Chapter 9 - System Desing and Scalability

// **Handling the questions**

- Communicate
- Go broad first
- Use the whiteboard
- Acknowledge interviewer concerns
- Be careful about assumptions
- State your assumptions explicitly
- Estimate when necessary
- Drive

// **Design** → step by step

- Scope the problem
- Make reasonable assumptions
- Draw the major components
- Identify the key issues
- Redesign for the key issues

// **Algorithms that scale** → step by step

- Ask questions
- Make believe
- Get real
- Solve problems

// **Key concepts**

// Horizontal vs. Vertical Scaling

- vertical → additional resources to one node
- horizontal → increase the number of nodes

// Load Balancer → distributes the load evenly

// Database denormalization adn NoSQL

- denormalization → adding redundant information into a database to spead up reads as JOIN statements are extremely slow

// Database partitioning (sharding)

- split up data on multiple machines while having a way to know which data is on which machine
- vertical partitioning → parition by feature
- key-based (hashed) partitioning → mod(key, n)
- directory-based partitioning → lookup table

// Caching → in memory, key value pairing

// Async Processing and Queues → slow operations

// Networking metrics

- bandwidth
- throughput → actual amount of data being transferred
- latency

// Map Reduce → process large amounts of data

- Map → takes in some data and emits a <key, value> pair
- Reduce → takes a key and a set of associated values → reduces them → emits a new key and value

// **Social Network** → design a social network with the ability to find the shortest path between two people

- design the social network as a graph
- BFS instead of DFS → remove unnecessary computation
- bidirectional BFS → from source and from destination → when they collide we found a path
- BFS will traverse 1 + B + B^2 + ... + B^k vertices
- Bi-directional BFS will traverse 2 + 2B^2 + ... + 2B^(k/2) vertices.

// **Web Crawler** → infinite loop occurs when a cycle occurs

- use a hash table to mark page v as visited
- use BFS
- create a signature for each page based on content and URL

// **Duplicate URLs** → data is too big to fit in memory → split data on disk in multiple files, create hash tables for each and look up in multiple hash tables → or split the data between multiple machines

// **Cache** → make use of linked lists and hash tables

// **Sales Rank** → slice data based on time period to remove bias from data → look into MapReduce

// **Pastebin** → get a randomly generated URL for every text snippet

- URL to file → uniformly distribute the docs → probability of popular files is uniformly distributed across servers
- caching popular files
- check if file already exists to avoid collisions