System Design: Fundamentals

Author: Ravi Tandon

Blog Link

Introduction

In this document, we discuss the basic building blocks of a distributed system. We also discuss the challenges of designing distributed systems. In the end, we discuss an application of the building blocks in building a typical website selling products.

Elements Of A Distributed Systems

A distributed system is defined as "*An interconnected set of nodes that are linked by a network and share information between them.*" Although a distributed system consists of multiple components, it serves a single purpose to the system's user. Some examples of distributed systems in computer science are the backends of Google, Facebook, LinkedIn, etc. Systems like the Internet, Blockchain or even the human nervous system is distributed system. For the sake of this blog series, we will focus on only those in Computer Systems 😊😊

We will discuss some of the critical elements of a typical distributed system for building user-facing applications next.

Microservices

The microservice architecture is a design paradigm where an application is structured to collect several independent services. The characteristics of these services are as follows:

- Organized around a specific business capability
- Own by a small team
- Independently deployable

- Highly scalable
- Loosely coupled
- Highly maintainable and testable

Load Balancers

A load balancer is a device that acts as a reverse proxy and distributes the network traffic across several different servers. Load balancers are used to scale distributed systems by distributing traffic between other servers. Load balancers are typically grouped into two distinct categories: Layer 4 and Layer 7. The load balancers at layer 4 act upon data found in network and transport layer protocols (IP, TCP, FTP, UDP). Layer 7 load balancers distribute requests based on application data (HTTP). Some of the standard algorithms that load balancers use are:

- Round Robin
- Weighted Round Robin
- Least connections
- Least response time

Databases

A database is a structured system to put your data under a pre-specified format and constraints. Some of the requirements when a database is needed are as follows:

- Scalability: Applications that have billions of rows or terabytes of data use DBMS.
- Security: Applications that require a strict deposit of access to data in an organization use DBMS.
- Consistency: Applications that require the data to remain consistent use DBMS. Some of how the data can become corrupt are as follows:
    - Lack of consistency constraints
    - Unsafe deletes
    - Overwriting

  A typical DBMS provides ACID capabilities to ensure that the data remains consistent.
- Availability: Applications that require the data to remain available throughout the lifetime use DBMS. A typical DBMS provides replication of servers to ensure the availability of data.

Caches

A cache is temporary storage that is relatively smaller in size with faster access time. Caches are used to reduce latency and reduce the load on your servers and databases. There are several levels at which caches are implemented. These are as follows:

A cache is temporary storage that is relatively smaller in size with faster access time. Caches are used to reduce latency and reduce the load on your servers and databases. There are several levels at which caches are implemented. These are as follows:

1. Client Caching: Caches are located on the client-side like browser, file-system, servers acting as reverse-proxy.
2. CDN Caching: CDN (Content Delivery Network) is also used as a cache for serving static content (e.g. HTML, CSS, Javascript, Image, Videos, etc.).
3. Web Server Caching: Webservers also implement local caches and can retrieve, return information without contacting downstream servers.
4. Database Caching: Databases by default include some level of caching so that they do not have to run queries repeatedly. This can boost the performance of databases when they are under a lot of load.
5. Application Caching: In application caching, the cache is placed between application and data stores. These caches usually store database query results and objects that the application uses. Typical application caches include Memcached, Redis, DynamoDB, etc.

One of the challenges with caching is ensuring consistency of the data between the cache and the underlying data layer (i.e. server or database).

File System Storage

A file is an unstructured collection of records. Typically file systems support two basic formats. These are as follows:

- Block Storage: Organize data in blocks on disk. The blocks are then divided into sectors and tracks. Current day personal computers (i.e., laptops and desktops) store data using Block Storage.
- Object Storage: Organize data into containers of flexible sizes, referred to as objects. Modern-day cloud systems use Object Storage for storing their data. E.g., Amazon S3. They are designed to be massively scalable, highly performant, and inexpensive.

The characteristics of a distributed file system (DFS) are as follows:

- Performance: A DFS should provide high throughput and low latency of access. The performance should be similar to what a local file system can provide.

- Availability: Distributed file systems are built across a network. Therefore, they can have several possibilities of failure. The data should remain accessible in case of partial node failures.
- Scalability: A DFS should scale horizontally and support the storage of petabytes of data. The system should scale as more nodes are added to the DFS. The client should not experience any disruption as more nodes are added to the system.
- Reliability: A DFS should be highly reliable and ensure that the probability of data loss is minimized. Typically, a DFS would create backup copies of the data to prevent it from being lost permanently.
- Ease Of Use: A DFS should provide a simple and easy-to-use interface for the client to use. Some of the common operations that clients use, such as reading, writing, copying, etc., should be easy to perform.
- Data Integrity: A DFS should ensure that the data remains consistent and shouldn't be lost if multiple users access it. Typically DFS should provide atomicity and consistency of operations in some form or the other.
- Heterogeneity: A DFS should support the storage of heterogeneous data (structured, unstructured, files, records, etc.). Different types of clients should be able to access the file system and store data on it.

Network

The final and central piece of a distributed system is the network that binds all the nodes together. Typically, clouds such as Amazon's AWS, Microsoft's Azure, etc., provide the infrastructure for networks on which the servers can run. These networks span Wide Area Networks and are spread across geographies in the world. The cloud networks are typically designed to enable the client/server communication model using protocols such as HTTP.

Messaging Queues

Messaging queues make it possible for services to communicate with each other asynchronously. The typical use cases of messaging queues tend to be sending notifications to clients. These notifications can be alerts, emails, messages, etc. The basic architecture of a messaging queue consists of the following:

a) Producer: The service that produces messages and writes them on the messaging queue.
b) Consumer: The service that consumes messages and reads them off the messaging queue.

The essential characteristics of a messaging queue are as follows:

a) Scalable: A messaging queue should be able to scale to millions of messages per hour. It should scale horizontally and preserve the basic guarantees when as more nodes are added to the system.
b) Distributed: A messaging queue should be distributed to scale up and support multiple consumers, producers, and messaging using a cluster of nodes.
c) Reliability: A messaging queue should be reliable and not drop messages. If the error rate is high, i.e., many messages get settled, then the burden falls on the client to perform checks on the data. This makes the message queue inefficient to use.
d) Performance: A message queue should support high throughput and low latency when delivering messages.
e) Easy to use interface: A messaging queue should provide a simple API interface to integrate with the client applications (i.e., producers and consumers) for broader adoption.

References

1. What are microservices?
2. Load Balancer
3. Database Introduction
4. ACID Guarantees Wikipedia
5. What is Caching?
6. What is DFS (Distributed File System)?
7. Distributed Networking (Wikipedia)
8. System Design - Message Queues
9. Kafka: A Distributed Messaging System For Log Processing
10. Building Scalable Distributed Systems: Part 1 — Introduction to Scalable Systems
11. Building Scalable Distributed Systems: Part 2 — Distributed System Architecture Blueprint: A Whirlwind Tour
12. Redis
13. Memcached
14. Consistency Guarantees In Distributed Systems Explained Simply
15. Deep Dive on S3 Consistency
16. A Comprehensive Guide To Distributed Systems
17. System Design Primer
18. CAP Theorem Revisited

The System Design Masterclass

[Website]

[Weekly Newsletter]
