System Design Notes

Ryan

July 2024

Contents

Part I

Common Knowledge

Chapter 1

Keywords

DNS; monolithic; tiered architecture; database management system; relational database; vertical scaling; horizontal scaling; failover; redundancy; load balancer; database replication; cache; eviction policy; expiration policy; consistent hashing; cache miss; cache hit; caching strategy; single point of failure; stateful webserver; stateless webserver; CDN; message queue; normalization; de-normalization; sharding; rate limiting; optimistic lock; eventual consistency; transaction log; lease; fan-out; rollback

Part II

Fundamentals and Building Blocks

Chapter 2

Consistent Hashing

Chapter 3

Cache

caching strategy; expiration policy; eviction policy; invalidation

Caches and cache layers are key components in almost every high-performance architecture. The idea is to save results that are expensive to obtain. Examples are prediction calculated by a complex machine learning algorithms or data extracted from a database.

When employing caches, it's important to remember two important characteristics of caches:

- It requires additional efforts to maintain the consistency between caches and the authoritative source of the data.
- Caches are ephemeral. When designing a system that uses caches, we need to assume we could lose the cached items at anytime.

3.1 Distributed Cache

Chapter 4

Lock

4.0.1 Default Locking

4.0.2 Optimistic Locking

4.0.3 Distributed Locks

Chapter 5

Message Queue

Chapter 6

Design Patterns

6.1 Partition

Partition or sharding is a direct application of the divide-and-conquer strategy. Instead of routing all the work to a single machine, we distribute the work to a cluster of machines. Partitioning the workload is a crutial step if we want to horizontally scale the system. The partitioning implementation needs to take into account the fact the size of cluster may change. For example, during a cluster maintenance, some machines may be shutdown, resulting in a decrease of the cluster capacity. Or we may need to add more machines to accommodate an increase in the request load. When the cluster size changes, requests may be routed to a different machines, which may not be ready to process the request efficiently. A typical solution to this problem is consistent hashing.

6.2 Leader Follower Pattern

Chapter 7

Classic Architecture

7.1 Basic Template

The diagram below demonstrates one of the basic structures of application services. Uses, such as mobile devices or applications running on servers send request to the endpoint of the target services. When the user request enters the service boundary, it usually hits a load balancer first. The load balancer is responsible for evenly distribute use requests to application servers in the cluster. The

For application servers form a cluster, there is usually a piece of code responsible for coordination. Servers may play different roles in a cluster, for example, some of the servers are coordinators and others are workers.

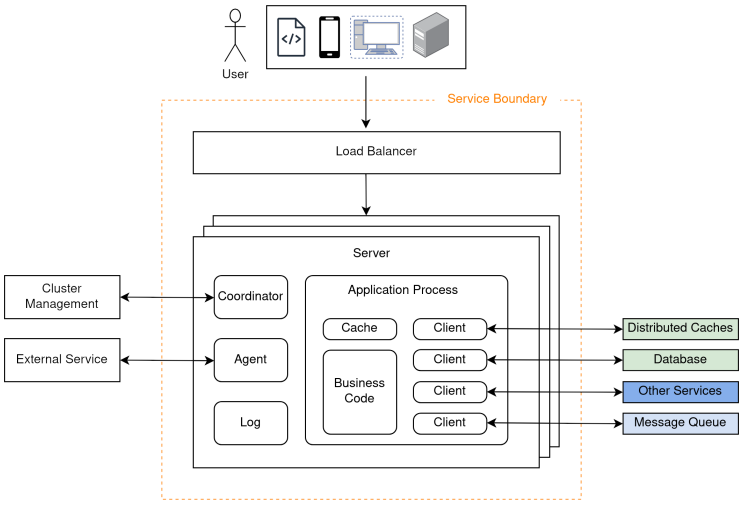

Figure 7.1: A basic template

Chapter 8

Data Modeling

normalization; denormalization

Part III

Cloud Technology

Chapter 9

AWS

Part IV

Database

Chapter 10

Relational Database

Chapter 11

Non-relational Database

Non-relational databases might be the right choice if:

- Your application requires super-low latency.
- Your data are unstructured, or you do not have any relational data.
- You only need to serizliae and deserialize data.
- You need to store a massive amount of data.