CHAPTER ONE

1. CHARACTERIZATION OF DISTRIBUTED SYSTEMS

1.1. Introduction

We know that, computer systems are undergoing a revolution. From 1945 until about 1985, computers were large and expensive that costs at least tens of thousands of dollars each. They were slow in processing instructions; and also there was lack of a way to connect them. They operated independently from one another.

However, after around the mid-1980s, two major developments in technology began to change that situation. The first was the development of cheap and powerful microprocessor-based computers. Initially, these were 8-bit machines, but soon 16-, 32-, and 64-bit CPUs became common. The second development was the invention of high-speed computer networks. Local-area networks (LANs) allow hundreds of machines within a building to be connected so that larger amounts of data can be moved between machines at rates of 100 million to 10 billion bits/sec. In addition, Wide-area networks (WANs) allow millions of machines all over the earth to be connected at speeds varying from 64 Kbps (kilobits per second) to gigabits per second.

The result of these technologies is that it is feasible to put together computing systems composed of large numbers of computers connected by a high-speed network to work for the same application. This is in contrast to the old centralized systems (or single processor systems) where there was single computer with its peripherals.

1.2. Definition of Distributed System

Definition 1:

> "A distributed system is a collection of independent computers that appears to its users as a single coherent system."*Andrew S. Tanenbaum*

This definition has two important aspects. The first one is that a distributed system consists of components that are autonomous (i.e. autonomous computers). A second aspect is that users (be the people or programs) think they are dealing with a single system (i.e. single system view for users). This means that one way or the other the autonomous components need to collaborate. Establishing this collaboration lies at the heart of developing distributed systems. Note that no assumptions are made concerning the type of computers.

Definition 2:

> "A distributed system is a system in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages." *Coulouris, et al*

This definition leads to the following especially significant characteristics of distributed systems: concurrency of components, lack of a global clock and independent failures of components.

- ***Concurrency***: In a network of computers, concurrent program execution is a norm. I can do my work on my computer while you do your work on yours, sharing resources such as web pages or files when necessary.

- ***No global clock***: When programs need to cooperate they coordinate their actions by exchanging messages. Close coordination often depends on a shared idea of the time at which the programs' actions occur. But it turns out that there are limits to the accuracy with which the computers in a network can synchronize their clocks – there is no single global notion of the correct time. This is a direct consequence of the fact that the only communication is by sending messages through a network.

- ***Independent failures***: Faults in the network result in the isolation of the computers that are connected to it, but that doesn't mean that they stop running. In fact, the programs on them may not be able to detect whether the network has failed or has become unusually slow. Similarly, the failure of a computer, or the unexpected termination of a program somewhere in the system (a crash), is not immediately made known to the other components with which it communicates. Each component of the system can fail independently, leaving the others still running.

Broader Definition

A system designed to support the development of applications and services which can exploit a physical architecture consisting of multiple, autonomous processing elements that do not share primary memory but cooperate by sending asynchronous messages over a communication network.

In order to support heterogeneous computers and networks while offering a single system view, distributed systems are often organized by means of a layer of software that is, logically placed between a higher-level layer consisting of users and applications, and a layer underneath consisting of operating systems and basic communication facilities, as shown below.

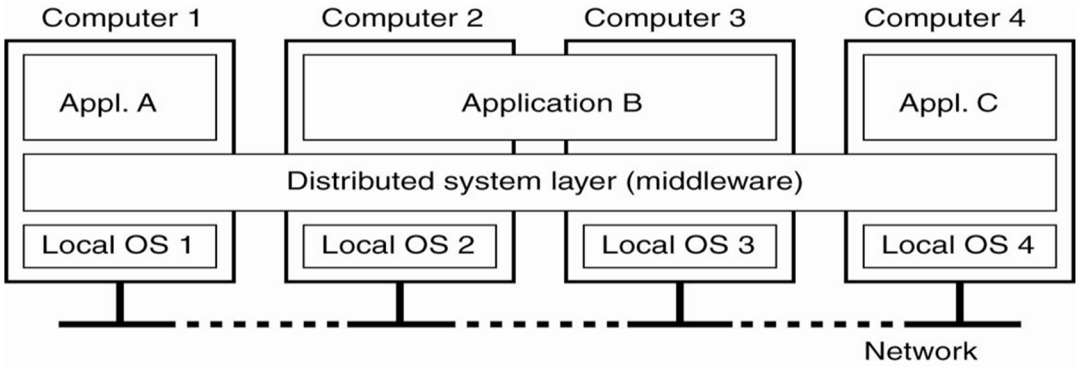

Figure 1.1. A distributed system organized as middleware

Note that the middleware layer extends over multiple machines, and offers each application the same interface. *(More on next chapter, Architecture & Communication)*

The above figure (Figure 1.1) shows four networked computers and three applications, of which application B is distributed across computers 2 and 3. Each application is offered the same interface. The distributed system provides the means for components of a single distributed application to communicate with each other, but also to let different applications communicate. At the same time, it hides, as best and reasonable as possible, the differences in hardware and operating systems from each application.

Challenges and Problems in Distribution

The challenges arising from the construction of distributed systems are the heterogeneity of their components, openness (which allows components to be added or replaced), security, scalability – the ability to work well when the load or the number of users increases – failure handling, concurrency of components, transparency and providing quality of service.

- ***Heterogeneity:*** Users access services and run applications over a heterogeneous collection of computers and networks. Heterogeneity (that is, variety and difference) applies networks, computer hardware, operating systems, programming languages, implementations by different developers.
- ***Openness:*** The openness of distributed systems is determined primarily by the degree to which new resource-sharing services can be added and be made available for use by a variety of client programs.
- ***Security:*** Security for information resources has three components: confidentiality (protection against disclosure to unauthorized individuals), integrity (protection against alteration or corruption), and availability (protection against interference with the means to access the resources).
- ***Scalable:*** A scalable distributed system remains effective when there is a significant increase in the number of resources and the number of users.
- ***Failure handling:*** Failures in a distributed system are partial – that is, some components fail while others continue to function. Therefore the handling of failures is particularly difficult like detecting failures, masking failures, tolerating failures and recovery from failures.
- ***Concurrency:*** There is a possibility that several clients can access a shared resource at the same time. Services and applications generally allow multiple client requests to be processed concurrently. These multiple requests may conflict with one another and produce inconsistent results.
- ***Transparency:*** Transparency is defined as the concealment from the user and the application programmer of the separation of components in a distributed system, so that the system is perceived as a whole rather than as a collection of independent components. The implications of transparency are a major influence on the design of the system software.
- ***Quality of service:*** Once users are provided with the functionality that they require of a service, such as the file service in a distributed system, we can go on to ask about the quality of the service provided. The main non-functional properties of systems that affect the quality of the service experienced by clients and users are reliability, security and performance. Adaptability to meet changing system configurations and resource availability has been recognized as a further important aspect of service quality.

1.3. Goals of Distributed System

This section emphasizes four important goals that should be met to make building a distributed system worth the effort. A distributed system should *make resources easily accessible*, it should reasonably *hide the fact that resources are distributed across a network*, it should be *open* and it should be *scalable*.

A. Making Resources Accessible

The main goal of a distributed system is to make it easy for the users (and applications) to access remote resources, and to share them in a controlled and efficient way. Resources can be just about anything, but typical examples include things like printers, computers, storage facilities, data, files, Web pages, and networks, to name just a few. There are many reasons for wanting to share resources. One obvious reason is that of ***economics***. It makes economic sense to share costly resources such as supercomputers, high-performance storage systems, printers, and other expensive peripherals. Connecting users and resources also makes it easier to collaborate and ***exchange information***.

❖*What could be the **limitations** while achieving this DSs goal?*

B. Distribution Transparency

An important goal of a distributed system is to hide the fact that its processes and resources are physically distributed across multiple computers. A distributed system that is able to present itself to users and applications as if it were only a single computer system is said to be transparent.

The concept of transparency can be applied to several aspects of a distributed system; the most important ones are the following:-

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed.
Location	Hide where a resource is physically located.
Migration	Hide that a resource may move to another location.
Relocation	Hide that a resource may be moved to another location while in use.
Replication	Hide that a resource is replicated.
Concurrency	Hide that a resource may be shared by several competitive users.
Failure	Hide the failure and recovery of a resource.

Table 1.1. Summary of different transparency concepts

Access transparency deals with hiding differences in data representation and the way that resources can be accessed by users. At a basic level, we wish to hide differences in machine architectures, but more important is that we reach agreement on how data is to be represented by different machines and operating systems. For example, a distributed system may have computer systems that run different operating systems, each having their own file-naming conventions. Differences in naming conventions as well as how files can be manipulated should all be hidden from users and applications.

Location transparency refers to the fact that users cannot tell where a resource is physically located in the system. Naming plays an important role in achieving location transparency. In

particular, location transparency can be achieved by assigning only logical names to resources, that is, names in which the location of a resource is not secretly encoded. An example of such a name is the URL http://www.prenhall.com/index.html which gives no clue about the location of Prentice Hall's main Web server.

The URL also gives no clue as to whether *index.html* has always been at its current location or was recently moved there. Distributed systems in which resources can be moved without affecting how those resources can be accessed are said to provide ***migration transparency***.

Even stronger is the situation in which resources can be relocated while they are being accessed without the user or application noticing anything. In such cases, the system is said to support ***relocation transparency***. An example of relocation transparency is when mobile users can continue to use their wireless laptops while moving from place to place without ever being (temporarily) disconnected.

Resources may be replicated to increase availability or to improve performance by placing a copy close to the place where it is accessed. ***Replication transparency*** deals with hiding the fact that several copies of a resource exist. To hide replication from users, it is necessary that all replicas have the same name. Consequently, a system that supports replication transparency should generally support location transparency as well, because it would otherwise be impossible to refer to replicas at different locations.

Concurrency transparency enables several processes to operate concurrently using shared resources without interference between them.

Failure transparency enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components. Making a distributed system failure transparent means that a user does not notice that a resource (he has possibly never heard of) fails to work properly, and that the system subsequently recovers from that failure.

C. Openness in a Distributed System

Another important goal of distributed systems is openness. An open distributed system is a system that offers services according to standard rules that describe the syntax and semantics of those services. For example, in computer networks, standard rules govern the format, contents, and meaning of messages sent and received. Such rules are formalized in protocols. In distributed systems, services are generally specified through interfaces, which are often described in an Interface Definition Language (IDL). This interface should properly have specified.

In addition to well defined interfaces, ***interoperability*** and ***portability*** are goals for open distributed system. Interoperability characterizes the extent by which two implementations of systems or components from different manufacturers can coexist and work together by merely relying on each other's services as specified by a common standard. Portability characterizes to what extent an application developed for a distributed system A can be executed without modification, on a different distributed system B that implements the same interfaces as A.

Another important goal for an open distributed system is that it should be easy to ***configure*** the system out of different components (possibly from different developers). Also, it should

be easy to add new components or replace existing ones without affecting those components that stay in place. In other words, an open distributed system should also be ***extensible***.

D. Scalability in Distributed Systems

Distributed systems operate effectively and efficiently at many different scales, ranging from a small intranet to the Internet. A system is described as scalable if it will remain effective when there is a significant increase in the number of resources and the number of users. It allows the system and applications to expand in scale without change to the system structure or the application algorithms.

Scalability of a system can be measured along at least three different dimensions. First, a system can be scalable with respect to its ***size***, meaning that we can easily add more users and resources to the system. Second, a ***geographically*** scalable system is one in which the users and resources may lie far apart. Third, a system can be ***administratively*** scalable; spanning that it can still be easy to manage even if it spans many independent administrative organizations.

1.4. Types of Distributed System

I. Distributed Computing Systems

It is class of distributed systems which is used for high-performance computing tasks. Distributed computing system consists of two sub groups: cluster computing and grid computing.

In ***cluster computing*** the underlying hardware consists of a collection of similar workstations or PCs (homogeneous), closely connected by means of a high speed local-area network. In addition, each node runs the same operating system. It is used for parallel programming in which a single compute intensive program is run in parallel on multiple machines.

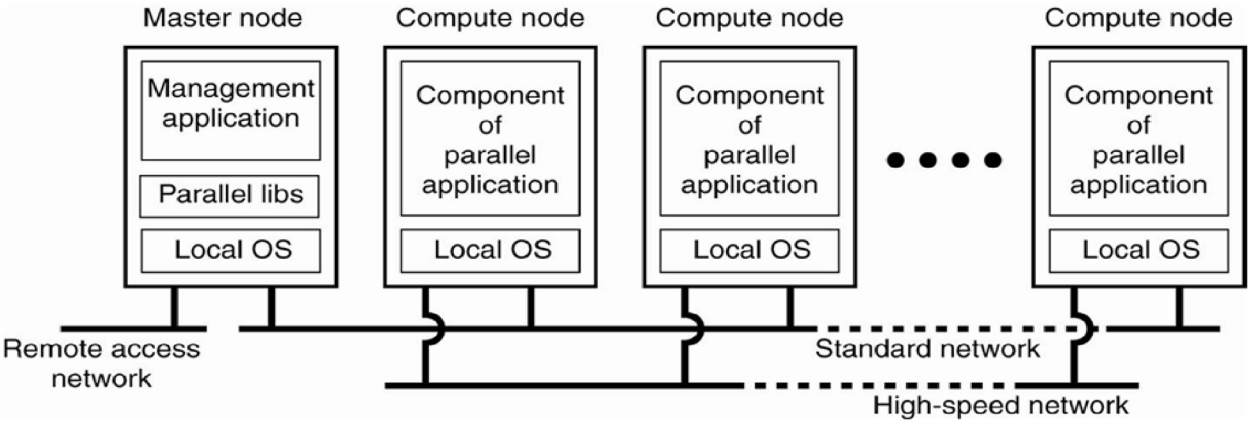

Figure 1.2: Example of cluster computing system

Each cluster consists of a collection of compute nodes that are controlled and accessed by means of a single master node. The master typically handles the allocation of nodes to a particular parallel program, maintains a batch queue of submitted jobs, and provides an interface for the users of the system. The master actually runs the middleware needed for the execution of programs and management of the cluster, while the compute nodes often need nothing else but a standard operating system.

Grid computing subgroup consists of distributed systems that are often constructed as a federation of computer systems. Grid computing systems have high degree of heterogeneity: no assumptions are made concerning hardware, operating systems, networks, administrative domains, security policies, etc. A key issue in a grid computing system is that resources from different organizations are brought together to allow the collaboration of a group of people or institutions.

From the perspective of grid computing, a next logical step is to simply *outsource* the entire infrastructure that is needed for compute-intensive applications. In essence, this is what **cloud computing** is all about: proving the facilities to dynamically construct an infrastructure and compose what is needed from available services. Unlike grid computing, which is strongly associated with high-performance computing, cloud computing is much more than just proving lots of resources. According to the National Institute of Standards and Technology (NIST), a cloud computing is defined as a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (E,g. networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal effort or service provider interaction.

II. Distributed Information Systems

In many cases, a networked application simply consisted of a server running that application and making it available to remote programs (clients). Such clients could send a request to the server for executing a specific operation, after which a response would be sent back. Integration at the lowest level would allow clients to wrap a number of requests into a single larger request and have it executed as a distributed transaction; ***all or none*** of the requests would be executed.

Transaction Processing Systems

For database applications, operations on a database are usually carried out in the form of transactions. Programming using transactions requires special primitives that must either be supplied by the underlying distributed system or by the language runtime system. The exact list of primitives depends on what kinds of objects are being used in the transaction. Typical examples of transaction primitives are shown below.

Primitive	Description
BEGIN_TRANSACTION	Mark the start of a transaction
END_TRANSACTION	Terminate the transaction and try to commit
ABORT_TRANSACTION	Kill the transaction and restore the old values
READ	Read data from a file, a table, or otherwise
WRITE	Write data to a file, a table, or otherwise

Assume the following two banking operations:

- withdraw an amount x from account 1

- deposit the amount x to account 2

What happens if there is a problem after the first activity is carried out? The solution is to group the two operations into one transaction; and then either both are carried out or neither. Transactions are (ACID properties):

1) ***Atomic***: to the outside world, the transaction happens indivisibly; a transaction either happens completely or not at all; intermediate states are not seen by other processes
2) ***Consistent***: the transaction does not violate system invariants; e.g., in an internal transfer in a bank, the amount of money in the bank must be the same as it was before the transfer (the law of conservation of money); this may be violated for a brief period of time, but not seen to other processes
3) ***Isolated***: concurrent transactions do not interfere with each other; if two or more transactions are running at the same time, the final result must look as though all transactions run sequentially in some order
4) ***Durable***: once a transaction commits, the changes are permanent.

III. Distributed Pervasive Systems

The distributed systems discussed so far are characterized by their stability; fixed nodes having high-quality connection to a network.

There are also mobile and embedded computing devices which are small, battery powered, mobile, and with a wireless connection. They are referred as distributed pervasive systems, in which instability is the default behavior.

There are three requirements for pervasive applications:

a) Embrace contextual changes: a device is aware that its environment may change all the time, e.g., changing its network access point
b) Encourage ad hoc composition: devices are used in different ways by different users
c) Recognize sharing as the default: devices join a system to access or provide information.

Examples of pervasive systems are:

- Home Systems that integrate consumer electronics
- Electronic Health Care Systems to monitor the well-being of individuals
- Sensor Networks

Further reading assignments

1. Parallel processing vs Distributed System
2. Distributed Operating System
3. Grid and cloud computing organization
4. Examples of a distributed systems application
5. Beginner common mistakes while developing a distributed system applications