Scalability

The Scalable System in Distributed System refers to the system in which there is a possibility of extending the system as the number of users and resources grows with time.

To exemplify, with the increasing number of users and workstations the frequency of file access is likely to increase in a distributed system. So, there must be some possibility to add more servers to avoid any issue in file accessing handling.

Scalability is generally considered concerning hardware and software. In hardware, scalability refers to the ability to change workloads by altering hardware resources such as processors, memory, and hard disc space. Software scalability refers to the capacity to adapt to changing workloads by altering the scheduling mechanism and parallelism level.

Scalability Framework

- Spring Framework
- JavaServer Faces(JSF)
- Struts
- Google Web Toolkit (GWT)

Measures of Scalability

Scalabiltiy of a system can be measured along at least three different dimensions.

- Size Scalability
- Geographical Scalability
- Administrative Scalability

1. Size Scalability: There will be an increase in the size of the system whenever users and
resources grow but it should not be carried out at the cost of performance and efficiency of the system. The system must respond to the user in the same manner as it was responding before scaling the system.

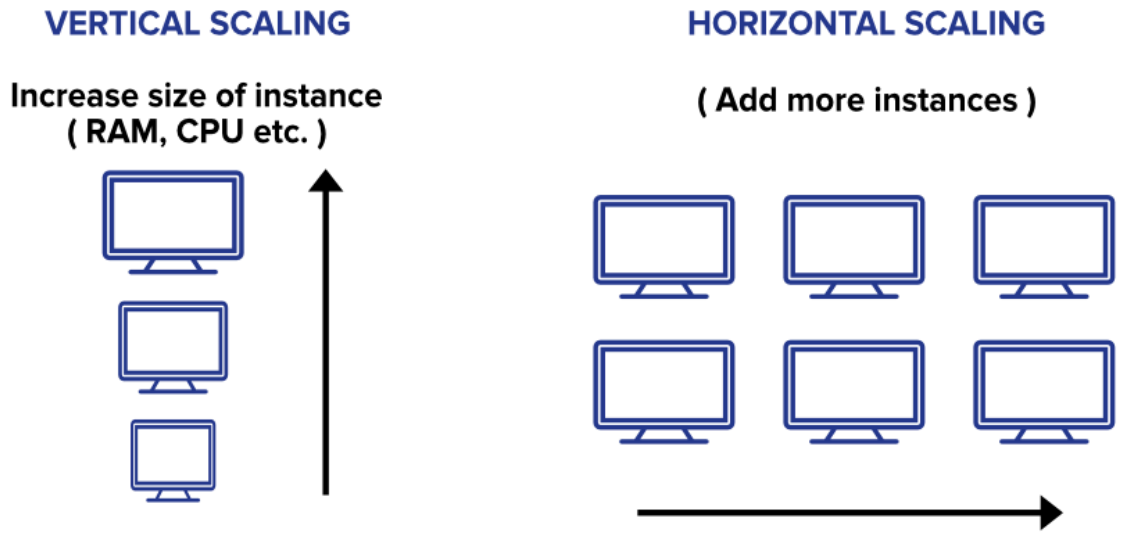
2. Geographical Scalability: Geographical scalability refers to the addition of new nodes in a physical space that should not affect the communication time between the nodes.

3. Administrative Scalability: adding more nodes should not increase the administrative costs of the system To exemplify, if there are multiple administrators in the system, the system is shared with others while in use by one of them.

Types of Scalability

1. Horizontal Scalability: Horizontal scalability implies the addition of new servers to the existing set of resources in the system. The major benefit lies in the scaling of the system dynamically. For example, Cassandra and MongoDB. Horizontal scaling is done in them by adding more machines. Furthermore, the load balancer is employed for distributing the load on the available servers
which increases overall performance.

2. Vertical Scalability: Vertical Scalability refers to the addition of more power to the existing pool
of resources like servers. For example, MySQL. Here, scaling is carried out by switching from smaller to bigger machines.



Horizontal Scaling	Vertical Scaling
When new server racks are added to the existing system to meet the higher expectation, it is known as horizontal scaling.	When new resources are added in the existing system to meet the expectation, it is known as vertical scaling
It is easier to upgrade.	It is harder to upgrade and may involve downtime.
It is difficult to implement	It is easy to implement
It is costlier, as new server racks comprise a lot of resources	It is cheaper as we need to just add new resources
It takes more time to be done	It takes less time to be done
High resilience and fault tolerance	Single point of failure
Examples of databases that can be easily scaled- Cassandra, MongoDB, Google Cloud Spanner	Examples of databases that can be easily scaled- MySQL, Amazon RDS

Horizontal Scaling in MongoDB - https://www.geeksforgeeks.org/scaling-in-mongodb/

Techniques for Scaling

1. Hide communication latencies
   ▶ Make use of asynchronous communication
   ▶ do other when waiting for a response from the server
   ▶ Have a separate handler for incoming response
   ▶ Problem: not every application fits this model
2. Facilitate solution by moving computations to client
3. Partition data and computations across multiple machines
   ▶ Move computations to clients (Java applets)
   ▶ Decentralized naming services (DNS)
   ▶ Decentralized information systems (WWW)
4. Replication and caching: Make copies of data available at different machines
   ▶ Replicated file servers and databases

▶ Mirrored Web sites
▶ Web caches (in browsers and proxies)
▶ File caching (at server and client)

Replication comes with a problem,

▶ Having multiple copies (cached or replicated), leads to inconsistencies: modifying one copy makes that copy different from the rest.
▶ Always keeping copies consistent and in a general way requires global synchronization on each modification.
▶ Global synchronization precludes large-scale solutions.

If we can tolerate inconsistencies, we may reduce the need for global synchronization, but tolerating inconsistencies is application dependent.