High Level Design Basics

Source: - Educative.io and System-design-primer

Distributed Systems

Scalability – Horizontal vs Vertical

Disadvantages of horizontal scaling,

- Scaling horizontally introduces complexity and involves cloning servers
- Servers should be stateless: they should not contain any user-related data like sessions or profile picture
- Sessions can be stored in databases or a persistent cache(Redis, Memcached)

Reliability – Probability of failure

Availability – Remaining Operational when required

Efficiency

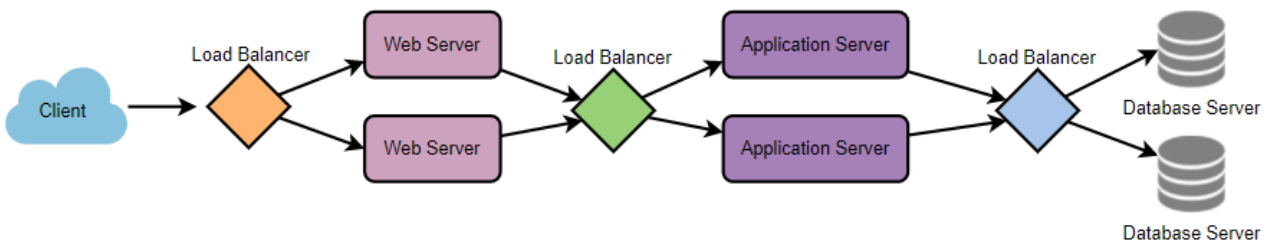
- Latency – Response time
- Throughput – Bandwidth

Serviceability/ Manageability – Easy to repair and maintain

Load Balancing

Load balancer can be kept at 3 places

- Between the user and the web server
- Between web servers and an internal platform layer, like application servers or cache servers
- Between internal platform layer and database.



Health Checks - Load balancers should only forward traffic to “healthy” backend servers

Different Algorithms for Load balancing,

1. Least Connection method (Least connections)
2. Least Response time (Least Avg. response time)
3. Least Bandwidth (Least traffic in server)
4. Round robin (cycles through a list of servers 1 by 1)
5. Weighted round robin (Assigns weight to server[Integer value])
6. IP hash (hash of the IP address of the client is calculated to redirect)

Redundant Load Balancers:-

Load balancer can be a single point of failure, so 2 load balancers can be designed hence in the event the main load balancer fails, the second load balancer takes over. Active and Passive load balancers.

Layer 4 vs Layer 7 Load balancer:-

Layer 4 LB	**Layer 7 LB**
looks at info at the *transport layer* to decide how to distribute requests	look at the *application layer* to decide how to distribute requests
this involves the source, destination IP addresses, and ports in the header, but not the contents of the packet	This can involve contents of the header, message, and cookies
Layer 4 load balancers forward network packets to and from the upstream server, performing Network Address Translation (**NAT**)	It can make a load-balancing decision based on the content of the message. It then makes a new TCP connection to the selected upstream server.
Transmission Control Protocol (TCP) is the Layer 4 protocol for HTTP traffic on the Internet	HTTP is the predominant Layer 7 protocol for website traffic on the Internet

The idea of **NAT** is to allow multiple devices to access the Internet through a single public address. To achieve this, the translation of a private IP address to a public IP address is required

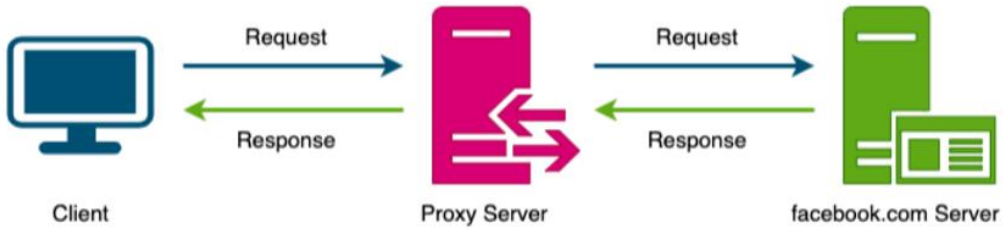
Web Server vs Application Server

Web server	Application server
Deliver static content	Delivers dynamic content
Content is delivered using the *HTTP protocol only*	Provides *business logic* to application programs using several protocols (including HTTP)
Serves only web-based applications	Can serve web and enterprise-based applications
Facilitates web traffic that is not very resource intensive.	Facilitates longer running processes that are very resource-intensive
No support for multi-threading	Uses multi-threading to support multiple requests in parallel
Apache tomacat	Jboss, Glassfish, Weblogic etc.

Proxies and Reverse Proxies

A proxy server is an intermediate piece of software or hardware that sits between the client and the server

Forward Proxy:-

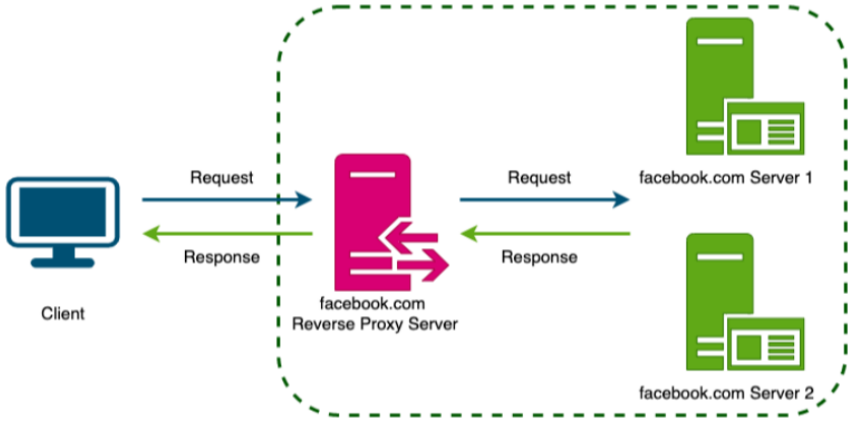


Hides the identity of the client from the server by sending requests on behalf of the client

Proxies can combine the same data access requests into one request and then return the result to the user; this technique is called **collapsed forwarding**.

Forward proxies are used to cache data, filter requests, log requests, or transform requests (by adding/removing headers, encrypting/decrypting, or compressing a resource)

Reverse Proxy:-



The reverse proxy hides the final server that served the request from the client.

A reverse proxy, just like a forward proxy, can be used for caching, load balancing, or routing requests to the appropriate servers

Serves static content [HTML/CSS, photos and videos]

Reverse proxy is a single point of failure. But keeping multiple Reverse proxies will make the system more complex.

Caching

Caching at various levels:

Client caching

Caches can be located on the client side (OS or browser), server side, or in a distinct cache layer.

CDN caching CDNs are considered a type of cache.

Web server caching

Reverse proxies and caches such as ***Varnish*** can serve static and dynamic content directly. Web servers can also cache requests, returning responses without having to contact application servers.

Database caching

Your database usually includes some level of caching in a default configuration, optimized for a generic use case.

Application caching

In-memory caches such as Memcached and Redis are key-value stores between your application and your data storage. Since the data is held in RAM, it is much faster than typical databases where data is stored on disk. Both Redis and Memcached are open source.

Redis has the following additional features:

- Persistence option (If enabled can store cache even data after restarts too)
- Built-in data structures such as sorted sets and lists

Cache Miss: A cache miss is an event in which a system or application makes a request to retrieve data from a cache, but that specific data is not currently in cache memory

CDN – Content Delivery Network:

Kind of cache that comes into play for sites serving large amounts of static media. If the system we are building is not large enough to have its own CDN, we can ease a future transition by serving the static media off a separate subdomain (e.g., static.yourservice.com) using a lightweight HTTP server like Nginx

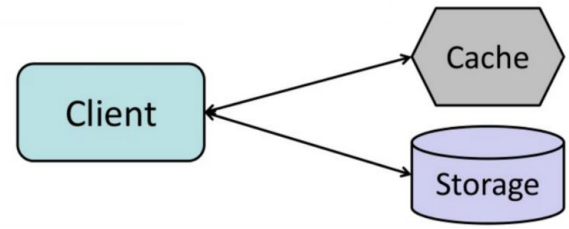
Cache Eviction:

To free up the space in cache. There are various cache eviction strategies

- First In First Out (FIFO): Evicts the 1st stored data from cache
- Last In First Out (LIFO): Evicts the last stored data from cache
- Least Recently Used (LRU): Discards the least recently used items first.
- Most Recently Used (MRU): Discards 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

Caching Invalidation Patterns:-

1. **Cache Aside Pattern:**



The application is responsible for reading and writing from storage. *The cache does not interact with storage directly*. The application does the following:

Look for entry in cache, resulting in a cache miss →Load entry from the database → Add entry to cache →Return entry

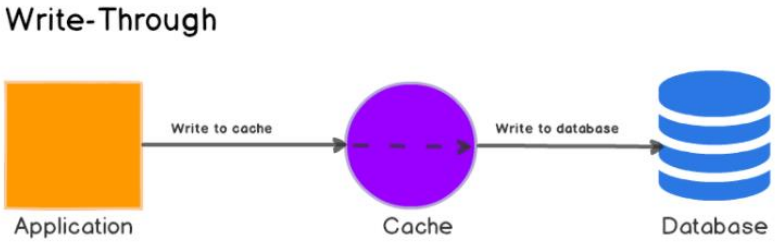
Cache-aside is also referred to as lazy loading. Only requested data is cached, which avoids filling up the cache with data that isn't requested.

In this patterns cache is not a single point of failure, even if cache fails app will fetch and write data from DB instead of crashing.

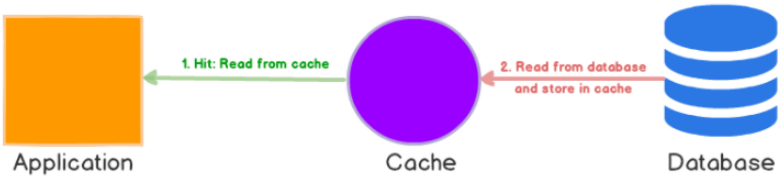
Disadvantage(s):

- Each cache miss results in three trips, which can cause a noticeable delay.
- Data can become stale if it is updated in the database. This issue is mitigated by setting a time-to-live (TTL) which forces an update of the cache entry, or by using write-through.

2. **Write/Read-through**



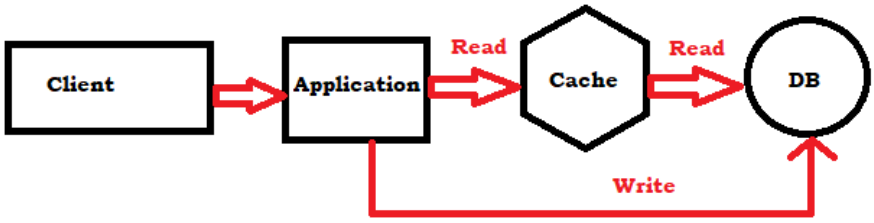
Read-Through



- In this write strategy, data is first written to the cache and then to the database. The cache sits in-line with the database and writes always go through the cache to the main database
- When there is a cache miss, it loads missing data from database, populates the cache and returns it to the application
- Read-through caches work best for read-heavy workloads when the same data is requested many times.
- DynamoDB Accelerator (DAX) is a good example of read-through / write-through cache. It sits inline with DynamoDB and your application

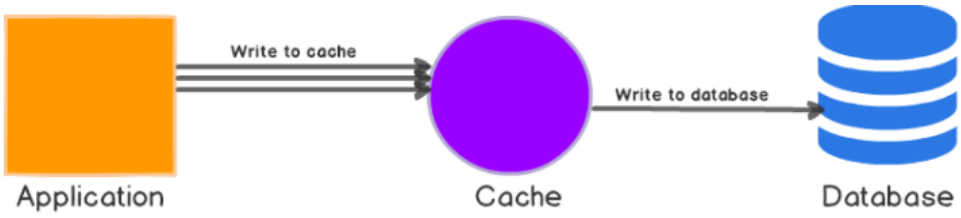
3. Write Around:

Data is written directly to the database and only the data that is read makes it way into the cache.



4. Write Back:

The application writes data to the cache which acknowledges immediately and after some delay, it writes the data back to the database



Good for write-heavy workloads. Can tolerate some database downtime. It can reduce overall writes to the database, which decreases the load and reduces costs, if the database provider charges by number of requests.

The main disadvantage is that if there's a cache failure, the data may be permanently lost.

Data Partitioning:

- A technique to break a big database (DB) into many smaller parts.
- It is the process of splitting up a DB/table across multiple machines to improve the manageability, performance, availability, and load balancing of an application
- It is more feasible to scale horizontally by adding more machines than to grow it vertically by adding beefier server

Partitioning Methods:

1. **Horizontal Partitioning:**
   - We put different rows into different tables
   - This is also called range-based Partitioning as we are storing different ranges of data in separate tables. Horizontal Partitioning is also known as Data Sharding
   - 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
2. **Vertical Partitioning:**
   - We divide our data to store tables related to a specific feature in their own server
   - Vertical Partitioning is straightforward to implement and has a low impact on the application
   - 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
3. **Directory-Based Partitioning:**

- A loosely coupled approach to work around issues mentioned in the above schemes is to **create a *lookup service*** that knows your current partitioning scheme and abstracts it away from the DB access code.
- So, to find out where a particular data entity resides, we query the directory server that holds the mapping between each tuple key to its DB server
- This loosely coupled approach means we can perform tasks like adding servers to the DB pool or changing our partitioning scheme without having an impact on the application

Original Table

CUSTOMER ID	FIRST NAME	LAST NAME	CITY
1	Alice	Anderson	Austin
2	Bob	Best	Boston
3	Carrie	Conway	Chicago
4	David	Doe	Denver

Vertical Shards

VS1

CUSTOMER ID	FIRST NAME	LAST NAME
1	Alice	Anderson
2	Bob	Best
3	Carrie	Conway
4	David	Doe

VS2

CUSTOMER ID	CITY
1	Austin
2	Boston
3	Chicago
4	Denver

Horizontal Shards

HS1

CUSTOMER ID	FIRST NAME	LAST NAME	CITY
1	Alice	Anderson	Austin
2	Bob	Best	Boston

HS2

CUSTOMER ID	FIRST NAME	LAST NAME	CITY
3	Carrie	Conway	Chicago
4	David	Doe	Denver

Partitioning Criteria:

1. **Key-based Sharding:**

- We apply a ***hash function*** to some key attributes of the entity we are storing; that yields the partition/sharding number. This approach should ensure a uniform allocation of data among servers
- Disadvantage is that it effectively fixes the total number of DB servers, since adding new servers means changing the hash function which would require redistribution of data and downtime for the service
- The Shard key and primary key of a data base can be same/ not same.
- The column which will be hashed should be a non-changeable column so that it provides same hash everytime when hit.
- It can be a single column or multiple columns to produce a shard key.