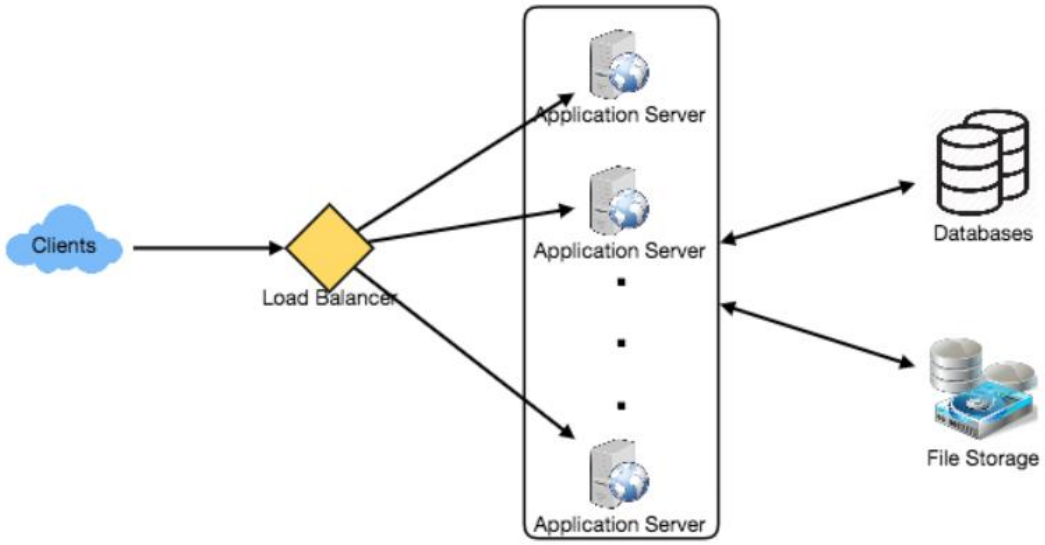
步骤 1：requirement clarify
例如设计 Twitter，需要问以下问题：
用户可以发文并 follow 别人么？
是否应该创建显示用户时间表？
推特包含照片和视频么？
只关注后端还是前后端都关注？
用户可以搜索推特文章么？
需要展示热搜话题么？
新文章需要有推送通知么？
。。。。。。

步骤 2：定义系统接口
例如推特的接口：
PostTweet(user_id, tweet_data, tweet_location, user_location, timestamp,…)
generateTimeline(user_id, current_time, user_location,…)
markTweetFavorite(user_id, tweet_id, timestamp,…)

步骤 3： back-of-the-envelop estimation
规模估算(推文数量，推文查看数量，每秒产生时间表数量，等等。。。)
需要使用多大内存，如果用户可以有照片和视频，内存占用会更大。
网络带宽使用期望是多少？ 这个对于决定如何管理 traffic 很重要。

步骤 4：定义数据模型
User： userId, name, email, DOB, creationdata, lastlogin,…
Tweet: tweetId, content, tweetLocation, numberoflikes, timestamp, …
userFollow: userID1, userID2…
favoriteTweets: userId, tweetId, timestamp…
使用 MySQL,还是 NoSQL??? 使用什么存储照片和视频？？？

步骤 5：high-level design
Draw a block diagram
例如 Twitter，



High-level 上，需要许多 app servers 响应 R/W，并且前面放上 load balancer 用来分配 traffic。
假设读流量很多，可以有 separate servers 用来 handle read。
后端，需要一个高效的 DB，存储所有推文和支持大量读请求。需要一个分布式文件存储系统存储照片和视频。

步骤 6：细节设计
如何 partition data?
对于热搜用户 如何处理?
在哪一层引入 cache?
哪一部分需要 load balance?

步骤 7：表明瓶颈，和如何解决瓶颈?
Try to discuss as many bottlenecks as possible and different approaches to mitigate them.
有没有 single point failure? 如何弱化?
有没有足够的 replica?
不同的 services 有没有足够的 copy?
如何监控系统性能？重要部分损坏或性能降级时有没有提醒?

系统设计基础知识

分布式系统的重要特性：
Scalability 扩展性：scalability is the capability of a system, process, or a network to grow and manage increased demand.
Scalable system would like to achieve this scaling without performance loss.
分为 horizontal/vertical scaling
水平：增加服务器数量 （Cassandra, mongoDB）
垂直：增加 CPU, RAM, 内存。。。(MySQL)

Reliability: probability of a system will fail in a given period. 通过 redundancy of software components and data.

Availability：the time a system remain operational to perform its required function in a specific period.正常工作时间的比例。

Rely 一定 available， 但是 available 不一定 reliability， high availability 也要有 repair time。

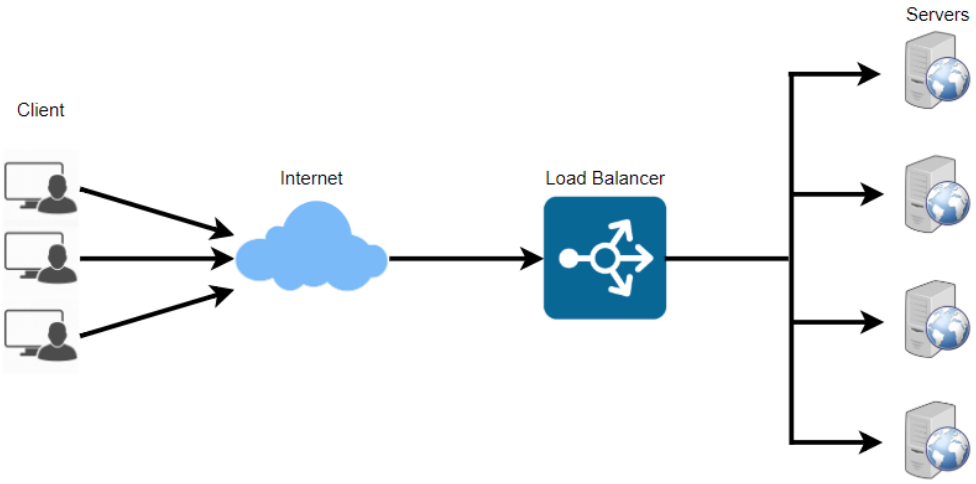
Efficiency：两个指标，response time/latency ,代表获得第一个响应的延迟。
Throughput(bandwidth)，代表单位时间内传递的 items 数量。

Serviceability or manageability: how easy it is to operate and maintain,代表系统维修时间，是否容易诊断错误，找到错误，容易更新，系统运行难度如何。。。

Load balancing:

Spread the traffic across a cluster of servers to improve responsiveness and availability of applications, websites or databases.还会检查所有服务器是否可用。

通常位于客户端和服务器之间，接收输入网络和应用流量，分配 traffic 到不同的后端服务器，使用各种算法。这样就减少了每个服务器的负荷，防止服务器出现 single point failure.提高了 availability。

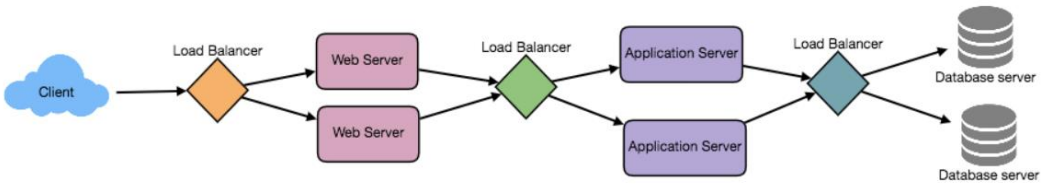


可以加 LB 的地方：

用户和 web server 之间

Web server 和 internal platform layer 之间。

Internal platform and database 之间。



LB 的好处：

Users experience faster, uninterrupted service.

Service providers experience less downtime and higher throughput.

LB makes it easy for system administrators to handle incoming requests while decreasing waiting time of users.

Smart LB can provide predictive analytics.

System administrators experience fewer failed or stressed components.

LB 算法有哪些?
先验证服务器是否正常工作，health check：周期性的试图连接后端服务器确保服务器正常。例如 hearthbeat.
各种各样的 LB 算法

Least connection method：map to server with fewest active connections

Least response time method: map to the server with the fewest active connections and the lowest average response time.

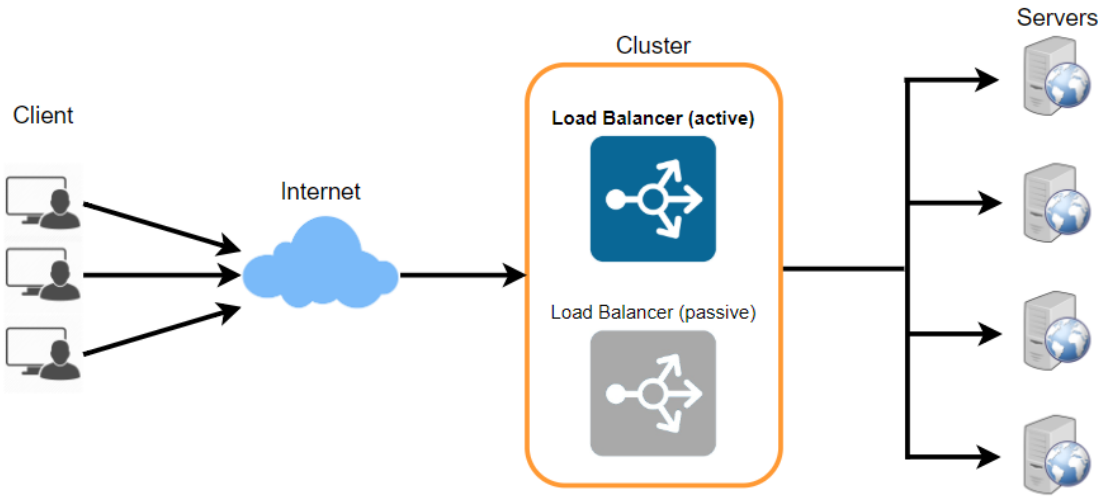
Least bandwidth method: map to the server that is currently sercing the least amount of traffic measured in MPS.

Round Robin method:一轮一轮挨个发，适合服务器完全一样，然后持久连接不多的情况。

Weighted round robin method:每个服务器都有 weight，代表处理能力。High weight 得到更多连接，处理更多 traffic。

IP hash: hash(IP ADDR) => redirect the request to the server.

为了避免 LB 的 single point failure：
加了 second LB，每个 LB monitors 另一个 LB 的健康情况。


Servers
Cluster
Client
Load Balancer (active)
Internet
Load Balancer (passive)

Cache:缓存
Based on temporary locality
Short-term memory。

Application server cache
Cache 存在了其中一台电脑上，但是 LB 每次都随机分配 server，增加了 cache Miss，如何解决？
两个方法：全局缓存，分布式缓存
Global cache and distributed caches.

Content distribution network(CDN)
CDN 是构建在现有网络基础之上的智能虚拟网络，依靠部署在各地的边缘服务器，通过中心平台的负载均衡、内容分发、调度等功能模块，使用户就近获取所需内容，降低网络拥塞，提高用户访问响应速度和命中率。CDN 的关键技术主要有内容存储和分发技术.

CDN 的基本原理是广泛采用各种缓存服务器，将这些缓存服务器分布到用户访问相对集中的地区或网络中，在用户访问网站时，利用全局负载技术将用户的访问指向距离最近的工作正常的缓存服务器上，由缓存服务器直接响应用户请求

Cache invalidation and evict
Write through and write back policy => evict need to update the cache，use MRU and LRU

Data partitioning
将一个大的 DB 表存储在不同的 machine 上。

Partitioning methods:

a. horizontal partitioning (by row)(range based)(data sharding)
   问题是，如果 range 选的不合适，导致 unbalanced servers.
b. vertical partitioning 一台服务器存用户信息，另一台存他的朋友，另一台存照片。。。
   问题是如果 application 日益发展，可能需要存的越来越多。
c. directory based partitioning
   lookup service 知道当前的 partition scheme。
   存储一个 map，key 是需要查找的信息，value 是存在了哪台服务器。

Partition 规则

a. Key or hash-based partition
   问题是如果增加了服务器，那么就要重写 hash function
   Redistribution

b. list partition
   For example, we can decide all users living in Iceland, Norway, Sweden, Finland, or Denmark will be stored in a partition for the Nordic countries.

c. Round-robin partition
   依次轮回排列

d. composite partition
   例如先 list partition，再 hash partition。

Partition 常见问题

a. joins and denormalization(NF1, NF2, NF3)

A common workaround for this problem is to denormalize the database so that queries that previously required joins can be performed from a single table. Of course, the service now has to deal with all the perils of denormalization such as data inconsistency.

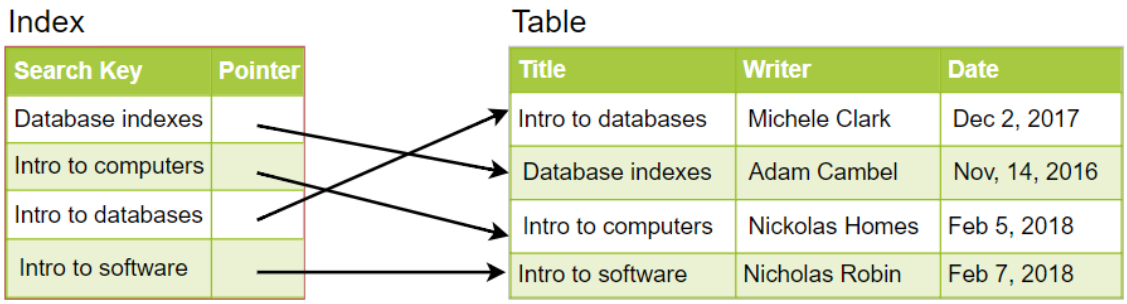
b. referential integrity

c. rebalancing

zip code 某一范围的数据特少，另一范围特别多，或者是负责用户照片的服务器接受了大量请求。可以考虑使用 lookup service?

Indexes

Find the row more quickly.
例子：图书馆目录
四列， book title, writer, subject, date of publication
通常有两个分类，按照书名分类，按照作者名分类。这就像是 index，加快了查找速度。



书名按照字母顺序排列

但是 index decrease write performance
slow down data insertion & update.

also have to update the index
This performance degradation applies to all insert, update, and delete operations for the table

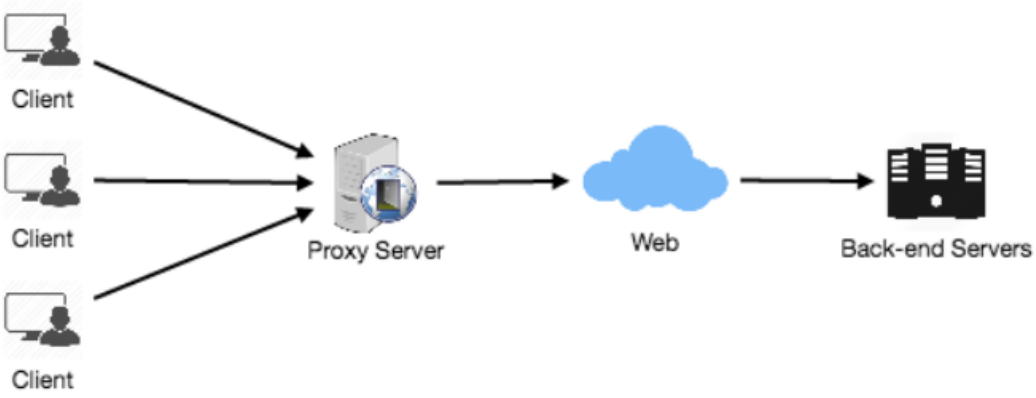
如果是经常写入，不怎么读，那不应该用 index!!!

Proxies
Proxy server is an intermediate server between the client and the back-end server.
Client connect proxy server to request a web page,file, connection...

用来 filter request, log request, transform requests(通过添加删除标头, 加密解密,

压缩。。)

优点： 它的 cache 可以满足很多请求。



代理服务器种类:
Open proxy：包括 anonymous proxy
开放代理，任何用户都可以使用。
匿名开放代理，帮助用户隐藏 IP 地址，安全。对于服务器操作者来说，开放代理有很高的风险

和 transparent proxy：主要是可以缓存 website。

反转代理服务器:
服务器根据客户端的请求，从其关联的一组或多组后端服务器（如 Web 服务器）上获取资源，然后再将这些资源返回给客户端，客户端只会得知反向代理的 IP 地址，而不知道在代理服务器后面的服务器簇的存在

前向代理 一般提到的是前向代理。表面上客户端 C 可以直接访问服务器 S，但实际上 C 在访问 S 的时候经过了中间的另一个中间的服务器 M，M 就是代理服务器。为什么说是前向代理？因为前向代理是面向客户端的，而不是服务器。M 接收了 C 的请求后，（有选择的）对请求进行简化或者其他处理，再向目标服务器请求数据。

C----------M----------S

反向代理 与前向代理相反，反向代理面向服务器，对于客户端 C 访问服务器 S 而言，好像 A 真的再访问 S 一样，其实真正的服务器是在 S 后面的 M.

C----------S-(反向代理服务器)----------M（真的服务器）

反向代理优点：LB， 加速访问静态内容，隐藏真实服务器。

Redundancy and replication

一个侧重 duplicate，一个侧重 consistency！！！

Redundancy is the duplication of critical components or functions of a system with the intention of increasing the reliability of the system, usually in the form of a backup or fail-safe, or to improve actual system performance

Replication means sharing information to ensure consistency between redundant resources, such as software or hardware components, to improve reliability, fault-tolerance, or accessibility.

SQL/NoSQL
Relational DB: MySQL,Oracle, MS SQL server, SQLite, Postgres, and MariaDB.
通过外键关联表与表之间的关系

关系型数据库存在的问题

1.网站的用户并发性非常高，往往达到每秒上万次读写请求，对于传统关系型数据库来说，硬盘 I/O 是一个很大的瓶颈
2.网站每天产生的数据量是巨大的，对于关系型数据库来说，在一张包含海量数据的表中查询，效率是非常低的
3.在基于 web 的结构当中，数据库是最难进行横向扩展的，当一个应用系统的用户量和访问量与日俱增的时候，数据库却没有办法像 web server 和 app server 那样简单的通过添加更多的硬件和服务节点来扩展性能和负载能力。当需要对数据库系统进行升级和扩展时，往往需要停机维护和数据迁移。
4.性能欠佳：在关系型数据库中，导致性能欠佳的最主要原因是多表的关联查询，以及复杂的数据分析类型的复杂 SQL 报表查询。为了保证数据库的 ACID 特性，必须尽量按照其要求的范式进行设计，关系型数据库中的表都是存储一个格式化的数据结构。

数据库事务必须具备 ACID 特性，ACID 分别是

Atomic 原子性，一个事务(transaction)中的所有操作，要么全部完成，要么全部不完成，**不会结束在中间某个环节**。事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。(oracle 通过 redo 和 undo 日志保证)

Consistency 一致性，

事务的一致性指的是在一个事务执行之前和执行之后数据库都必须处于一致性状态。如果事务成功地完成，那么系统中所有变化将正确地应用，系统处于有效状态。如果在事务中出现错误，那么系统中的所有变化将自动地回滚，系统返回到原始状态。

Isolation 隔离性，

指的是在并发环境中，当不同的事务同时操纵相同的数据时，每个事务都有各自的完整数据空间。由并发事务所做的修改必须与任何其他并发事务所做的修改隔离。事务查看数据更新时，数据所处的状态要么是另一事务修改它之前的状态，要么是另一事务修改它之后的状态，**事务不会查看到中间状态的数据**。

Durability 持久性。

指的是只要事务成功结束，它对数据库所做的更新就必须**永久保存**下来。即使发生系统崩溃，重新启动数据库系统后，数据库还能恢复到事务成功结束时的状态。

NoSQL:

Key-value store:例如 Redis, Voldemort, and Dynamo 获取的是一个数组

优点

1.用户可以根据需要去添加自己需要的字段，为了获取用户的不同信息，不像关系型数据库中，要对多表进行关联查询。仅需要根据 id 取出相应的 value 就可以完成查询。
2.适用于 SNS(Social Networking Services)中，例如 facebook，微博。系统的升级，功能的增加，往往意味着数据结构巨大变动，这一点关系型数据库难以应付，需要新的结构化数据存储。由于不可能用一种数据结构化存储应付所有的新的需求，因此，非关系型数据库严格上不是一种数据库，应该是一种数据结构化存储方法的集合。

不足：

只适合存储一些较为简单的数据，对于需要进行较复杂查询的数据，关系型数据库显的更为合适。不适合持久存储海量数据

Consistent hashing
Distributed hash table(DHT)
index = hash_function(key)

假设 N 个 cache 服务器，通常是 key % N

但是这样做有缺点：not horizontally scalable，加了服务器 N 增加 就得重新设计 Hash function, redistribution...

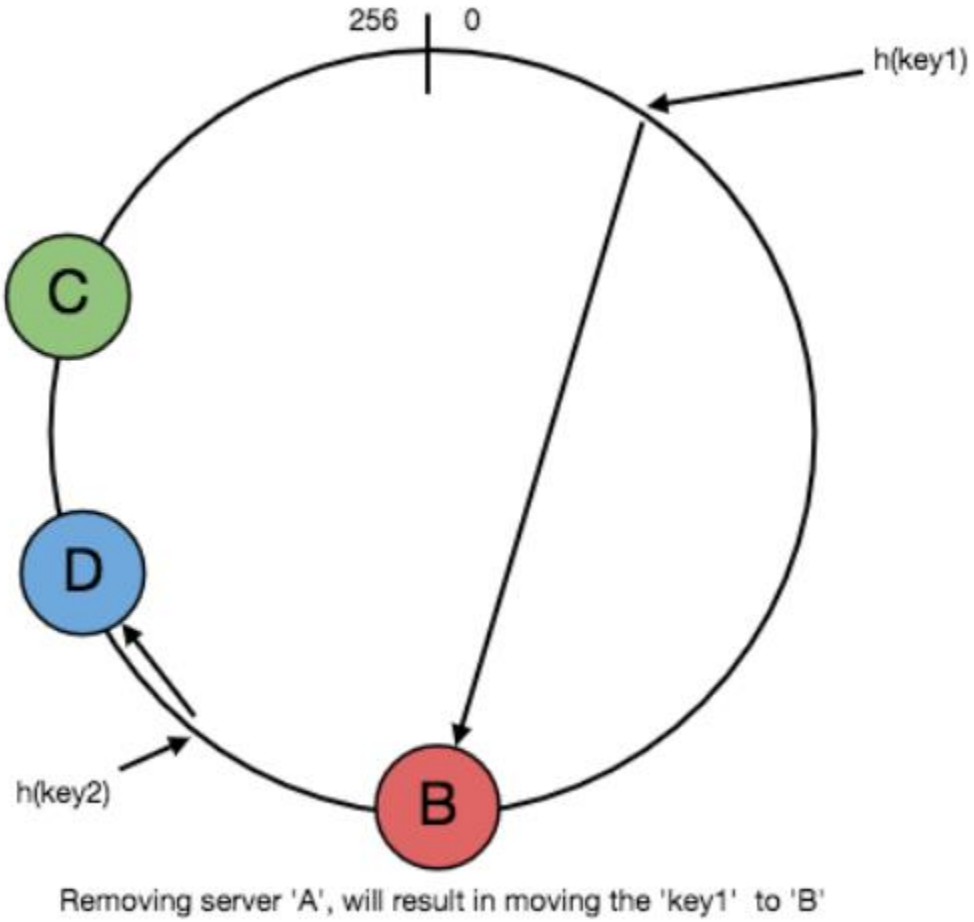
而且，不能 LB， 尤其是对 non-uniformlly distributed data，比如有些人热搜 有些事情是热点新闻。

===》 consistent hashing.

增加服务器 N=》minimize reorganization(redistribution).=>>easy to scale horizontally➔ 只有 k/n key 需要 remapped. K 是 key 数量, N 是服务器数量。

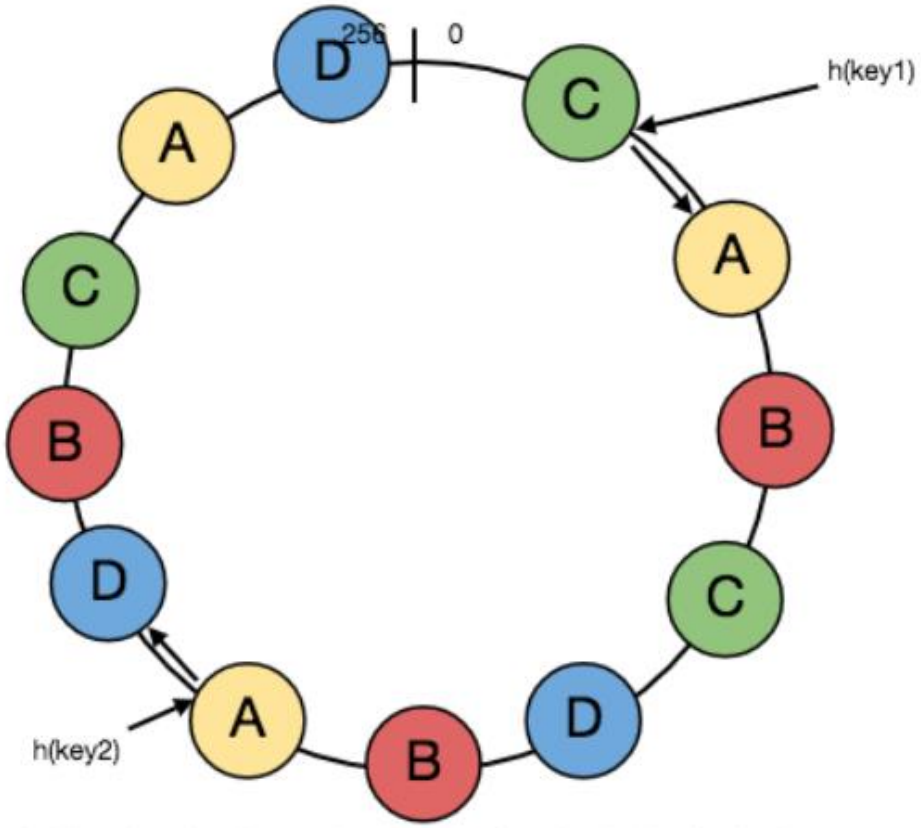
Consistent hashing 如何工作?

Hash the key to an interger，并且顺时针旋转，遇到的第一个服务器就是存它的地方。



加了 D 之后，只有 B->d 之间的需要 remap，移除 D 之后，只有 B-D 之间的需要 remap.

因为 data 不是均匀分布的，所以增加 LB=》方法就是增加 virtual replicas!!!



Adding virtual replicas to the ring to uniformly distributing the keys

Long-Polling vs WebSockets vs Server-Sent Events

Long-Polling, WebSockets, and Server-Sent Events are popular communication protocols between a client like a web browser and a web server

Ajax Polling

1. The client opens a connection and requests data from the server using regular HTTP.
2. The requested webpage sends requests to the server at regular intervals (e.g., 0.5 seconds).
3. The server calculates the response and sends it back, just like regular HTTP traffic.
4. The client repeats the above three steps periodically to get updates from the server.

The problem with Polling is that the client has to keep asking the server for any new data. As a result, a lot of responses are empty, creating HTTP overhead.

HTTP Long-Polling

with the expectation that the server may not respond immediately. That's why this technique is sometimes referred to as a "Hanging GET".

- If the server does not have any data available for the client, instead of sending an empty response, the server holds the request and waits until some data becomes available.
- Once the data becomes available, a full response is sent to the client. The client then immediately re-request information from the server so that the

server will almost always have an available waiting request that it can use to deliver data in response to an event.

The basic life cycle of an application using HTTP Long-Polling is as follows:

1. The client makes an initial request using regular HTTP and then waits for a response.
2. The server delays its response until an update is available or a timeout has occurred.
3. When an update is available, the server sends a full response to the client.
4. The client typically sends a new long-poll request, either immediately upon receiving a response or after a pause to allow an acceptable latency period.
5. Each Long-Poll request has a timeout. The client has to reconnect periodically after the connection is closed due to timeouts.

WebSockets

http 协议的缺点是通信只能由客户端发起，最典型的场景就是聊天室，假如用 HTTP 协议的话，就只能去轮询获取服务端有没有消息了，而用 WebSocket 的话，服务端有新消息可以自动推送。

（1）服务端可以主动推送信息，属于服务器推送技术的一种。

（2）建立在 TCP 协议之上，服务端的实现比较容易。

（3）与 HTTP 协议有着良好的兼容性，默认端口也是 80 和 443，并且握手阶段采用 HTTP 协议，因此握手时不容易屏蔽，能通过各种 HTTP 代理服务器。

（4）数据格式比较轻量，性能开销小，通信高效。

（5）可以发送文本，也可以发送二进制数据。

（6）没有同源限制，客户端可以与任意服务器通信。

（7）协议标识符是 ws（如果加密，则为 wss），服务器网址就是 URL。

Server-sent Events(SSEs)

1. Client requests data from a server using regular HTTP.
2. The requested webpage opens a connection to the server.
3. The server sends the data to the client whenever there's new information available.