***Introduction:***

*A distributed system is a collection of physically separate, possibly heterogeneous, computer systems (nodes) that are networked to provide the users with access to the various resources that the system maintains.* From the point of view of a specific node in a distributed system, the rest of the nodes and their respective resources are remote, whereas its own resources are local.

Major reasons for building **distributed systems:**

* **resource sharing**

If a number of different sites (with different capabilities) are connected to one another, then a user at one site may be able to use the resources available at another

* **computational speedup**

If a particular computation can be partitioned into subcomputations that can run concurrently, then a distributed system allows us to distribute the subcomputations among the various sites. The subcomputations can be run concurrently and thus provide computation speedup.

* **Reliability**

If one site fails in a distributed system, the remaining sites can continue operating, giving the system better reliability

**World Wide Web has many aspects of a distributed computing environment. Certainly it provides data migration (between a web server and a web client). It also provides computation migration. For instance, a web client could trigger a database operation on a web server. Finally, with Java, Javascript, and similar languages, it provides a form of process migration: Java applets and Javascript scripts are sent from the server to the client, where they are executed. A network operating system provides most of these features, but a distributed operating system makes them seamless and easily accessible. The result is a powerful and easy-to-use facility —one of the reasons for the huge growth of the World Wide Web.**

Distinguish between the client–server and peer-to-peer models of distributed systems.

Answer: The client-server model firmly distinguishes the roles of the client and server. Under this model, the client requests services that are provided by the server. The peer-to-peer model doesn’t have such strict roles. In fact, all nodes in the system are considered peers and thus may act as either clients or servers—or both. A node may request a service from another peer, or the node may in fact provide such a service to other peers in the system. For example, let’s consider a system of nodes that share cooking recipes. Under the client-server model, all recipes are stored with the server. If a client wishes to access a recipe, it must request the recipe from the specified server. Using the peer-to-peer model, a peer node could ask other peer nodes for the specified recipe. The node (or perhaps nodes) with the requested recipe could provide it to the requesting node. Notice how each peer may act as both a client (it may request recipes) and as a server (it may provide recipes).

**Network operating system (NOS):**

**What?SSSSSSSSSSSSSSSSSSSSSSSSS**

Network operating systems are simpler to implement but generally more difficult for users to access and use than are distributed operating systems, which provide more features. Currently, all general-purpose operating systems, and even embedded operating systems such as Android and iOS, are network operating systems.

Distributed operating systems are designed to get rid of required command change when user login other machine remotely. An important point about SSH, FTP, and cloud-based storage applications is that they require the user to change paradigms. FTP, for example, requires the user to know a command set entirely different from the normal operating system commands. With SSH, the user must know appropriate commands on the remote system. For instance, a user on a Windows machine who connects remotely to a UNIX machine must switch to UNIX commands for the duration of the SSH session. With cloud-based storage applications, users may have to log into the cloud service (usually through a web browser) or native application and then use a series of graphical commands to upload, download, or share files. In a distributed operating system, users access remote resources in the same way they access local resources. Access to a shared resource can be provided by data migration, computation migration, or process migration. Distributed operating systems host control for the following:

* Data migration:
  1. to transfer the entire file to local site. When the user no longer needs access to the file, a copy of the file (if it has been modified) is sent back to remote site. This approach was used in the Andrew file system, but it was found to be too inefficient
  2. file is divided into portion. only those portions of the file that are actually necessary for the immediate task will be transferred. any part of it that has been modified must be sent back to the remote site.
* computation migration:

to transfer the computation, rather than the data, across the system. It called RPC might result in the invocation of another RPC or even in the transfer of messages to another site.

process migration:

A logical extension of computation migration is process migration. When a process is submitted for execution, it is not always executed at the site at which it is initiated, examples for process migration:

* Load balancing
* Computation speedup

**Distributed System Issues:**

A distributed system may suffer from various types of hardware failure. The failure of a link, a host, or a site and the loss of a message are the most common types. To ensure that the system is robust, we must detect any of these failures, reconfigure the system so that computation can continue, and recover when the failure is repaired.

Making the multiple processors and storage devices in a distributed system transparent to the users has been a key challenge to many designers. Ideally, a distributed system should look to its users like a conventional, centralized system. The user interface of a transparent distributed system should not distinguish between local and remote resources

scalability— the capability of a system to adapt to increased service load. Systems have bounded resources and can become completely saturated under increased load. For example, with respect to a file system, saturation occurs either when a server’s CPU runs at a high utilization rate or when disks’ I/O requests overwhelm the I/O subsystem

Even perfect design however cannot accommodate an ever-growing load. Adding new resources might solve the problem, but it might generate additional indirect load on other resources (for example, adding machines to a distributed system can clog the network and increase service loads). As the amount of data, I/O workload, and processing expands, so does the need for a DFS to be fault-tolerant and scalable.

**The Distributed File System (DFS)** whose clients, servers, and storage devices are dispersed among the machines of a distributed system. The most important performance measure of a DFS is the amount of time needed to satisfy service requests

The basic architecture of a DFS depends on its ultimate goals. Two widely used architectural models we discuss here are the client–server model and the cluster-based model. The main goal of a client–server architecture is to allow transparent file sharing among one or more clients as if the files were stored locally on the individual client machines. The distributed file systems NFS and OpenAFS are prime examples.

Large bottlenecks cannot be tolerated, and system component failures must be expected. Cluster-based architecture was developed in part to meet these needs. presented by the Google file system (GFS) and the Hadoop distributed fil system (HDFS)

There are two main types of DFS models: the client–server model and the cluster-based model. The client-server model allows transparent file sharing among one or more clients. The cluster-based model distributes the files among one or more data servers and is built for large-scale parallel data processing.