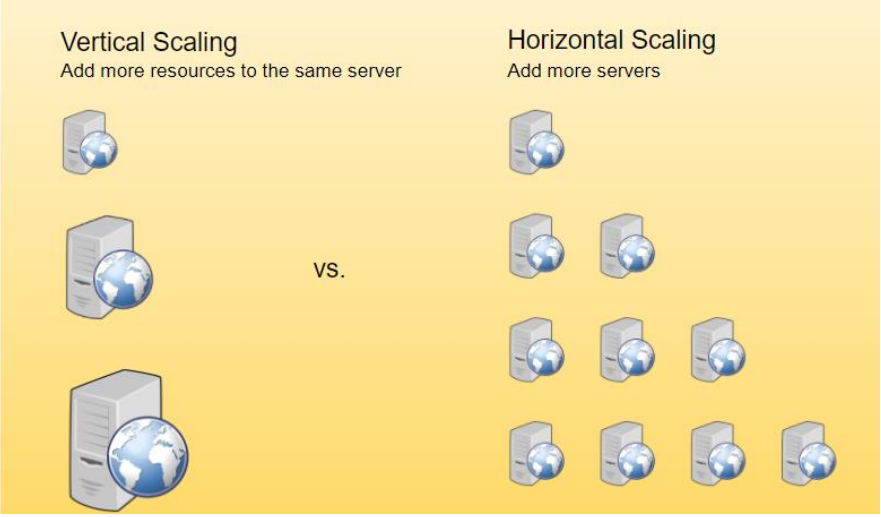
System Design

一．系统设计的几个概念

1. Scalability

A scalable system would like to achieve this scaling without performance loss.

Horizontal scaling means that you scale by adding more servers into your pool of resources whereas Vertical scaling means that you scale by adding more power (CPU, RAM, Storage, etc.) to an existing server.



Vertical scaling : MySQL (Vertical-scaling is usually limited to the capacity of a single server)

Horizontal scaling : MongoDB （it is often easier to scale dynamically by adding more machines into the existing pool）

2.Reliability (可靠性)

Reliability is the probability a system will fail in a given period.

3.Availability

Availability is the time a system remains operational to perform its required function in a specific period.

High reliability contributes to high availability.

4.Efficiency

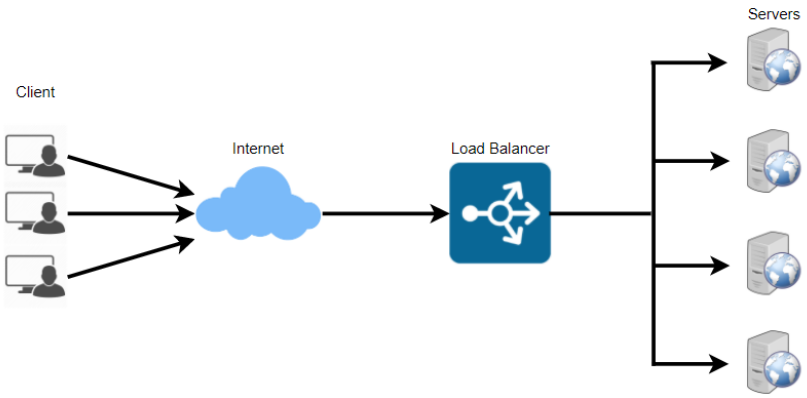
Latency, bandwidth;

5.Serviceability or Manageability

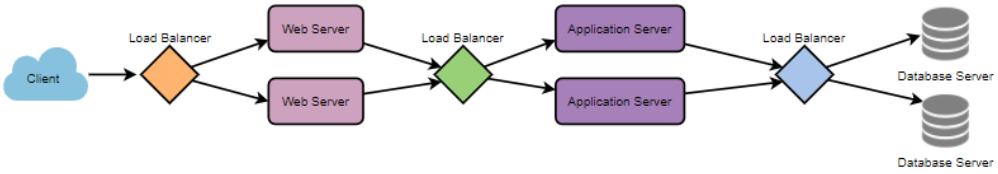
Serviceability or manageability is the simplicity and speed with which a system can be repaired or maintained.

6. Load Balancing

It helps to spread the traffic across a cluster of servers to improve responsiveness and availability of applications, websites or databases.



We can add LBs at three places:

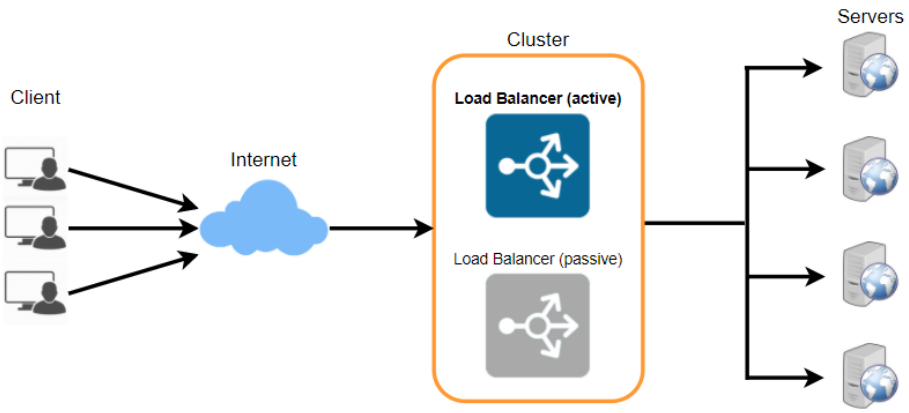


Benefits:

1) Users experience faster, uninterrupted service. Users won't have to wait for a single struggling server to finish its previous tasks.
2) Load balancing makes it easier for system administrators to handle incoming requests while decreasing wait time for users.

Redundant Load Balancers 冗余负载均衡器

Each LB monitors the health of the other and, since both of them are equally capable of serving traffic and failure detection, in the event the main load balancer fails, the second load balancer takes over.



Load Balancing Algorithms.

1) They will first ensure that the server they choose is actually responding appropriately to requests and then use a pre-configured algorithm to select one from the set of healthy servers.
2) If a server fails a health check, it is automatically removed from the pool, and traffic will not be forwarded to it until it responds to the health checks again.
3) **Least Connection Method** — This method directs traffic to the server with the fewest active connections. This approach is quite useful when there are a large number of persistent client connections which are distributed between the servers.

   Least Response Time Method — This algorithm directs traffic to the server with the fewest active connections and the lowest average response time.

   Least Bandwidth Method - This method selects the server that is currently serving the least amount of traffic measured in megabits per second (Mbps).

   Round Robin Method — This method cycles through a list of servers and sends each new request to the next server. When it reaches the end of the list, it starts over at the beginning. It is most useful when the servers are of equal specification and there are not many s.

   Weighted Round Robin Method — The weighted round-robin scheduling is designed to better handle servers with different processing capacities. Each server is assigned a weight (an integer value that indicates the processing capacity). Servers with higher weights receive new connections before those with less weights and servers with higher weights get more connections than those with less weights.

   IP Hash — Under this method, a hash of the IP address of the client is calculated to redirect the request to a server.

7. Cache

Application server cache

If your load balancer randomly distributes requests across the nodes, the same request will go to different nodes, thus increasing cache misses. Two choices for overcoming this hurdle are global caches and distributed caches.

Content Distribution Network (CDN)

CDNs are a kind of cache that comes into play for sites serving large amounts of static media.

Cache Invalidation 缓存无效

Write-through cache

Write-back cache

Write-around cache

Cache eviction policies 缓存逐出策略

First In First Out (FIFO): The cache evicts the first block accessed first without any regard to how often or how many times it was accessed before.

Last In First Out (LIFO): The cache evicts the block accessed most recently first without any regard to how often or how many times it was accessed before.

Least Recently Used (LRU): Discards the least recently used items first.

Most Recently Used (MRU): Discards, in contrast to LRU, the most recently used items first.

Least Frequently Used (LFU): Counts how often an item is needed. Those that are used least often are discarded first.

Random Replacement (RR): Randomly selects a candidate item and discards it to make space when necessary.

8. Data partitioning

is a technique to break up a big database (DB) into many smaller parts.

Partitioning Method

a. Horizontal partitioning: In this scheme, we put different rows into different tables.

The key problem with this approach is that if the value whose range is used for partitioning isn't chosen carefully, then the partitioning scheme will lead to unbalanced servers.

b. Vertical Partitioning:

In this scheme, we divide our data to store tables related to a specific feature in their own server. The main problem with this approach is that if our application experiences additional growth, then it may be necessary to further partition a feature specific DB across various servers

c: Directory Based partitioning

Partitioning Criteria:

a.Key or Hash-based partitioning: Under this scheme, we apply a hash function to some key attributes of the entity we are storing; that yields the partition number.

b.List partitioning 列表分区: In this scheme, each partition is assigned a list of values, so whenever we want to insert a new record, we will see which partition contains our key and then store it there.

c. Round-robin partitioning 循环分区: This is a very simple strategy that ensures uniform data distribution. With 'n' partitions, the 'i' tuple is assigned to partition (i mod n).

d. Composite partitioning 综合分区: Under this scheme, we combine any of the above partitioning schemes to devise a new scheme.

Common Problems of Data Partitioning

a. Joins and Denormalization:

b. Referential integrity:

c. Rebalancing:

In such cases, either we have to create more DB partitions or have to rebalance existing partitions, which means the partitioning scheme changed and all existing data moved to new locations. Doing this without incurring downtime is extremely difficult.

9. Indexes

Indexes are well known when it comes to databases. Sooner or later there comes a time when database performance is no longer satisfactory. **One of the very first things you should turn to when that happens is database indexing.**

The goal of creating an index on a particular table in a database is to make it faster to search through the table and find the row or rows that we want. Indexes can be created using one or more columns of a database table, providing the basis for both rapid random lookups and efficient access of ordered records.

Example:

Index

Search Key	Pointer
Database indexes	
Intro to computers	
Intro to databases	
Intro to software	

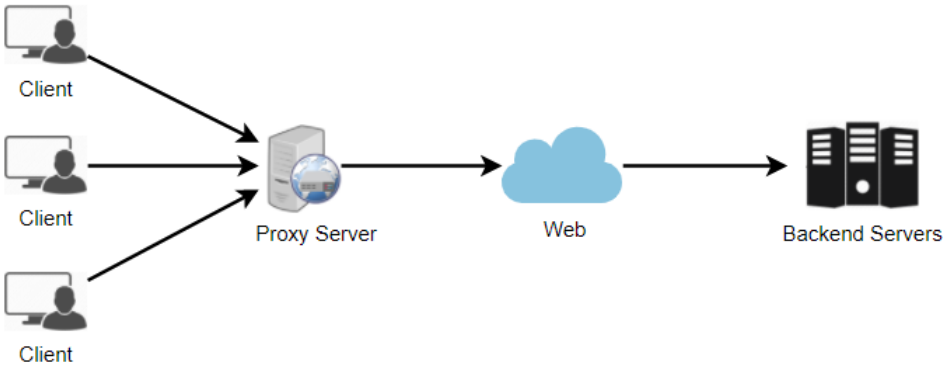
Table

Title	Writer	Date
Intro to databases	Michele Clark	Dec 2, 2017
Database indexes	Adam Cambel	Nov, 14, 2016
Intro to computers	Nickolas Homes	Feb 5, 2018
Intro to software	Nicholas Robin	Feb 7, 2018

An index can dramatically speed up data retrieval but may itself be large due to the additional keys, which slow down data insertion & update.

10. Proxies

A proxy server is an intermediate server between the client and the **back-end server.** Typically, proxies are used to filter requests, log requests, or sometimes transform requests (by adding/removing headers, encrypting/decrypting, or compressing a resource). Another advantage of a proxy server is that its cache can serve a lot of requests.



Types.

Open Proxy

1)Anonymous Proxy-This proxy reveals its identity as a server but does not disclose the initial IP address. Though this proxy server can be discovered easily it can be beneficial for some users as it hides their IP address.

2)Transparent Proxy–This proxy server again identifies itself, and with the support of

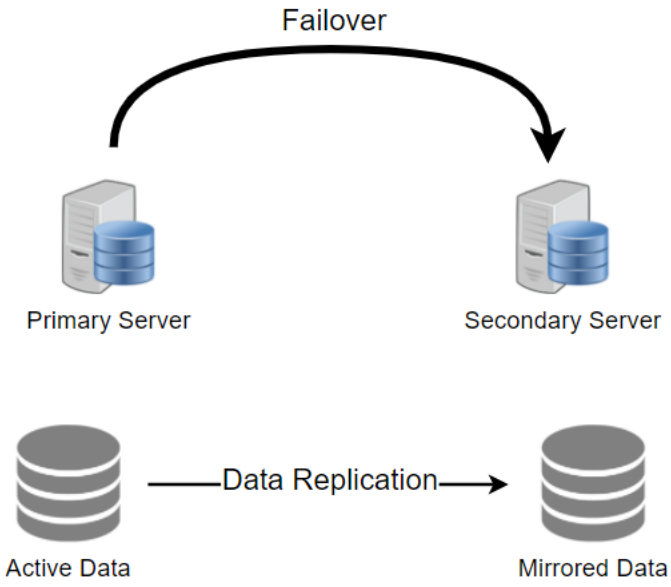
HTTP headers, the first IP address can be viewed. The main benefit of using this sort of server is its ability to cache the websites.

Reverse Proxy

A reverse proxy retrieves resources on behalf of a client from one or more servers. These resources are then returned to the client, appearing as if they originated from the proxy server itself.

11. Redundancy and Replication

Redundancy is the duplication of critical components or functions of a system with the intention of increasing the reliability of the system.



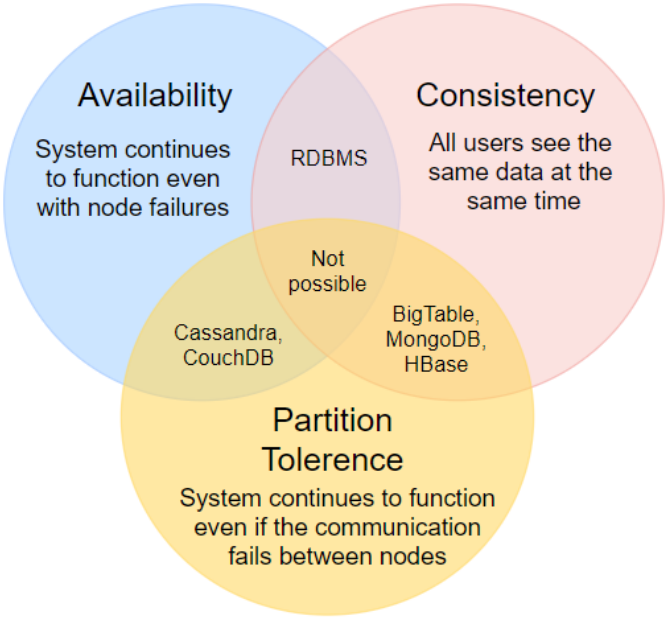
12. CAP Theorem

CAP theorem states that it is impossible for a distributed software system to simultaneously provide more than two out of three of the following guarantees (CAP): Consistency, Availability, and Partition tolerance.

Consistency: All nodes see the same data at the same time. Consistency is achieved by updating several nodes before allowing further reads.

Availability: Every request gets a response on success/failure. Availability is achieved by replicating the data across different servers.

Partition tolerance: The system continues to work despite message loss or partial failure. A system that is partition-tolerant can sustain any amount of network failure that doesn't result in a failure of the entire network. Data is sufficiently replicated across combinations of nodes and networks to keep the system up through intermittent outages.



13. Consistent Hashing 一致性哈希算法（明天上网查补充资料）

Distributed Hash Table (DHT) is one of the fundamental components used in distributed scalable systems. Hash Tables need a key, a value, and a hash function where hash function maps the key to a location where the value is stored.

$$index = hash\_function(key)$$

补充：普通哈希算法如下

键	哈希值	节点编号
semlinker	2650	1
kakuqo	1847	2
test	1356	0

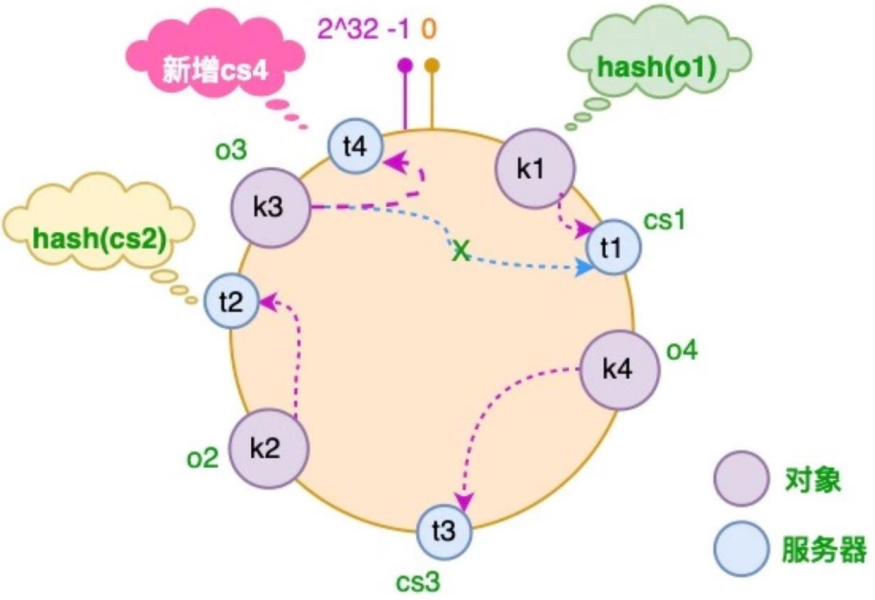
node_number = hash(key) % 3

键	哈希值	节点编号
semlinker	2650	0
kakuqo	1847	1
test	1356	0

node_number = hash(key) % 2

在分布式多节点系统中，出现故障很常见。任何节点都可能在没有任何事先通知的情况下挂掉，针对这种情况我们期望系统只是出现性能降低，正常的功能不会受到影响。 对于原始示例，示例中有的 3 个节点，假设其中 1 个节点出现故障，这时节点数发生了变化，节点个数从 3 减少为 2。很明显节点的减少会导致键与节点的映射关系发生变化，这个变化对于新的键来说并不会产生任何影响，但对于已有的键来说，将导致节点映射错误，以 “semlinker” 为例，变化前系统有 3 个节点，该键对应的节点编号为 1，当出现故障时，节点数减少为 2 个，此时该键对应的节点编号为 0。

一致性哈希算法：增加一台服务器 cs4，经过同样的 hash 运算，该服务器最终落于 t1 和 t2 服务器之间，具体如下图所示

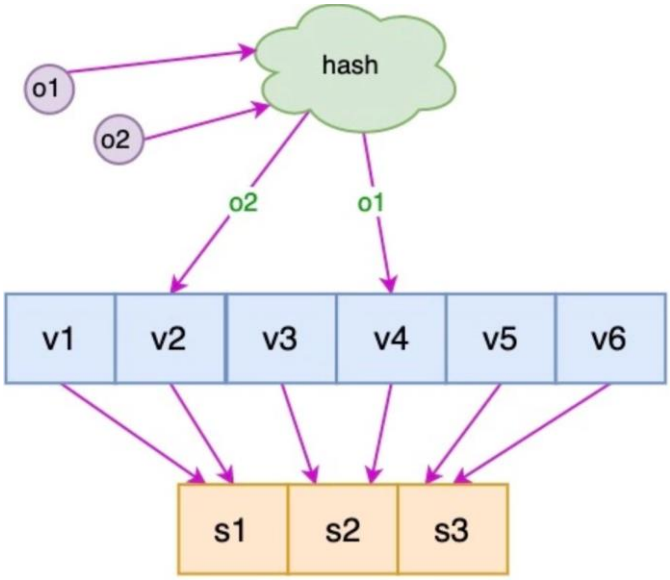


可能出现的问题：

对于新增服务器的情况还存在一些问题。新增的服务器 cs4 只分担了 cs1 服务器的负载，服务器 cs2 和 cs3 并没有因为 cs4 服务器的加入而减少负载压力。如果 cs4 服务器的性能与原有服务器的性能一致甚至可能更高，那么这种结果并不是我们所期望的。

针对这个问题，我们可以通过引入虚拟节点来解决负载不均衡的问题。即将每台物理服务器虚拟为一组虚拟服务器，将虚拟服务器放置到哈希环上，如果要确定对象的服务器，需先确定对象的虚拟服务器，再由虚拟服务器确定物理服务器。

图中 o1 和 o2 表示对象，v1 ~ v6 表示虚拟服务器，s1 ~ s3 表示物理服务器。



14. SQL vs. NoSQL

SQL：Relational databases store data in rows and columns. Each row contains all the information about one entity and each column contains all the separate data points. Some popular relational databases are MySQL, Oracle, MS SQL Server, SQLite, Postgres, and MariaDB.

NoSQL：

Key-Value Stores: eg. Redis, Voldemort, and Dynamo.

Document Databases: eg. CouchDB and MongoDB.

Wide-Column Databases: eg. Cassandra and HBase.

Graph Databases: eg. Neo4J and InfiniteGraph.

High level differences between SQL and NoSQL （后面复习的时候重新回忆）

Storage

Schema

Querying

Scalability

Reliability or ACID Compliancy (Atomicity, Consistency, Isolation, Durability)

SQL VS. NoSQL - Which one to use?

Reasons to use SQL database #

1. We need to ensure ACID compliance.
2. Your data is structured and unchanging.

Reasons to use NoSQL database

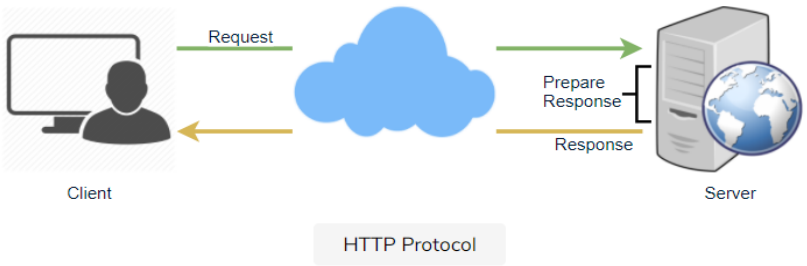
1. Storing large volumes of data that often have little to no structure.
2. Making the most of cloud computing and storage.
3. Rapid development.

15. 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.

Http protocols:

1. The client opens a connection and requests data from the server.
2. The server calculates the response.
3. The server sends the response back to the client on the opened request.



Ajax Polling:

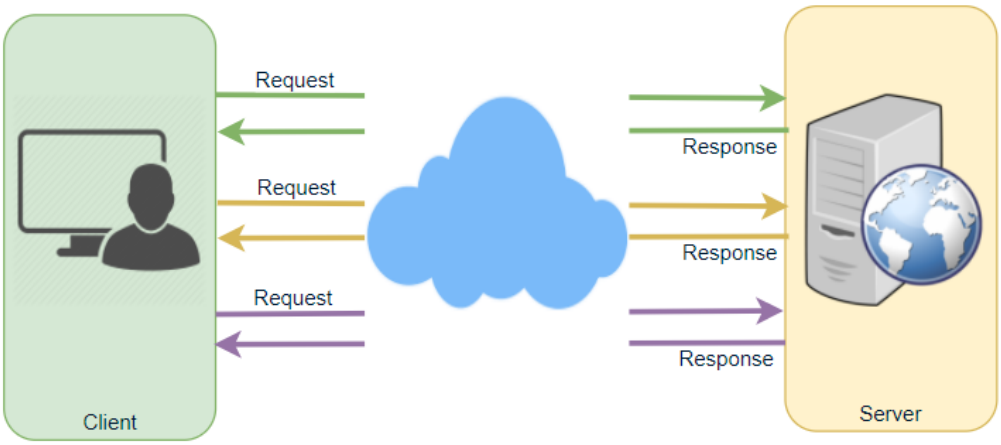
1.The client opens a connection and requests data from the server using regular HTTP.

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.

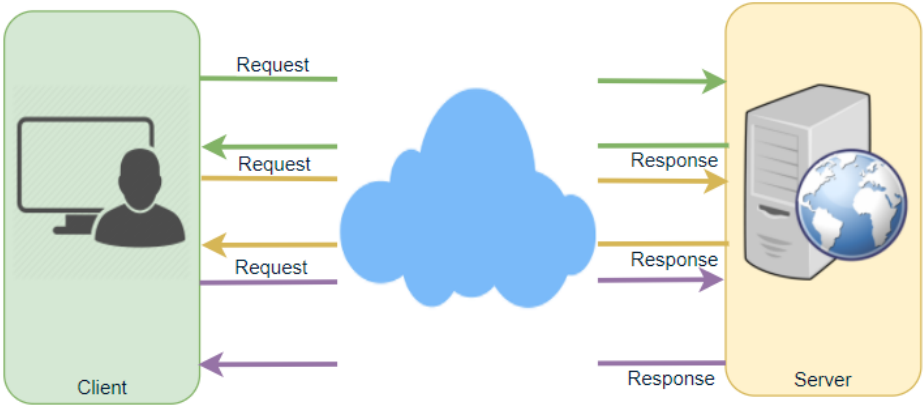
The problem with Polling is that the client has to keep asking the server for any new data. As a result, a lot of responses are empty, creating HTTP overhead.



HTTP Long-Polling:

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.



Long Polling Protocol

WebSockets:

WebSocket provides communication channels over a single TCP connection. It provides a connection between a client and a server that both parties can use to start sending data at any time.

WebSockets Protocol

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.