# 1 CheatSheet: System Design For Job Interview INTERVIEW

- PDF Link: cheatsheet-systemdesign-A4.pdf, Category: interview
- Blog URL: `https://cheatsheet.dennyzhang.com/cheatsheet-systemdesign-A4`
- Related posts: CheatSheet: Leetcode For Code Interview, CheatSheet: Well-Known Papers For IT Industry, #denny-cheatsheets

File me Issues or star this repo.

## 1.1 Reference

| Name | Summary |
|---|---|
| Papers | CheatSheet: Well-Known Papers For IT Industry, Github: papers-we-love |
| Github | Github: system-design-primer, Github: puncsky/system-design-and-architecture |
| Website | Website: hiredintech - System Design, Website: blog.gainlo.co |
| Website | Website: interviewing.io, Website: interviewbit.com |
| Cheatsheet | CheatSheet: Leetcode For Code Interview, CheatSheet: Common Code Problems & Follow-ups |
| Cheatsheet | CheatSheet: System Design For Job Interview, CheatSheet: SRE/DevOps/Sysadmin |
| Cheatsheet | CheatSheet: Behavior Questions For Coder Interview |
| Coding | Code problems for #oodesign |
| Company Tech Blog | Website: Facebook Engineering, Website: Google Developers |
| Company Tech Blog | Medium: Netflix Blog, Medium: Airbnb Engineering & Data Science |
| Company Tech Blog | Shopify Engineering, Github Engineering |
| Individual Tech Blog | Blog: All Things Distributed - Amazon CTO, Blog: highscalability |
| YouTube | YouTube: Intro to Architecture and Systems Design Interviews, YouTube: System Design Interview |
| YouTube | YouTube Channel: Success in Tech, YouTube: Scalability Harvard Web Development |
| Reference | Link: Facebook Engineering Interview, Link: The System Design Process |

## 1.2 Process Of System Design

| Num | Name | Summary |
|---|---|---|
| 1 | Outline use cases: List major and focus on some | Show good sense. The questions you asked define your level |
| 2 | Estimate scale: **Data** + **Traffic** | Back-of-the-envelope estimation |
| 3 | Defining data model | It helps to clarify how data will flow among different components |
| 4 | Abstract design | Sketch main components, explain workflow, avoid too deep for details |
| 5 | Detailed design + discussion with interviewers | Explain trade-off of your proposal + on-demand deep dive |
| 6 | Identify and resolve Bottlenecks | **Key challenges** + **Trade-Offs**. Usuaully no optimal solution(s) |
| 7 | Scale your design | Availability, Resiliency, Scalability, Security, Serviceability, etc |

## 1.3 Design Problems Per Category

| Num | Name | Summary |
|---|---|---|
| 1 | Design a small scale MIS system | Design: Flight booking service, Design a payment processor |
| 2 | Design an API gateway | |
| 3 | Design a logging & metrics system | Pull vs Push model |
| 4 | Resource/Task scheduling | Design web crawler, Delayed task scheduling |
| 5 | K/V DB store | Design K/V DB; Design memcache/redis |
| 6 | Recommendation system | Design amazon book recommendation system |
| 7 | Data synchronization | Design dropbox client sync |
| 8 | Design a distributed component | Design a distributed counter, Design a distributed UUID generator |
| 9 | Design a SNS system | Design Twitter News Feed |
| 10 | Design a communication system | Design a message chat room |
| 11 | Design a gaming system | Design: Leaderboard |
| 12 | Design an ads system | |
| 13 | Design API Gateway | Design An API Rate Limiter |

## 1.4 Common Mistakes Of System Design

| Num | Name | Summary |
|---|---|---|
| 1 | Run into an opinioned solutions before clarification | Inexperienced; Hard to communicate |
| 2 | Not driving the conversation | Inexperienced |
| 3 | General answers without your personal experience/thinking | |
| 4 | Makes interviewers feeling you're stubborn | |

## 1.5 Algorithms Questions In System Design

| Num | Name | Summary |
|---|---|---|
| 1 | Give three 1TB disks, how to store 2TB data with redundancy | Store as A, B and A XOR B |
| 2 | How to support feature of "diff big1.bin big2.bin" | #lcs - Longest Common Subsequence |
| 3 | How to support "rsync big1.bin ssh:/big2.bin" in a doggy network | delta-transfer algorithm |
| 4 | How to release 1GB binary to 10000 servers | |
| 5 | Avoid double payment in a distributed payment system | link |

## 1.6 Top 20 Design Problems For Technical Modules

| Num | Name | Summary |
|---|---|---|
| 1 | Design a distributed counter | |
| 2 | Delayed task scheduling | |
| 3 | Design a thread-safe Hashmap | |
| 4 | Design An API Rate Limiter | |
| 5 | Design a distributed UUID generator | |
| 6 | Design a distributed Hashmap | |
| 7 | Design data sync for a distributed system | |
| 8 | Design A big file transfer feature | |
| 9 | Top K Frequent Elements in Recent X mins | LFU cache |
| 10 | Design a distributed transactions | |
| 11 | Design: A Parking Lot Service | |
| 12 | Design a distributed transaction | |
| 13 | Design: A URL Redirecting Feature | |
| 14 | Top URL hits | |
| 15 | Unique url hits | |
| 16 | Design RSS news reader | |
| 17 | Design a load balancer | |
| 18 | Design a client-server API to build a rich document editor | |
| 19 | Design online/offline status system | |
| 20 | Design dropbox client sync | |
| 21 | Design a circuit breaker | |
| 22 | Design a service auto-discovery feature | |
| 23 | Design a secrets management system | |
| 24 | Design a online contest system like leetcode.com | |

## 1.7 Top 20 Design Problems For A Product

| Num | Name |
|---|---|
| 1 | Design: TinyURL - A URL Shorterner Service |
| 2 | Design Twitter News Feed |
| 3 | Design K/V DB |
| 4 | Design: Leaderboard |
| 5 | Design: Flight booking service |
| 6 | Design: Uber Backend |
| 7 | Design an API gateway |
| 8 | Design: An Elevator Service |
| 9 | Design web crawler |
| 10 | Design amazon shopping cart |
| 11 | Design: Google Suggestion Service |
| 12 | Design a payment processor |
| 13 | Design google doc |
| 14 | Design gmail |
| 15 | Design instagram, a photo sharing app |
| 16 | Design Yelp, a location-based system |
| 17 | Design Pastebin.com |
| 18 | Design amazon book recommendation system |
| 19 | Google autocomplete |
| 20 | Design Google PageRank |
| 21 | Design messaging/notification system |
| 22 | Design search post system |
| 23 | Design memcache/redis |
| 24 | Design typeahead |
| 25 | Design a voice conference system |
| 26 | Design Google Adsense fraud detection |
| 27 | Design slack |

## 1.8 Top 30 Concepts For Feature/System Design

| Num | Name | Summary |
|---|---|---|
| 1 | Caching | Stores data so that future requests of data retrieval can be faster |
| 2 | Message Queue | Provides an asynchronous communications protocol, |
| 3 | Data Partition & Sharding | Break up a big data volume into many smaller parts |
| 4 | DB Indexing | Create indexes on multiple columns to speed up table look up |
| 5 | DB replication | Duplicate data to increase service availability |
| 6 | CAP: Consistency/Availability/Partition | A distributed database system can only have 2 of the 3 |
| 7 | DB: SQL & NoSQL | Relational databases and non-relational databases |
| 8 | Concurrency & Communication | |
| 9 | Pessimistic And Optimistic Locking | |
| 10 | Consistency Module | weak consistency, eventual consistency, strong consistency |
| 11 | Conflict resolution | Quorum, vector lock, reconcile on read/write, CRDTs |
| 12 | B+ Tree | |
| 13 | Networking: HTTP | |
| 14 | Pull vs Push model | |
| 15 | Garbage Collection | |
| 16 | Memory Management | |
| 17 | Heartbeats | |
| 18 | Self Protection | API Rate limit, Circuit breaker, bulkhead, throttling |
| 19 | Filesystem | |
| 20 | API: gRPC vs REST | |
| 21 | Load balancer | |
| 22 | Scale up vs Scale out | Vertical scaling and Horizontal scaling |
| 23 | API Design | |
| 24 | Session management | |
| 25 | Networking: TCP vs UDP | |
| 26 | Consistency patterns | Weak consistency, Eventual consistency, Strong consistency |
| 27 | Availability patterns | Fail-over vs Replication |
| 28 | CDN - Content Delivery Network | Edge caching |
| 29 | Monitoring | |
| 30 | Security | |
| 31 | Networking: DNS | |
| 32 | Linux signals | |

## 1.9 Top 15 Advanced Data Structure & Algorithms

| Num | Name | Summary |
|---|---|---|
| 1 | Consistent Hash | |
| 3 | Bloom filter | A space-effcient query returns either "possibly in set" or "definitely not' |
| 4 | CRDTs (Conflict-Free Replicated Data Types) | |
| 5 | SSTable (Sorted Strings Table) | |
| 6 | LSM (Log Structured Merge Trees) | |
| 7 | Gossip | Propagate cluster status |
| 8 | Two-phase commit/Three-phase commit | |
| 10 | Vector Clocks/Version Vectors | |
| 11 | Paxos and raft protocol | |
| 12 | Merkle Tree | |

https://raw.githubusercontent.com/dennyzhang/cheatsheet.dennyzhang.com/master/cheatsheet-featuredesign-A4/dynamo-summary.png

## 1.10 Explain workflow: What happens when XXX?

| Num | Name | Summary |
|---|---|---|
| 1 | When happens when I search in google? | |
| 2 | How loadbalancer works | |
| 3 | Explain three phase commit model | |
| 4 | Explain HTTP return code | Link: Series of posts on HTTP status codes e.g, 401 vs 405, 500 vs 503 vs 504 |
| 5 | Explain Mysql DB replication model | |
| 6 | Explain gossip protocol | |
| 7 | Explain CAP | |
| 8 | Explain Hadoop file system | |
| 9 | [Linux] Explain OS booting process | |

## 1.11 Explain tools: how XXX supports XXX?

| Num | Name | Summary |
|---|---|---|
| 1 | How JDK implement hashmap? | |
| 2 | Explain java garbage collection model | |
| 3 | Explain raft/etcd | |
| 4 | How OS supports XXX? | |

## 1.12 Cloud Design Principles

| Num | Name | Summary |
|---|---|---|
| 1 | Fail fast | |
| 2 | Design for failure | |
| 3 | Immutable infrastructure | |
| 4 | Cats vs Cattle | Avoid snowflake servers |
| 5 | Auto healing | |
| 6 | Async programming | |
| 7 | GitOps operational model | |
| 8 | Event-Driven Architectures | |

## 1.13 Cloud Design Patterns

| Num | Name | Summary |
|---|---|---|
| 1 | Ambassador pattern | Create helper service to send network requests, besides the main sevice |
| 2 | Cache-Aside pattern | Load data on demand into a cache from a data store |
| 3 | Circuit Breaker pattern | If a request takes too many reousrce, abort it |
| 4 | Bulkhead pattern | Isolate elements into pools, so that one fire won't burn all |
| 5 | Gateway Aggregation pattern | Aggregate multiple individual requests into a single request |
| 6 | Priority Queue pattern | Support different SLAs for different individual clients |
| 7 | Strangler pattern | Incrementally migrate a legacy system piece by piece |

## 1.14 Engineering Of Well-Known Products

| Name | Summary |
|---|---|
| Google | Link: Google Architecture |
| Facebook | Link: Facebook Live Streams |
| Twitter | Link: Twitter Image Service, YouTube: Timelines at Scale |
| Uber | Link: Lessons Learned From Scaling Uber |
| Tumblr | Link: Tumblr Architecture |
| StackOverflow | Link: Stack Overflow Architecture |

## 1.15 Grow Design Expertise In Daily Work

| Num | Name | Summary |
|---|---|---|
| 1 | Keep the curiosity | Thinking about interesting/weird questions helps |
| 2 | Deep dive into your daily work | Unify and normalize problems from daily work |
| 3 | Learn the work of your coleagues | Indirect working experience also help |
| 4 | Popular products under the hood | Once you notice an interesting feature, think about how it's supported? |
| 5 | Read engineering blogs | Especially for big companies |
| 6 | Tools under the hood | Common tools/frameworks |
| 7 | Try tools | Use cases; Alternatives; Pros and Cons |
| 8 | Read papers | Best practices in papers |
| 9 | Try new things | Gain hands-on experience; evaluate alternatives |
| 10 | Datastore & OS | Learn how databases and operating systems work |
| 11 | Language implementation | Deep dive into one programming language. Java, Python, Golang, etc |

## 1.16 More Resources



- https://github.com/binhnguyennus/awesome-scalability
- https://github.com/donnemartin/system-design-primer
- https://github.com/checkcheckzz/system-design-interview
- https://github.com/binhnguyennus/awesome-scalability
- https://docs.microsoft.com/en-us/azure/architecture/patterns/
- https://github.com/sdmg15/Best-websites-a-programmer-should-visit