* [1. Domain Name System](#1-domain-name-system)
  + [Important Details](#important-details)
  + [DNS hierarchy](#dns-hierarchy)
  + [DNS caching](#dns-caching)
  + [DNS as a distributed system](#dns-as-a-distributed-system)
  + [Test](#test)
* [2. Load balancers](#2-load-balancers)
  + [Introduction](#introduction)
  + [Global server load balancers (GSLB)](#global-server-load-balancers--gslb-)
  + [Local Load balancers](#local-load-balancers)
  + [Advanced Load balancers](#advanced-load-balancers)
* [3. Databases](#3-databases)
  + [Introduction](#introduction)
  + [Types of Database](#types-of-database)
  + [1. Relational Database](#1-relational-database)
    - [Flexibility](#flexibility)
    - [Reduced redundancy](#reduced-redundancy)
    - [Concurrency](#concurrency)
    - [Integration](#integration)
    - [Backup and disaster recovery](#backup-and-disaster-recovery)
    - [Drawbacks of Relational databases](#drawbacks-of-relational-databases)
  + [2. Non-Relational Database](#2-non-relational-database)
    - [Simple design](#simple-design)
    - [Horizontal scaling](#horizontal-scaling)
    - [Availability](#availability)
    - [Support for unstructured and semi-structured data](#support-for-unstructured-and-semi-structured-data)
    - [Cost](#cost)
  + [Data Replication](#data-replication)
  + [**Synchronous Vs Asynchronous**](#synchronous-vs-asynchronous)
  + [Data replication models](#data-replication-models)
    - [1. Single leader/primary-secondary replication](#1-single-leaderprimary-secondary-replication)
    - [2. Multi-leader replication](#2-multi-leader-replication)
    - [3. Peer to peer/leaderless replication](#3-peer-to-peerleaderless-replication)
  + [Data Partition](#data-partition)
  + [Sharding](#sharding)
    - [1. Vertical Sharding](#1-vertical-sharding)
    - [2. Horizontal Sharding](#2-horizontal-sharding)
    - [3. Rebalance the partitions](#3-rebalance-the-partitions)
    - [4. Partitioning and secondary indexes](#4-partitioning-and-secondary-indexes)
  + [Request Routing](#request-routing)
    - [1. ZooKeeper](#1-zookeeper)
  + [Trade-offs in Databases](#trade-offs-in-databases)
  + [Centralized database](#centralized-database)
  + [Distributed database](#distributed-database)
  + [Query optimization and processing speed in a distributed database](#query-optimization-and-processing-speed-in-a-distributed-database)
  + [Parameters assumption](#parameters-assumption)
  + [Possible approaches](#possible-approaches)
* [4. Key-Value Store](#4-key-value-store)
  + [System Design - The Key-value store](#system-design---the-key-value-store)
  + [Design of a key-value store](#design-of-a-key-value-store)
  + [Ensure Scalability and Replication](#ensure-scalability-and-replication)
  + [Versioning Data and Archiving](#versioning-data-and-archiving)
  + [Enable Fault Tolerance and Failure Detection](#enable-fault-tolerance-and-failure-detection)
* [5. Content Delivery Network](#5-content-delivery-network)
  + [System Design: The Content Delivery Network (CDN)](#system-design--the-content-delivery-network--cdn-)
  + [Introduction to CDN](#introduction-to-cdn)
  + [In-depth Investigation of CDN:](#in-depth-investigation-of-cdn-)
  + [Evaluation of CDN Design](#evaluation-of-cdn-design)
* [6. Sequencer](#6-sequencer)
  + [System Design Sequencer](#system-design-sequencer)
  + [Design of a Unique ID Generator](#design-of-a-unique-id-generator)
  + [Unique IDs with Causality](#unique-ids-with-causality)
* [7. Distributed Monitoring](#7-distributed-monitoring)
  + [Distributed Monitoring](#distributed-monitoring)
  + [System Design](#system-design)
  + [Pre-requisites](#pre-requisites)
  + [Monitor Service-side errors](#monitor-service-side-errors)
  + [System Design](#system-design)
  + [Visualize](#visualize)
  + [Monitor Client-side errors](#monitor-client-side-errors)
  + [System Design](#system-design)
* [8. Distributed Caching](#8-distributed-caching)
  + [System Design](#system-design)
  + [High-level Design](#high-level-design)
  + [Detailed Design](#detailed-design)
  + [Evaluation](#evaluation)
  + [Memcached vs Redis](#memcached-vs-redis)
* [9. Distributed Messaging Queue](#9-distributed-messaging-queue)
  + [Requirements](#requirements)
  + [Considerations](#considerations)
  + [System Design](#system-design)
  + [Evaluation](#evaluation)
* [10. Publish-Subscribe System](#10-publish-subscribe-system)
  + [System Design](#system-design)
* [11. Rate Limiter](#11-rate-limiter)
  + [Requirements](#requirements)
  + [System Design](#system-design)
  + [Algorithms](#algorithms)
* [12. Blob Store](#12-blob-store)
  + [Requirements](#requirements)
  + [System Design](#system-design)
  + [Evaluation](#evaluation)
* [13. Distributed Search](#13-distributed-search)
  + [Requirements](#requirements)
  + [Indexing](#indexing)
  + [System Design](#system-design)
  + [Scaling](#scaling)
  + [Evaluation](#evaluation)
* [14. Distributed Logging](#14-distributed-logging)
  + [System Design](#system-design)
* [15. Distributed Task Scheduling](#15-distributed-task-scheduling)
  + [Requirements](#requirements)
  + [System Design](#system-design)
  + [Evaluation](#evaluation)
* [16. Shared Counters](#16-shared-counters)
  + [High-level Design](#high-level-design)
  + [System Design](#system-design)

## 1. Domain Name System

DNS is the Internet's naming service that maps human-friendly domain names to machine-readable IP addresses. Typically, DNS service is transparent to users. The browser translates the domain name to IP address via DNS Infra.

### Important Details

* Name servers: DNS isn't a single server. It's a complete infrastructure with numerous servers. DNS servers that respond to the users' queries are called name servers.
* Resource records: The DNS database stores domain name to IP address mappings in the form of record names (RR). The RR is the smallest unit of information that users request from the name servers. The 3 important pieces of information are type, name, and value.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Description | Name | Value | Example(Type, Name, Value) |
| A | Provides the hostname to IP address mapping | Hostname | IP address | (A, relay1.main.educative.io,104,18.2.119) |
| NS | Provides the hostname that is the authoritative DNS for a domain name | Domainname | Hostname | (NS,educative.io, dns.educative.io) |
| CNAME | Provides the mapping from alias to canonical hostname | Hostname | Canonical name | (CNAME, educative.io, server1.primary.educative.io) |
| MX | Provides the mapping of mail server from alias to canonoical hostname | Hostname | Canonical name | (MX, mail.educative.io, mailserver1.backup.educative.io) |

* Caching: DNS uses caching at different layers to reduce request latency for the user.
* Hierarchy: DNS name servers are hierarchical form. The hierarchical structure allows DNS to be highly scalable because of its increasing size and query load.

### DNS hierarchy

* DNS isn't a single server that accepts requests and responds to queries. It's a complete infrastructure with "Name Servers" at different hierarchies. Below are the different types of servers in the DNS hierarchy.
  + DNS resolver: Resolvers initiate the querying sequence and forward requests to the other DNS name servers. Typically, DNS resolvers lie within the premis of the user's network. These servers can also be called as local or defaults servers. Caching technique is used fo cater DNS queries.
  + Root-level name servers: These servers receives requests from the local servers. Root name servers maintain name servers based on top-level domain names, such as .com, .edu, .us, and so on. When a user requests the IP address of "docker.io", root-level name servers will return a list of top-level domain (TLD) servers that hold the IP addresses of the .io domain.
  + Top-level domain(TLD) name servers: These servers hold the IP addresses of authoritative name servers. The querying party will get a list of IP addresses that belong to the authoritative servers of the organization.
  + Authoritative name servers: These are the organization's DNS name servers that provide the IP addresses of the web or application servers.

* Typically, an iterative query is preferred to reduce query load on DNS infrastructure.

### DNS caching

* A record is a data unit within the DNS database that shows a name-to-value binding. Caching reduces response time to the user and decreases network traffic.

### DNS as a distributed system

* Highly Scalable There are 13 logical root name servers (named letter A to M) with over 1,000 replicated instances spread throughout the world strategically to handle user queries and managed by 12 different organizations.
* Reliable
  + Caching is done in the browser, the OS, and the local name server, and the ISP DNS resolvers also maintain a rich cache of freqently visited services. Even if the servers are down, cached records can be served.
  + Server replications DNS has replicated copies of each logical server spread systematiclly across the globe to entertain users requests at low latency.
  + Protocol Many clients rely on the unrealiable UDP to request and recevie DNS response, but is fast and improves DNS performance. DNS queries are usually retransmitted at the transport layer if there's no response for the previous one. Therefore, request-response might need additional round trips, which provides a shorter delay as compared to TCP, which needs a three-way handshake every time before data exchange.
* Consistent DNS provides eventual consistency and updates records on replicated servers lazily. Typically, it can take from few seconds up to three days on the DNS servers across the Internet. Consistency can suffer because of caching too. Since authoritativce servers are located within the ogranization, it may be possible that certain resource records are updated on the authoritative servers in case of server failures at the organization. To mitigate this issue, each cached record comes with an expiration time called (TTL)

### Test

nsloop => Non-Authoritative answer --> Refererence to cached response.  
dig => 53 refers to TTL and "Query time: 24 msec" refers to DNS response time

## 2. Load balancers

### Introduction

The job of the load balancer is to faily divide all client's requests among the pool of available servers. Load balancers perform this job to avoid overloading or crashing servers.

The load balancing layer is the first point of contact within a data center after the firewall. They provide following capabilities:-

* Scalability - Load balancers make upscaling or downscaling of servers transparent to the end users.
* Availability - One of the jobs of the load balancers is to hide faults and failuers of servers. Even if some servers go down or suffer a fault, the system still remains available.
* Performance - Load balancers can forward requests to servers with a lesser load so the user can get a quicker response time. This not only improves performance but also improves resource utilization.

Services offered by Load Balancers

* Health checking: LBs use the heartbeat protocol to monitor the health and, therefore, reliability of end-servers.
* TLS termination: LBs reduce the burden on end-servers by handling TLS termination with the client.
* Predictive analytics: LBs can predict traffic patterns through analytics performed over traffic passing through them or using statistics of traffic obtained over time.
* Service discovery: Client's requests are forwarded to appropriate hosting servers by inquiring about the server registry.
* Security: LBs improve security by mitigating attacks like DoS at different layers of the OSI model (3,4, 7).

### Global server load balancers (GSLB)

GSLB involves the distribution of traffic load across multiple geographical regions. GSLB ensures that globally arriving traffic load is intelligently forwarded to a data center.

GSLB takes forwarding decisions based on the users' geographic locations, the number of hosting servers in different locations, the health of data centers and so on. GSLB offers automatic zonal failover. Example: power or network failure in a data center requires that all the traffice be routed to another data center.

GSLB service can be installed on-premises or obtained through "Load balancing as a Service(LBaaS)".

### Local Load balancers

Load balancing achieved within a data center. This type of load balancing focuses on improving efficieny and better resource utliziation of the hosting servers in a data center. They behave like a "reverse proxy" and make their best effort to divide incoming requests among the pool of available servers. Incoming clients' requests seamlessly connect to the LB that uses a virtual IP address (VIP).

### Advanced Load balancers

* **Algorithms of load balancers**
  + **Round-robing scheduling** - each request is forwarded to a server in the pool in a repeating sequential manner.
  + **Weighted round-robin** - If some servers have a higher capability of serving clients' requests, then it's preferred to use a weighted round-robin algorithm. Each node is assigned a weight. LBs forward requests according to the weight of the node. The higher the weight, higher the number of assignments.
  + **Least connections** - If all the servers have the same capacity to serve clients, uneven load on certain servers is still a possibility. Newer requests are assigned to servers with fewer existing connections. LBs keep a state of the number and mapping of existing connections in such a scenario.
  + **Least response time** - In performance-sensitive services, algorithms such as least response time are required. This algorithm ensures that the server with the least response time is requested to serve the clients.
  + **IP hash** - Some applications provide a different level of service to users based on their IP addresses. In that case, hashing the IP address is performed to assign users' requests to servers.
  + **URL hash** - It may be possible that some services within the application are provided by specific servers only. In that case, a client requesting service from a URL is assigned to a certain cluster or set of servers. The URL hasing algorithm is used in those scenarios.
  + **Static vs dynamic algorithms**
    - Static algorithms don't consider the changing state of the servers. Therefore, task assignment is carried out based on existing knowledge about the server's configuration.
    - Dynamic algorithms consider the current or recent state of the servers. Dynamic algorithms maintain state by communicating with the server, which adds a communication overhead. State maintenance makes the design of the algorithm much more complicated.
  + In practice, dynamic algorithms provide far better results because they maintain a state of the serving hosts and are, therefore, worth the effort and complexity.

**Stateful load balancers versus stateless LBs**

* Stateful load balancing involved maintaining a state of the sessions established between clients and hosting servers. The stateful LB incorporates state information in its algorithm to perform load balancing. Stateful LBs increase complexity and limit scalability because session information of all the clients is maintained across all the load balancers.That is, load balancers share their state information with each other to make forwarding decisions.

* Stateless load balancing maintains no state and is, therefore, faster and light weight. Stateless LBs use consistent hashing to make forwarding decisions. However, if infrastructure changes (a new application server is added), stateless LBs may not be resilient as stateful LBs because consistent hashing alone isn't enough to route a request to the correct application server. Therefore, local state may still be required along with consistent hashing.

* **Types of Load balancers**
  + **Layer 4 load balancer**: Layer 4 refers to load balancing performed on the basis of transport protocol like tcp/udp. These types of LBs maintain conneciton/session with the clients and ensure that the same TCP/UDP communication ends up being forwarded to the same back-end server. Even though TLS termination is performed at layer 7 LBs, some layer 4 LBs also support it.
  + **Layer 7 load balancer**: Layer 7 load balancers are based on the data of application layer protocols. It's possible to make application-aware forwarding decisions based on HTTP headers, URLs, cookies, and other application-specific data- for example, user ID. Apart from TLS termination, these LBs can take responsibility like rate limiting users, HTTP routing, and header rewriting.
* Layer 7 load balancers are smart in terms of inspection. However layer 4 load balancers are faster in terms of processing.
* **Load balancer deployments** In practice, a single layer LB isn't enough for a large data center. In fact, multiple layers of load balancers coordinate to take informed forwarding decisions as shown below:-
  + **Tier-0 and Tier-1 LBs**: If DNS can be considered as the tier-0 load balancer, equal cost multipath (ECMP) routers are the tier-1 LBs. ECMP routers divides incoming traffic on the basis of IP or some other algo like round-robin or weighted round-robin. Tier-1 LBs will balance the load across different paths to higher tiers of load balancers. ECMP routers play a vital role in the horizontal scalability of the higher-tier LBs.
  + **Tier-2 LBs**: The Tier-2 LBs include layer 4 load balancers. They make sure that for any connection, all incoming packets are forwarded to the same tier-3 LBs. Consistent Hashing can be used for this purpose. Since hashing has limiation with infra changes, local or global state needs to be maintained. Tier-2 LBs glues Tier-1 and Tier-3 LBs.
  + **Tier-3 LBs**: These LBs are in direct contact with the back-end servers, they perform health monitoring of servers at HTTP level. This tier enables scalability by evenly distributing requests amoung the set of healthy back-end servers. This tier also reduces the burden on end-servers by handling low-level details like TCP-congestion control protocols, the discover of Path MTU (maximum transmission unit), the difference in application protocol between client and back-end servers, and so on.
* Tier-1 balances the load among the load balancers themselves. Tier 2 enables a smooth transition from tier-1 and tier-3 in case of failures Tier-3 does the actual load balancing between the back-end servers.

## 3. Databases

### Introduction

* For an application like WhatsApp, why can't we store information in a File?
  1. We can't offer concurrent managment to seperate users accessing the storage files fromd different loc.
  2. We can't grant different access rights to different users.
  3. How will the system scale and be available when adding thousands of entires?
  4. How will we search content for different users in a short time?

A database is an organized collection of data that can be managed and accessed easily. Databases are created to make it easier to store, retrieve, modify, and delete data in connection with different data-processing procedures.

* Two basic types of databases:
  + SQL (relational database) Relational databases, like phone books that record contact numbers and addresses, are organized and have predetermined schemas.
  + NoSQL (non-relational database) Non-Relational databases, like file directories that store anything from a person's constant information to shopping preferences, are unstructured, scattered, and feature a dynamic schema.

* Reasons why a database is important
  + Managing large data: A large amount of data can be easily handled with a database, which wouldn't be possible using other tools.
  + Retrieving accurate data (data consistency): Due to different constraints in databases, we can retrieve accurate data whenever we want.
  + Easy Updation: It is quite easy to update data in databases using data manipulation language (DML).
  + Security: Databases ensure the security of the data. A database only allows authorized users to access data.
  + Data Integrity: Databases ensure data integrity by using different constraints of data.
  + Availability: Databases can be replicated on different servers, which can be concurrently updated. These replicas ensure availability.
  + Scalability: Databases are divided (using data paritioning) to manage the load on a single node. This increases scalability.

### Types of Database

#### 1. Relational Database

Relational databases adhere to particular schemas before stroing the data. The data stored in relational databases  
has prior structure. Mostly, this model organizes data into one or more relations called tables, with a unique key  
for each tuple  
Since each tuple has a unique key, a tuple in one table can be linked to a tuple in other tables by storing the  
primary keys in other tables, generally known as foreign keys.  
  
A SQL (Structural Querying Language) is used for manipulating the database. This includes insertion, deletion, and  
retrieval of data.  
  
Relational databases provide Atomicity, consistency, isolation and durablity (ACID) properties to maintain the  
integrity of the database.   
 - Atomicity: A transasction is considered an atomic unit. If a statement fails within a transaction, it should  
 be aborted and rolled back.  
 - Consistency: At any given time, the database should be in a consistent stated, and it should remain consistent  
 after every transaction.  
 - Isolation: In the case of multiple transactions running concurrently, they shouldn't be affected by each  
 other. The final state of the database should be the same as the transactions were executed sequentially.  
 - Durability: The system should guarantee that completed transactions will survive permanently in the database  
 even in system failures.  
  
 Examples - MySQL, Oracle Database, Microsoft SQL Server, IBM DB2, Postgres, SQLite  
  
One of the greatest powers of the relational database is its abstractions of ACID transactions and related   
programming semantics.

##### Flexibility

In the context of SQL, DDL (Data Definition Language) provides us the flexibility to modify the database, including  
tables, columns, renaming the tables, and other changes. DDL even allows us to modify the schema while other   
queries are happening and the database server is running.

##### Reduced redundancy

One of the biggest advantages of the relational database is that it eliminates data redundancy. The information  
related to a specific entity appears in one table while the relevant data to that specific entity apprears in the  
other tables linked through foriegn keys. This process is called normalization and has the additional benefit of  
removing an inconsistent dependency.

##### Concurrency

A transaction is considered an atomic operation, so it also works in error handling to either roll back or commit  
a transaction on successful execution.

##### Integration

The process of aggregating data from multiple sources is a common practice in enterprise applications. A common  
way to perform this aggregation is to integrate a shared database where multiple applications store their data.

##### Backup and disaster recovery

Relational databases gurantee the state of data is consistent at any time. Most cloud-based relational databases  
perform continous mirroing to avoid loss of data and make the restoration process easier and quicker.

##### Drawbacks of Relational databases

**Impedance mismatch** Impedance mismatch is the difference between the relational model and the in-memory data structures. One is a structured data and other is a complex in-memory data structure. so, the impedence mismatch requires translation between two representations.

#### 2. Non-Relational Database

A NoSQL database is designed for a variety of data models to access and manage data. These databases are used in applications that require a large volume of semi-structured and unstructured data, low latency, and flexible data models.

##### Simple design

Unlike relational databases, NoSQL doesn't require dealing with the impedance mismatch - for ex storing all the employees' data in one document instead of multiple tables that require join operations. This strategy makes it simple and easier to write less code, debug, and maintain.

##### Horizontal scaling

Primarily, NoSQL is preferred due to its ability to run databases on a large scale. As the data is stored in one document. NoSQL databases often spread data across multiple nodes and balance data and queries across nodes automatically. In case of a node failure, it can be transparently replaced without any application disruption.

##### Availability

To enhace the availability of data, node replacement can be performed without application downtime. Data replication is supported to ensure high availability and disaster recovery.

##### Support for unstructured and semi-structured data

Many NoSQL databases work with data that doesn't have schema at the time of database configuration or data writes. For ex: document databases are structureless; they allow documents (JSON, XML, BSON and so on) to have different fields.

##### Cost

Licenses for many RDBMSs are pretty expensive, while many NoSQL databases are opensource and freely availble

**Types of Databases**

**Key-Value database**: use key-value methods like hash tables to store data in key-value pairs. Key servers as a unique or primary key, and the values can be anything ranging from simple scalar values to complex objects. Allow easy partitioning and horizontal scaling of the data. Examples: Amazon DynamoDB, Redis, and Memcached DB.

**Use Case**: Key-value databases are efficient for session-oriented applications. Session oriented-applications, such as web applications, store users' data in the main memory or in a database during a session.

**Document database**: is designed to store and retrieve documens in formats like XMl, JSON, BSON, and so on. These documents are composed of hierarchical tree data structure that can include maps, collections, and scalar values. Documents in this type of database may have varying structures and data. Examples : MongoDB, Google Cloud Firestore.

**Use Case**: Suitable for unstructured catalog data, like JSON files or other complex structured hierarchical data. for example in e-commerce applications, a product has thousands of attributes, which is unfeasible to store in a relational database due to its impact on the reading performance.

**Graph database**: uses graph data structure to store data, where nodes represent entities, and edges show relationships between entities. The organizaton of nodes based on relationships leads to interesting patterns between the nodes. This database allows us to store the data once and then interpret it differently based on relationships. Graph data is kept in store files for persistent storage. Examples: Neo4J, OrientDB, InfiniteGraph.

**UseCase**: can be used in social applications and provide interesting facts and figures among different kinds of users and their activities. The focus of graph database is to store data and pave the way to drive analyses and decisions based on relationships between entities. The nature of graph databases makes them suitable for various applications such as data regulation and privacy, machine learning research, financial services-based applications and many more.

**Columnar database**: store data in columns instead of rows. They enable across to all entries in the database column quickly and efficiently. Popular columnar databases include Cassandra, HBase, Hypertable, and Amazon redshift.

**UseCase**: Columnar database are efficient for a large number of aggregation and data analytics queries. It drastically reduces the disk I/O requirements and the amount of data required to load from the disk. For example, in applications related to financial institutions, there's a need to sum the financial transaction over a period of time. Columnar databases make this operation quicker by just reading the column for the amount of money, ignoring other attributes of customers.

**Choosing the right database**

|  |  |
| --- | --- |
| Relational database | Non-relational database |
| If the data to be stored is structured | If the data to be stored is unstructured |
| If ACID properties are required | If there's a need to serialize and deserialize data |
| If the size of the data is relatively smalland can fit on a node | If the size of the data to be stored is large |

**Drawbacks of NoSQL databases**

**Lack of standardization** - NoSQL doesn't follow any specific standard, like how relational database follow relational algebra. Porting applications from one type of NoSQL database to another might be a challenge.

**Consistency** - NoSQL provides different products based on the specific trade-offs between consistency and availability when failures can happen. We won't have strong data integrity, like primary and referential integrities in a relational database.

Data might not be strongly consistent but slowly converging using a weak model like eventual consistency.

### Data Replication

Following characteristics are required from a data store :- - Availability under faults (failure of some disk, nodes, and network and power outages). - Scalability (with increasing reads, writes, and other operations). - Performance (low latency and high throughput for the clients).

It's challenging, or even impossible, to achieve the above characteristics on a single node.

**Replication** refers to keeping multiple copies of the data at various nodes (preferably geographically distributed) to achieve availability, and performance.

Replication is relatively simple if the replicated data doesn't frequent changes. The main problem in replication arises when we have to maintain changes in the replicated data over time.

Additional complexities :- 1. How do we keep multiple copies of data consistent with each other? 2. How do we deal with failed replica nodes? 3. Should we replicate synchronously or asynchronously? \* How to deal with replication lag in case of asynchronous replication? 4. How do we handle concurrent writes? 5. What consistency model needs to be exposed to the end programmers?

#### **Synchronous Vs Asynchronous**

#### Data replication models

##### 1. Single leader/primary-secondary replication

##### 2. Multi-leader replication

##### 3. Peer to peer/leaderless replication

### Data Partition

#### Sharding

##### 1. Vertical Sharding

##### 2. Horizontal Sharding

##### 3. Rebalance the partitions

##### 4. Partitioning and secondary indexes

#### Request Routing

##### 1. ZooKeeper

### Trade-offs in Databases

#### Centralized database

#### Distributed database

#### Query optimization and processing speed in a distributed database

#### Parameters assumption

#### Possible approaches

## 4. Key-Value Store

### System Design - The Key-value store

### Design of a key-value store

### Ensure Scalability and Replication

### Versioning Data and Archiving

### Enable Fault Tolerance and Failure Detection

## 5. Content Delivery Network

### System Design: The Content Delivery Network (CDN)

### Introduction to CDN

### In-depth Investigation of CDN:

### Evaluation of CDN Design

## 6. Sequencer

### System Design Sequencer

### Design of a Unique ID Generator

### Unique IDs with Causality

## 7. Distributed Monitoring

### Distributed Monitoring

#### System Design

#### Pre-requisites

### Monitor Service-side errors

#### System Design

#### Visualize

### Monitor Client-side errors

#### System Design

## 8. Distributed Caching

### System Design

### High-level Design

### Detailed Design

### Evaluation

### Memcached vs Redis

## 9. Distributed Messaging Queue

### Requirements

### Considerations

### System Design

### Evaluation

## 10. Publish-Subscribe System

### System Design

## 11. Rate Limiter

### Requirements

### System Design

### Algorithms

## 12. Blob Store

### Requirements

### System Design

### Evaluation

## 13. Distributed Search

### Requirements

### Indexing

### System Design

### Scaling

### Evaluation

## 14. Distributed Logging

### System Design

## 15. Distributed Task Scheduling

### Requirements

### System Design

### Evaluation

## 16. Shared Counters

### High-level Design

### System Design