and agree on the same state.

- When to go for the Master-Slave Architecture? Firstly, when a replica of the DB is available. Secondly, when read operations can be scaled!

- Analytics requires real-time data polling. Is its limitation!

- Distinguish transactions which need to show up in real-time versus those which can be committed later.

- Asynchronous communication remains better for Master-Slave architectures, but synchronous communication remains better for peer-to-peer architectures.

1.1 Quad Trees; Delivery; Range Queries

- Uniformity in distribution of points
- Scalable granularity
- Proximity (measure of closeness : like measures in R2-metric spaces)
- Measure distance (Euclidean)
- 2-digit binary representtion of quadrants : fails case of close proximity but massive quarant differs =¿ big mismatch
- Looking for proximity in a general sense : not exactness!
- Continuity of line -¿ infinite partitioning! Partioning/the recursive functions will never end.
- Focus on building the quadrant tree to represent points on a 2D grid : then on the Hilbert curves to snake and map the 2D-grid into the 1D-line.
- Decimals/digits towards LHS entail that scalable granularity.

1.2 SQL Queries

- Practice 1-many and many-1 relationships
- Practice joins and database normalizations again!

1.3 The Key Abstractions

- Gateways
- Messaging Queues
- Load Balancers
- Sharding and Consistent Hashmaps
- Network routers/switches.
- PAAS-SAAS-IAAS
- Caches and their cache lines
- RAID External Disks

1.4 A few Notes on Abstractions

- PAAS = Platform-as-a-Service (e.g. Facebook). It aggregates all that SAAS stuff into one centralized, consolidated location to make things easy for the user. Without a platform, users would have a difficult time navigating across each provided service, and additionally, would have to set up independent connection with each server (versus having a gateway !)

1.5 Gateways

- Condense and optimize on network requests.
- Also known as a reverse proxy : proxies tend to pair up to gateway applications.
- Gateways server as a de-coupler, help to separate public facing client-side networks from server-side private networks, and bridge different network protocols.
- Gateways help to handle public-facing external protocols and external security mechanisms (this is why VPNs exist : they are the lazy solution to security across networks).
- Easy to scale up connections (and their connection pools) in Gateway. They streamline connections from client to internal networks.

1.6 General Tips and Tricks

- Whatever first features you mention, will remain the direction your interviewer will take you upon!
- Define network protocols and services as correctly as possible.

1.7 Optimizing thy Network and Protocols

- Use internal (proprietary) protocols. Less header information and fewer security mechanisms.
- TCP is needed to persist the connections over the network! (esp. via WebSockets)
- The key is this : CSA is one direction (req, resp). P2P is bi-directional (req, resp) patterns.
- HTTP is limited by being a client-server protocol only. Requests can go client-¿server and responses can go only server-¿client. Not the other way around!
- Seperate out servers/applications to conserve thy memory footprint.
- Utilize a parser/unparser service : decouple from GW, if too many users are connected to GW!
- Decouple responsibilities from Gateway to reduce memory footprint.

1.8 The Message Queues

- Mesasge queues not only ensure deliverability of messages, but they also help with scalability
- Easy to configure them - retry intervals and message delay intervals.

1.9 Chat Applications

- Group messaging will be the toughest problem here
- Peer-to-Peer communication needed too!

1.10 Further Questions

- How would protocols be handled if their range closes up (e.g. no more HTTP ports or SSH ports available)? Re-issue network requests and network responses again?
- How to handle the scalability of multiple connections too?
- Notice how Images, E-mails, SMS, and Profiles are all designated as seperate services hosted on seperate servers! Each be their own application.

1.11 Caching

-