System Design Trade-offs and Guidelines

Today's modern technological revolution is happening because of large-scale software systems. Big enterprise companies like Google, Amazon, Oracle, and SAP have all built large-scale software systems to run their (and their customers') businesses. Building and operating such large-scale software systems requires first-principles thinking to design and develop the technical architecture before actually putting the system into code. This is because we don't want to be in a state where these systems will not work or scale as more users need them.

If the design is right in the first place, the rest of the implementation journey becomes smooth. This requires looking at the business requirements, understanding the needs and objectives of the customer, evaluating different trade-offs, thinking about error handling and edge cases, and contemplating future changes and robustness while worrying about basic details like algorithms and data structures. Enterprises can avoid the mistake of wasted software-development effort by carefully thinking about systems and investing time in understanding bottlenecks, system requirements, users being targeted, user access patterns, and many such decisions, which, in short, is system design.

This chapter covers the basics of system design with the goal of helping you understand the fundamental concepts, the trade-offs that naturally arise in such large-scale software systems, the fallacies to avoid in building such systems, and the guidelines of system design—those lessons that have been learned from building large-scale software systems over the years. This chapter is intended to introduce you to the essentials—we'll dig into the details in later chapters, but we want you to have a good foundation to start. Let's begin with the basic concepts of system design.

System Design Concepts

To understand the building blocks of system design, we should understand the fundamental concepts around systems. We can leverage abstraction here: the concept in computer science of obfuscating the inner details to create a model of these system design concepts, which can help us grasp the bigger picture. The concepts in system design, be it any software system, revolve around communication, consistency, availability, reliability, scalability, system maintainability, and fault tolerance. We will go over each of these concepts in detail, creating a mental model while also exploring how they are applied in large-scale system design.

Communication in System Design

Communication represents the foundational mechanism through which distributed system components exchange information and coordinate their operations. In the context of large-scale software architectures, effective communication patterns are paramount to ensuring seamless inter-service interactions and data consistency across heterogeneous systems.

Synchronous vs. Asynchronous Communication

Communication protocols in distributed systems can be classified into two primary paradigms: synchronous and asynchronous communication. Synchronous communication, exemplified by REST

APIs and gRPC, requires the requesting component to await a response before proceeding, thereby establishing a tight temporal coupling between services. Conversely, asynchronous communication mechanisms, such as message queues and event-driven architectures, decouple temporal dependencies, allowing components to operate independently while ensuring eventual delivery of messages.

Consistencia

Network Reliability and Latency

Communication within distributed systems is inherently subject to network constraints including latency, bandwidth limitations, and potential packet loss. These constraints necessitate careful consideration of timeout mechanisms, retry strategies, and circuit breaker patterns to mitigate communication failures and enhance system resilience.

Consensus and Distributed Coordination

In multi-node systems, achieving consensus on shared state requires sophisticated coordination protocols. Algorithms such as Raft and Paxos address the challenge of maintaining consistency across distributed nodes while tolerating partial failures, thus enabling robust distributed decision-making and state management.

Fonte: System Design on AWS, Jayanth Kumar, Mandeep Singh