Horizontal Scaling vs Vertical Scaling

	Horizontal Scaling (Scale Out)	**Vertical Scaling (Scale Up)**
Definition	Involves adding more nodes (servers) to a system to increase capacity.	Involves adding more power (CPU, RAM, storage) to an existing machine.
Details	Data is distributed across multiple servers. The load is balanced across multiple nodes, which can be complex to manage but provides high scalability.	The system performance is increased by adding more computing power, RAM, and disk space to the existing machine.
Benefits	High availability due to data distribution, easier to scale, potentially more cost-effective in the long run.	Simplicity in configuration and management, improved performance for single-threaded processes, consistency is easier to maintain.
Drawbacks	More complex to manage due to data distribution, potential consistency issues due to data propagation delay.	Limited by the maximum capacity of a single machine, potential downtime for upgrades, single point of failure.

Database Partitioning

Database Partitioning:

Database partitioning enhances the performance, manageability, and scalability of databases. It revolves around subdividing a database into smaller segments called partitions. Each partition contains a subset of data, which can be stored on separate servers or storage devices. This approach simplifies data management, optimizes query performance, and streamlines maintenance.

Vertical Partitioning:

Vertical partitioning involves dividing data into subsets based on attributes or columns. These attribute-focused subsets are created to optimize storage and query performance. By selecting specific attributes for each partition, vertical partitioning minimizes resource usage and maximizes query efficiency.

Horizontal Partitioning/Sharding:

Horizontal partitioning entails segmenting data into subsets based on predefined criteria such as ranges, attribute values, or other logical divisions. Each subset contains a portion of data that aligns with the chosen criteria. This method simplifies data distribution, enhances scalability, and improves overall system performance.

Logical Sharding vs Physical Sharding

	Logical Sharding	**Physical Sharding**
Definition	Data is split into discrete shards based on a logical condition, but all shards may reside on the same physical server.	Each shard is stored on a separate physical server or machine.
Benefits	Easier to manage and more flexible, improves query efficiency.	Allows for greater scalability, can handle larger data volumes and higher load.
Drawbacks	Limited by the capacity and performance of a single machine.	More complex to manage, involves dealing with network latency, failures, and data consistency.
Best Used When	The size of the database is manageable within a single server and the main goal is to improve efficiency of data management.	The size of the database exceeds the storage capacity of a single machine, or the workload needs to be distributed across multiple machines.

Algorithmic Sharding vs Dynamic Sharding

Attribute	**Algorithmic Sharding**	**Dynamic Sharding**
Definition	Uses a consistent algorithm to determine where data should go.	Adapts to changes in data distribution and load.
Examples	Range-based Sharding, Hash-based Sharding	Directory-based Sharding, Geolocation-based Sharding
Benefits	Straightforward and fast, easy to determine where data is located.	Flexibility in adding/removing shards, better scalability
Drawbacks	Inflexible, difficult to add or remove shards.	Complexity, potential consistency issues.
Best Used When	The number of shards is stable and the distribution of data is relatively uniform.	The distribution of data changes frequently, or there are requirements for more granular control.

Different Strategies for Sharding

Strategy	Description	Example Use-case
Range-based Sharding	Data is distributed based on a certain range of a key.	User IDs: Users with IDs 1 to 1000 might go to Shard 1, IDs 1001 to 2000 to Shard 2, and so on.
Hash-based Sharding	A hash function is applied to a certain data key, and the output of the function determines the shard.	Usernames: A hash function applied to usernames could distribute them evenly across shards.
Directory-based Sharding	A lookup table or service keeps track of where data is stored.	Document Database: A lookup table can keep track of which documents are stored in which shards.
Attribute-based Sharding	Data is sharded based on specific attributes of the data.	Geographic Regions: Data could be sharded based on the geographic region of users to optimize data locality.
Random Sharding	Data is distributed across shards randomly.	High write throughput scenarios: When the write distribution needs to be even and quick, and there's less emphasis on read speed or complex queries.

Consistent Hashing

- **Purpose**: Consistent hashing is a technique used in distributed systems to efficiently distribute data across multiple servers while minimizing data movement when servers are added or removed.
- **Hash Ring:** It employs a virtual "hash ring," which is a circle with many equally spaced points. Each point represents a server or a virtual server.
- **Data Mapping:** Data is hashed (using a hash function) to produce a numeric value. This numeric value is placed on the hash ring to determine the server responsible for storing or retrieving the data.
- **Virtual Servers:** Virtual servers are multiple "imaginary" representations of physical servers. Each physical server is associated with multiple virtual servers, spread around the hash ring. This enhances load balancing and fault tolerance.
- **Load Balancing:** Distributes data uniformly across servers, preventing overloading of specific servers and achieves even data distribution.
- **Fault Tolerance:** Virtual servers and data replication ensure fault tolerance. If a server fails, its data is still accessible through replicas on neighboring servers.
- **Scalability:** When servers are added, only a fraction of data needs to be remapped due to the consistent hashing properties.
- **Adding/Removing Servers:** When a server is added or removed, minimal data needs to be redistributed. Only data assigned to a specific server or its virtual servers is affected.

- **O(log N) Complexity:** Achieves average O(log N) time complexity for locating the responsible server, where N is the number of servers. Sorted ring and uniform hash function distribution enable this efficiency.
- **Data Replication:** Replicates data on neighboring servers for redundancy and fault tolerance. Neighboring servers in the clockwise direction typically store replicas.
- **Wrap-around Property:** The circular nature of the ring simplifies operations and minimizes data remapping when servers are added or removed.
- **Applications:** Used in Distributed Hash Tables (DHTs), load balancing, Content Delivery Networks (CDNs), distributed caching, and sharding in databases.
- **Benefits:** Efficient data distribution, fault tolerance, load balancing, and scalability make it vital in large-scale, distributed systems.
- **Challenges:** Choosing an appropriate hash function and handling corner cases (e.g., data hotspots) require careful consideration.