**Chapter 1**

Scale From Zero to Millions of Users

**RDBMS** (Relational database management system)

* MySQL
* Oracle database
* PostgreSQL

**Non-Relational**

* CouchDB
* Neo4j
* Cassandra
* HBase
* Amazon DynamoDB

**4 Categories**

* Key-value stores
* Graph stores
* Column stores
* Document stores

|  |  |
| --- | --- |
| Relational DB | Non-relational DB |
| * Fixed schema * Been around over 40 years * Supports join operations | * Super-low latency * Good when data is unstructured or non-relational * Only need to serialize and deserialize data (JSON, XML, YAML, etc.) * Can store a massive amount of data |

Vertical scaling – adds more power (CPU, RAM, etc.)

* Has a hard limit (impossible to add unlimited CPU/memory to a single server)
* Does not have failover and redundancy

Horizontal scaling – add more servers

* Generally better scaling approach

Load balancer

* Reverse proxy - a server middleware to protect other servers. Typically implemented to help increase security, performance, and reliability.
* Web servers are no longer reachable directly by clients
* Private IPs used for communication between servers
* Improves server availability – directs traffic to prevent one server from overloading

Database replication

* Master/slave relationship
  + Master db – supports write operations
  + Slave db – copies master and supports read operations
  + Advantages
    - Performance – allows more queries to be processed in parallel
    - Reliability – data is preserved even if one database is destroyed
    - High availability – keeps app operational when one db goes offline
* Other replication methods
  + Multi-masters
  + Circular

Cache – temporarily stores of expensive or frequently accessed data in memory

* Aim to improve response times

Cache tier – temporary data store layer

* Faster than database
* Improves system performance, reduce database workload
* Can be scaled independently
* Read-through cache strategy - checks if cache has the available response before sending queries to database
* Strategies
  + Good when data is read frequently but modified infrequently
  + Needs expiration policies to remove unused data
  + Keep data store and cache in sync. Maintain consistency across multiple regions can be challenging
  + Avoid single points of failure (SPOF) with multiple cache servers
  + Cache eviction policy – handling cache requests when full
    - LRU – least recently used (most popular)
    - LFU – least frequently used
    - FIFO – First in First out

Content delivery network (CDN)

* network of geographically dispersed servers
* delivers static content (images, videos, CSS, JavaScript files, etc.)
* websites will use the closest CDN server since more distance = slower loading
* often run by third-party providers
* Considerations
  + Cost and size of CDN storage – how much do you really need?
  + Set an appropriate cache expiry
  + Fallback – clients should request from the source if there is a CDN outage
  + Invalidating files – removing files before their expiration date

Dynamic content caching – caching of HTML pages that are based on request path, query strings, cookies, and request headers

Stateless web tier – scaling the web tier horizontally

* Move state (e.g. user session data) out of the web tier
* Store session data in persistent storage like relational database or NoSQL

**Stateful architecture**

* Stateful server – remember client data from one request to another
  + Server A contains authentication data for user A
* Stateless system – simpler, more robust, and scalable
  + Shared data store could be:
    - Relational database
    - Memcached/Redis
    - NoSQL
* Autoscaling web tier – adding or removing web servers automatically based on load

**Data centers**

* geoDNS-routing – requests from the user is routed to the nearest data center
* geoDNS – DNS service delivering domain names based on the location of the user
* Challenges
  + Traffic redirection – requires geoDNS routing
  + Data synchronization – local database and caches will replicate data across multiple data centers to provide consistency for failover cases
  + Testing and automation at different locations can be challenging

Message queue – (decoupling) durable component stored in memory, supporting asynchronous communications

* Producers/publishers – (input) create and publish messages to the queue
* Consumers/subscribers – (output) connects to queue and execute messages
* Allows producers and consumers can be scaled independently

**Logging, metrics, automation**

* Logging – monitor errors at per server level or aggregate them to a centralized service for easy search and viewing
* Metrics – data collection to gain business insight and system health
  + Host level metrics – CPU, memory, disk I/O
  + Aggregated level metrics – database tier performance, cache tier
  + Key business metrics – daily active users, retention, revenue
* Automation
  + Continuous integration - each code check in is verified through automation

**Database scaling**

* Vertical scaling
  + Amazon Relational Database Service – one db server has 24TB of RAM
* Horizontal scaling
  + Sharding – adding more servers, separates a large database into “shards”
  + Shards share the same schema but different data0
  + Sharding key – allows data to be retrieved and modified by routing queries to the correct database
  + Challenges
    - Resharding – needed when a shard could no longer hold more data due to rapid growth or shard experiencing exhaustion faster than others due to uneven data distribution
    - Celebrity problem (hotspot key problem) – excessive access to a specific shard causes overload
    - Join and de-normalization – hard to perform join operations across database shards

**Summary**

* Methods to scale systems to increase more users:
  + Keep web tier stateless
  + Build redundancy at every tier
  + Cache data as much as you can
  + Support multiple data centers
  + Host static assets in CDN
  + Scale data tier with sharding
  + Split tiers into individual tiers
  + Monitor the system and use automation tools