Introduction to Distributed System

➢ Characterization of Distributed Systems

Definition:

"A distributed system is a collection of independent computers that appears to its users as a single coherent system."

Important aspects of this definition:

- Distributed system consist of components
- Users think they are dealing with a single system.

Differences between the various computers and the ways in which they communicate are mostly hidden from users. Users and applications can interact with a distributed system in a consistent and uniform way, regardless of where and when interaction takes place. Distributed systems should also be relatively easy to expand or scale.

Networks of computers are everywhere. The Internet is one, as are the many networks of which it is composed. Mobile phone networks, corporate networks, factory networks, campus networks, home networks, in-car networks – all of these, both separately and in combination, share the essential characteristics that make them relevant subjects for study under the heading distributed systems.

Distributed system is the one in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages. This simple definition covers the entire range of systems in which networked computers can usefully be deployed.

Characteristics of Distributed Systems are,

Concurrency: In a network of computers, concurrent program execution is the 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. The capacity of the system to handle shared resources can be increased by adding more resources (for example. computers) to the network. The coordination of concurrently executing programs that share resources is also an important and recurring topic.

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: All computer systems can fail, and it is the responsibility of system Designers to plan for the consequences of possible failures. Distributed systems can fail in new ways. 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.

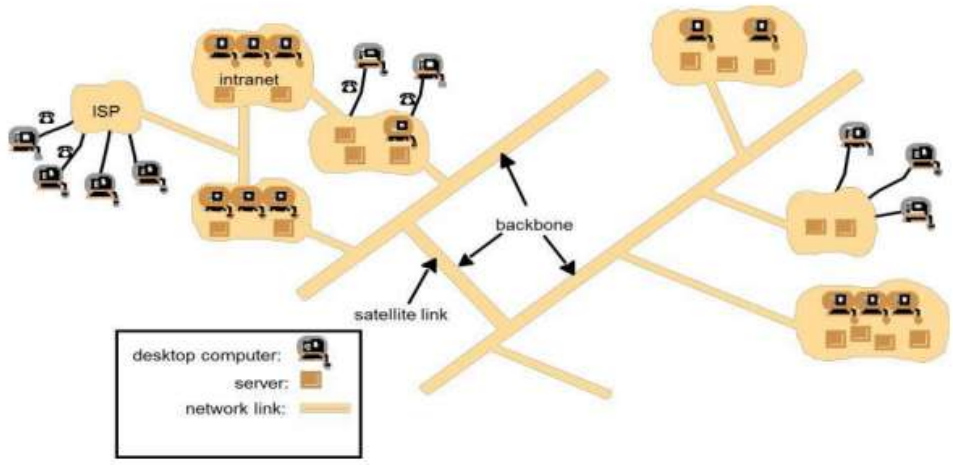
EXAMPLES OF DISTRIBUTED SYSTEMS

Typical examples of Distributed systems are,

- The Internet
- Intranets
- Mobile and Ubiquitous computing.

The Internet:

The Internet is a very large distributed system. It enables users, wherever they are, to make use of services like www, email, file transfer. The set of services is open-ended. Refer figure below which shows a typical portion of the internet. Internet connects millions of LANs and MANs to each other.

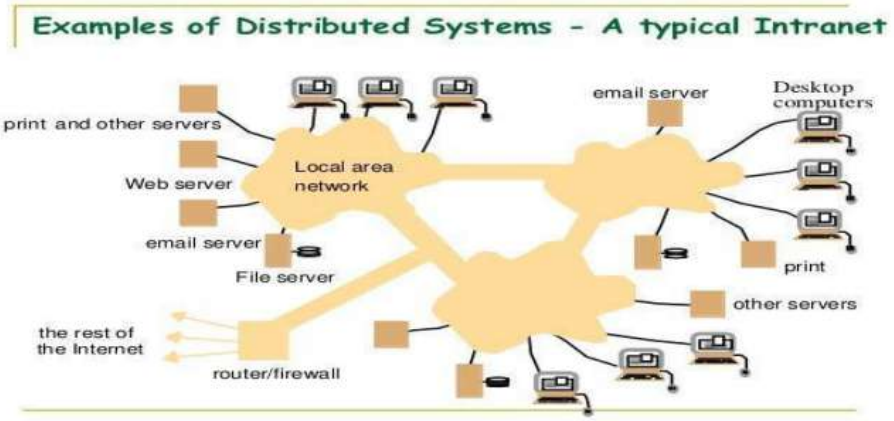


Intranet

An intranet is a portion of the internet that is separately administered and has a boundary that can be configured to enforce local security policies. It may be composed of several LANs linked by backbone connections. The n/w configuration of a particular intranet is

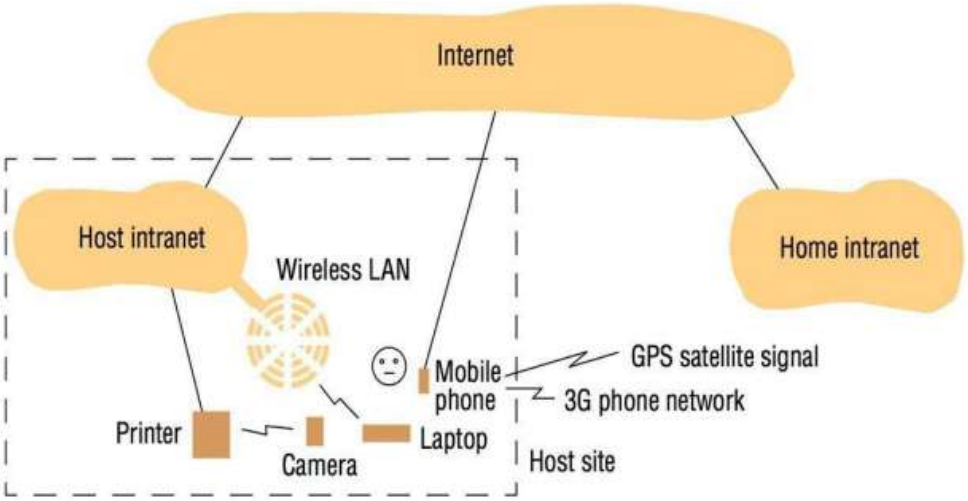
the responsibility of the organization that administers it. An intranet is connected to the Internet via router, which allows the users to use the services available on the Internet.

Firewall is used to protect the intranet by preventing unauthorized messages leaving or entering. Some organizations do not wish to connect their internal networks to the Internet at all. E.g. Police and other security and law enforcement agencies are likely to have at least some internal networks that are isolated from the outside world. These organizations can be connected to the Internet to avail the services by dispensing with the firewall.



Mobile and Ubiquitous computing:

Integration of portable computing devices like Laptops, smartphones, handheld devices, pagers, digital cameras, smart watches, devices embedded in appliances like refrigerators, washing machines, cars etc. with the distributed systems became possible because of the technological advances in device miniaturization and wireless networking. These devices can be connected to each other conveniently in different places, makes mobile computing possible. Figure below shows how a user from home intranet can access the resources at Host intranet using mobile devices.



In mobile computing, users who are away from home intranet, are still allowed to access resources via the devices they carry.

Ubiquitous computing is the harnessing of many small, cheap computational devices that are present in user's physical environments, including home, office and others. The term ubiquitous is intended to suggest that small computing devices will eventually become so pervasive in everyday objects that they are scarcely noticed. The presence of computers everywhere is useful only when they can communicate with one another.

E.g. It would be convenient for users to control their washing machine and hi-fi system using a "Universal remote control" device at home. The mobile user can benefit from computers that are everywhere.

➢ Issues, Goals, and Types of distributed systems

CHALLENGES IN DISTRIBUTED SYSTEMS

HETEROGENEITY:

The Internet enables users to access services and run applications over a heterogeneous collection of computers and networks. Heterogeneity (that is, variety and difference) applies to all of the following:

• Networks;

• Computer hardware;

• Operating systems;

• programming languages;

• Implementations by different developers.

Although the Internet consists of many different sorts of network their differences are masked by the fact that all of the computers attached to them use the Internet protocols to communicate with one another. For example, a computer attached to an Ethernet has an implementation of the Internet protocols over the Ethernet, whereas a computer on a different sort of network will need an implementation of the Internet protocols for that network.

Data types such as integers may be represented in different ways on different sorts of hardware – for example, there are two alternatives for the byte ordering of integers. These differences in representation must be dealt with if messages are to be exchanged between programs running on different hardware.

Although the operating systems of all computers on the Internet need to include an implementation of the Internet protocols, they do not necessarily all provide the same application programming interface to these protocols. For example, the calls for exchanging messages in UNIX are different from the calls in Windows.

Different programming languages use different representations for characters and data structures such as arrays and records. These differences must be addressed if programs written in different languages are to be able to communicate with one another. Programs written by different developers cannot communicate with one another unless they use common standards, for example, for network communication and the representation of primitive data items and data structures in messages. For this to happen, standards need to be agreed and adopted – as have the Internet protocols.

OPENNESS:

Openness cannot be achieved unless the specification and documentation of the Key software interfaces of the components of a system are made available to software developers. In a word, the key interfaces are published. This process is akin to the standardization of interfaces, but it often bypasses official standardization procedures, which are usually cumbersome and slow-moving.

However, the publication of interfaces is only the starting point for adding and extending services in a distributed system. The challenge to designers is to tackle the complexity of distributed systems consisting of many components engineered by different people. Systems that are designed to support resource sharing in this way are termed open distributed systems to emphasize the fact that they are extensible. They may be extended at the hardware level by the addition of computers to the network and at the software level by the introduction of new services and the reimplementation of old ones, enabling application programs to share resources. A further benefit that is often cited for open systems is their independence from individual vendors.

SECURITY:

Many of the information resources that are made available and maintained in distributed systems have a high intrinsic value to their users. Their security is therefore of

considerable importance. 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).

In a distributed system, clients send requests to access data managed by servers, which involves sending information in messages over a network. For example:

1. A doctor might request access to hospital patient data or send additions to that data.

2. In electronic commerce and banking, users send their credit card numbers across the Internet.

In both examples, the challenge is to send sensitive information in a message over a network in a secure manner. But security is not just a matter of concealing the contents of messages – it also involves knowing for sure the identity of the user or other agent on whose behalf a message was sent. In the first example, the server needs to know that the User is really a doctor, and in the second example, the user needs to be sure of the identity of the shop or bank with which they are dealing. The second challenge here is to identify a remote user or other agent correctly. Both of these challenges can be met by the use of encryption techniques developed for this purpose.

SCALABILITY:

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. The number of computers and servers on the Internet has increased dramatically. Figure below shows the increasing number of computers and web servers during the 12-year history of the Web up to 2005.

Date	*Computers*	*Web servers*	*Percentage*
1993, July	1,776,000	130	0.008
1995, July	6,642,000	23,500	0.4
1997, July	19,540,000	1,203,096	6
1999, July	56,218,000	6,598,697	12
2001, July	125,888,197	31,299,592	25
2003, July	~200,000,000	42,298,371	21
2005, July	353,284,187	67,571,581	19

It is interesting to note the significant growth in both computers and web servers in this period, but also that the relative percentage is flattening out – a trend that is explained by the growth of fixed and mobile personal computing. One web server may also increasingly be hosted on multiple computers.

The design of scalable distributed systems presents the following challenges:

- Controlling the cost of physical resources
- Controlling the performance loss
- Preventing software resources running out
- Avoiding performance bottlenecks

FAILURE HANDLING

Computer systems sometimes fail. When faults occur in hardware or software, programs may produce incorrect results or may stop before they have completed the intended computation.

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. The following are techniques for dealing with failures.

Detecting failures: Some failures can be detected. For example, checksums can be used to detect corrupted data in a message or a file. Chapter 2 explains that it is difficult or even impossible to detect some other failures, such as a remote crashed server in the Internet. The challenge is to manage in the presence of failures that cannot be detected but may be suspected.

Masking failures: Some failures that have been detected can be hidden or made less severe. Two examples of hiding failures:

1. Messages can be retransmitted when they fail to arrive.

2. File data can be written to a pair of disks so that if one is corrupted, the other may still be correct.

Just dropping a message that is corrupted is an example of making a fault less severe – it could be retransmitted. The reader will probably realize that the techniques described for hiding failures are not guaranteed to work in the worst cases; for example, the data on the second disk may be corrupted too, or the message may not get through in a reasonable time however often it is retransmitted.

Tolerating failures: Most of the services in the Internet do exhibit failures – it would not be practical for them to attempt to detect and hide all of the failures that might occur in such a large network with so many components. Their clients can be designed to tolerate failures, which generally involves the users tolerating them as well. For example, when a web browser cannot contact a web server, it does not make the user wait forever while it

keeps on trying – it informs the user about the problem, leaving them free to try again later. Services that tolerate failures are discussed in the paragraph on redundancy below.

Recovery from failures: Recovery involves the design of software so that the state of permanent data can be recovered or 'rolled back' after a server has crashed. In general, the computations performed by some programs will be incomplete when a fault occurs, and the permanent data that they update (files and other material stored in permanent storage) may not be in a consistent state.

Redundancy: Services can be made to tolerate failures by the use of redundant components

CONCURRENCY

Both services and applications provide resources that can be shared by clients in a distributed system. There is therefore a possibility that several clients will attempt to access a shared resource at the same time. For example, a data structure that records bids for an auction may be accessed very frequently when it gets close to the deadline time.

The process that manages a shared resource could take one client request at a time. But that approach limits throughput. Therefore services and applications generally allow multiple client requests to be processed concurrently. To make this more concrete, suppose that each resource is encapsulated as an object and that invocations are executed in concurrent threads. In this case it is possible that several threads may be executing concurrently within an object, in which case their operations on the object may conflict with one another and produce inconsistent results.

For example, if two concurrent bids at an auction are 'Smith: $122' and 'Jones: $111', and the corresponding operations are interleaved without any control, then they might get stored as 'Smith: $111' and 'Jones: $122'.

The moral of this story is that any object that represents a shared resource in a distributed system must be Responsible for ensuring that it operates correctly in a concurrent environment. This applies not only to servers but also to objects in applications. Therefore any programmer who takes an implementation of an object that was not intended for use in a distributed system must do whatever is necessary to make it safe in a concurrent environment.

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.

Access transparency enables local and remote resources to be accessed using identical operations. Location transparency enables resources to be accessed without knowledge of their physical or network location (for example, which building or IP address).

Concurrency transparency enables several processes to operate concurrently using shared resources without interference between them. Replication transparency enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.

Failure transparency enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components.

Mobility transparency allows the movement of resources and clients within a system without affecting the operation of users or programs.

Performance transparency allows the system to be reconfigured to improve performance as loads vary.

Scaling transparency allows the system and applications to expand in scale without change to the system structure or the application algorithms.

The two most important transparencies are access and location transparency; their presence or absence most strongly affects the utilization of distributed resources. They are sometimes referred to together as network transparency.

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 nonfunctional 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.

Reliability and security issues are critical in the design of most computer systems. The performance aspect of quality of service was originally defined in terms of responsiveness and computational throughput, but it has been redefined in terms of ability to meet timeliness guarantees, as discussed in the following paragraphs.

Some applications, including multimedia applications, handle time-critical data – streams of data that are required to be processed or transferred from one process to another at a fixed rate. For example, a movie service might consist of a client program that retrieves a film from a video server and presents it on the user's screen. For a satisfactory result the successive frames of video need to be displayed to the user within some specified time limits.

In fact, the abbreviation QoS has effectively been commandeered to refer to the ability of systems to meet such deadlines. Its achievement depends upon the availability of the necessary computing and network resources at the appropriate times. This implies a requirement for the system to provide guaranteed computing and communication resources that are sufficient to enable applications to complete each task on time (for example, the task of displaying a frame of video).

The networks commonly used today have high performance – for example, BBC iPlayer generally performs acceptably – but when networks are heavily loaded their performance can deteriorate, and no guarantees are provided. QoS applies to operating systems as well as networks. Each critical resource must be reserved by the applications that require QoS, and there must be resource managers that provide guarantees. Reservation requests that cannot be met are rejected.

Goals of Distributed System

A distributed system should make resources easily accessible; it should be reasonably hide the fact that resources are distributed across a network; it should be open; and it should be scalable.

- Making resources accessible
- Distribution Transparency
- Openness
- Scalability
- Reliability
- Performance

Making resources accessible

The main goal of DS is to make it easy for users to access remote resources and to share them in a controlled and efficient way. Resources can be anything: printers, computers, storage facilities, data, files, web pages, networks etc. There are many reasons for wanting to share resources. One obvious reason is economics.

E.g. it is cheaper to let a printer be shared by several users in a small office than having to buy and maintain a separate printer for each user.

Distribution Transparency

An important goal of a DS is to hide the fact that its processes and resources are physically distributed across multiple computers.

DS that is able to present itself to users and applications as if it were only a single computer system is said to be transparent.

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is 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

Figure 1-2. Different forms of transparency in a distributed system (ISO, 1995).

Openness

An open DS is a system that offers services according to standard rules that describe the syntax and semantics of those services.

E.g. in CN, standard rules govern the format, contents, and meaning of messages sent and received. Such rules are formalized in protocols.

Types of Distributed System

- Distributed Computing Systems
  - Cluster Computing Systems
  - Grid Computing Systems
- Distributed Information Systems
  - Transaction Processing Systems
  - Enterprise Application Integration
- Distributed Pervasive Systems
  - Home Systems
  - Electronic Health Care System
  - Sensor Networks

Distributed Computing System

An important class of distributed systems is the one used for high-performance computing tasks.

Roughly speaking, one can make a distinction between two subgroups.

In cluster computing the underlying hardware consists of a collection of similar workstations or PCs, closely connected by means of a high- speed local-area network. In addition, each node runs the same operating system.

The situation becomes quite different in the case of grid computing.

This subgroup consists of distributed systems that are often constructed as a federation of computer systems, where each system may fall under a different administrative domain,

and may be very different when it comes to hardware, software, and deployed network technology.

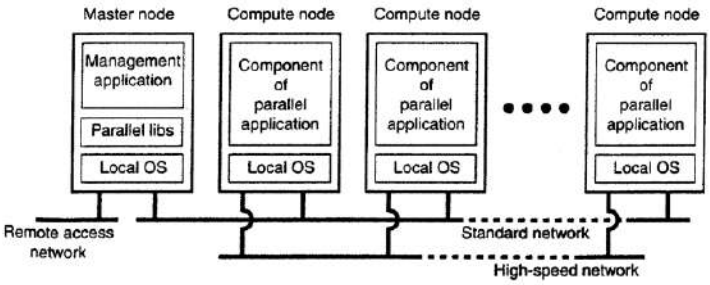


Figure 1-6. An example of a cluster computing system.

Cluster Computing System

Cluster computing is used for parallel programming in which a single (compute intensive) program is run in parallel on multiple machines.

One well-known example of a cluster computer is formed by Linux-based Beowulf clusters, of which the general configuration is shown in Fig. 1-6.

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.

As such, 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 System

Grid computing systems have a 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.

Such a collaboration is realized in the form of a virtual organization.

The people belonging to the same virtual organization have access rights to the resources that are provided to that organization.

Typically, resources consist of computer servers (including supercomputers, possibly implemented as cluster computers), storage facilities, and databases. In addition, special networked devices such as telescopes, sensors, etc., can be provided as well.

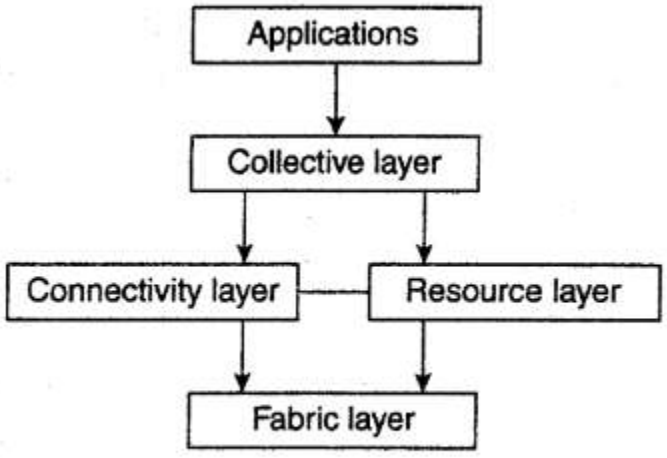


Figure 1-7. A layered architecture for grid computing systems.

Given its nature, much of the software for realizing grid computing evolves around providing access to resources from different administrative domains, and to only those users and applications that belong to a specific virtual organization.

For this reason, focus is often on architectural issues.

An architecture proposed by Foster et al. (2001). is shown in Fig. 1-7

The lowest fabric layer provides interfaces to local resources at a specific site. Note that these interfaces are tailored to allow sharing of resources within a virtual organization.

Typically, they will provide functions for querying the state and capabilities of a resource

The connectivity layer consists of communication protocols for supporting grid transactions that span the usage of multiple resources. For example, protocols are needed to transfer data between resources, or to simply access a resource from a remote location.

In addition, the connectivity layer will contain security protocols to authenticate users and resources. Note that in many cases human users are not authenticated; instead, programs acting on behalf of the users are authenticated.

In this sense, delegating rights from a user to programs is an important function that needs to be supported in the connectivity layer. The resource layer is responsible for managing a single resource. It uses the functions provided by the connectivity layer and calls directly the interfaces made available by the fabric layer.

For example, this layer will offer functions for obtaining configuration information on a specific resource, or, in general, to perform specific operations such as creating a process or reading data. The resource layer is thus seen to be responsible for access control, and hence will rely on the authentication performed as part of the connectivity layer.

The next layer in the hierarchy is the collective layer. It deals with handling access to multiple resources and typically consists of services for resource discovery, allocation and scheduling of tasks onto multiple resources, data replication, and so on. Unlike the connectivity and resource layer, which consist of a relatively small, standard collection of protocols,

The collective layer may consist of many different protocols for many different purposes, reflecting the broad spectrum of services it may offer to a virtual organization. Finally, the application layer consists of the applications that operate within a virtual organization and which make use of the grid computing environment.

The collective, connectivity, and resource layer form the heart of what could be called a grid middleware layer. These layers jointly provide access to and management of resources that are potentially dispersed across multiple sites.

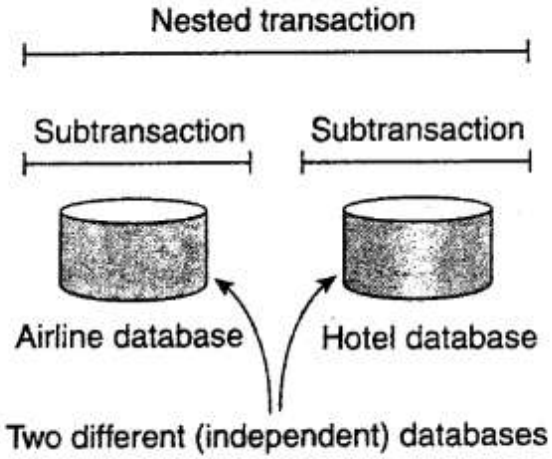
Distributed Information System

Distributed Transaction Processing:

It works across different servers using multiple communication models.

The four characteristics that transactions have:

- Atomic: the transaction taking place must be indivisible to the others.
- Consistent: The transaction should be consistent after the transaction has been done.
- Isolated: Concurrent transactions do not interfere with each other.
- Durable: Once a transaction commits, the changes are permanent.



A nested transaction is constructed from a number of subtransactions.

The top-level transaction may fork off children that run in parallel with one another, on different machines, to gain performance or simplify programming. Each of these children may also execute one or more subtransactions, or fork off its own children.

Nested transactions are important in distributed systems, for they provide a natural way of distributing a transaction across multiple machines. They follow a logical division of the work of the original transaction.

For example, a transaction for planning a trip by which three different flights need to be reserved can be logically split up into three subtransactions. Each of these subtransactions can be managed separately and independently of the other two.

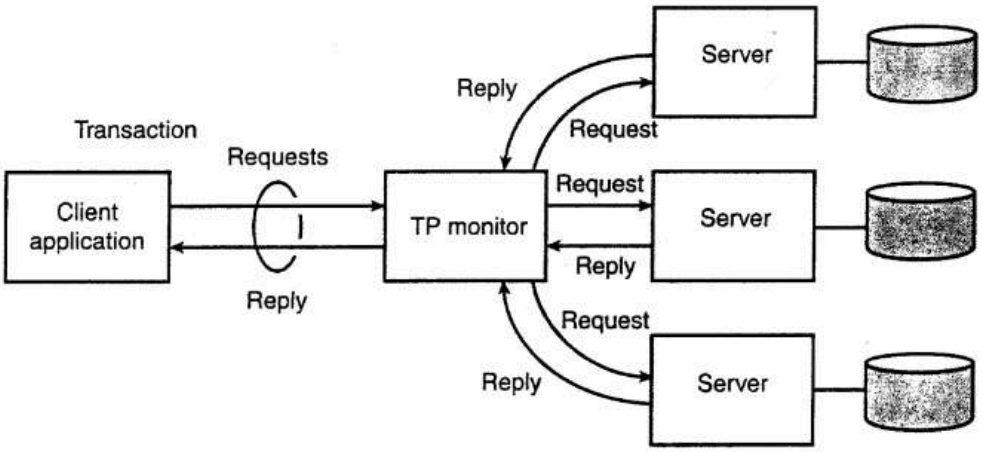


Figure 1-10. The role of a TP monitor in distributed systems.

In the company's middleware systems, the component that manages distributed (or nested) transactions has formed the application integration core at the server or database.

This was referred to as the Transaction Processing Monitor(TP Monitor). Its main task was to allow an application to access multiple servers/databases by providing a transactional programming model.

Many requests are sent to the database to get the result, to ensure each request gets successfully executed and deliver the result to each request, this work is handled by the TP Monitor.

Enterprise Application Integration

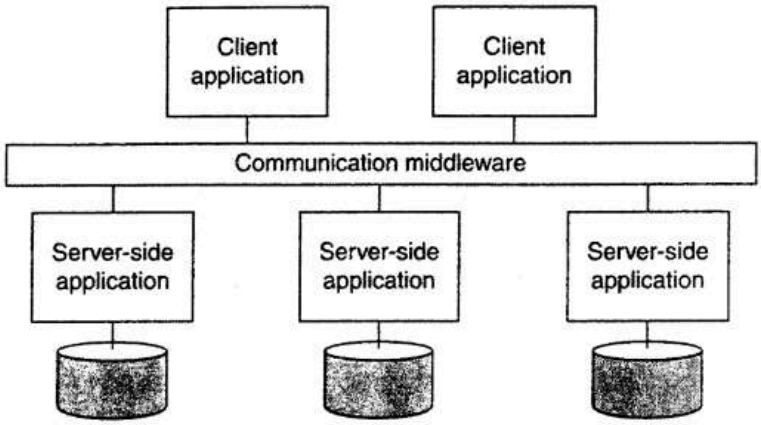


Figure 1-11. MiddJeware as a communication facilitator in enterprise application integration.

Enterprise Application Integration (EAI) is the process of bringing different businesses together.

The databases and workflows associated with business applications ensure that the business uses information consistently and that changes in data done by one business application are reflected correctly in another's.

Many organizations collect different data from different plate forms in the internal systems and then they use those data in the Trading system /physical medium.

Remote Procedure Calls (RPC), a software element that sends a request to every other application component.

An app can have a different database for managing different data and then they can communicate with each other on different platforms.

Suppose, if you login into your android device and watch your video on YouTube then you go to your laptop and open YouTube you can see the same video is in your watch list.

Distributed Pervasive System

Pervasive Computing is also abbreviated as ubiquitous (Changed and removed) computing

it is the new step towards integrating everyday objects with microprocessors so that this information can communicate.

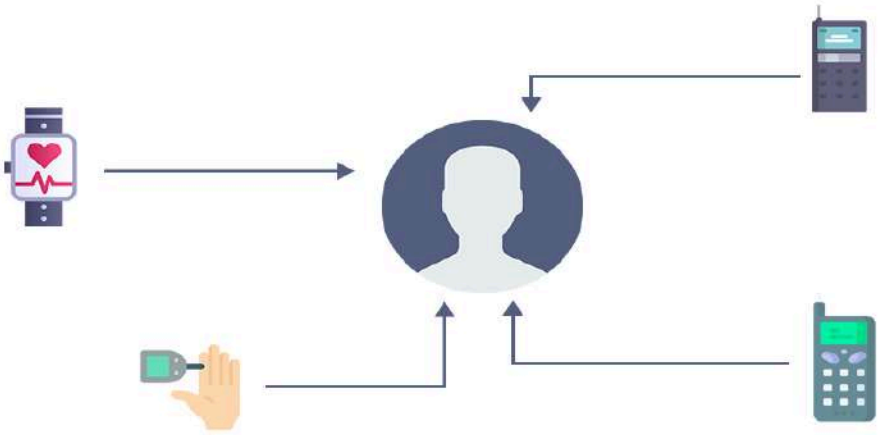
a computer system available anywhere in the company or as a generally available consumer system that looks like that same everywhere with the same functionality but that operates from computing power, storage, and locations across the globe.

Home System

Nowadays many devices used in the home are digital so we can control them from anywhere and effectively.

Electronic Health System

Nowadays smart medical wearable devices are also present through which we can monitor our health regularly.



Sensor Network (IoT devices)

Internet devices only send data to the client to act according to the data send to the device.

They can store and process the data to manage it efficiently.