A distributed system is a collection of autonomous computers that are connected through a network and

distribution software called middleware, which enables computers to coordinate their activities and to

share the resources of the system. A distributed system’s users perceive the system as a single integrated

computing facility.

A distributed system has several characteristics: Its components are autonomous, scheduling and

other resource management and security policies are implemented by each system, there are multiple

points of control and multiple points of failure, and the resources may not be accessible at all times.

Distributed systems can be scaled by adding additional resources and can be designed to maintain

availability even at low levels of hardware/software/network reliability.

Distributed systems have been around for several decades. For example, distributed file systems and

network file systems have been used for user convenience and to improve reliability and functionality of

file systems for many years. Modern operating systems allow a user to mount a remote file system and

access it the same way a local file system is accessed, yet with a performance penalty due to larger communication

costs. The remote procedure call (RPC) supports inter-process communication and allows a

procedure on a system to invoke a procedure running in a different address space, possibly on a remote

system. RPCs were introduced in the early 1970s by Bruce Nelson and used for the first time at Xerox;

the Network File System (NFS) introduced in 1984 was based on Sun’s RPC. Many programming languages

support RPCs; for example, Java Remote Method Invocation (Java RMI) provides a functionality

similar to that of UNIX RPC methods, and XML-RPC uses XML to encode HTML-based calls.

The middleware should support a set of desirable properties of a distributed system:

• Access transparency. Local and remote information objects are accessed using identical operations.

• Location transparency. Information objects are accessed without knowledge of their location.

• Concurrency transparency. Several processes run concurrently using shared information objects

without interference among them.

• Replication transparency. Multiple instances of information objects are used to increase reliability

without the knowledge of users or applications.

• Failure transparency. The concealment of faults.

• Migration transparency. The information objects in the system are moved without affecting the

operation performed on them.

• Performance transparency. The system can be reconfigured based on the load and quality of service

requirements.

• Scaling transparency. The system and the applications can scale without a change in the system

structure and without affecting the applications.