Weaknesses and Bottlenecks Noticed while solving Interview Problems and Coding Problems

Hari Sridhar

June 27, 2021

Contents

Abstract

This document stores some things that I have encountered, while solving problems on Leetcode. I hope to address these bottlenecks better, and improve performance, over time too.

Keywords : Leetcode, Interview, Bottleneck, etc.,

1 Observed weaknesses

- If you are ever interviewing, JOT down what you know and do not know. Iterate on that!
- Build System Knowledge
- Linking libraries. How to
- Compiler Errors Handling
- Is allocated memory shared or not?
- Prototyping
- Dynamic versus static libraries. Pros versus cons.
- Templating
- API development.
- HOZ vs VERT scalability. Amazon is the former. SWIFT is the later.
- Everyone wants to know if you've done TESTING. Not just unit testing too
- OOP [C++] is not OOP [Java] either
- Failure to read problems descriptions fully.
- Problem reduction : can we reduce a problem given, to problems previously solved before?

2 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.

2.1 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.

2.2 SQL Queries

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

2.3 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

2.4 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 !)

2.5 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.

2.6 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.

2.7 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.

2.8 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.

2.9 Chat Applications

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

2.10 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.

2.11 Caching

-

3 Big Data - Things to Remember

Build System - Maven - builds the *.jar* files, *pom.xml* files store configurations. Computations were performed by uploading to Amazon Web Services [AWS] S3, and creating EMR (Elastic Map Reduce) clusters

Optimizations around the chain

- Functionality between map and reduce stages : SHUFFLE, SORT, COPY, MERGE

4 Data Formats

RDD is a data representation format in SPARK.

5 Big Data Patterns and MetaPatterns

- Count [e.g. word counts in a book]
- Reverse Index

- Filtering
- Summations
- Joins
- Job chaining and merging - metapattern

6 Qualities of a Good/Strong Leader

Engagement : do company bosses and business leaders communicate and work with employees effectively (or do they take a hands-off approach, at times)?
Resolution - How effectively do leaders engage in compromise?? Can they negotiate? Can they manage, not just their teams, but also across teams, especially given different goals across members?
Understanding - EFfectively leaders allow for their team members to engage in failures, and reckognize if they face any challenges or not too.

7 BASIC PROBABILIY RULES - SIG ASSESSMENT.

Review basic probability principles - bayes rule - conditional probabilities - exponents - event spaces

8 OS Locks, Semaphores

Remmeber how they actually work again.

9 Data Structures

9.1 Trees

- Even a well-chosen hash function will fail you if thy keys are not randomly well-distributed!
- Utilize sort key in NoSQL

Remember Time Complexity of recursive approaches again. E.g. the BFS Flood-Fill
Iteration via a queue is faster than recursion. Handle stack overflow with recursion.
1. Binary Search Trees (BSTs) are aptly named so because the tree structure, due to its SORTED nature, enables binary search performance for operations - deletion/insertion/search. Even though arrays can support binary search, insertion/deletion is O(n) performance, and for LL, search is O(n) even though insertion/deletion is O(1). Trees combine the best of both worlds here.

10 Coding Hacks

1. Use formatted, tab-delimited printing E.g. System.out.printf(”x = [%p] y = [%d]”, x, y);

11 Method Overloading vs Overriding

Overriding [across classes]. Names and parameters [signature] stay the same. Implementation differs. Stands to reason this is run-time polymorphism (usually the polymorphism/polymorphic behavior of our interest!).
Overloading [within class]. The method names same; the signatures/signature lists are not. Remember that this is how polymorphism is done [these two means!]. It stands to reason that this really just is compile-time polymorphism.
We also deserve to know this by the better terminology *static dispatch* and *dynamic dispatch.* This informs us if behavior is decided at compile-time, or run-time.

12 Graph Problems

Make sure to study other data structures in depth before approach graphs, as graphs utilize said data structures. Review cycle-detection algorithms. Common ünique except for 1p̈roblems tend to bias towards this algorithm class.

13 Polymorphism versus Inheritance

IT's less "one object, multiple forms", and more so, "one function, multiple forms", really! - hence, the importance of overloading and overwriting [function invocation/dispatch]. Without these two, it [polymorphism] is non-existent in the language.
I guess inheritance does not imply polymorphism naturally [a.k.a. a language, where you can have a pet, dog, and superdog class, but can't instantiate an object reference to superdog, using pet or dog, but just superdog]. And inheritance can be done if you lack ANY functions [imagine your classes lacked ANY methods!].
It is inheritance, though, which extends to code reusability/redundancy elimination. Polymorphism does not do this! Polymorphism TECHNICALLY [not by itself] allows you to treat objects, of different types, in a similar manner.

14 Behavioral Interviews - Thing Noticed

The interviewers do not expect candidates to possess all skills.
It is fine, to admit that you lack experience in specific domains, or admit, that you know things in theory [versus practice], or have heard of things, but are not too experienced in them.

15 Relational Database Theory

- Databases utilize hashmaps underneath!
- Views - The table generated, after your query.
- Triggers- if database changes, you set an event and update things.
- Normalization - reduce elimination, joining, redundancy reduction.
- Just remember : you trade-off normalization for redundancy (in SQL).
- Remember 1-many and many-1 relationships in SQL. The Select Query language is limited in datatype support : it can not store a list in a column (e.g. Integers or String). Can serialize/deserialize as strings, but this requires computation for sure!
- Hashmap optimizations : To avoid weird amortized time complexity on SLL buckets, try either BSTs, or self-balancing RBTs (Red-Black Trees). Set up thresholds too!

16 SDLC - Software Development Life Cycle

Remember the 7 stages here.
Waterfall versus Agile. Agile allows revisiting the 7 stages. Waterfall does not,

17 OOP Principles

Inheritance, Polymorphism, Encapsulation, Abstraction Encapsulation = Abstraction = Interfaces - list of methods to be overriden by classes extended it - multiple inheritance supported. Common use cases - Iterable, Comparable. Abstract classes - just like a class, but, methods in it declared abstract. Only one abstract class is inheritable from, and methods must be overridden too! Common use case - data structures in OOP language such as List, Set, Maps.

18 Java - Tidbits to Know For Later

Java Memory Mangament, Runtime with JVM, Garbage Collection, Multi-Threatding and Threading Abstractions, thread provisioning and safety.

19 Downcasting versus Upcasting

If you are going to down cast, or upcast, you should take note of this in your codebase as you type things out. This will help clarify things too.

20 More Coding Things to Note Down

[1] SLL access - you can't access LinkedLists like Arrays dude. - always code method parameters with most generic data type. In place of arraylist, use list instead. in place of tree set, use set instead. [2] Set orders - treesets, are technically visited. hashsets, are not. can use to advantage again, in the future. [3] Misunderstood coding level experience - what I have, does not match, what the company position actually desires.

21 More Internet Questions

[1] how to ARCHITECT specific parts of a website [2] HOW to handle traffic granularity requests

22 Exception Handling

Akin to signal handling.
Focus is on your user-space, programm errors though.
Used when you expect errors to occur in specific code segments. Delineate as *protected* code.
Aim to incorporate now, when programming. Some cases I can think of now

- Range exceptions
- Division exceptions
- Allocation
- Invalid args
- Overflow [when implementing middle or summing two large values].

23 Catching Static Versus Non-Static Errors

This has been an insane weakness, in the past.
The good assumption, is that non-static and static things belong in seperate worlds/scopes, and that static allows you to operate in a world WITHOUT objects [think as if back to C-programming, method-function flows].

Remember this

- Static variables for setting properties common to all objects.
- Static methods to update and operate on static variables, OR, operate in a class without needing an instance [main methods make sense - exclusive of classes].

And ignore the following too :

- Static blocks. It's a computational thing.
- Static nested classes.

24 Memory and Storage Issues

24.1 Memory Limit Exceeded

MLE - Memory Limit Exceeded. Reasons this can occur include the following

- Accessing memory that you cannot [kernel-user land issues]
- Literally memory unavailable
- Your memory is so fragmented. Good luck finding space.

Usually, the website specifies it [e.g. could be 256 MB].

25 Recognizing when to classify a problem as *Ad-Hoc*

Sometimes, one will end up with problems that are actually *ad-hoc* in their nature. That is, the problem solving strategy, used to solve the problem, does not fall any under existing problem solving strategies.

26 Presentation Error [PE]

I also end up running into *Presentation Error* issues often too. Usually, this arises from not displaying the output data in the correct, proper format, that it deserves, to have been shown in.

27 Failing to read the C++ Documents and Reference Manuals Carefully

One of your friends did not have this issue, but you did. You like to peruse the documents, stack overflow, and existing solutions quickly, substitute in, and then go solve your problem at hand. In addition, you have ignored actually understand how the documents are actually formed here. And you like to just *plug-in-chug* the example code provided, rather than actually reading up the *programming-language* specs. If you get more used to reading the *programming-language* specs, then you will have less of a need, to actually read the example codes posted. Try to develop, a sort of intuition, to the construction of code, from a language, based on the specs. Do some exercises here too. It can also go a long way.
You will also learn a bit about *templating*, *generic-programming*, understand how some of *PL* material matters here, and learn to stop always needing to reference these documents each and every time too.

28 Datatype issues

I keep running into this stupid issue. Things noticed

- Ask if datatype if signed, or not
- Assess the number of bits, or bytes, in said datatype
- Assert that arithmetic respects said data type [e.g. bSearch example, say, low = 5, high = max value, good luck finding a middle]
- When in doubt, shoot for larger data types
- For any mathematical operation, such as abs, ask if there is an equivalent function, for other data type support too
- Ask if there are conversion methods. But, also ask if there are easy ones too. Casting and converting is annoying. **Also I have no idea what is happening under the hood**.

29 Input-Output Stream characteristics

When assessing input, ask what type of stream you will work with. It is usually the following:

- Standard input/output
- A file. You must be concerned with how to read from files.
- Directly from memory.

30 Files characteristics to remember

There are always a couple of things to remember, when operating on files

- File type. Most are usually *csv*, *txt*
- Files usually end with an EOF character.
- Files usually contain a field seperator. They are easy to access via *awk*.
- Files lines usually end with the newline,
  n character.
- Files consist of blocks, and thus, remember how they are linked to memory too.
- Files may be too larger to actually read into working memory. **is it an issue?**
- Files lines, can vary, in the number of fields compromising them.

31 Quality Control of Test Data

Always remember to perform quality control on your test data of interest. Sometimes, your input data can actually prove invalid. Or your input data needs to be reformatted differently.

32 Case Management

You also need to take a look into case management later. This can also arise.

33 Past Projects, Coding Samples

You need to reprepare these again. Set a *ReadMe* later.

34 Generating useful test data

I've noticed significant struggles, when it comes to actually producing test data, when solving problems. And in executing tests too. I've noticed that I never invest the time to set up a testbench [e.g. I fail to say, write up 4-8 test cases], and somewhat just submit problems, to the application, and hope that it passes and succeeds. And even if it does, I only account for the provided test cases [**Write up yo test bench dude!**].
So, here is a new practice. Aside from easy base cases, write up at least 5 unique test cases. And come up with reasons for recognizing the uniqueness of said test cases [e.g. input order, input size, boundaries, edge cases]. And in addition,

to your inputs, write out the expected outputs too, since you can not expect to always know the proper outputs.

35 Quick Big-O Evaluations

When it comes to MLE or TLE issues, it can prove useful to perform a quick assessment of Big-O here.

36 Really Understanding the *this* Pointer

Turns out *this* is actually easy to understand!
this, is just a *self-referential* pointer. Think of it like that.
It makes it easy to access your objects data, within its own class.

37 Re-Understanding Regex

Turns out the characters are not as complicated as thought out earlier.
ˆ and *$* refer to beginning and end of line (before new-line character) [or starting,ending positions of strings] - just like VIM!
Note those similarities!
Remember - Vim, is a line-based tool!

38 Other Suggestions

- Know about the idea of a balanced binary search tree (e.g., AVL tree or red-black tree), but don't worry about being able to implement one during an interview.
- N element gap, between *cur* and *prev* pointers, sometimes needed.
- At a lower level, if not HM, use a bitset.
- Reverse-mapping technique, of array elements, to indices
- Remember the idea for topological sorting.