## Definition

A Distributed Computing System (DCS) is a collection of processors interconnected by a communication network in which each processor has its own local memory and other peripherals and communication between any two processors of the system takes place by message passing over a communication network (i.e. for a particular processor, its own resources are local).

# Why Distributed Systems Are Becoming Popular

1. **Inherently Distributed Applications:** Many applications are by nature, inherently distributed thus requiring distributed computing for their realization e.g. for collecting, preprocessing and accessing data. Examples include computerized worldwide airline reservation system, banking system, loaning system etc.
2. **Information Sharing Among Distributed Users:** there is desire for efficient person-to-person communication facility by sharing information over great distances e.g. two far-off users can work on the same project.
3. **Resource Sharing:** resources both h/w and s/w can be shared.
4. **Better Price-Performance Ratio:** they have this quality as compared to the centralized systems due to the rapidly increasing power and reduction in the price of microprocessors together with the increasing speed of communication networks. They facilitate resource sharing among multiple computers.
5. **Shorter response time and higher throughput:** They have better performance due to multiple processors as compared to single-processor centralized systems.
6. **Higher reliability:** due to the multiplicity of processors and storage devices, multiple copies of critical information is maintained and redundancy achieved. Also geographical distribution limits scope of failures caused by national disasters. Its important aspect is “availability” i.e. the fraction of time a system is available for use.
7. **Extensibility and Incremental growth:** they are capable of incremental growth i.e. additional resources both s/w and h/w can be added. Distributed systems with these qualities are referred to as “open distributed systems.”
8. **Better flexibility in meeting users’ needs:** a distributed system may have a pool of different types of computers, so that the most appropriate one can be selected for processing a user’s job depending on the nature of the job.

# Q: What is A Distributed Operating System (DOS)?

**Definition:** *“ Is one that looks to its users like an ordinary centralized OS but runs on multiple, independent CPUs.” (Tanenbaum and Van Renesse [1985])*

The OS commonly used for DCS are broadly classified into two namely “network OS” and “distributed OS”. Three features that differentiate them are:

* + **System image** – in a network OS, users view the DCS as a collection of distinct machines connected by a communication subsystem i.e. users are aware that multiple computers are being used, but DOS hides the existence of multiple computers and provides a single-system image to its users as a “virtual uniprocessor.”
  + **Autonomy** – for NOS, each computer has its own local OS and functions independent of others except when they must intercommunicate whereby they must use a mutually agreed on communication protocol. For a DOS, there is a single system-wide OS and each computer runs a part of this global OS. Processes and resources are managed globally and there is a single set of system calls (globally valid) available on all computers of the DCS.
  + **Fault tolerance capability** – this is usually high for a DOS than a NOS e.g. a 10% loss in NOS affects 10% users but in DOS only 10% loss in performance is experienced,

**Issues in Designing a DOS**

A DOS is more difficult to design than a centralized OS (COS) for several reasons e.g. in COS, it is assumed that the OS has access to complete and accurate information about the environment in which it is functioning but DOS must be designed with the assumption that complete information will never be available i.e. resources are physically separated, there are no common clocks among the multiple processors, delivery of messages is delayed and messages could even be lost.

The following therefore are some of the key design issues:

1. **Transparency:** One main aim of DOS is to display the existence of multiple computers as invisible (transparent) and provide a single system image to users (virtual uniprocessor). The eight forms of transparency include:
   1. **Access transparency –** users should not need or be able to recognize whether a resource, both h/w and s/w is remote or local i.e. the user interface should not distinguish between remote and local resources.
   2. **Location transparency –** which has two aspects, *name transparency*, i.e. the name of a resource s/w and h/w should not reveal any hint as to the physical location of the resource i.e. name should be independent of the physical connectivity/topology or the current location of the resource, and must be unique system wide. *User mobility*, i.e. no matter which machine a user is logged onto, he/she should be able to access a resource with the same name.
   3. **Replication transparency –** All DOS have the provision to create replicas of files and other resources on different nodes and both the existence of multiple copies of a replicated resource and the replication activity should be transparent to users.
   4. **Failure transparency –** this should mask from the users, partial failures of the system, e.g. a communication link failure, a machine failure or a storage device crash. The DOS should continue to function during partial failures, probably in a degraded form.
   5. **Migration transparency –** aims at ensuring that the movement of an object (e.g. processor or file) is handled automatically by the system in a user-transparent manner. To achieve this objective, the following three issues must be looked into:
      1. Migration decisions as to which object is to be moved should be made automatically by the system
      2. Migration of an object should not require any change in its name
      3. When the migrating object is a process, the interprocess communication mechanisms should ensure that a message sent to the migrating process reaches it without the need for the sender process to resend it if the receiver process moves to another node before the message is received.
   6. **Concurrency transparency –** allows each user to feel that he/she is the sole user to the system. To achieve this, the resource sharing mechanisms of the DOS must display the following four properties:
      1. An event-ordering property ensures that all access requests to various system resources are properly ordered to provide a consistent view to all users of the system.
      2. A mutual-exclusion property ensures that at any time at most one process accesses a shared resource, which must not be used simultaneously by multiple processes if program operation is to be correct.
      3. A non-starvation property ensures that if every process that is granted a resource, which must not be used simultaneously by multiple processes, eventually releases it, every request for that resource is eventually granted.
      4. A non-deadlock property ensures that a situation will never occur in which competing processes prevent their mutual progress even though no single one requests more resources than available in the system.
   7. **Performance transparency** – allows the system to be automatically reconfigured to improve performance, as loads vary dynamically in the system e.g. process must be evenly distributed with jobs.
   8. **Scaling transparency** – allows the system to expand in scale without disrupting the activities of the users
2. **Reliability:** is possible in DOS due to the existence of multiple instances of resources. A fault may occur (either mechanical or algorithmic) generating an error. System failures could either be *fail-stop* where system stops functioning after changing to a state in which a failure can be detected, or *Byzantine failure* where system continues to function but produces wrong results.

To achieve higher reliability, the fault handling mechanism must be designed to avoid faults, tolerate faults and detect and recover from faults as explained below.

* 1. **Fault avoidance** – deals with designing the components of the system such that the occurrence of faults is minimized
  2. **Fault tolerance** – is the ability of a system to continue functioning in the event of partial failure. This can be achieved through
     1. **Redundancy techniques** – which avoids single points of failure by replicating critical h/w and s/w components, such that if one fails, the others can be used to continue.
     2. **Distributed control** – many of the particular algorithms or protocols used in a DOS must employ a distributed control mechanism to avoid single points of failure.
     3. **Fault detection and recovery** – is the use of h/w and s/w mechanisms to determine the occurrence of a failure and then to correct the system to a state acceptable for continued operation. Techniques include:\
        1. **Atomic transactions** – is a computation consisting of a collection of operations that take place indivisibly in the presence of failures and concurrent computations.
        2. **Stateless servers** – the stateless service paradigm makes crash recovery very easy because no client state information is maintained by the server.
        3. **Acknowledgements and timeout-based retransmissions of messages** – the receiver must return an acknowledgement message for every message received, and if sender does not receive any acknowledgement for a message within a fixed timeout period, it assumes that the message was lost and retransmits.

1. **Flexibility:** allows the following to be achieved:
   1. **Ease of modification** – incorporating changes in a user-transparent manner or with minimum interruption on users.
   2. **Ease of enhancements** – e.g. adding new functionalities from time to time to make it more powerful and easy to use.
2. **Performance:** its performance must be at least, as good as that for a centralized system. Design principles considered include:
   1. **Batch if possible** – e.g. transferring large chunks of data at once instead of individual pages.
   2. **Cache whenever possible** – makes data available whenever it is currently being used thereby saving time and bandwidth. Reduces contention on centralized systems.
   3. **Minimize copying of data** – e.g. moving data in and out of buffers.
   4. **Minimize network traffic** – e.g. migrating a process closer to the resources it is using most heavily.
   5. **Take advantage of fine-grain parallelism for multiprocessing** –e.g. using threads for server processes thereby allowing the servers to simultaneously service requests from several clients.
3. **Scalability**: is the capability of a system to adapt to increased service load. Guiding principles for item design include:
   1. **Avoid centralized entities:-** use of centralized entities e.g. a single central file server or database for the entire system makes the DS non-scalable due to:
      1. Failure of centralized entity brings down whole system
      2. Performance becomes a system bottleneck when contention for it increases with increasing number of users.
      3. Capacity of the network that connects the centralized entity with other nodes of the system often gets saturated when the contention for the entity increases beyond a certain level.
      4. In a WAN compromising several LANS, it is inefficient to always get a particular type of request serviced at a server node that is several gateways away.
   2. **Avoid central algorithms**: centralized algorithms operate by collecting information from all nodes, processing it on a single node and then distributes the result to other nodes. It suffers similar problems mentioned above.
   3. **Perform most operations on client workstations:** -Perform client operations on the workstation rather than on a server machine. Server is common resource and so its cycles are more precious. It allows graceful degradation of the system performance as it grows in size by reducing contention for shared resources.
   4. **Heterogeneity**:- A heterogeneous DS consists of interconnected sets of dissimilar hardware or software system. It allows users flexibility to use various platforms for their work.
   5. **Security**:- Enforcing it is difficult due to lack of single point of control and the use of insecure networks for data communication. Since the client–server model is often used for requesting and providing services, when a client sends a request message to a server, the server must have some way of knowing whom the client is, which is not that simple. To enforce security hence requires the following additional requirements, as compared to a centralized system:-
      1. It should be possible for the sender of the message to know that the message was received by the intended receiver.
      2. It should be possible for the receiver to know that the message was sent by the genuine sender.
      3. It should be possible for both the sender and receiver to be guaranteed that the contents of the message were not changed while it was in transit.
   6. **Emulation of Existing Operation System**:- For commercial success, its vital that a newly designed DOS be able to emulate existing popular OS e.g. Unix so that, new software can be written using the system call interface of the new OS to take full advantage of its special features of distribution, while old software can also be run on without rewriting them.

**CLUSTERED COMPUTING**

A **computer cluster** is a set of loosely or tightly connected [computers](https://en.wikipedia.org/wiki/Computers) that work together so that, in many respects, they can be viewed as a single system. Unlike [grid computers](https://en.wikipedia.org/wiki/Grid_computing), computer clusters have each [node](https://en.wikipedia.org/wiki/Node_(computer_science)) set to perform the same task, controlled and scheduled by software.

The components of a cluster are usually connected to each other through fast [local area networks](https://en.wikipedia.org/wiki/Local_area_network), with each *node* (computer used as a server) running its own instance of an [operating system](https://en.wikipedia.org/wiki/Operating_system).

In most circumstances, all of the nodes use the same hardware and the same operating system, although in some setups (e.g. using [Open Source Cluster Application Resources](https://en.wikipedia.org/wiki/Open_Source_Cluster_Application_Resources) (OSCAR)), different operating systems can be used on each computer, or different hardware.

Clusters are usually deployed to improve performance and availability over that of a single computer, while typically being much more cost-effective than single computers of comparable speed or availability.

Computer clusters emerged as a result of convergence of a number of computing trends including the availability of low-cost microprocessors, high-speed networks, and software for high-performance [distributed computing](https://en.wikipedia.org/wiki/Distributed_computing).

They have a wide range of applicability and deployment, ranging from small business clusters with a handful of nodes to some of the fastest [supercomputers](https://en.wikipedia.org/wiki/Supercomputer) in the world such as [IBM's Sequoia](https://en.wikipedia.org/wiki/IBM_Sequoia).

The desire to get more computing power and better reliability by orchestrating a number of low-cost [commercial off-the-shelf](https://en.wikipedia.org/wiki/Commercial_off-the-shelf) computers has given rise to a variety of architectures and configurations.

The computer clustering approach usually (but not always) connects a number of readily available computing nodes (e.g. personal computers used as servers) via a fast [local area network](https://en.wikipedia.org/wiki/Local_area_network).

The activities of the computing nodes are orchestrated by "clustering middleware", a software layer that sits atop the nodes and allows the users to treat the cluster as by and large one cohesive computing unit, e.g. via a [single system image](https://en.wikipedia.org/wiki/Single_system_image) concept.

Computer clustering relies on a centralized management approach which makes the nodes available as orchestrated shared servers. It is distinct from other approaches such as [peer to peer](https://en.wikipedia.org/wiki/Peer_to_peer) or [grid computing](https://en.wikipedia.org/wiki/Grid_computing) which also use many nodes, but with a far more [distributed nature](https://en.wikipedia.org/wiki/Distributed_computing).

A computer cluster may be a simple two-node system which just connects two personal computers, or may be a very fast [supercomputer](https://en.wikipedia.org/wiki/Supercomputer). A basic approach to building a cluster is that of a [Beowulf](https://en.wikipedia.org/wiki/Beowulf_(computing)) cluster which may be built with a few personal computers to produce a cost-effective alternative to traditional [high performance computing](https://en.wikipedia.org/wiki/High_performance_computing).

An early project that showed the viability of the concept was the 133-node [Stone Soupercomputer](https://en.wikipedia.org/wiki/Stone_Soupercomputer). The developers used [Linux](https://en.wikipedia.org/wiki/Linux), the [Parallel Virtual Machine](https://en.wikipedia.org/wiki/Parallel_Virtual_Machine) toolkit and the [Message Passing Interface](https://en.wikipedia.org/wiki/Message_Passing_Interface) library to achieve high performance at a relatively low cost.

**Attributes of clusters**

Computer clusters may be configured for different purposes ranging from general purpose business needs such as web-service support, to computation-intensive scientific calculations. In either case, the cluster may use a [high-availability](https://en.wikipedia.org/wiki/High-availability_cluster) approach. Note that the attributes described below are not exclusive and a "computer cluster" may also use a high-availability approach, etc.

**"**[Load-balancing](https://en.wikipedia.org/wiki/Load_balancing_(computing))**"** clusters are configurations in which cluster-nodes share computational workload to provide better overall performance. For example, a web server cluster may assign different queries to different nodes, so the overall response time will be optimized.[[11]](https://en.wikipedia.org/wiki/Computer_cluster#cite_note-Sloan-11) However, approaches to load-balancing may significantly differ among applications, e.g. a high-performance cluster used for scientific computations would balance load with different algorithms from a web-server cluster which may just use a simple [round-robin method](https://en.wikipedia.org/wiki/Round-robin_scheduling) by assigning each new request to a different node.[[11]](https://en.wikipedia.org/wiki/Computer_cluster#cite_note-Sloan-11)

Computer clusters are used for computation-intensive purposes, rather than handling [IO-oriented](https://en.wikipedia.org/wiki/Input/output) operations such as web service or databases.[[12]](https://en.wikipedia.org/wiki/Computer_cluster#cite_note-VECPAR-12) For instance, a computer cluster might support [computational simulations](https://en.wikipedia.org/wiki/Computer_simulation) of vehicle crashes or weather. Very tightly coupled computer clusters are designed for work that may approach "[supercomputing](https://en.wikipedia.org/wiki/Supercomputing)".

**"**[High-availability clusters](https://en.wikipedia.org/wiki/High-availability_cluster)**"** (also known as [failover](https://en.wikipedia.org/wiki/Failover) clusters, or HA clusters) improve the availability of the cluster approach. They operate by having redundant [nodes](https://en.wikipedia.org/wiki/Node_(networking)), which are then used to provide service when system components fail. HA cluster implementations attempt to use redundancy of cluster components to eliminate [single points of failure](https://en.wikipedia.org/wiki/Single_point_of_failure). There are commercial implementations of High-Availability clusters for many operating systems. The [Linux-HA](https://en.wikipedia.org/wiki/Linux-HA) project is one commonly used [free software](https://en.wikipedia.org/wiki/Free_software) HA package for the [Linux](https://en.wikipedia.org/wiki/Linux) operating system.

Clusters are primarily designed with performance in mind, but installations are based on many other factors. Fault tolerance (*the ability for a system to continue working with a malfunctioning node*) allows for scalability, and in high performance situations, low frequency of maintenance routines, resource consolidation, and centralized management. Advantages include enabling data recovery in the event of a disaster and providing parallel data processing and high processing capacity.

In terms of scalability, clusters provide this in their ability to add nodes horizontally. This means that more computers may be added to the cluster, to improve its performance, redundancy and fault tolerance.

This can be an inexpensive solution for a higher performing cluster compared to scaling up a single node in the cluster. This property of computer clusters can allow for larger computational loads to be executed by a larger number of lower performing computers.

When adding a new node to a cluster, reliability increase because the entire cluster does not need to be taken down. A single node can be taken down for maintenance, while the rest of the cluster takes on the load of that individual node.

If you have a large number of computers clustered together, this lends itself to the use of [distributed file systems](https://en.wikipedia.org/wiki/Clustered_file_system#distributed_file_systems) and [RAID](https://en.wikipedia.org/wiki/RAID), both of which can increase the reliability, and speed of a cluster.

**GRID COMPUTING**

**Grid computing** is the use of widely distributed computer resources to reach a common goal. The **grid** can be thought of as a [distributed system](https://en.wikipedia.org/wiki/Distributed_system) with non-interactive workloads that involve a large number of files.

Grid computing is distinguished from conventional high-performance computing systems such as [cluster](https://en.wikipedia.org/wiki/Cluster_(computing)) computing in that grid computers have each node set to perform a different task/application. Grid computers also tend to be more [heterogeneous](https://en.wikipedia.org/wiki/Heterogeneous) and geographically dispersed (thus not physically coupled) than cluster computers.

Although a single grid can be dedicated to a particular application, commonly a grid is used for a variety of purposes. Grids are often constructed with general-purpose grid [middleware](https://en.wikipedia.org/wiki/Middleware) software libraries. ***Grid sizes can be quite large***.

Grids are a form of [distributed computing](https://en.wikipedia.org/wiki/Distributed_computing) whereby a "**super virtual computer**" is composed of many networked [loosely coupled](https://en.wikipedia.org/wiki/Loose_coupling) computers acting together to perform large tasks.

For certain applications, distributed or grid computing can be seen as a special type of [parallel computing](https://en.wikipedia.org/wiki/Parallel_computing) that relies on complete computers (with onboard CPUs, storage, power supplies, network interfaces, etc.) connected to a [computer network](https://en.wikipedia.org/wiki/Computer_network) (private or public) by a conventional [network interface](https://en.wikipedia.org/wiki/Network_interface_controller), such as [Ethernet](https://en.wikipedia.org/wiki/Ethernet). This is in contrast to the traditional notion of a [supercomputer](https://en.wikipedia.org/wiki/Supercomputer), which has many processors connected by a local high-speed [computer bus](https://en.wikipedia.org/wiki/Computer_bus).

Grid computing combines computers from multiple administrative domains to reach **a common goal**,**to solve a single task**, and may then disappear just as quickly.

The size of a grid may vary from small—confined to a network of computer workstations within a corporation, for example—to large, public collaborations across many companies and networks. "The notion of a confined grid may also be known as an intra-nodes cooperation whilst the notion of a larger, wider grid may thus refer to an inter-nodes cooperation".

Grids are a form of [distributed computing](https://en.wikipedia.org/wiki/Distributed_computing) whereby a “super virtual computer” is composed of many networked [loosely coupled](https://en.wikipedia.org/wiki/Loose_coupling) computers acting together to perform very large tasks.

This technology has been applied to computationally intensive scientific, mathematical, and academic problems through [volunteer computing](https://en.wikipedia.org/wiki/Volunteer_computing), and it is used in commercial enterprises for such diverse applications as [drug discovery](https://en.wikipedia.org/wiki/Drug_discovery), [economic forecasting](https://en.wikipedia.org/wiki/Economic_forecasting), [seismic analysis](https://en.wikipedia.org/wiki/Seismic_analysis), and [back office](https://en.wikipedia.org/wiki/Back_office) data processing in support for [e-commerce](https://en.wikipedia.org/wiki/E-commerce) and [Web services](https://en.wikipedia.org/wiki/Web_service).

Coordinating applications on Grids can be a complex task, especially when coordinating the flow of information across distributed computing resources. [Grid workflow](https://en.wikipedia.org/wiki/Scientific_workflow_system) systems have been developed as a specialized form of a workflow management system designed specifically to compose and execute a series of computational or data manipulation steps, or a workflow, in the Grid context.

“Distributed” or “grid” computing in general is a special type of [parallel computing](https://en.wikipedia.org/wiki/Parallel_computing) that relies on complete computers (with onboard CPUs, storage, power supplies, network interfaces, etc.) connected to a [network](https://en.wikipedia.org/wiki/Computer_network) (private, public or the [Internet](https://en.wikipedia.org/wiki/Internet)) by a conventional [network interface](https://en.wikipedia.org/wiki/Network_interface_controller) producing commodity hardware, compared to the lower efficiency of designing and constructing a small number of custom supercomputers.

The primary performance disadvantage is that the various processors and local storage areas do not have high-speed connections. This arrangement is thus well-suited to applications in which multiple parallel computations can take place independently, without the need to communicate intermediate results between processors.

The high-end [scalability](https://en.wikipedia.org/wiki/Scalability) of geographically dispersed grids is generally favorable, due to the low need for connectivity between [nodes](https://en.wikipedia.org/wiki/Node_(computer_science)) relative to the capacity of the public Internet.

There are also some differences in programming and MC. It can be costly and difficult to write programs that can run in the environment of a supercomputer, which may have a custom operating system, or require the program to address [concurrency](https://en.wikipedia.org/wiki/Concurrency_(computer_science)) issues.

If a problem can be adequately parallelized, a “thin” layer of “grid” infrastructure can allow conventional, standalone programs, given a different part of the same problem, to run on multiple machines. This makes it possible to write and debug on a single conventional machine and eliminates complications due to multiple instances of the same program running in the same shared [memory](https://en.wikipedia.org/wiki/Computer_memory) and storage space at the same time.

**UTILITY COMPUTING**

**Utility computing**, or **The Computer Utility**, is a service provisioning model in which a service provider makes computing resources and infrastructure management available to the customer as needed, and charges them for specific usage rather than a flat rate.

Like other types of on-demand computing (such as grid computing), the utility model seeks to maximize the efficient use of resources and/or minimize associated costs. Utility is the packaging of [system resources](https://en.wikipedia.org/wiki/System_resource), such as computation, storage and services, as a metered service.

This model has the advantage of a low or no initial cost to acquire computer resources; instead, resources are essentially rented.

This repackaging of computing services became the foundation of the shift to "[on demand](https://en.wikipedia.org/wiki/Code_on_demand)" computing, [software as a service](https://en.wikipedia.org/wiki/Software_as_a_service) and [cloud computing](https://en.wikipedia.org/wiki/Cloud_computing) models that further propagated the idea of computing, application and network as a service.

There was some initial skepticism about such a significant shift. However, the new model of computing caught on and eventually became mainstream.

IBM, HP and Microsoft were early leaders in the new field of utility computing, with their business units and researchers working on the architecture, payment and development challenges of the new computing model. Google, Amazon and others started to take the lead in 2008, as they established their own utility services for computing, storage and applications.

Utility computing can support grid computing which has the characteristic of very large computations or sudden peaks in demand which are supported via a large number of computers.

"Utility computing" has usually envisioned some form of [virtualization](https://en.wikipedia.org/wiki/Platform_virtualization) so that the amount of storage or computing power available is considerably larger than that of a single [time-sharing](https://en.wikipedia.org/wiki/Time-sharing) computer.

Multiple servers are used on the "back end" to make this possible. These might be a dedicated [computer cluster](https://en.wikipedia.org/wiki/Computer_cluster) specifically built for the purpose of being rented out, or even an under-utilized [supercomputer](https://en.wikipedia.org/wiki/Supercomputer). The technique of running a single calculation on multiple computers is known as [distributed computing](https://en.wikipedia.org/wiki/Distributed_computing).

The term "[grid computing](https://en.wikipedia.org/wiki/Grid_computing)" is often used to describe a particular form of distributed computing, where the supporting nodes are geographically distributed or cross [administrative domains](https://en.wikipedia.org/wiki/Administrative_domain). To provide utility computing services, a company can "bundle" the resources of members of the public for sale, who might be paid with a portion of the revenue from clients.

One model, common among [volunteer computing](https://en.wikipedia.org/wiki/Volunteer_computing) applications, is for a central server to dispense tasks to participating nodes, on the behest of approved end-users (in the commercial case, the paying customers). Another model, sometimes called the [Virtual Organization](https://en.wikipedia.org/wiki/Virtual_Organization_(Grid_computing)) (VO), is more decentralized, with organizations buying and selling [computing resources](https://en.wikipedia.org/wiki/Computational_resource) as needed or as they go idle.