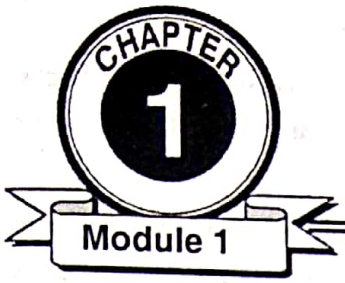


Introduction to Distributed System

Syllabus

Characterization of Distributed Systems: Issues, Goals, and Types of distributed systems, Distributed System Models, Hardware concepts, Software Concept.

Middleware: Models of Middleware, Services offered by middleware, Client Server model.

1.1 Introduction

- The development of powerful microprocessors and invention of the high speed networks are the two major developments in computer technology. Many machines in the same organization can be connected together through local area network and information can be transferred between machines in a very small amount of time.
- As a result of these developments, it became easy and practicable to organize computing system comprising large number of machines connected by high speed networks. Over the period of last thirty years, the price of microprocessors and communications technology has constantly reduced in real terms.
- Because of this, the distributed computer systems appeared as a practical substitute to uniprocessor and centralized systems. The networks of computers are present all over.
- Internet is composed of many networks. All these networks separately and in combination as well, share the necessary characteristics that make them pertinent topics to focus under distributed system.
- In distributed system, components located on different computers in network communicate and coordinate by passing messages. As per this argument, following are the characteristics of the distributed system.
  - Internet
  - Intranet, which is small part of internet managed by individual organization.
  - Mobile and ubiquitous computing.
- Resources required for computation are not available on single computer. These resources which are distributed among many machines can be utilized for computation and it is main motivation behind distributed computing. Followings are some of the important issues that need to be considered.

1. Concurrency

- The different cooperating applications should run in parallel and coordinate by utilizing the resources which are available at different machines in network.

2. Global Clock not Available

- Cooperation among applications running on different machines in network is achieved by exchanging the messages. For cooperation, applications action at particular time is exchanged. But, it is not easy to have all the machines clocks with same time.
- It is difficult to synchronize the different machines clock in network. Therefore, it is necessary to take into consideration that single global clock is not available and different machines in LAN, MAN or WAN have different clock time.

3. Independency in Failure of System Component

- Any individual component failure should not affect the computation. It is obvious that, any software or hardware component of the system may fail.
- This failure should not affect the system from running and system should take appropriate action for recovery.

Definition of Distributed System

- A computer network is defined as a set of communicating devices that are connected together by communication links. These devices include computers, printers and other devices capable of sending and/or receiving information from other devices on the network. These devices often called as node in the network. So computer network is interconnected set of autonomous computers.
- A distributed system is defined as set of autonomous computers that appears to its users as a single coherent system. Users of distributed system feel that, they are working with a single system.
- Following are the main characteristics of distributed system.
  - A distributed system comprises computers with distinct architecture and data representation. These dissimilarities and the ways all these machines communicate are hidden from users.
  - The manner in which distributed system is organized internally is also hidden from the users of the distributed system.
  - The interaction of users and applications with distributed system is in consistent and identical way, in spite of where and when interaction occurs.
  - A distributed system should allow for scaling it.
  - Distributed system should support for availability. It should be always available to the users and applications in spite of failures.
  - Failure handling should be hidden from users and applications.
- Network contains computers with different architectures (heterogeneous). To offer the single system view of distributed system in this heterogeneous environment, a middleware layer is often placed between higher layer that comprises of user and applications and lower layer comprising the operating system and communication facilities.

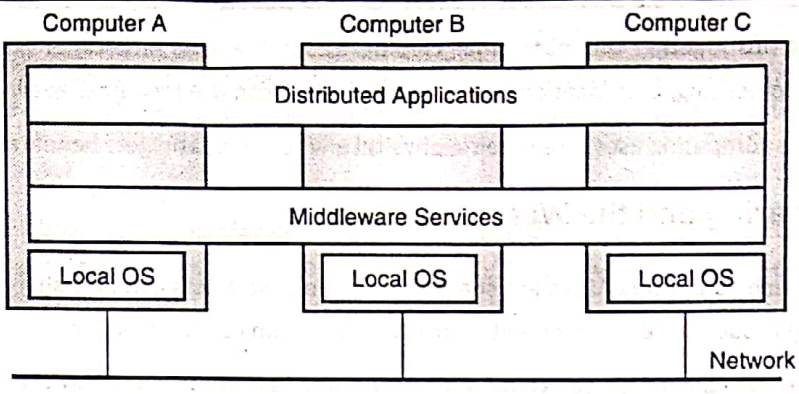


Fig. 1.1.1 : Distributed system organized as middleware

In Fig. 1.1.1, applications are running on three different machines A, B and C in network.

1.2 Examples of Distributed System

1.2.1 Internet

- Applications running on different machines communicate with each other by passing messages. This communication is possible due to different protocols designed for the same purpose.
- Internet is a very large distributed system. User at any location in network can use services of World Wide Web, email, file transfer etc. These services can be extended by addition of servers and new types of services.
- Internet Service Provider (ISP) provides internet connection to individual users and small organizations. Email and web hosting like local services are provided by ISP and it also enable the users to access services available at different locations in internet.

1.2.2 Intranet

- Intranet is small part of internet belonging to some organization and separately administrated to enforce local security policies. Intranet may comprise many LANs which are linked by backbone connection.
- Intranet can be administrated by administrator of single organization and its administration scope may vary ranging from LAN on single site connected to a connected.
- LANs that is part of many braches of companies. Router connects intranet to internet and user in intranet then can use services available in rest of the internet.
- Firewall which filters incoming and outgoing messages protects unauthorized messages to enter or leave the intranet.

1.2.3 Mobile and Ubiquitous Computing

- Many small and portable devices manufactured with advancement in technology is now integrated into distributed system. Examples of these devices are Laptop, PDAs, Mobile phones, Cameras, pagers, smart watches, devices embedded in appliances, cars and refrigerators etc.
- As these devices are portable and easily gets connected, mobile computing became possible. Mobile users can easily use resources available in network and services available in internet.

- Ubiquitous computing ties together the many small chip devices in available physical environment of home, office or elsewhere. In mobile computing, user location can be anywhere but can use resources available everywhere.
- Whereas, in ubiquitous computing users are in same physical environment and gets benefited.

1.3 Resource Sharing and the Web

- Hardware resources such as printers, disks are shared to reduce cost. Users can also share web pages or databases. The pattern of sharing in such large environment of distributed system cannot be same.
- All users can share search engine for searching purpose or small group of users can share their files or document among them. The resources available at different locations in large network can be used by users or applications in it.
- These resources are treated as services and present its functionality to users and applications. For example, shared files can be accessed by using file service. Sharing of resources available on different computers of large size system cannot be possible without communication.
- Thus, resource to be shared must be managed by some program which should provide communication interface which allows effective access and updating of this resource with reliability and in consistent way.
- Server application provides services. Client application running on other machine, request the service of server by sending message. In this scenario, we say that client is invoking an operation on server.
- Server processes the request and sends back response to client. This is called remote invocation by client.
- Requesting application is client and responding application is server. Server may request the service of other server also. Here, client and servers are processes and not computers.

1.3.1 World Wide Web (WWW)

- This system publishes, helps in accessing resources and services available in internet. Internet user uses web browser application to retrieve documents listen audio or view video streams.
- In any document, hyperlink is available to organize, view the contents or can be used as references to other documents and resources which are already available in web.
- Thus, users may go indefinite way to view required information as hyperlinks are provided in each viewed document for further information of their use. Web is open to extend, add new services and open to implement in new way with assurance about not affecting the existing system. For example, any browser can access any web server.
- This flexibility of web is applicable for all types of contents to publish. If new format or type of content is invented by any user then web can publish it. For this new content presentation, plug-ins in browser can be added.
- Webs working is based on three basic components which are HTML (Hypertext marks up language), HTTP (Hypertext transfer protocol) and URLs (Uniform resource locators).
  1. **HTML :** Content of web page like text and image is presented by using HTML. These structured items in web page are organized as headings, paragraphs, tables and images. Links are also provided to access the resources present at other location in network.
  2. **HTTP :** Web browser or client applications use this protocol to interact with web servers. Following are main features of HTTP.

- **Request-reply Interaction :** Client send request to server using URL of the resource to be accessed.
- **Content Types :** The client sends request to server which the returns response to client. Browsers are not necessarily capable to process this received content from server. Server mentions content type in reply message so that browser can process it. Browser may take other applications help to process the content if it is not capable to process it.
- **Resource access :** Browser sends many requests concurrently to server to minimize the delay to the user.
- **Access Control :** Access to particular resource can be restricted for unauthorized users. Users should be authenticated to use this resource.

3. **URLs :** Browser uses URL to locate the resource on web server. URL contains two components. These are scheme and scheme-specific-location. The first component declares type of URL. For example, email address, URL to access the file using FTP protocol, nntp to specify news group and telnet for remote login are the examples of scheme.

1.4 Issues and Goals

Following are the issues and goals related to design of distributed system.

1.4.1 Heterogeneity

- We cannot have homogeneous environment in wide area network. The types of networks, architecture of the different computers in different networks, operating system running on different computers, programming languages used and implementation of products by different developers cannot be expected to be same. Internet enables users to access services and run applications in heterogeneous environment.
- There are different implementations of internet protocols for different type of networks. If the machines architecture is different then data representation supported by architecture is also different. Hence, different computers in distributed system have different representation for integer, characters and other data items. For example, IBM mainframe uses EBCDIC character code and IBM PC uses ASCII.
- It is necessary to deal with such differences in data representation when messages need to be exchanged between applications running on such different hardware. Different programming languages also use different data representations for data structures such as array and records.
- Applications developed in different languages should be able to communicate with each other. This issue also needs to be addressed. It is also necessary that products developed by different developers should communicate with each other. This is for example for network communication and representation of primitive data items and data structures in messages.
- Internet protocols have these agreed and adopted standards. In future, design of distributed system, these standards should be developed and adopted for communication among heterogeneous hardware and software components.

1.4.2 Making Resources Accessible

- Users and applications should be able to access the remote resources in easy way. Distributed system should allow sharing these remote resources in efficient and controlled manner.

- Resource sharing offers saving in cost. One printer can be shared among many users in office instead of having one printer to each individual user. There can be more saving in cost if expensive resources are shared.
- By connecting users and resources, it becomes easier to work together and exchange information. The success of the Internet is due to its straightforward protocols for exchanging files, mail, documents, audio, and video. The worldwide spread people can work together by means of groupware. Electronic commerce permits us to purchase and sell verity of goods without going to shop or even leaving home.
- The increase in connectivity and sharing also increases the security risk and to deal with it is equally important. Presently, systems offer fewer defenses against eavesdropping or intrusion on communication.
- A communication can be tracked to construct a favorite profile of a particular user. This clearly violates privacy, particularly if it is done without informing the user. A allied problem with increased connectivity can also cause unnecessary communication, for example electronic junk mail, called as spam. Special information filters can be used to select inward messages based on their content.

1.4.3 Distribution Transparency

The second main goal of a distributed system is to hide the actuality of physical distribution of processes and resources across several computers. A transparent distributed system offers its feel to users and applications as a single computer system. Following types of transparency is present in distributed system.

- **Access Transparency :** It hides the dissimilarities of data representation of different machines in the network and the manner in which remote resources are accessed by the user. Intel machines use little endian format to order bytes and SPARC uses big endian format. So Intel machines transfer high order bytes in beginning and in case of SPARC low order bytes are transmitted first. Different operating systems also have their own file name convention. All these dissimilarities should be hidden from users and applications.
- **Location Transparency :** This transparency hides the location of the resources in distributed system. For example, URL used to access web server and file does not give any idea about its location. Also name of the resource remains same although it changes the location when moved between machines in the distributed system.
- **Migration Transparency :** It hides the fact that resources are moved from one location to other. It does not affect the way in which these resources are accessed. Processes or files often are migrated by system to improve the performance. All this should remain the hidden from user of the distributed system.
- **Relocation Transparency :** If the user or application is using the resource and during use of it is if moved to other location then it remains hidden from user. For example if user is traveling in car and changing the location frequently, still he or she continuously uses the laptop without getting disconnected.
- **Replication Transparency :** It hides the fact that many copies of the resource are placed at different locations. Often resources are replicated to achieve availability or placed its copy near to location of its access. This activity of the replication should be transparent to the user.
- **Concurrency Transparency :** It hides the sharing of the resource by many users of the distributed system. If one user is using the resource then other should remain unknown about it. Many users can have stored their files on file server. It is essential that each user does not become aware of the fact that other is using the same resource. This is called concurrency transparency.

- **Failure Transparency :** It hides the failure and recovery of the resource from user. User does not become aware of failure of resource to work appropriately, and that the system then recovers from that failure. It is not possible to achieve complete failure transparency. Because for example, if network fails user can notice this failure.
- **Performance Transparency :** In order to improve performance system automatically takes some action and it user remains unaware of this action taken by system. In case of overloading of processor jobs gets transfer to lightly loaded machine. If many requests are coming from users for a particular resource then resource gets placed to nearest server of those users.
- **Scaling Transparency :** The main goal of this transparency is to permit the system to expand in scale without disturbing the operations and activities of existing users of distributed system.

1.4.4 Scalability

- If system adapt to increased service load then it is said to be a scalable system. There are three ways to measure the scalability of the system.

(i) Scalable with respect to its size
(ii) Geographically scalable
(iii) Administratively scalable

- Practically, scalable system leads to some loss of performance as the system scales up. Following are the problems related with scalability.

(i) Scalable with respect to its size

- System easily adapts addition of more users and resources to the system.
- In order to supports for more number of users and resources, it is necessary to deal with centralized services, data and algorithms. If centralized server is used to provide services to more number of users and applications then server will become bottleneck.
- In spite of having more processing power, large storage capacity with centralized server, more number of requests will be coming to server and increased communication will restrict further growth.
- Like the centralized services, centralized data also is not good idea. Keeping a single database would certainly saturate all the incoming and outgoing communication lines.
- In the same way above, centralized algorithms also a not good idea. If centralized routing algorithm is used, it will collect the information about load on machines and communication lines. The same information algorithm processes to compute optimal routes and distribute to all machines. Collecting such information and sending processed information will also overload the part of network. Therefore use of such centralized algorithms must be avoided. Preference should be given to only decentralized algorithms.

(ii) Geographically scalable

- Although users and resources may lie far distance apart geographically, still system allows the users to use resources and system.

- In synchronous communication client blocks until reply comes from server. Distributed system designed for local area networks (LANs) are based on synchronous communication due to small distances between machines.
- Therefore it is hard to scale such systems. A care should be taken while designing interactive applications using synchronous communication in wide area systems as machines are far apart and time required for communication is three orders magnitude slower with compare to LAN.
- A further problem that get in the way of geographical scalability is that communication in wide-area networks is intrinsically unreliable, and virtually always point-to-point. On the contrary, local-area networks generally offer highly reliable communication facilities based on broadcasting, which ease development of the distributed systems.
- If system has several centralized components then geographical scalability will be restricted because of the performance and reliability problems resulting from wide-area communication. Use of centralized components will lead to waste of network resources.

(iii) Administratively Scalable

- Although system spans many independent administrative organizations, still its management is easy.
- In order to scale a distributed system across multiple, independent administrative domains, a key problem that requires to be resolved is that of conflicting policies with respect to resource usage (and payment), management and security.
- Distributed system components residing in one domain can always be trusted by users that work within that same domain. In this scenario, system administration may have tested and authorized applications, and may have taken special measures to make sure that such components cannot be tampered with. Here, the users trust their system administrators. But, this trust does not cross domain boundaries naturally.
- If a distributed system crosses the domain boundaries, two forms of security measures require to be taken. First is, the distributed system on its own must protect against malicious attacks from the other new domain. Second, the other new domain on its own must protect against malicious attacks from the distributed system.

Scaling Techniques

Following are the three techniques for scaling :

1. Hiding communication latencies
2. Distribution
3. Replication

1. Hiding communication latencies

- Geographical scalability can be achieved by hiding communication latencies. In this case, waiting by client for response from geographically remote server can be avoided by using asynchronous communication for implementation of requester's application. In asynchronous communication instead of blocking client carry out other work until response is received.
- Batch-processing systems and parallel applications make use of asynchronous communication, in which if one task is waiting for communication to finish, some independent tasks can be scheduled for execution.

- On the other hand, a new thread of control can be started to carry out the request. Even though it blocks waiting for the reply, other threads in the process can carry on.
- In case of interactive applications, asynchronous communication cannot be used effectively. In this case to minimize the communication cost, server side computation of the request can be moved to the client side. Client side validation of form is the best example of this approach. Instead of carrying out validation at server side, it is better to shift it at client side.

2. Distribution

- Distribution is important scaling technique. In distribution component of the system divided into smaller parts, and then kept those parts across the system by spreading it. A good example of distribution is the Internet Domain Name System (DNS).
- There is hierarchical organization of DNS name space into a tree of **domains.** These domains are divided into non-overlapping **zones.** The names belonging to each zone are managed by a single name server.

3. Replication

- Although problems related to scalability degrade the performance, it is in general a good thought in fact to **replicate** components across a distributed system. Apart from increasing availability of the component, replication also balances the load between components which result in achieving the better performance. In WAN based system, placing a copy close to accessing location hides much of the communication latency problems described above.
- Similar to replication, caching leads to make a copy of a resource in the near to the client accessing that resource. But, contrary to replication, caching decision is taken by the client of a resource instead of owner of a resource.
- Caching is carried out on demand while replication is often planned in advance. The disadvantage of caching and replication that may have negative effect on scalability. Since multiple copies of a resource exist, making change in one copy makes that copy different from the others. As a result, caching and replication leads to consistency problems.

1.4.5 Openness

- Openness is the important goal of the distributed system. An **open distributed** system provides services as per standard rules that tell the syntax and semantics of those services. As, in computer networks, standard rules state the message format, its contents and meaning of sent and received messages. All these rules are present in protocols.
- Similar to above, in distributed systems, services are usually specified through interfaces, which are expressed in an Interface Definition Language (IDL). The definitions of the interfaces written in an IDL almost capture only the syntax of these services.
- They state exactly the names of the available functions together with parameters type; return values, exceptions likely to be raised etc. The semantics of interfaces means specification of what these interfaces can perform. Actually, these specifications are specified in an informal way through natural language.
- Processes make use of interfaces to communicate with each other. There can be different implementation of these interfaces leading to different distributed system that function in the same manner.

- Appropriate specifications are complete and neutral. Complete specifications specify the whole thing that is essential to make an implementation. But, many interface definitions does not obey completeness.
- So that it is required for a developer to put in implementation-specific details. If specifications do not impose what an implementation should look like : they should be neutral. Completeness and neutrality are significant for interoperability and portability.
- An open distributed system should allow configuring the system out of different components from different developers. Also, it should be trouble-free to put in new components or replace existing ones without disturbing those components that stay in place. It means, an open distributed system should also be extensible. Open systems interfaces are published.
- Monolithic approach should be avoided for building the system. There should be separation between policy and mechanism. For example, apart from storing the documents by browser, users should be able to make a decision which documents are stored and for how much duration. User should be able to set it dynamically.
- User should be able to implement his own policy as a component that can be plugged into the browser. Certainly, implemented component must have an interface that the browser can recognize so that it can call procedures of that interface.
- Open distributed system provides uniform communication mechanism and it is also based on published interfaces in order to access the common resources. It also considers heterogeneous environment in terms of hardware and software.

1.5 Types of Distributed Systems

- There are a different distributed computing systems
  1. Distributed Information Systems
  2. Distributed Embedded Systems
- The following discussion describes these systems.

1.5.1 Distributed Computing Systems

- One class of distributed system is intended for high-performance computing tasks. Cluster computing comprises the hardware that consists of a set of similar workstations or PCs running same operating systems and closely connected through a high speed local-area network.
- In case of grid computing, distributed systems are built as a group of computer systems. Here administrative domain of each system may be distinct, and may be very dissimilar in hardware, software, and deployed network technology.

1.5.1(A) Cluster Computing Systems

- Cluster computing systems were accepted because of reduced cost and improved performance of computers and workstations. From technical and cost point of view, it became reliable to build supercomputer using off-the-shelf technology by simply connecting a set of comparatively simple computers in a high-speed network.

- In almost all cases, cluster computing is used for parallel programming in which a single program is run in parallel on several machines.
- The common configuration of Linux-based Beowulf clusters is shown in Fig. 1.5.1. Each cluster comprises a set of compute nodes. All these compute nodes are controlled and accessed through a single master node.
- The master node characteristically manages the allocation of nodes to a specific parallel program, holds a batch queue of submitted jobs, and offers an interface for the users of the system.
- The master in fact runs the middleware required for the execution of programs and management of the cluster, whereas the compute nodes requires only a standard operating system. Libraries offer advanced message based communication facilities.

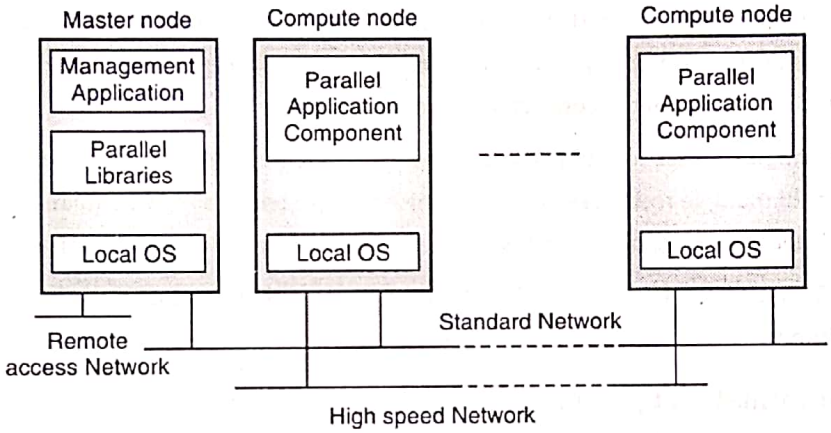


Fig. 1.5.1 : Cluster computing system

1.5.1(B) Grid Computing Systems

- In contrast to homogeneous environment of cluster computing systems, grid computing systems includes a high degree of heterogeneity. In this case hardware, operating systems, networks, administrative domains, security policies are different.
- In a grid computing system, resources from distinct associations are brought together to let the collaboration of a group of people or institutions. This collaboration leads to virtual organization. The people from same virtual organization are given access rights to the resources that are offered to that organization.
- These resources comprise compute servers (together with supercomputers, probably implemented as cluster computers), storage facilities, and databases. Also, special networked devices like telescopes, sensors, etc., can be made available as well. A layered architecture for grid computing systems is shown in following Fig. 1.5.2.

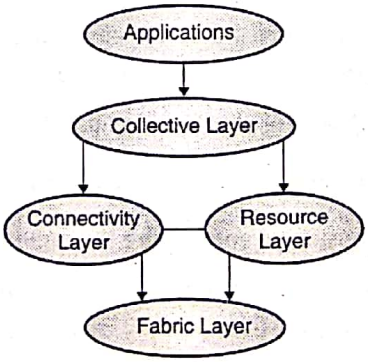


Fig. 1.5.2 : A layered architecture for grid computing systems

- **Fabric Layer :** It is a lowest layer and offers interfaces to local resources at a specific site. These interfaces are customized to permit sharing of resources within a virtual organization.
- **Connectivity layer :** This layer contains communication protocols to offer support for grid transactions that span the usage of multiple resources. Protocols are required to send data among resources or to access a resource from a distant location. Also, this layer will have security protocols to authenticate users and resources. Instead of user, if users program is authenticated then the handover of rights from user to program is carried out by connectivity layer.
- **Resource layer :** This layer manages a single resource. It makes use of functions offered by the connectivity layer and calls straight way the interfaces made available by the fabric layer. As an example, this layer will provide functions for getting configuration information on a particular resource or generally, to carry out specific operations such as process creation or reading data. Hence this layer is responsible for access control, and hence will be dependent on the authentication carried out as part of the connectivity layer.
- **Collective Layer :** It handles access to multiple resources and contains services for resource discovery, allocation and scheduling of tasks onto multiple resources, data replication, and so on. This layer contains many diverse protocols for verity of functions, reflecting the broad range of services it may provide to a virtual organization.
- **Application Layer :** This layer contains applications that functions within a virtual organization and which utilizes the grid computing environment.

1.5.2 Distributed Information Systems

- Many organizations deal with networked applications facing the problems in interoperability. Many existing middleware solutions provide easy way to integrate applications into an enterprise-wide information system.
- In most of the cases networked application is a client server application in which server run the application (database) and clients sends request to it. After processing request server sends reply to client.
- If integration is done at lowest level, it would permit clients to enclose a many requests, perhaps for different servers, into a single larger request and let it be executed as a distributed transaction. The basic scheme was that either all or none of the requests would be executed.
- Such integration should also happen by allowing applications to communicate straight way with each other. This has shown the way to enterprise application integration (EAI).Above two forms of distributed systems is explained below.

1.5.2(A) Transaction Processing Systems

Programming using transactions involves primitives that are either provided by underlying distributed system or by the language runtime system. Following are the examples of primitives for transactions.

- **BEGIN_TRANSACTION :** For marking the start of transaction.
- **END_ TRANSACTION :** For terminating the transaction and try to commit.
- **ABORT_ TRANSACTION :** For killing the transaction and restore the previous state.
- **READ :** To read the data from table or file.
- **WRITE :** To write the data to table or file.

The main property of a transaction is either all of its operations are executed or none are executed. The operations that are present in the body of transaction can be system calls, library procedures, or programming language statements. Following are the ACID properties of transaction.

1. **Atomic :** Execution of transaction is individual.
2. **Consistent :** There is no violation of system invariants by transaction.
3. **Isolated :** Concurrent transactions do not interfere with each other.
4. **Durable :** After commit is done by transaction, the changes are permanent.

Nested Transaction

- In nested transactions, top level transaction fork off children, which run in parallel to one another on different computers for improving the performance. Every forked child may also execute one or more subtransactions, or fork off its own children. Fig. 1.5.3 shows nested transaction.

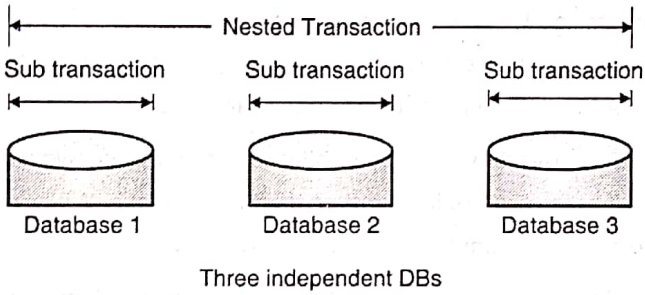


Fig. 1.5.3 : A nested Transaction

- Consider that, out of several subtransactions running in parallel one of the commits and submit the result to the parent transaction. If parent aborts after further computation and restores entire system to the previous state it had before the top-level transaction began.
- In this case the committed result of subtransaction must nevertheless be undone. Therefore the permanence referred to above applies only to top-level transactions. Any transaction or subtransaction basically works on a private copy of all data in the entire system.
- They manipulate the private copy only. If transaction or subtransaction aborts then private copy disappears. If it commits, its private copy replaces the parent's copy. Hence, if after commit of subtransaction a new subtransaction is started, then new one notice results created by the first one. Similarly, if higher-level transaction aborts then all its underlying subtransactions have to be aborted as well.
- Nested transactions present a natural means of distributing a transaction across multiple machines in distributed system. They exhibit logical splitting up of the work of the original transaction.

1.5.3 Enterprise Application Integration

- It became obvious to have some facilities to integrate applications independent from their databases. The reason behind this was the decoupling of applications from databases. There is requirement of straightway communication among application components instead of using the request/reply pattern that was supported by transaction processing systems.

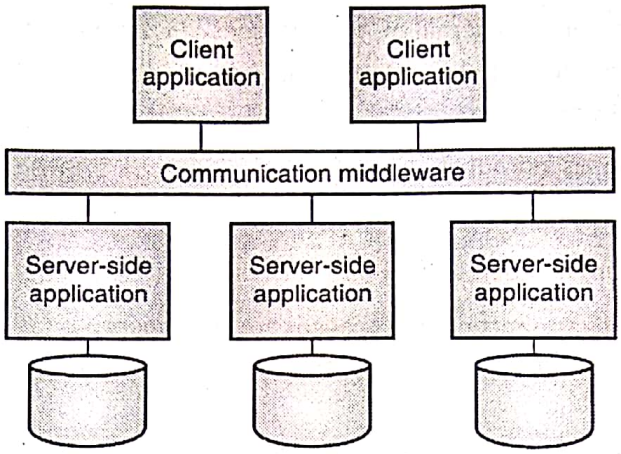


Fig. 1.5.4 : Middleware as a communication facilitator in enterprise application integration

- There are many examples of communication middleware. Using Remote procedure calls (RPC), an application component at client side can efficiently send a request to another application component at server side by making a local procedure call. This request at client side is packed as a message and sent to the called application. In the same way, the result will be packed as message and sent back to the calling application as the result of the procedure call.
- Like RPC, it is also possible to call the remote objects using remote method invocations (RMI). An RMI is basically is similar to RPC, except that it operates on objects rather than applications. While communicating using RPC or RMI both caller and callee have to be up and running. This is the drawback of it.
- In case of message-oriented middleware, applications just send messages to logical contact points, describe by means of a subject. Similarly, applications can demand for a specific type of message and communication middleware ensure that those messages are delivered to those applications.

1.5.4 Distributed Pervasive Systems

- The distributed system discussed above is stable and nodes are fixed and permanent in it. The distributed system in which mobile and embedded computing devices are present, instability is default behavior. These devices in the system called as distributed pervasive systems are small in size with battery-powered, mobile, and connected through wireless connection.
- An important characteristic of this system is absence of human administrative control. One possible best solution is owner configures their devices. These devices require discovering their environment automatically.
- In pervasive systems devices usually connect to the system with the intention of accessing and probably for providing the information. This calls for means to without difficulty read, store, manage, and share information. Considering irregular and altering connectivity of devices, the memory space where accessible information resides will most likely alter continually.

1.6 Distributed System Models

Basically distributed system models are categorized as :

- **Architectural Models :** These models concern about placement of different components of distributed system and relationship between these components.

- **Fundamental Models :** These models concern with formal description about properties that often appears and common in above architectural model.

1.6.1 Architectural Models

- These models of distributed system deals with placement of different components and with how these components are related to each other. Client server model is one of the examples of the architectural model. It is possible to modify client server model with following decisions. These are for examples :
  - Decision for partitioning the data and placing it at cooperating server machines.
  - Proxy client and servers for caching.
  - Client server model can also be modified by use of mobile and mobile agents.
  - As per requirement, adding and removal of mobile devices should support in suitable manner.
- The distributed system with decided components for design should meet current needs and future requirements. The designed distributed system should also supports for reliability. The system should be enough flexible to make changes, easy to manage, adaptable and should be cost effective.
- The architectural models considered here makes use of processes and objects.
- Three types of processes that are considered are (i) server (ii) client (iii) peer processes. Peer processes cooperate and interact with each other in symmetrical way to carry out the task.
- These processes can be placed at different machines in network in order to improve reliability and performance of the system. As a variation to client server model, it is possible to build dynamic system. Processes can move from one machine to other. Client can download code from server to run it locally. Communication traffic and access delays can be reduced by moving data from server to client machine.
- The design of some distributed system also allows adding or removing machines or mobile devices seamlessly. These devices can offer services to other or find out available services in the system.

1.6.1(A) Software Layers

- Software architecture defines software layers in single computer or it defines services that are provided or requested by processes running locally or on remote machines. This view can be expressed in terms of service layers.

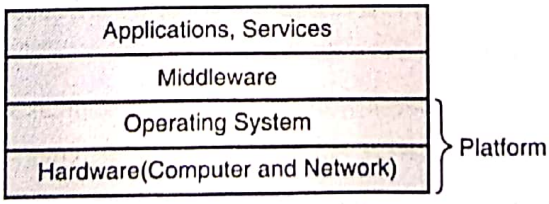


Fig. 1.6.1 : Service Layers In Distributed System

- Server process offers services to requesting client processes. Distributed Service can be placed on one or more servers which interact with each other and client processes by maintaining the consistent view of the service resources.
- In Fig. 1.6.1, lowest two layers are hardware and software layers which offer services to upper layers. These layers are platform for applications and distributed system. Middleware layer masks heterogeneity and offer communication and programming model to programmers.

- Middleware offers distribution transparency. It also provides services needed by application programs. Remote procedure call (RPC), Remote method invocation is the examples of middleware models.

1.6.1(B) System Architecture

- The key aspect in design of distributed system is placement of different system components such as applications, services servers and other processes on different computers in network. This decision affects performance, reliability and security of the resulting system.
- Following are the main types of architectural models.

Client Server Model

- This is basic and widely employed model. In this model server and client processes are on different computers. Client processes interacts with server processes to access the shared resources or to get services.
- In this model, requesting process is considered as client. Server process in turn may send request to other server process on different machine. For example web server may request local file server which manages files that contains web pages.

Services by Multiple Servers

- Several server processes, each running on separate computer may implement particular service. These server processes communicates with each other to provide the services to requesting client process. For example, client can access resource available on any particular web server. Here resources are partitioned and kept on different servers.
- Objects on which service is dependent can be distributed among many servers. In other case, objects can be replicated on many computers where server processes are running. The replication improves performance, availability and fault tolerance. Consistent copies are maintained at replicated serves. For example, web service can be mapped onto many servers having replicated database.

Proxy Servers and Caches

- Recently used data objects are maintained at cache. The newly received objects are added in cache by replacing existing objects there if required. Cached objects needed by client are first checked by caching service to confirm it as up-to-date copy. If not then up-to-date copy is accessed. These caches are collocated with other each client copies or may be fetched from proxy server shared by many clients.
- For example, Web browsers always caches recently used web pages, resources and maintain at client local file system. Before displaying these pages, browser uses HTTP request to check consistency of cached pages with server copies.
- Proxy servers can also maintain these shared caches to share among many clients. These proxy servers are maintained to reduce load on network and to improve performance. Proxy servers may also access remote web servers.

Peer Processes

- In this model, all processes are considered as equal and cooperate with each other to achieve a designated task. No process is client or server and code in these processes maintains the consistency in resources which are at application level. Application level activities are also synchronized by these processes whenever required

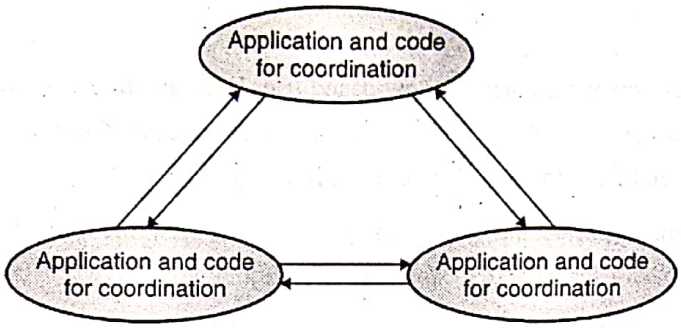


Fig. 1.6.2 : Distributed Computation with three peer processes

- In this computation, a delay incurred in communication with server processes is reduced. It leads to better interactive response among peer processes than server based architecture.

1.6.1(C) Variations in Basic Client Server Model

It is necessary to consider the following factors in design of client server model.

Mobile code and Mobile Agent

- Example of mobile code is downloading the applet code from web server by browser and running it locally. Sometimes server itself pushes up-to-date information to client. In this case client does not request server for the same information.
- Mobile agent is running program (code and data) which migrates from one machine to other in different network to perform a particular task.
- Mobile agent makes use of resources at machine where it is transferred. It may access database entries from database at this machine. Hence, communication cost can be reduced by avoiding the transfer of large size data from one machine to other machine.
- Mobile agents can be security threat at target machine where it is transferred. The environment receiving this mobile agent should check authentication of user of the mobile agent to access a particular resource. If mobile agent does not get required resource, it may not be able to complete the execution. Hence, mobile agents have limited applicability.

Network Computers

- Network computer downloads operating system and required application software from file server which is remote machine. In this case, although applications run on local machine but file server manages the files.
- The advantage of such configuration is that, user can work on any network machine as application data and code is stored at remote file server. The network computer need not have large memory capacity or processing power and hence cost can be reduced.
- Network computer uses resources at server. If disk is included then it stores minimum of software and used as cache to store recently accessed objects from remote file server. Cached objects at client are invalidated if gets updated at server.

The Clients

- Thin client is software layer which supports window-based user interface on computer where user is working whil application is executing on remote server. This remote computer is powerful machine capable of executing large number of applications. It can be cluster computer or multiprocessor machine.
- Applications like CAD or image processing when executes at server, user at thin client notices delay as image and vector information needs to transfer from thin client to application.

Mobile Devices and Spontaneous Networking

- Laptops, PDAs, digital cameras, smart watches, mobile phones, washing machines and many such devices which can be connected with wireless networking can be part of distributed system.
- If these devices are integrated in distributed system then it supports for mobile computing. In this environment, user with mobile devices can access local or remote services.
- Spontaneous networking integrates these mobile devices in network. Spontaneous networking encompasses applications that involve connection of both mobile and non-mobile devices in ore informal manner.
- Spontaneous networking offers easy connection to local network and easy integration with local services. A communication link is provided to devices and after connection established devices access the required services available in network.
- Spontaneous networking design raises issues such as convenient connection and integration if machine or device moves to other network. Addressing and routing algorithms differs in other networks and users may get limited connectivity while traveling. Security and privacy issues need to be addressed as user gets connected in new network.

1.6.1(D) Interfaces and Objects

- Server processes can be implemented in object oriented languages such as C++ and Java. These server processes o peer processes encapsulates the objects which can be accessed by remote processes. The object reference can b passed to remote process so that methods implemented in objects can be invoked by remote processes with remot invocation.
- CORBA and Java supports such remote method invocation (RMI). Set of functions for invocation in server or pe process is specified by one or more interface definitions. The client server model can be static or more dynam depending on decision about how to distribute responsibilities between processes and between machines in th network.
- In static architecture pre-allocation of responsibilities is done statically. For example, file server for file managemen Objects can be dynamically instantiated whenever required for invocation.

1.6.1(E) Design Requirements for Distributed Architectures

Performance Issues

Limited communication and processing capabilities of computers and networks is considered to discuss performan issues. These are as follows :

Responsiveness

- Users of interactive applications expect fast and consistent response. To access remote service, fast response is not dependent only on load and performance of server, but it can be due to delay in software components. These software components are client and server operating systems, middleware and code that implements remote service.
- There should be less software layers and less amount of data transfer between client and server to improve response time. Caching of data at client side also gives faster response. Accessing remote data may result in variation of delay. For text data short delay and for graphical images longer delay will be incurred.

Throughput

- It is computational work carried out per unit time. Throughput is affected by processing speed at client and server side and data transfer rate between client and server.
- Data transfer between various software layers in both the sender and receiving machine also affects the overall throughput. Network should also support for faster data transfer rate to improve throughput.

Balancing Loads

- Instead of running all the services, applications, processes components at server, it is better to shift some execution at client side. For example, web servers load is reduced by running applets at client side.
- Information can be replicated on many web servers to distribute the requests from many clients. Partially completed jobs can be migrated to other machine for execution to balance the load.

Quality of Service

- Clients and users experiences quality of service due to system properties such as reliability, security and performance. System configuration may change or resources may not available. This also affects the quality of service. How users or applications adapt to changing system configuration is important.
- Reliability and security aspects of system are important design issue. System should supports for failure free operations and failure should not affect the system performance. System should also provide security to get quality of service.
- Performance factors which are responsiveness and throughput is also important that affect quality of service. Faster response and better throughput improves quality of service.
- Stream oriented communication involves time critical data transfer and processing. For example, fetching movie video from web server. To guarantee quality of service, network resources should be available along the route of data transfer. These resources involve computational capacity, bandwidth, buffer space etc.

Caching and Replication

- Data replication and caching improves performance of the distributed system. Replication is placing same data or information on several machines. Response from web server is cached by client and proxy server. The cache consistency protocol configured with web browser ensures to get up-to-date copy of resource cached by client.

- In order to take care of performance, availability and operations that are disconnected, this up-to-date condition c be relaxed. Client and proxy server sends request to original web server to validate cached copy. In case of fail response, server responds with up-to-date copy of cached data.
- Web server keeps record of browsers and proxies which have accessed the resource. The expiry time for this cache resource is set by server. This estimated expiry time is based on last updated time of accessed resource. Respons from web server always contains expiry time of resource and current time.
- This cached response enables the browsers and proxies to know if it is likely to become stale for future request Cached response age gets compared with expiry time. Age of cached response involves sum of time of response fro server that has been cached and server time. This agreement of time is independent of the clock time at client, prox or web server.

Dependability

- The users of the designed system are dependent on correctness, security and fault tolerance of system. Security and fault tolerance is need and these parameters have impact on the architectural design of the system.
- Although fault occurs in hardware, software or network, the applications which are dependent on the system should carry out the designated task correctly. Multiple copies of the resources should be provided so that system and application software can be reconfigured to complete its task.
- There is practical limitation to make available redundant resources and hence limitations to achieve expected degree of fault tolerance. Redundancy can be possible if at architectural level, multiple machines are available to run the component process of the system. In the same way, multiple communication paths are also required to exchange multiple messages between processes.
- Data can be replicated at several locations so that data access can be possible from machine which is currently running and with no any types of fault in it. Replication of data also incurs the cost of maintaining consistency between al copies.
- Architectural impact due to need of security enforces keeping of sensitive data and other resources on machine which is secured against many attacks and vulnerabilities.

1.6.2 Fundamental Models

- Fundamental models are based on fundamental properties that permit the users to be more specific about th characteristics, risk of failure and security.
- These models mainly consider primary entities of the system, their interaction and characteristics which impacts th individual or cooperative behavior of the system. In fundamental model, the main aspects focused are interaction failure and security.

1.6.2(A) Interaction Model

- Server processes may interact with each other to provide services to client as per their requests. On the other hand peer processes may communicate with each other to achieve a common goal.

- In this scenario, many messages gets exchanged between these processes and state of these each interacting processes is private which cannot be accessed or updated by other process. Following factors affects interacting processes.

Communication Performance

- Latency, bandwidth and delay affect communication performance. Latency is time elapsed between start of transmission of message by sender process and when receiver process begins the receiving of the same message.
- Delay incurred in accessing the network that increases when network is heavily loaded due to traffic. Delay can be due to load at operating system at sender and receiver machines.
- Bandwidth of network decides transfer of information in given time. If more connections or channels use such network then bandwidth gets divided. Jitter is variation in consecutive packet or message delivery time. It is more related to multimedia data transfer.

Machine Clocks and Timing Events

- Different machines in the network have internal clock which is used by processes running on these machines to obtain current time. Practically, these clocks drift with respect to perfect time. Some machines clock can be leading or lagging with respect to perfect or standard time. Hence, processes running on different machines read different clock values as this machine's clock time are not same.
- The drift rate of different machines clock is not similar. The drift rate is defined as relative amount the machine clock differs with respect to perfect clock. Hence, although all machines clock are set at same time initially, in future these clocks would differ in time.
- Several algorithms are used to synchronize the clocks of computer in the network.

Two variants of interaction models

It is difficult to set time limit for execution of processes, delivery of messages to processes running on other machine and drift rate of clocks of different machines. Following are two examples of interaction model. One model requires strong assumption about time and other requires no assumption about time.

Synchronous Distributed Systems

- In synchronous distributed system, following bounds are defined :
- Lower and upper bounds of each process for their execution of each step are known in advance.
- Message will be received by each process from other process running on different machine in bounded time.
- The machine on which process is running have known bound about its local clocks drift rate with respect to real time clock.
- It is possible to model such synchronous distributed system. Practically, the above defined bounds should be guaranteed. It is necessary to give guarantee about required processor cycles and network capacity to use resources to complete tasks by processes within defines time bounds.

Asynchronous Distributed Systems

Internet is example of asynchronous distributed system. This system does not consider timing bounds. For example, i user is experiencing delay in receiving response from web server then web browser allows user to carry out other tasks. I this type of distributed systems, there are no bounds on :

- **Speed of process execution :** Processes which are communicating with each other may take different times to finis the particular step. For example, requesting process step may execute in nanosecond and responding process step o other machine may take longer time (may be second, minutes and more than that).
- **Delays in message transmission :** One message from process A may take time in picoseconds to deliver to process whereas other message from the same process may take hours.
- **Drift rate of clock :** It can be arbitrary for different clocks of different machines.
- Practically, asynchronous distributed system is very often as it is necessary to share processors and communicatio channels in network to accomplish the designated task. Hence, in asynchronous distributed system it is possible t implement the bounds defined above for synchronous distributed system. For example, multimedia data stream ca be delivered within bounded time in asynchronous distributed system.

Ordering of Events

- Ordering of events between communicating processes is very important to achieve consistency at different sites. Fo example, two servers in different cities A and B maintains bank database. For example, customer accounts balanc initially is Rs 1000. If customer deposited Rs 100 in city A and at the same time branch manager gives 1% interest o account balance in city B.
- These are two events related to same copy of database that is present on two different servers. Lets us say these events as event A (updating account balance by adding Rs 100) and event B (updating account balance by adding 1% interest). If these events executes in different order at these databases then inconsistency in account balances woul occur.
- In city A, if event A is executed first and then suppose event B is executed. In this case account balance in city A will b updated as: 1000 + 100 = 1100 and then adding Rs 11 interest = **Rs 1111.** In city B, if event B is executed first and the suppose event A is executed. In this case, account balance in city B will be updated as : 1000 + 10 = 1010 and the adding Rs 100 = Rs **1110.**
- These Event A and B messages arrive at both sites A and B in order explained above due to communication delay i network. If clocks of all machines in network show same time then ordering of messages can be done as pe timestamp when message was sent. Practically, it is not possible. Lamport proposed notion of logical time to arrang the messages at different sites in same order for execution. The logical clock concept will be explained in chapter 3.

1.6.2(B) Failure Model

In distributed system, communication channel, processes, machines can fail during the course of execution. Failur model suggest the ways in which failures in different components of the system may occurs and to provide th understanding of this failure.

Omission Failures

The omission failure refers to the faults that occur due to failure of process or communication channel. Because of this failure, process or communication channel fails to carry out the action or task that it is supposed to do.

Process Omission Failures

- The main omission failure of process is when it crashes and never executes its further action or program step. In this case, process completely stops. When any process does not responds to requesting process repeatedly, the requesting process detects or concludes this crash.
- The detection of crash in above manner relies on use of timeouts. In asynchronous system, timeout can point towards only that process is not responding. The process may have crashed, may be executing slowly or messages have yet not delivered to the system.
- If other process surely detects the crash of process then this process crash is called as fail-stop. This fail-stop behaviour can be formed in synchronous system when processes uses time out to know when other process fail to respond and delivery of messages are guaranteed.

Communication Omission Failures

- Sending process P executes *send* primitive and puts message in its outgoing buffer. Communication channel transport this message to receiving process Q's incoming buffer. Process Q then executes *receive* primitive to take this message from its incoming buffer. It then deliver the message.
- If communication channel is not able to transport message from process P's outgoing buffer to process Q's its incoming buffer then it produces omission failure. The message may be dropped due to non-availability of buffer space at receiving side, no buffer space at intermediate machine or network error detected by checksum calculation with data in message.
- Send-omission failures refer to loss of messages between sending process and its outgoing buffer. Receive- omission failures refer to loss of messages between incoming buffer and the receiving process.

Arbitrary Failures

- In this type of failure process or communication channel behaves arbitrarily. In this failure, the responding process may return wrong values or it may set wrong value in data item. Process may arbitrarily omits intentional processing step or carry out unintentional processing step.
- Arbitrary failure also occur with respect to communication channels. The examples of these failures are : message content may change, repeated delivery of the same messages or non-existent messages may be delivered. These failures occur rarely and can be recognized by communication software.

Timing Failures

- In synchronous distributed system, limits are set on process execution time, message delivery time and clock drift rate. Hence, timing failures are applicable to this system.
- In timing failure, clock failure affects process as its local clock may drift from perfect time or may exceeds bound on its rate.

- Performance failure affects process if it exceeds the defined bounds on the interval between two steps.
- Performance failure also affects communication channels if transmission of message take longer time than defined bound.

Masking Failures

- Distributed system is collection of many components and components are constructed from collection of other components. Reliable services can be constructed from the components which exhibit failures.
- For example, suppose data is replicated on several servers. In this case, if one server fails then other servers would provide the service. Service mask failure either by hiding it or by transforming it in more acceptable type of failure.

1.6.2(C) Security Model

- The architectural model discussed offers the basis for security model. The distributed system can be secured by providing security to processes and to the communication channels through which these processes communicates. Security also can be achieved by protecting the objects which these processes encapsulate against unauthorized access.
- **Protecting Objects :** Server keeps objects to which invocations come from clients at different machines. Invocations may be from process or user. These objects at server hold users private data or shared data such as web pages. It is necessary to provide access right to users with each invocation. Server should verify this authority of invocation and access rights for it.
- **Securing Processes and their Interactions :** Processes communicates with each other by sending the messages to each other. These messages are exposed against various attacks while under travel. Peer processes and server processes expose their interfaces through which other processes send them invocations. Integrity should be maintained for applications which handle financial transaction or confidential data.
- **Adversary :** Threat may come from legitimate machine in network or from machine which is connected in unauthorized way. Attacker may be capable of sending the messages to any process or may read or copy messages while in transmission. Incoming request to process may be from unauthorized source. This source address can be forged by attacker. Server also gets invocation requests from many clients. Server cannot accept or reject these requests without reliable knowledge of senders.
- **Uses of Security Models :** It is not true that, distributed system can be secured by securing communication channels and by applying access rights only. Encryption or access right techniques incur high processing and management cost. This analysis can help to develop threat model by considering all attacks, its sources and by minimizing the cost above.

1.7 Hardware Concepts

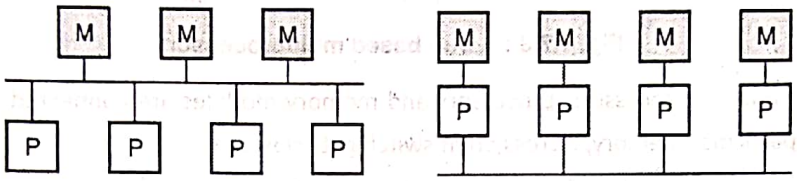
Hardware concepts illustrate the organization of hardware, their interconnection and the manner in which it communicates with each other. Multiple CPUs exist in distributed system.

The machines are basically divided in two groups

- **Multiprocessors :** Processors share the memory.
- **Multicomputers :** Each processor has its own private memory.

Both multiprocessors and multicomputers further divided in two categories on the basis of architecture of interconnection network. The basic organization is shown in Figs. 1.7.1 and 1.7.2.

- **Bus-based :** Machines are connected by single cable or processors are connected by bus and share the memory by communicating over the bus.
- **Switch-based :** Machines are connected with different wiring patterns and messages are routed along outgoing line by switching the switch.

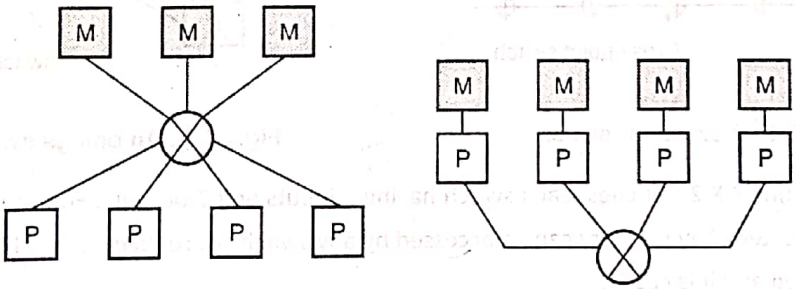


Bus-based shared memory multiprocessor Processors share the memory

Bus-based multicomputer. Each processor has its private memory

Fig. 1.7.1

- Multicomputers can be homogeneous or heterogeneous. Homogeneous multicomputers use same technology for interconnection network and all processors are similar, accessing the same amount of private memory.
- Heterogeneous multicomputers have different architecture for different machines. Machines are connected by different types of networks using different technology.



Processors share the memory and interconnected through switch

Processors have private memory and interconnected through switch

Fig. 1.7.2

1.7.1 Multiprocessors

- Multiple CPUs and memory modules are connected through bus. Single memory is available. CPU1 writes memory and CPU2 reads written value. CPU2 gets the same value which was written by CPU1. Memory having such property is said to be coherent.
- The disadvantage of the scheme is that, if few numbers of processors are connected then bus becomes overloaded and performance degrades. To overcome the problem, a high cache memory is placed between CPU and bus. If referenced word is in cache then it is accessed by CPU and bus is avoided. Typical size of cache is 512 KB to 1 MB.

- Again memory can be incoherent if updated word in one cache is not propagated to other copies of the word present in different cache memory : otherwise other copies will have old values. A bus-based multiprocessor provides less scalability.

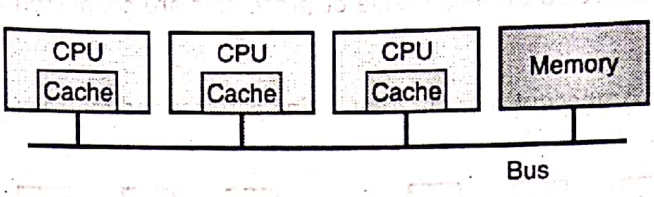


Fig. 1.7.3 : A bus-based multiprocessor

- To allow for more number of processors, processors and memory modules are connected by crossbar-switch. When CPU wants to access particular memory, a cross point switch gets closed.
- Several CPU can access different memory simultaneously but for two CPUs to access same memory at the same time one has to wait. It is shown in Fig. 1.7.4 Here for n CPUs and n memories, n^2 cross point switches are required.

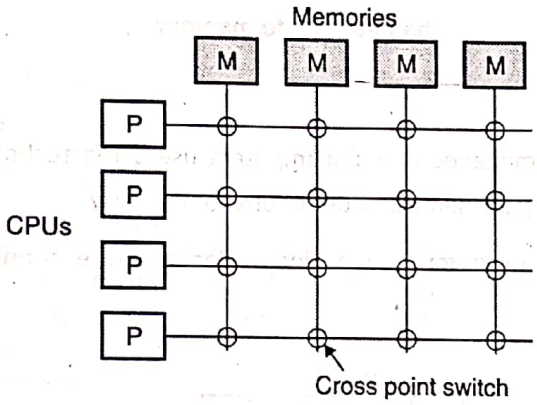


Fig. 1.7.4 : A crossbar switch

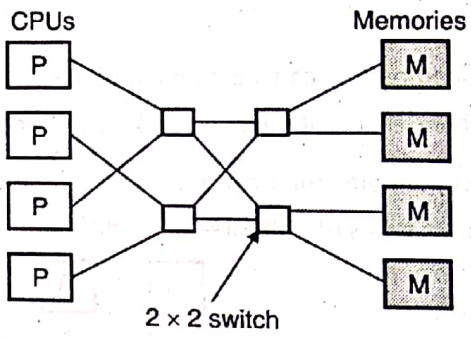
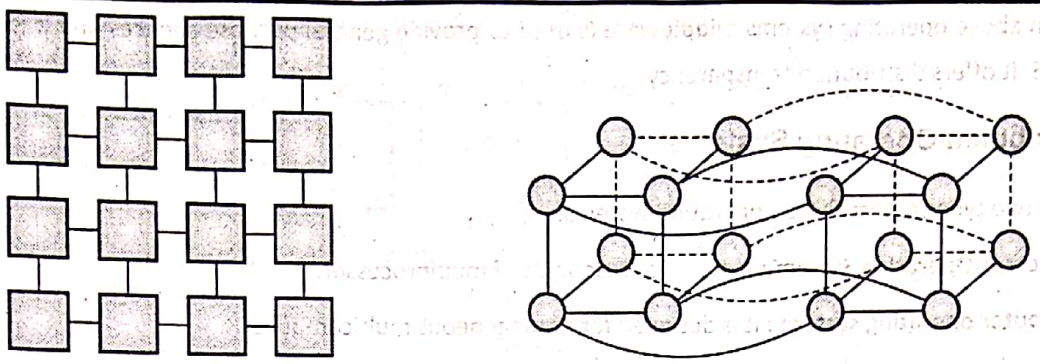


Fig.1.7.5 : An omega switching network

- Omega network contains 2 X 2 switches, each switch having 2 inputs and 2 outputs. For any input any one out of two output line can be selected. Any memory can be accessed by any switch. To reduce latency between CPU and memory, fast switching is needed and it is costly.
- Non-uniform memory access architecture allows the CPU to access its local memory fast and other CPU's memory is slowly accessed.

1.7.2 Homogeneous Multicomputer Systems

- Nodes are connected through high speed interconnection network. Processors communicate through interconnection network; therefore traffic is low with compare to traffic between processor and memory.
- If nodes are connected through interconnection networks like fast Ethernet then it is bus-based multicomputer. It has limited scalability and performance is low if few number of processors (25-100 nodes) added.
- Instead of broadcasting like in bus based multicomputers, routing of messages can be done through interconnection network. A grid is shown in Fig. 1.7.6 (a) and suitable for graph theory related problems. A 4-dimentional hypercube is shown in Fig. 1.7.6 (b). Vertices represent CPU and edges represent connection between CPUs.



(a) Grid (b) Hypercube

Fig. 1.7.6

- Massively parallel processors (MPPs) and Cluster of workstations are the examples of the switched multicomputers.

1.7.3 Heterogeneous Multicomputer Systems

- Today most of the distributed systems are built on the top of multicomputers having different processors, memory size etc. Interconnection network also have a different technology. These are called as heterogeneous multicomputers. In large scale heterogeneous multicomputer environment, services to application and its performance varies at different locations.
- Absence of global system view, intrinsic heterogeneity, scaling are the factors due to which there is a requirement of sophisticated software to built the distributed applications for heterogeneous multicomputers.

1.8 Software Concepts

Distributed systems and traditional operating systems provides the similar services such as :

- Both play the role of resource **managers** for the underlying hardware.
- Both permit the sharing of resources like CPUs, memories, peripheral devices, the network, and data of all kinds among multiple users and applications.
- Distributed systems try to hide the details and heterogeneous nature of the underlying hardware by offering a virtual machine on which applications can be executed without trouble.
- In a network multiple machines may have different operating system installed on it. Operating systems for distributed computers are categorized as :
- **Tightly coupled systems :** It keeps a single, global view of the resources it manages. Such tightly coupled operating system is called as distributed operating system (DOS). It is useful for the management of multiprocessors and homogeneous multicomputer. DOS hides details of underlying hardware. This hardware can be shared by many processes and details remains hidden.
- **Loosely-coupled systems :** In a set of computers, each has its own OS and there is coordination between operating systems to make their own services and resources available to the others. This loosely coupled OS is called as network operating system (NOS) and is used for heterogeneous multicomputer systems.

- Apart from above operating systems **middleware** is used to provide general purpose services and it is installed on th top of NOS. It offers distribution transparency.

1.8.1 Distributed Operating Systems

There are two types of distributed operating systems.

- **Multiprocessor Operating System :** It manages resources of multiprocessor.
- **Multicomputer operating system :** It is designed for homogeneous multicomputer.
- Distributed operating system is similar in functions to the traditional uniprocessor operating system. DOS handles the multiple processors.

Uniprocessor Operating Systems

- These are traditional operating systems and developed for single CPU. It permits the user and applications to share the different resources available in system. It allows the different applications to use same hardware in isolated manner. Simultaneously several applications executing on the same machine gets their required resources.
- Operating system protects data of one application from the other application if both are simultaneously executing. Applications should use only facilities provided by OS. To send the messages, applications should use communication primitives offered by OS.
- Operating system takes care of the way in which hardware resources are used and shared. For this purpose CPU supports for two modes of operation, kernel mode and user mode.
- In **kernel mode**, execution of all instructions is allowed to be carried out, and the entire memory and set of all registers is accessible throughout execution. On the contrary, in **user mode**, there is a restriction on memory and register access. When CPU executes operating system code, it switches to kernel mode. This switching from user mode to kernel mode occurs through system calls that are implemented by the operating system.
- If virtually all operating system code is executes in kernel mode then operating system is said to have monolithic architecture and it runs in single address space. The disadvantage of monolithic design approach is lack of flexibility. In this design, it is difficult to replace the components.
- Monolithic operating systems are not a good scheme from point of view of openness, software engineering, reliability, or maintainability. If the operating system is organized into two parts, it can provide more flexibility. In the first part, set of modules for managing the hardware is kept which can uniformly well be executed in user mode.
- For example, memory management module keeps track allocated and free space to the processes. It is required to execute in kernel mode at the time when registers of the Memory Management Unit (MMU) are set.
- A small **microkernel** contains only code that must execute in kernel mode. It is the second part of the operating system. Actually, a microkernel require only contain the code for, context switching, setting device registers, manipulating the MMU, and capturing hardware interrupts.
- Microkernel also contains the code to pass system calls to calls on the proper user level operating system modules, and to return their results. Following is the organization of operating system with this approach.

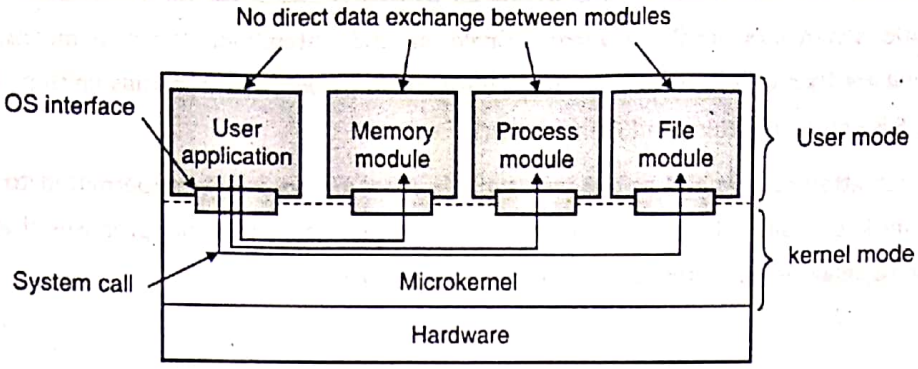


Fig. 1.8.1 : Separating applications from operating system code through a microkernel

Multiprocessor Operating Systems

- Unlike the uniprocessor operating systems, multiprocessor operating systems offers support for multiple processors having access to a shared memory. In this case, all data structures required by the operating system to deal with the hardware, together with the multiple CPUs, are placed into shared memory. Multiple processors can access these data. Hence protection against simultaneous access is needed to promise consistency.
- Modern operating systems are designed with the intension of handling multiple processors. The main aim of multiprocessor operating systems is to achieve high performance by means of multiple CPUs. A key goal is hide the presence of number of CPUs from the application.
- Such transparency can be attained and is relatively straightforward as different parts of applications communicates by using same primitives as those in multitasking uniprocessor operating systems. All the communication is made by manipulating data at shared memory locations, and it is required protect that data against concurrent access. This protection is achieved through synchronization primitives : semaphores and monitors.

Semaphores

- A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations : wait and signal. These operations were firstly termed P (for wait) and V (for signal).
- A semaphore S is integer variable whose value can be accessed and changed only by two operations wait (P or sleep or down) and signal (V or wakeup or up). Wait and signal are atomic operations.
- Binary semaphores do not assume all the integer values. It assumes only two values 0 and 1. On the contrary, counting semaphores (general semaphores) can assume only non-negative values.
- The wait operation on semaphores S, written as wait(S) or P(S), operates as follows :

```
wait(S) : IF  S > 0
          THEN S : = S – 1
          ELSE  (wait on S)
```

- The signal operation on semaphore S, written as signal(S) or V(S), operates as follows :

```
signal(S) : IF (one or more process are waiting on S)
            THEN (let one of these processes proceed)
            ELSE  S : = S + 1
```

- The two operations wait and signal are done as single indivisible atomic operation. It means, once a semaphore operation has initiated, no other process can access the semaphore until operation has finished. Mutual exclusion on the semaphore S is enforced within **wait(S)** and **signal(S)**.
- If many processes attempt a wait(S) at the same time, only one process will be permitted to proceed. The other processes will be kept waiting in queue. The implementation of wait and signal promises that processes will not undergo indefinite delay. Semaphores solve the lost-wakeup problem.

Monitors

- If semaphores are used incorrectly, it can produce timing errors that are hard to detect. These errors occur only if some particular execution sequences results and these sequences do not always occur.
- In order to handle such errors, researchers have developed high-level language constructs called as monitor.
- A monitor is a set of procedures, variables, and data structures that are all grouped together in a particular type of module or package. Processes may call the procedures in a monitor if required, but direct access to the monitor's internal data structures from procedures declared outside the monitor is restricted to the processes.

Following is the example of the monitor.

```
monitor example
integer i ;
condition c ;
procedure producer ( );
end ;
procedure consumer ( );
end ;
end monitor ;
```

- Monitors can achieve the mutual exclusion: only one process can be active in a monitor at a time.
- As monitors are a programming language construct, the compiler manages the calls to monitor procedures differently from other procedure calls.
- Normally, when a process calls a monitor procedure, if any other process is currently executing within the monitor, it gets checked.
- If so, the calling process will be blocked until the other process has left the monitor. If no other process is using the monitor, the calling process may enter.

The characteristics of a monitor are the following

- Only the monitor's procedures can access the local data variables. External procedures cannot access it.
- A process enters the monitor by calling one of its procedures.
- Only one process may be running in the monitor at a time; any other process that has called the monitor is blocked, waiting for the monitor to become available.

- Synchronization is supported by monitor with the use of **condition variables** that are contained within the monitor and accessible only within the monitor. Condition variables are a special data type in monitors, which are operated on by two functions :
- **cwait (c) :** the calling process's execution gets suspended on condition c. The monitor is now accessible for use by another process.
- **csignal (c) :** blocked process after a cwait (on the same condition) resumes its execution. If there are many such processes, choose one of them; if there is no such process, do nothing.
- Monitor wait and signal operations are dissimilar from those for the semaphore. If a process in a monitor signals and no task is waiting on the condition variable, the signal is lost.
- When one process is executing in monitor, processes that trying to enter the monitor join a queue of processes blocked waiting for monitor availability.
- Once a process is in the monitor, it may temporarily block itself on condition x by issuing cwait (x); it is then placed in a queue of processes waiting to reenter the monitor when the condition changes, and resume execution at the point in its program following the cwait (x) call.
- If a process that is executing in the monitor detects a change in condition variable x, it issues csignal (x), which alerts the corresponding condition queue that the condition has changed.
- A producer can add characters to the buffer only by means of the procedure append inside the monitor; the producer does not have direct access to buffer.
- The procedure first-checks the condition not full to determine if there is space available in the buffer. If not, the process executing the monitor is blocked on that condition.

Multicomputer Operating Systems

- In multicomputer operating systems data structures for systemwide resource management cannot simply shared by keeping them in physically shared memory. In its place, the only way of communication is through **message passing**. Following is the organization of multicomputer operating systems shown in Fig 18.2.

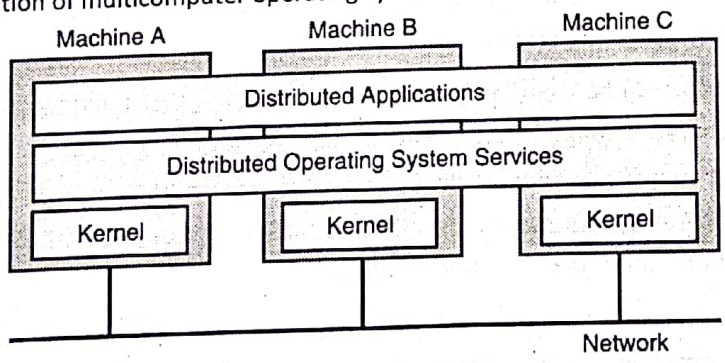


Fig. 1.8.2 : General structure of a multicomputer operating system

- Kernel on each machine manages local resources, for example memory, the local CPU, a local disk and others. Each machine contains separate module for sending and receiving messages to and from other machines.

- On the top of each local kernel there is a common software layer that implements the operating system as a virtual machine supporting parallel and concurrent execution of various tasks.
- This software layer offers a complete *software implementation* of shared memory. Further facilities usually implemented in this layer are, such as, assignment of task to processor, masking hardware failures, providing transparent storage, and general interprocess communication.
- Some multicomputer operating systems offers only message-passing facilities to applications and do not offer shared memory implementation. But different system can have different semantics of message-passing primitives. Their dissimilarities can be explained by taking into consideration whether or not messages are buffered. When should be blocking of sending or receiving process is also need to be considered as well.
- Buffering of the messages can be done at the sender's side or at the receiver's side. There are four synchronization points at which a sender or receiver can possibly block. At the sender's side, sender is blocked if buffer is full. If sender buffer is not present then three other points to block the sender are :
  - The message has been sent by sender.
  - The message has arrived at the receiver side.
  - The message has been delivered to the receiver application.
- Another issue that is significant to know message-passing semantics is whether or not communication is reliable. In reliable communication, a assurance of receiving the message by receiver is given to the sender.

Distributed Shared Memory Systems (DSMs)

- Programming for multicomputers is complex with compare to multiprocessors. As only message passing is available with multicomputers, programming becomes difficult with multicomputers. On the other hand, multiprocessor uses semaphores and monitors to access the shared data. So programming with multiprocessors is simple. In multicomputer case buffering, blocking, and reliable communication needs to be consider as well which leads to complex task of programming.
- Implementing the shared memory on multicomputers offers a virtual shared memory machine, running on a multicomputer. This shared memory model can be used to write the applications on multicomputers. The role of multicomputer operating system is important in this case.
- A large virtual address space can be created by using virtual memory of each individual node. With this approach a page based **distributed shared memory (DSM)** is realized. In DSM system, the address space is broken into pages of size 4 KB or 8 KB and kept over all the nodes in the system. If a processor reference fails to an address which is locally unavailable, a trap takes place, and the operating system fetches the page having the referenced address and starts again the faulting instruction, which now completes successfully.
- Following Fig. 1.8.3 shows an address space divided in 16 pages and four processors are available. It is just like normal paging. Here RAM of other machine is being used as the backing store instead of the local disk.

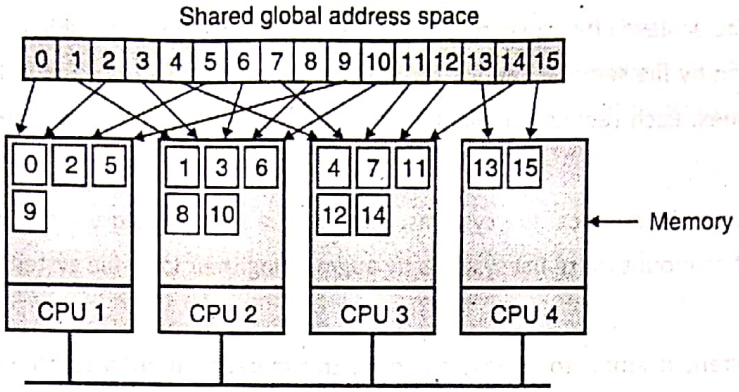


Fig. 1.8.3 : Pages are spread over 4 machines

- As shown in figure, if processor 4 references the instruction or data belonging to the pages 13 and 15 then it is done locally. If references to instructions and data are which are in the pages placed on other machines then trap to the operating system will occur.
- All read only pages can be replicated so that references will be done locally and performance can be improved. If read-write pages are replicated then if one of the copies gets modified other copies should also reflect the same changes. Write invalidate can be used to perform write on the page. Before performing write other copies are invalidated.
- If size of the pages kept large then it can reduce the total number of transfers when large section of contiguous data needs to be accessed. Conversely, if a page comprises data needed by two independent processes on different processors, the operating system may need to repetitively transfer the page between those two processors. This is called as **false sharing**.

1.8.2 Network Operating Systems

- Network operating systems assumes underlying hardware as heterogeneous. Whereas DOS assumes underlying hardware as homogeneous. In heterogeneous environment, machines and operating systems installed on them may be different. All these machines are connected to each other in computer network.
- NOS permit users to use services available on a particular machine in the network. Remote login service provided by NOS allows the user to log in remote machine from his/her terminal. Using command for remote copy user can copy the file from one machine to other.

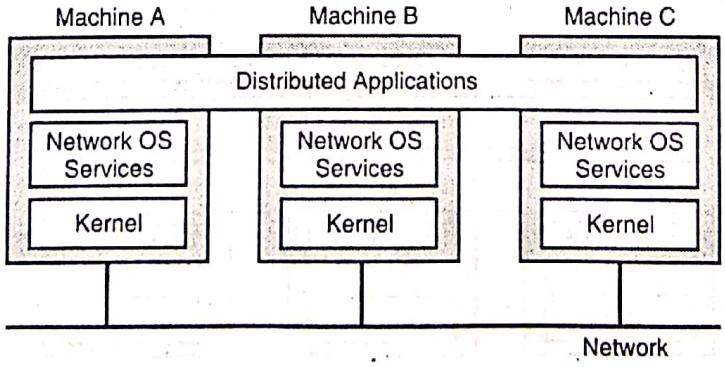


Fig. 1.8.4 : General structure of a network operating system

- Information sharing can be achieved by providing a shared, global file system accessible from all the workstations. The file system is implemented by **file servers**. The file servers accept requests to read and write files from user programs running on **client machines.** Each request of client request is executed by file server, and the reply is sent back to client.
- File servers usually supports hierarchical file systems, each with a root directory including subdirectories and files. Workstations can import or mount these file systems by augmenting their local file systems with those located on the servers.
- Distributed operating system attempts to achieve complete transparency in order to offer a single system view to the user. Whereas achieving full transparency with network operating system is not possible. As we have seen for remote login, user has to explicitly log into remote machine. In remote copy user knows the machine to which he/she is copying the files. The main advantage of network operating system is that it provides scalability.

1.9 Middleware

- Both DOS and NOS do not fulfil the criteria required for distributed system. A distributed operating system cannot manage a set of *independent* computers, while a network operating system does not provide transparency. A distributed system consisting of advantages of both DOS and NOS : the scalability and openness of network operating systems (NOS) and the transparency and related ease of use of distributed operating systems (DOS) is possible to realize by using middleware.
- A middleware layer can be added on the top of network operating systems which hide the heterogeneity of the collection of underlying platforms but as well get the better distribution transparency. The most of the modern distributed systems are build by means of such an additional layer of what is called **middleware.**

1.9.1 Positioning Middleware

- It is not possible to achieve distribution transparency as many distributed applications make direct use of the programming interface provided by network operating systems. Applications always make use of interfaces to the local file system as well.
- If additional layer of software is placed between applications and the network operating system then distribution transparency can be achieved and higher level of abstraction is provided. This layer is called **middleware** as shown in Fig. 1.9.1.

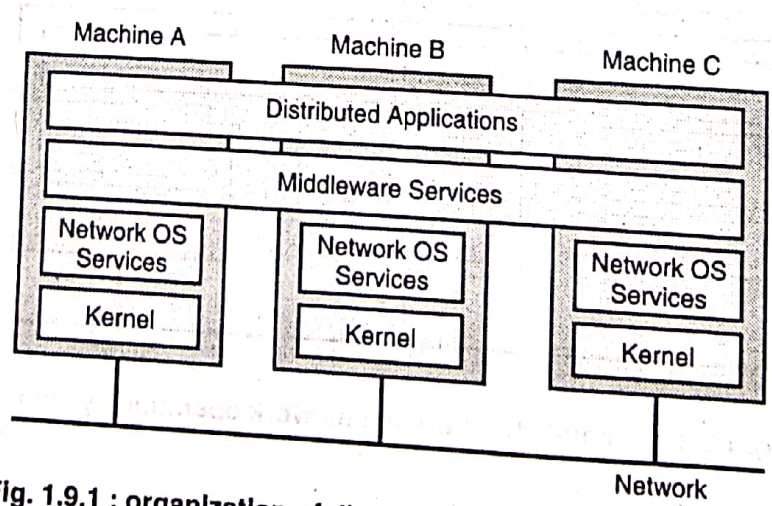


Fig. 1.9.1 : organization of distributed system as middleware

- In this organization local resource management is handled by NOS. It also provides simple communication means to connect to other computers in the network. A major goal is to hide heterogeneity of the underlying platforms from applications. As a result, many middleware systems offer an essentially complete set of services.

1.10 Models of Middleware

- In order to ease the development and integration of distributed applications most middleware is uses some model to express distribution and communication. A relatively simple model is that of treating everything as a file.
- The approach of treating everything as a file is introduced in UNIX and also followed in plan 9. Here, all devices such as keyboard, mouse, disk, network interface, etc, are treated as files. In essence, whether a file is local or remote makes no difference. For reading or writing bytes in file, an application first opens the file and then perform read or write operation. Once the operation is done it closes file again. As files can be shared by many processes, communication reduces to just accessing the same file.
- Another model of middleware is based on **distributed file systems**. This model supports distribution transparency only for files that only stores data. This model became popular as it is reasonably scalable.
- In **Remote Procedure Calls (RPCs)** based middleware model, a client side process calls a procedure implemented on a remote machine. In this call, parameters are transparently sent to the remote machine (server) where the procedure actually executed. The result of execution then server sent back to the caller. Although called procedure is executed on remotely, it appears as call was executed locally. In this case the communication with remote process remains hidden from calling process.
- Similar to calling the remote procedure in RPC, it is also possible to invoke objects residing on remote computers in a transparent manner. Many middleware systems offer a concept of **distributed objects**. The main idea behind distributed objects is that each object implements an interface that hides all the internal details of the object from its users. An interface contains the methods that the object implements. The process notices of an object are its interface.
- In the implementation of distributed objects, object resides on single machine and its interface is kept on several other machines. The available interface of the object on invoking process's machine converts its method invocation into a message that is sent to the object. After receiving method invocation message of process, object executes the invoked method and sends back the result. The interface implementation then converts the reply message into a return value, and submits to the invoking process. Similar to RPC, the invoking process remains totally unaware of the network communication.
- The World Wide Web became successful due to the very simple but very much effective model of **distributed documents**. In web model, information is organized in the form of documents. Each of the document is kept on a machine transparently sited anywhere in the wide area network.
- Links in the documents refer to other documents. By using link referring to particular document, that document is obtained from its location and gets displayed on the user's machine. Apart from textual documents, web also supports for audio, video and interactive graphic-based documents.

1.11 Services Offered by Middleware

- All middleware implements access transparency. For this, they provide high-level **communication facilities** to hide the low-level message transmitting through computer networks. How communication is supported depends very much on the model of distribution the middleware offers to users and applications.

- **Naming** service is offered by almost all middleware. Due to name services offered by middleware it is possible to share and search for the entities. Naming service becomes complex to deal with if scalability is considered. For efficient searching of a name in a large-scale system, the place of the entity that is named must be assumed to be fixed. This main difficulty which is needed to be handled. In World Wide Web (WWW), URL is used to refer to the document. Server on which this document is stored, its name is present in URL. Therefore, if the document is migrated to another server, its URL fails to search the document.
- **Persistence** service is offered for storage purpose. Persistence service is provided through a distributed file system in simple way, but many advanced middleware have integrated databases into their systems. If not, it provides facilities for applications to connect to databases.
- Many middleware offers facility for **distributed transactions** if data storage is important. Transactions permit multiple read and write operations to take place atomically. Atomicity is the property where either transaction commits so all its write operations are actually performed, or it fails, leaving all referenced data unchanged. Distributed transactions operate on data which is distributes across multiple machines.
- The important service provided by middleware is **security.** Instead of depending on the underlying local operating systems to sufficiently support security for the entire network, security has to be partly implemented additionally in the middleware layer itself.

Middleware and Openness

- Modern distributed systems are built as middleware for different operating systems. Due to this, applications developed for a specific distributed system become operating system independent and more dependent on specific middleware. The difficulty arises due to less openness of middleware.
- Actual open distributed system is specified through interfaces that are complete. Completeness states that the details required for implementing the system, has surely been specified. If interfaces are not complete then system developers must add their own interfaces. As a result, situation arises in which two middleware systems developed by different teams follow the same standard, but applications developed for one system cannot ported to the other without trouble.
- Due to incompleteness, although two different implementations implement precisely the same set of interfaces but different underlying protocols, they cannot interoperate.

Comparison between DOS, NOS and Middleware

Sr. No.	Distributed OS (DOS)	Network OS(NOS)	Middleware based DS
1.	1. Degree of transparency is very high for multiprocessor OS. 2. Degree of transparency is high for multicomputer OS.	Degree of transparency is low.	Degree of transparency is high
2.	Same operating system is present on all nodes in both cases (Multiprocessor OS and Multicomputer OS).	Operating system on different nodes is different.	Operating system on different nodes is different.

Sr. No.	Distributed OS (DOS)	Network OS(NOS)	Middleware based DS
3.	1. As multiple processors present in single machine, number of copies of multiprocessor OS is one. 2. As multiple machines are present multiple (n) copies of multicomputer OS are required.	As multiple machines are present multiple (n) copies of NOS are required.	Middleware is installed on the top of NOS on every machine. Hence multiple machines are preset so multiple copies of OS are required.
4.	1. In multiprocessor OS basis for communication is shared memory. Processors share the memory to communicate with each other. 2. In case of multicomputer OS, Many machines communicate with each other by passing the messages. Message passing is basis for communication	In network OS, basis for communication is files.	In middleware based DS, basis for communication is model specific.
5.	1. In multiprocessor OS resource management is global and central. 2. In multicomputer OS resource management is global and distributed	In network OS resource management is per node. Hence it is easy to scale the system.	Resource management is per node
6.	1. In multiprocessor OS there is no scalability as single machine in which multiple processors are presents 2. Multicomputer OS supports for moderate scalability..	Network operating system offers the scalability.	In middleware based distributed system scalability varies.
7.	Both multiprocessor OS and multicomputer OS are not open. These OS are developed to optimize the performance.	Network operating system is openness. Modern operating systems are built with microkernel design.	It is also open.

1.12 Client Server Model

1.12.1 Clients and Servers

- In client-server model, there are two types of processes. A server process implements the particular type of service. A client process requiring the service implemented by server process sends request to server. Client after sending request waits until reply comes from the server.

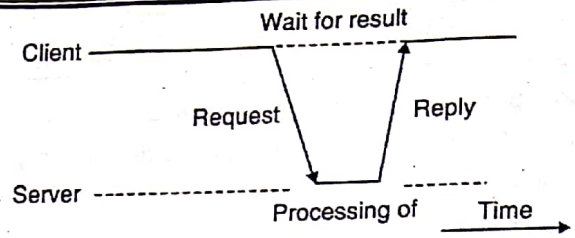


Fig. 1.12.1 : Request-reply based interaction between client and server

- If underlying network is reliable like LAN then connectionless protocol can be used to built client and server. Us connectionless protocol communication is efficient and if there is no loss of packets then request reply interacti work well.0
- Reliable connection oriented protocol cannot be proper solution in LAN but it works well in wide area networks whe communication is basically reliable. In connection oriented protocol setting up and terminating the connection expensive if less number of request and reply messages are exchanged.

1.12.2 Application Layering

Client server application is partitioned between following three layers :

1. User-interface level
2. The processing level
3. The data level

1. User- interface level

- User-interface level is implemented in clients. This level comprises the code that permits the end user to interact wi application. The simple user-interface program is character based screen.
- Graphical displays with pop-up and pull down menus can be used at client side. X-Windows-interfaces is one of t example.

2. The processing level

- The core part of the application which operates on database or file system is considered at processing level. In sear engines example, user type keyword to be searched is the user interface.
- At back end there is huge database of web pages. The core logic here is to convert the typed keywords by user database queries. Then ranking the result into list and transforming the list in html pages.

3. Data level

- In client-server model, this level contains programs which maintain the data on which applications perform th operations. This data is persistent.
- Although application is not running, still data remains stored at somewhere.

1.12.3 Client Server Architectures

- The above discussed three levels suggest the different ways to distribute client server application across different machines in the distributed system.
- The simple organization is to keep interface level on client machine and other two levels (processing and data level) on server machine.

Multitiered Architectures

The programs in different levels described above can be distributed among different machines to organizing the client and servers. Following are the different possibilities for two tier architecture :

- Terminal dependent part of the user interface on client machine and user interface application and database on server machine.
- Place entire user interface on client machine and application and database on server machine.
- Place entire user interface and some part of application (front end) on client machine and other part of application and database on server machine. Validation of filled forms is done at client side.
- Place entire user interface and entire application on client machine and database on server machine. In this case client contact only for operations of data to server.
- Place entire user interface and entire application and some part of database (for example cached web pages on local disk) on client machine and database on server machine.

Modern architectures

- One way is vertical distribution where logically different components are placed on different machines. This is just like multitiered architecture. This is related to vertical fragmentation concept in distributed relational databases. In this case, table columns are split and kept on many machines.
- In horizontal distribution, client or server is physically split up in logical parts. Here each part operates on its own share of data set to balance the load. For example, web pages are distributed among many servers and client request is forwarded to these server in round robin fashion.

Review Questions

Q. 1 What is distributed System? What are the main characteristics of distributed system?

Q. 2 Give and explain examples of distributed system.

Q. 3 Write note on "Resource Sharing and Web".

Q. 4 Explain different issues and goals related to design of distributed system.

Q. 5 What is meant by distribution transparency? Explain different types of transparency.

Q. 6 Explain different scaling techniques.

Q. 7 Explain different types of distributed system.

Q. 8 What is open distributed system? Explain in detail.

Q. 9 Explain main types of architectural models of distributed system.

Q. 10 Explain software layers in architectural model of distributed system.

Q.11 What are the different variations in basic client server model of distributed system?

Q. 12 Write note on "interfaces and objects".

Q. 13 What are the design requirements for distributed architectures?

Q. 14 Explain different fundamental models of distributed system.

Q. 15 Explain interaction model of distributed system.

Q. 16 Explain two variants of interaction model.

Q. 17 Write note on "Failure Model".

Q. 18 Explain different types of failures in distributed system.

Q. 19 Write note on "security model" of distributed system.

Q. 20 What is multiprocessor and multicomputer? Explain its basic organization with diagram.

Q. 21 Write note on "Homogeneous Multicomputer System".

Q. 22 Write note on "Heterogeneous Multicomputer System".

Q. 23 What is distributed operating system (DOS)? What are the types of DOS?

Q. 24 Write note on "Distributed Shared Memory Systems (DSMs)"

Q. 25 Explain network operating system. (NOS).

Q. 26 What are the different models of middleware?

Q. 27 Explain different services provided by middleware.

Q. 28 Differentiate between DOS, NOS and middleware based distributed system.

Q. 29 Explain different client server models.