Distributed Operating System(DOS)

Operating System

An **operating system** (OS) is system software that manages computer hardware, software resources, and provides common services for computer programs.

OR

Operating System acts as an interface between user and the hardware.

USER
APPLICATION
OPERATING SYSTEM
HARDWARE

1.Hardware – provides basic computing resources (CPU, memory, I/O devices).

2.Operating system – controls and coordinates the use of the hardware among the various application programs for the various users.

3.Applications programs – define the ways in which the system resources are used to solve the computing problems of the users (compilers, database systems, video games, business programs).

4.Users (people, machines, other computers).

Types of Operating System

1. Batch OS
2. Distributed OS
3. Multitasking OS
4. Network OS
5. Real-OS
6. Mobile OS

Batch OS

Batch OS is the first operating system for second-generation computers. This OS does not directly interact with the computer. Instead, an operator takes up similar jobs and groups them together into a batch, and then these batches are executed one by one based on the first-come, first, serve principle.

Multitasking OS

The multitasking OS is also known as the time-sharing operating system as each task is given some time so that all the tasks work efficiently. This system provides access to a large number of users, and each user gets the time of CPU as they get in a single system. The tasks performed are given by a single user or by different users. The time allotted to execute one task is called a quantum, and as soon as the time to execute one task is completed, the system switches over to another task.

Network operating systems

Network operating systems are the systems that run on a server and manage all the networking functions. They allow sharing of various files, applications, printers, security, and other networking functions over a small network of computers like LAN or any other private network.

Real-Time operating systems

Real-Time operating systems serve real-time systems. These operating systems are useful where many events occur in a short time or certain deadlines, such as real-time simulations.

Types of the real-time operating system are:

- **Soft real-time OS**

The soft real-time OS is the operating system for applications where time constraint is not very strict.

In a soft real-time system, an important task is prioritized over less important tasks, and this priority remains active until the completion of the task. Furthermore, a time limit is always set for a specific job, enabling short time delays for future tasks, which is acceptable. For Example, virtual reality, reservation system, etc.

- **Hard real-time OS**

The hard real-time OS is the operating system for mainly the applications in which the slightest delay is also unacceptable. The time constraints of such applications are very strict. Such systems are built for life-saving equipment like parachutes and airbags, which immediately need to be in action if an accident happens.

Mobile OS

A mobile OS is an operating system for smartphones, tablets, and PDA’s. It is a platform on which other applications can run on mobile devices.

Distributed Operating System

- A DOS is a system which contains multiple components located on different machines, which coordinate and communicate actions in order to appear as a single coherent working system to the user.

- A distributed operating system (DOS), are systems which model where distributed applications are running on multiple computers, linked by communications

WHAT IS A DISTRIBUTED COMPUTING SYSTEM?

Computer architectures consisting of interconnected, multiple processors are basically of two types:

I. Tightly coupled systems. In these systems, there is a single systemwide primary memory (address space) that is shared by all the processors [Fig. I.1(a)]. If any processor writes, for example, the value 100 to the memory location x, any other processor subsequently reading from location x will get the value 100. Therefore, in these systems, any communication between the processors usually takes place through the shared memory.

Tightly coupled systems are referred to as parallel processing systems, and loosely coupled systems are referred to as distributed computing systems, or simply distributed systems.

In contrast to the tightly coupled systems, the processors of distributed computing systems can be located far from each other to cover a wider geographical area. Furthermore, in tightly coupled systems, the number of processors that can be usefully deployed is usually small and limited by the bandwidth of the shared memory.

- 2. Loosely coupled systems. In these systems, the processors do not share memory, and each processor has its own local memory [Fig. I.1(b)]. If a processor writes the value 100 to the memory location x, this write operation will only change the contents of its local memory and will not affect the contents of the memory of any other processor. Hence, if another processor reads the memory location x, it will get whatever value was there before in that location of its own local memory. In these systems, all physical communication between the processors is done by passing messages across the network that interconnects the processors.

In short, a distributed computing system is basically a collection of processors interconnected by a communication network in which each processor has its own local memory and otherperipherals, and the communication between any two processors of the system takes place by message passing over the communication network. For a particular processor, its own resources are local, whereas the other processors and their resources are remote. Together, a processor and its resources are usually referred to as a node or site or machine of the distributed computing system.

Advantages of Distributed Computing

1. Cost Effective

Although distributed systems consists of high implementation costs, they are cost effective in the long run. Compared to a mainframe computer, where a single system is composed of several processors, the distributed system is made up of several computers together. This type of infrastructure is far more cost effective than a mainframe system.

2. Efficiency

Distributed systems are made to be efficient in every aspect since they posses multiple computers. Each of these computers could work independently to solve problems. This not only considered to be efficient, it significantly saves time of the user.

3. Scalability

Distributed systems are made on default to be scalable. Whenever there is an increase in workload, users can add more workstations. There is no need to upgrade a single system. Moreover, no any restrictions are placed on the number of machines. Which means that, these machines will be able to handle high demand workload easily.

4. Reliability

The distributed systems are far more reliable than single systems in terms of failures. Even in the case of a single node malfunctioning, it does not pose problems to the remaining servers. Other nodes can continue to function fine.

5. Latency

Distributed systems results on low latency. If a particular node is located closer to the user, the distributed system makes sure that the system receives traffic from that node. Thus, the user could notice much less time it takes to serve them.

Disadvantages of Distributed Computing

Difficult troubleshooting: Troubleshooting and diagnostics are more difficult due to distribution across multiple servers.

Less software support: Less software support is a major drawback of distributed computer systems.

High network infrastructure costs: Network basic setup issues, including transmission, high load, and loss of information.

Security issues: The characteristics of open systems make data security and sharing risks in distributed computer systems.

Examples of Distributed Computing

1: Telecommunication networks:

- telephone networks and cellular networks
- computer networks such as the Internet
- wireless sensor networks
- routing algorithms

2: Network Applications:

- World Wide Web and peer-to-peer networks
- massively multiplayer online games and virtual reality communities
- Distributed databases and distributed database management systems
- network file systems
- distributed cache such as burst buffers
- distributed information processing systems such as banking systems and airline reservation systems

3: Real-Time Process Control:

- aircraft control systems
- industrial control systems

4: Parallel Computation:

- scientific computing, including cluster computing,
- Grid computing
- Coud computing, and various volunteer computing projects
- distributed rendering in computer graphics

Distributed Computing Models

Various models are used for building distributed computing systems. These models can be broadly classified into five categories-

1. Minicomputer
2. Workstation
3. Workstation-server
4. Processor-pool
5. Hybrid

Minicomputer model

- The minicomputer model is a simple extension of the centralized time-sharing system.
- A distributed computing system based on this model consists of a few minicomputers interconnected by a communication network were each minicomputer usually has multiple users simultaneously logged on to it.
- Several interactive terminals are connected to each minicomputer.Each user logged on to one specific minicomputer has remote access to other minicomputers.

- The network allows a user to access remote resources that are available on some machine other than the one on to which the user is currently logged.The minicomputer model may be used when resource sharing with remote users is desired.
- The early ARPA net is an example of a distributed computing system based on the minicomputer model.

Minicomputer model

Workstation Model

- A distributed computing system based on the workstation model consists of several workstations interconnected by a communication network.
- An organization may have several workstations located throughout an infrastructure were each workstation is equipped with its own disk & serves as a single-user computer.
- In such an environment,at any one time a significant proportion of the workstations are idle which results in the waste of large amounts of CPU time.

- Therefore,the idea of the workstation model is to interconnect all these workstations by a high-speed LAN so that idle workstations may be used to process jobs of users who are logged onto other workstations & do not have sufficient processing power at their own workstations to get their jobs processed efficiently.
- Example:Sprite system & Xerox PARC.

Workstation Model

Workstation–Server Model

- The workstation model is a network of personal workstations having its own disk & a local file system.
- A workstation with its own local disk is usually called a diskful workstation & a workstation without a local disk is called a diskless workstation.Diskless workstations have become more popular in network environments than diskful workstations,making the workstation-server model more popular than the workstation model for building distributed computing systems.
- A distributed computing system based on the workstation-server model consists of a few minicomputers & several workstations interconnected by a communication network.

- In this model,a user logs onto a workstation called his or her home workstation.Normal computation activities required by the user's processes are performed at the user's home workstation,but requests for services provided by special servers are sent to a server providing that type of service that performs the user's requested activity & returns the result of request processing to the user's workstation.
- Therefore,in this model,the user's processes need not migrated to the server machines for getting the work done by those machines.
- Example:The V-System.

Workstation–Server Model

Processor–Pool Model:

- The processor-pool model is based on the observation that most of the time a user does not need any computing power but once in a while the user may need a very large amount of computing power for a short time.
- Therefore,unlike the workstation-server model in which a processor is allocated to each user,in processor-pool model the processors are pooled together to be shared by the users as needed.
- The pool of processors consists of a large number of microcomputers & minicomputers attached to the network.

- Each processor in the pool has its own memory to load & run a system program or an application program of the distributed computing system.
- In this model no home machine is present & the user does not log onto any machine.
- This model has better utilization of processing power & greater flexibility.
- Example:Amoeba & the Cambridge Distributed Computing System.

Processor–Pool Model:

Hybrid Model:

- The workstation-server model has a large number of computer users only performing simple interactive tasks &-executing small programs.
- In a working environment that has groups of users who often perform jobs needing massive computation,the processor-pool model is more attractive & suitable.
- To combine Advantages of workstation-server & processor-pool models,a hybrid model can be used to build a distributed system.
- The processors in the pool can be allocated dynamically for computations that are too large or require several computers for execution.
- The hybrid model gives guaranteed response to interactive jobs allowing them to be more processed in local workstations of the users

Why are distributed computing systems gaining popularity

1. **Inherently Distributed Applications:**

   Several applications are inherently distributed in nature and require distributed computing system for realization. These applications might have distributed locations for collecting, processing and accessing data resulting in need of distributed computing systems.

   Example: A computerized banking system, airline reservation

2. **Information sharing among Distributed users:**

   - Another reason for emergence of distributed computing systems was the desire for efficient person-to-person communication facility by sharing information over distances. In distributed systems information generated by one user can be easily shared by other users for many ways.

   Example: Exchange msg by email

3. Resource Sharing:

- Sharing of software resources such as software libraries and databases can be done in an efficient way among the users of a single distributed computing system. Cloud storage can be an example.

4. Price-Performance Ratio:

- This is one of the most important reasons for growing popularity as the increased power and reduced microprocessor price is combined with fast communication network that provide better price-performance ratio.
- Resource sharing is allowed in distributed computing which makes it cost-effective.

5. Shorter Response Time and higher Throughput:

- Multiplicity of processors enable distributed computing systems to have better performance. Most common performance metrics are response tine and throughput of user processes. Example is the workstation model where the users have multiple processes that run simultaneously and interact with each other.

6. Higher Reliability:

- Reliability refers to the degree of tolerance against errors and component failures in a system. A reliable system prevents loss of information even in the event of component failure. The multiplicity of storage devices and processors in a distributed computing system allow the maintenance of multiple copies of information which still provide data if one site is down.

7. Extensibility and Incremental Growth:

- It is possible to extend the power and functionality of a distributed computing system by adding additional resources to the system. Extensibility is also easier because addition of resources can be done without disruption of the normal functioning of the system.

8. Better flexibility in Meeting Users Needs:

- A distributed computing system may have a pool of different types of computers from which the most appropriate one can be selected for processing a user's job. Hybrid model can be an example for this.

Issues in designing distributed systems:

1. Heterogeneity

The Internet enables users to access services and run applications over a heterogeneous collection of computers and networks. Internet consists of many different sorts of network their differences are masked by the fact that all of the computers attached to them use the Internet protocols to communicate with one another.

2. Openness

The openness of a computer system is the characteristic that determines whether the system can be extended and re-implemented in various ways.The openness of distributed systems is determined primarily by the degree to which new resource-sharing services can be added and be made available for use by a variety of client programs.

3. Security

Many of the information resources that are made available and maintained in distributed systems have a high intrinsic value to their users.Their security is therefore of considerable importance. Security for information resources has three components: confidentiality, integrity, and availability.

4. Scalability

Distributed systems operate effectively and efficiently at many different scales, ranging from a small intranet to the Internet. A system is described as scalable if it will remain effective when there is a significant increase in the number of resources and the number of users.

5. Failure handling

Computer systems sometimes fail. When faults occur in hardware or software, programs may produce incorrect results or may stop before they have completed the intended computation. Failures in a distributed system are partial – that is, some components fail while others continue to function. Therefore the handling of failures is particularly difficult.

6. Concurrency

Both services and applications provide resources that can be shared by clients in a distributed system. There is therefore a possibility that several clients will attempt to access a shared resource at the same time. Object that represents a shared resource in a distributed system must be responsible for ensuring that it operates correctly in a concurrent environment.

7. Transparency: The distributed systems should be perceived as a single entity by the users or the application programmers rather than as a collection of autonomous systems, which are cooperating. The users should be unaware of where the services are located and also the transferring from a local machine to a remote one should also be transparent

Types of Transparencies

1. Access Transparency: Clients should be unaware of the distribution of the files. The files could be present on a totally different set of servers which are physically distant apart and a single set of operations should be provided to access these remote as well as the local files. Applications written for the local file should be able to be executed even for the remote files. The examples illustrating this property are the File system in Network File System (NFS), SQL queries, and Navigation of the web.

2. Location Transparency: Clients should see a uniform file name space. Files or groups of files may be relocated without changing their pathnames. A location transparent name contains no information about the named object's physical location. This property is important to support the movement of the resources and the availability of services. The location and access transparencies together are sometimes referred as Network transparency. The examples are File system in NFS and the pages of the web.

3. Concurrency Transparency: Users and Applications should be able to access shared data or objects without interference between each other. This requires very complex mechanisms in a distributed system, since there exists true concurrency rather than the simulated concurrency of a central system. The shared objects are accessed simultaneously. The concurrency control and its implementation is a hard task. The examples are NFS, Automatic Teller machine (ATM) network. CSE 6306 Advanced Operating Systems

4. Replication Transparency: This kind of transparency should be mainly incorporated for the distributed file systems, which replicate the data at two or more sites for more reliability. The client generally should not be aware that a replicated copy of the data exists. The clients should also expect operations to return only one set of values. The examples are Distributed DBMS and Mirroring of Web pages.

5. Failure Transparency: [4] Enables the concealment of faults, allowing user and application programs to complete their tasks despite the failure of hardware or software components. Fault tolerance is provided by the mechanisms that relate to access transparency. The distributed system are more prone to failures as any of the component may fail which may lead to degraded service or the total absence of that service. As the intricacies are hidden the distinction between a failed and a slow running process is difficult. Examples are Database Management Systems.

6. Migration Transparency: This transparency allows the user to be unaware of the movement of information or processes within a system without affecting the operations of the users and the applications that are running. This mechanism allows for the load balancing of any particular client, which might be overloaded. The systems that implement this transparency are NFS and Web pages.

7. Performance Transparency: Allows the system to be reconfigured to improve the performance as the load varies.

8. Scaling Transparency: A system should be able to grow without affecting application algorithms. Graceful growth and evolution is an important requirement for most enterprises. A system should also be capable of scaling down to small environments where required, and be space and/or time efficient as required. The best-distributed system example implementing this transparency is the World Wide Web.

8. Quality of service

Once users are provided with the functionality that they require of a service, such as the file service in a distributed system, we can go on to ask about the quality of the service provided. The main nonfunctional properties of systems that affect the quality of the service experienced by clients and users are reliability, security and performance. Adaptability to meet changing system configurations and resource availability has been recognized as a further important aspect of service quality.

9. Reliability

The idea is that if a machine goes down, some other machine takes over the job. A highly reliable system must be highly available, but that is not enough. Data entrusted to the system must not be lost or garbled in any way, and if files are stored redundantly on multiple servers, all the copies must be kept consistent. In general, the more copies that are kept, the better the availability, but the greater the chance that they will be inconsistent, especially if updates are frequent.

10. Performance

Always the hidden data in the background is the issue of performance. Building a transparent, flexible, reliable distributed system, more important lies in its performance. In particular, when running a particular application on a distributed system, it should not be appreciably worse than running the same application on a single processor. Unfortunately, achieving this is easier said than done.

Computer Networks

Computer Networks

- A computer network is defined as the interconnection of two or more computers.
- A computer network is a communication system that links end systems by communication lines and software protocols to exchange data between two processes running on different end systems of the network.
- The end systems are often referred to as nodes, sites, hosts, computers, machines, and so on. The nodes may vary in size and function.

NETWORKS TYPES

Networks are broadly classified into two types:

- local area networks (LANs)
- wide-area networks (WANs). OR long-haul.

The key characteristics that are often used to differentiate between these two types of networks are

1. Geographic distribution: The main difference between the two types of networks is the way in which they are geographically distributed. A LAN is restricted to a limited geographic coverage of a few kilometers, but a WAN spans greater distances and may extend over several thousand kilometers.

2. Data rate. Data transmission rates are usually much higher in LANs than in WANs .Transmission rates in LANs usually range from 0.2 megabit per second (Mbps) to 1 gigabit per second (Gbps). On the other hand, transmission rates in WAN usually range from 1200 bits per second to slightly over 1Mbps.

3. Error rate. Local area networks generally experience fewer data transmission errors than WANs do.

4. Communication link. The most common communication Jinks used in LANs are twisted pair, coaxial cable, and fiber optics. On the other hand, since the sites in a WAN are physically distributed over a large geographic area, the communication links used are by default relatively slow and unreliable. Typical communication links used in WANs are telephone lines, microwave links, and satellite channels.

5. Ownership. A LAN is typically owned by a single organization because of its limited geographic coverage. A WAN, however, is usually formed by interconnecting multiple LANs each of which may belong to a different organization. Therefore, administrative and maintenance complexities and costs for LANs are usually much lower than for WANs.

6. Communication cost. The overall communication costs of a LAN is usually much lower than that of a WAN.

LAN Topologies

Topology is physical or logical arrangement of computer. The physical topology of a network refers to the configuration of cables, computers, and other peripherals.

Types of Topologies:

- Bus Topology
- Star Topology
- Ring Topology
- Tree Topology
- Mesh Topology

Bus Topology

- Bus Topology Alternatively referred to as a line topology, a bus topology is a network setup where each computer and network device are connected to a single cable or backbone.
- Bus networks are useful in small networks (like those setup in a small offices) and have the advantage of using less cable.
- Their main disadvantage is that if any segment of the network fails, all transmissions do as well.

- Below is a visual example of a simple computer setup on a network using the bus topology.

Star Topology

- Star Topology Alternatively referred to as a star network,
- star topology is one of the most common network setups.
- In this arrangement, every node connects to a central hub, switch or computer.
- The hub acting as a server and the peripheral devices as clients.
- A major disadvantage of this network topology is that if the central hub fails, all of the connected devices are disconnected. The visual example shows how this setup gets its name as it is shaped very much like a star.

Ring Topology

- Ring Topology Alternatively referred to as a ring network
- Ring topology is a computer network configuration where the devices are connected to each other in a circular shape.
- Each packet is sent around the ring until it reaches its final destination.

Ring Topology

ComputerHope.com

Tree Topology

- Tree Topology Alternatively referred to as a star bus topology
- Tree topology is one of the most common network setups that is similar to a bus topology and a star topology.
- A tree topology connects one star network to other star networks.
- Below is a visual example of a tree topology, with a simple computer set up on a network using the star topology, connected to another network using the star topology.

TREE TOPOLOGY

Mesh Topology

- A network setup where each computer and network device is interconnected with one another, allowing for most transmissions to be distributed, even if one of the connections go down.
- This topology is not commonly used for most computer networks as it is difficult and expensive to have redundant connection to every computer.
- However, this topology is commonly used for wireless networks. Below is a visual example of a simple computer setup on a network using a mesh topology.

Mesh Topology

Medium access control protocols

- Medium access control protocols are **mechanisms that allow several users or transmitters to access a common medium or channel**. They play an important role in the development of both wired and wireless networks.
- The objectives of multiple access protocols are optimization of transmission time, minimization of collisions and avoidance of crosstalks.

Several protocols have been developed for medium-access control in a multiaccess environment. Of these,

1. the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocol is the one most commonly used for multi access bus networks,
2. and the token ring and slotted ring are the two commonly used protocols for ring networks.

Collision Detection in CSMA/CD

Carrier Sense Multiple Access with Collision Detection (CSMA/CD) is **a network protocol for carrier transmission that operates in the Medium Access Control (MAC) layer**. ... On detection of a collision, the station stops transmitting, sends a jam signal, and then waits for a random time interval before retransmission.

The CSMA/CD Protocol employs decentralized control of the shared medium, In this scheme, each site has equal status in the sense that there is no central controller site. The sites contend with each other for use of the shared medium and the site that first gains access during an idle period of the medium uses the medium for the transmission of its own message. Obviously, occasional collisions of messages may occur when more than one site senses the medium to be idle and transmits messages at approximately the same time. The scheme uses **collision detection, recovery, and controlled retransmission** mechanisms to deal with this problem.

1. Carrier sense and defer mechanism.
2. Collision detection mechanism.
3. Controlled retransmission mechanism.

Token Ring Protocol

The virtual ring is made up of stations that pass tokens during a sequence in a virtual ring. Each node or station in a token bus network knows both the predecessor station's and successor station's addresses. In this scenario, a node transmits the data only if it has a token. Although it is a virtual ring, its working rule is analogous.

Advantages of token rings

Token rings reduce the chances of data collision.

Token passing performs better than bus topology under heavy traffic.

A server is not needed to control connectivity among the nodes.

Slotted Ring Protocol

It can also use the slotted ring protocol, which circulates, instead of a single token, a number of fixed length slots, which may be busy or free. A sending device waits for a slot marked free, sends the message in it, and marks it as busy.

In this scheme, a constant number of fixed-length message slots continuously circulate around the ring. Each slot has two parts-control and data. The control part usually has fields to specify whether the slot is full or empty, the source and destination addresses of the message contained in a full slot, and whether the message in it was successfully received at the destination. On the other hand, the data part can contain a fixed-length message data.

ISO- OSI 7 Layer Model

The OSI Reference Model

- There are many users who use computer network that are located all over the world. For national and worldwide data communication, systems must be developed which are compatible to communicate with each other. ISO has developed this.
- ISO stands for **International Standard Organization.** This is called a model for **Open System Interconnection (OSI)** and is commonly known as OSI model.
- It gives a layered networking framework that conceptualizes how communications should be done between heterogeneous systems
- The ISO-OSI model is a seven layer (7 Layer)architecture developed in 1984.

- To reduce the design complexity, most of the networks are organized as a series of layers or levels, each one build upon one below it. The basic idea of a layered architecture is to divide the design into small pieces. Each layer adds to the services provided by the lower layers in such a way that the highest layer is provided has a full set of services.
- The basic elements of a layered model are services protocols and interfaces.
- A service is a set of actions that a layer offers to another (higher) layer.
- A Protocol is a set of rules that a layer uses to exchange information.
- A Interface is communication between the layers.

Layer 7
APPLICATION LAYER
Layer 6
PRESENTATION LAYER
Layer 5
SESSION LAYER
User Support Layers
Layer 4
TRANSPORT LAYER
Layer 3
NETWORK LAYER
Layer 2
DATA LINK LAYER
Layer 1
PHYSICAL LAYER
Network Support Layers
Physical Link

- The physical layer, data link layer and the network layer are the network support layers.
- The layers manage a physical transfer of data from one device to another.
- Session layer, presentation layer, and application layer are the user support layers.
- These layers allow communication among unrelated software in dissimilar environments. Transport layer links the two groups.
- The main functions of each of the layers are as follows −

1. **Physical Layer −**

- **Physical Layer** function is to transmit individual bits from one node to another over a physical medium.
- Physical layer is the lowest layer of all. It is responsible for sending bits from one computer to another. This layer is not concerned with the meaning of the bits and deals with the physical connection to the network and with transmission.
- This layer defines electrical and physical details represented as 0 or a 1

Functions of Physical Layer:

- Representation of Bits
- Data Rate
- Synchronization
- Line Configuration
- Transmission Modes

2. Data Link Layer

- It is responsible for the reliable transfer of data frames from one node to another connected by the physical layer.
- Data link layer is most reliable node to node delivery of data. It forms frames from the packets that are received from network layer and gives it to physical layer. Error controlling is easily done. Error detection bits are used by the data link layer. It also corrects the errors.

- **FUNCTIONS OF DATA LINK LAYER:**
- **Framing:** Frames are the streams of bits received from the network layer into manageable data units.
- **Physical Addressing:** The Data Link layer adds a header to the frame in order to define physical address of the sender or receiver of the frame, if the frames are to be distributed to different systems on the network.
- **Flow Control:** A flow control mechanism to avoid a fast transmitter from running a slow receiver by buffering the extra bit is provided by flow control. This prevents traffic jam at the receiver side. Error Control: Error control is achieved by adding a trailer at the end of the frame. Duplication of frames are also prevented by using this mechanism.
- **Access Control:** Protocols of this layer determine which of the devices has control over the link at any given time.
- Two Parts of Data Link Layer Data link layer is divided into two subparts

  1.Logical Link Control Layer (LLC) 2.Media Access Control Layer (MAC)

3. Network Layer

- It manages the delivery of individual data packets from source to destination through appropriate addressing and routing.
- It routes the signal through different channels from one node to other. It acts as a network controller. It manages the Subnet traffic.It decides by which route data should take. It divides the outgoing messages into packets.
- **Functions of Network Layer**
- It translates logical network address into physical address. .
- Routers and gateways operate in the network layer.
- Mechanism is provided by Network Layer for routing the packets to final destination. Connection services are provided.
- Breaks larger packets into small packets.

4. Transport Layer

- It is responsible for delivery of the entire message from the source host to destination host.
- It decides if data transmission should be on parallel path or single path. It receives messages from the Session layer above it, convert the message into smaller units and passes it on to the Network layer.
- Transport layer breaks the message (data) into small units so that they are handled more efficiently by the network layer.

- **Function of Transport Layer**:
- **Service Point Addressing :**Transport Layer header includes service point address which is port address. Segmentation and Reassembling : A message is divided into segments; each segment contains sequence number, which enables this layer in reassembling the message.
- **Flow Control :** In this layer, flow control is performed end to end.
- **Error Control :** Error Control is performed end to end in this layer to ensure that the complete message arrives at the receiving transport layer without any error. Error Correction is done through retransmission.

5. Session Layer –

- It establishes sessions between users and offers services like dialog control and synchronization.
- Session layer manages and synchronize the conversation between two different applications. Transfer of data from source to destination session layer streams of data are marked and are resynchronized properly, so that the ends of the messages are not cut and data loss is avoided.

- **Functions of Session Layer:**
- **Dialog Control :** This layer allows two systems to start communication with each other in half-duplex or full-duplex.
- **Synchronization :** This layer allows a process to add checkpoints which are considered as synchronization points into stream of data. 28
- Example: If a system is sending a file of 800 pages, adding checkpoints after every 50 pages is recommended. This ensures that 50 page unit is successfully received and acknowledged. This is beneficial at the time of crash as if a crash happens at page number 110; there is no need to retransmit 1 to 100 pages.

6. Presentation Layer −

- It monitors syntax and semantics of transmitted information through translation, compression, and encryption.
- The primary goal of this layer is to take care of the syntax and semantics of the information exchanged between two communicating systems.
- Presentation layer takes care that the data is sent in such a way that the receiver will understand the information (data) and will be able to use the data.

Functions of Presentation Layer:

- **Translation :** Before being transmitted, information in the form of characters and numbers should be changed to bit streams.
- **Encryption :** It carries out encryption at the transmitter and decryption at the receiver.
- **Compression :** It carries out data compression to reduce the bandwidth of the data to be transmitted. It is important in transmitting multimedia such as audio, video, text etc.

7. Application Layer :

- It is the topmost layer.
- Transferring of files disturbing the results to the user is also done in this layer.
- Mail services, directory services, network resource etc are services provided by application layer.
- This layer mainly holds application programs to act upon the received and to be sent data.

- **Functions of Application Layer:**
- **Mail Services :** This layer provides the basis for E-mail forwarding and storage.
- **Directory Services :** This layer provides access for global information about various services.
- **File Transfer, Access and Management (FTAM) :** It is a standard mechanism to access files and manages it.
- **Protocols at Application Layer**
- **HTTP:**Hyper Text Transfer Protocol
- **FTP:**File Transfer Protocol
- **SMTP**:Simple Mail Transfer Protocol
- **NFS:**Network File Services
- **NVT**:Network Virtual Terminal

ATM TECHNOLOGY

ATM and ATM Networks

- ATM stands for Asynchronous Transfer Mode.
- It is a switching technique that uses time division multiplexing (TDM) for data communications.
- It encodes data into small fixed - size cells so that they are suitable for TDM and transmits them over a physical medium.
- It is a high-speed, connection-oriented switching and multiplexing technology that uses short, fixed-length packets (called cells) to transmit different types of traffic simultaneously, including voice, video, and data.
- The size of an ATM cell is 53 bytes: 5 byte header and 48 byte payload. There are two different cell formats - user-network interface (UNI) and network-network interface (NNI).

- The ATM is a connection-oriented technology because a sender first establishes a connection with the receiver. However, unlike circuit switching, in which a physical circuit is established between the sender and the receiver and reserved for them for the entire duration of their communication session,
- In ATM technology, a virtual circuit is established between the sender and the receiver.
- That is, ATM does not reserve the path for one user exclusively. Any time a given user is not occupying a channel, another user is free to use it.
- Connection establishment in ATM means that a route is determined from the sender to the receiver and routing information is stored in the switches along the route during connection establishment.
- All cells of messages from the sender to the receiver follow this virtual path stored in the switches. When the connection is no longer needed, it is released and the routing information for this connection is purged from the switches.

ATM Endpoint
ATM Switch
ATM Endpoint
ATM Adaptation
ATM Layer
Physical Layer
ATM Layer
Physical Layer
ATM Adaptation
ATM Layer
Physical Layer
Physical Medium

Physical Layer

- The physical layer is the bottom-most layer of the ATM protocol suite. It is concerned with putting bits on the wire and taking them off again.
- It has two sublayers: the **physical medium dependent (PMD)** sublayer and the **transmission convergence (TC)** sublayer.
- The PMD sublayer defines the actual speed for traffic transmission on the physical communication medium (electrical/optical) used.
- On the other hand, the TC sublayer defines a protocol for the encoding and decoding of cell data into suitable electrical/optical waveforms for transmission and reception on the physical communication medium defined by the PMD sublayer.

The physical layer can transfer cells from one user to another in one of the following two ways:

1**. By carrying cells as a synchronous data stream.** In this case, the user-network interface (UNI), which takes the form of an ATM adaptor board plugged into a computer, puts out a stream of cells on to a wire or fiber. The transmission stream must be continuous, and when there is no data to be sent, empty cells are transmitted.

2. **By carrying cells in the payload portion of an externally framed transmission structure.** In this case, the UNI uses some standard transmission structure for framing and synchronization at the physical layer. SONET (Synchronous Optical NETwork) [Omidyar and Aldridge 1993],

The SONET format is currently supported by single-mode fiber, multimode fiber, and twisted pair.

- SONET stands for Synchronous Optical Network. SONET is a communication protocol, developed by Bellcore – that is used to transmit a large amount of data over relatively large distances using optical fibre.
- With SONET, multiple digital data streams are transferred at the same time over the optical fibre.
- a SONET frame consists of a total of 810 bytes (9 X 90), out of which 27 bytes (9 X 3) are overhead and the remaining 783 bytes (9 X 87) are payload.

Fig. 2.13 SONET frame format.

ATM Layer

- The ATM layer handles most of the cell processing and routing activities.
- These include building the cell header, cell multiplexing of individual connections into composite flows of cells, cell demultiplexing of composite flows into individual connections, cell routing, cell payload type marking and differentiation, cell loss priority marking and reduction, cell reception and header validation, and generic flow control of cells.
- These functions are designed to be carried out in hardware at very high data rates.
- The ATM layer is independent of the physical medium used to transport the cells.
- The functionality of the ATM layer is defined by the fields present in the ATM cell header (Fig. 2.14).
- These fields and their functions are as follows:

Fig. 2.14 ATM cell format: GFC, Generic Flow Control; VPI, Virtual Path Identifier; VCI, Virtual Channel Identifier; PTI, Payload Type Identifier; CLP, Cell Loss Priority; HEC, Header Error Control.

- **Generic Flow control (GFC)field.** This field occupies 4 bits in an ATM cell header. The default setting of four 0's indicates that the cell is uncontrolled. An uncontrolled cell does not take precedence over another cell when contending for a virtual circuit. The bits in the GFC field can be suitably set to implement some form of prioritized congestion control.
- **Virtual path identifier (V**PI) and virtual channel identifier (VCI) fields. The VPI field occupies 8 bits and theVCI field occupies 16 bits in an ATM cell header. These two fields are used by the routing protocol to determine the path(s) and channel(s) the cell will traverse.
- **Payload-type identifier (PTI)** field. This field occupies 3 bits in an ATM cell header. It is used to distinguish data cells from control cells so that user data and control data can be transmitted on different subchannels.
- **Cell loss priority (CLP) field**. This field occupies 1 bit in an ATM cell header. When set, it indicates that the cell can be discarded, if necessary, during periods of network congestion.
- **Header error control (HEC)**field. This field occupies 8 bits in an ATMcell header. It is used to protect the header field from transmission errors. It contains a checksum of only the header (not the payload).

ATM Adaptation Layer

- It provides facilities to the existing packet switched networks to connect to ATM network and use its services. It accepts the data and converts them into fixed sized segments. The transmissions can be of fixed or variable data rate.
- The functionality of the physical and the ATM layers of the ATM protocol suite is not tailored to any application.
- We saw that ATM can support various types of traffic, including voice, video, and data.
- The AAL is responsible for providing different types of services to different types of traffic according to their specific requirements.
- It packages various kinds of user traffic into 48-byte cells, together with the overhead needed to meet specific quality-of-service requirements of different types of traffic. To reflect the spectrum of applications, four service classes were defined by the ITV

ATM CBR-Constant Bit Rate

- Used for applications requiring fixed data rate available always during the lifetime of connection.

- APPLICATIONS:Telephony, Videoconferencing, Audio and Video distribution and retrieval etc.

ATM rt-VBR, Real time Variable Bit Rate

- Used for applications requiring tight limit on delay i.e. used for time sensitive applications

- It allows more flexibility to the network compare to CBR.

- APPLICATIONS: LIVE TV Transmission having video information

To reflect the spectrum of applications, four service classes were defined :

1. **Class A**. Applications having delay-sensitive CBR traffic that require connection. oriented service belong to this class. Video and voice applications normally belong to this class.
2. **Class B.** Applications having delay-sensitive VBR traffic that require connection oriented service belong to this class. Some video and audio applications belong to this class.
3. **Class C.** Applications having VBR traffic that are not delay-sensitive but that require connection - oriented service belong to this class. Connection-oriented file transfer is a typical example of an application that belongs to this class.
4. **Class D.** Applications having VBR traffic that are not delay-sensitive and does not require connection-oriented service belong to this class. LAN interconnection and electronic mail are typical examples of applications that belong to this class.

Service class	Class A	Class B	Class C	Class D
Bit rate type	CBR	VBR	VBR	VBR
Delay sensitive?	Yes	Yes	No	No
Connection oriented?	Yes	Yes	Yes	No
AAL protocol to be used	AAL1	AAL2	AAL3/4 or AAL5	AAL3/4 or AAL5

Fig. 2.15 Service classes for the ATM Adaptation Layer (AAL).

Main Features of ATM Technology

Asynchronous Transfer Mode (ATM) is a transfer protocol with the following characteristics:

- It is scalable and flexible. It can support megabit-to-gigabit transfer speeds and is not tied to a specific physical medium.
- It efficiently transmits video, audio, and data through the implementation of several adaptation layers.
- Bandwidth can be allocated as needed, lessening the impact on and by high-bandwidth users.
- It transmits data in fixed-length packets, called cells, each of which is 53 bytes long, containing 48 bytes of payload and 5 bytes of header.
- It is asynchronous in the sense that although cells are relayed synchronously, particular users need not send data at regular intervals.
- It is connection oriented, using a virtual circuit to transmit cells that share the same source and destination over the same route.

4 It uses the concept of virtual networking to pass traffic between two locations. This concept allows the available bandwidth of a physical channel to be shared by multiple applications, enabling multiple applications to simultaneously communicate at 92 Chap. 2 • Computer Networks different rates over the same path between two end points. That is, it allows the total bandwidth available to be dynamically distributed among a variety of user applications.

5. In addition to point-to-point communication in which there is a single sender and a single receiver, it can easily support multicasting facility in which there is a single sender but multiple receivers. Such a facility is needed for transmitting broadcast television to many houses at the same time. Many collaborative distributed applications also require frequent use of this kind of facility.

6. It enables the use of a single network to efficiently transport a wide range of multimedia data such as text, voice, video, broadcast television, and so on. Therefore, the use of separate networks such as a telephone network for voice, an X.25 network for data, and a cable television network for video can now be replaced by a single ATM network that provides a means for integrating voice, video, data, and other information. This integration will in tum lead to substantial cost savings and simplification in the design of communication networks.

7. It is flexible in the way it grants access to bandwidth. That is, it enables supply of bandwidth on demand and allows billing network users on a per-cell basis (more probably on a .giga-cell basis, given the speed and transfer capabilities of ATM). A user can grab as big or as small a chunk of network bandwidth as he or she needs and pay only for as much as he or she uses.

8. It is a scalable technology. It enables increase or decrease of such things as bandwidths and data rates and still maintains the architecture of the signaling process. 0Moreover, the same switching technology (cell switching) and the same cell format is used

for the transport of all types of information (voice, video, and data) in both LAN and WAN

environments.

9. It has a fairly solid base of standards. It has been adopted by the International

Telecommunications Union (lTV) (formerly CCITT) and internationally standardized as

the basis for the Broadband Integrated Services Digital Network (B-ISDN).

Benefits or advantages of Asynchronous Transfer Mode (ATM)

Following are the benefits or **advantages of Asynchronous Transfer Mode (ATM)**:

➨It is optimized to transport voice, data and video i.e. single network for everything. It is used for mixed traffic, real-time and non real time traffic types.

➨It is easy to integrate with LAN, MAN and WAN network types i.e. seamless integration.

➨It is QoS oriented and high speed oriented.

➨It enables efficient use of network resources using bandwidth on demand concept.

➨It uses simplified network infrastucture.

Drawbacks or disadvantages of Asynchronous Transfer Mode (ATM)

➨Overhead of cell header (5 bytes per cell)

➨Complex mechanisms are used to achieve QoS.

➨Congestion may cause cell losses

➨ATM switch is very expensive compare to LAN hardware. Moreover ATM NIC is more expensive compare to ethernet NIC.

➨As ATM is connection oriented technology, the time required for connection setup and tear down is larger compare to time required to use it.

Message Passing

- **A process is a program in execution**
- **When we say that two computers of a distributed system are communicating with each other, we mean that two processes, one running on each computer, are in communication with each other.**
- **Resource manager process to monitor the current status of usage of its local resources**
- **All resource managers communicate each other from time to time to dynamically balance the system load**
- **Therefore a DOS needs to provide interprocess communication (IPC) mechanism for communication activities**
- **IPC basically requires information sharing among two or more processes.**
- **Two basic methods for information sharing is:**
  1. **Original sharing, or shared-data approach**
  2. **Copy sharing, message-passing approach.**

- The *shared-data* paradigm gives the conceptual communication pattern

- In *message-passing approach*, the information to be shared is physically copied from the sender process address space to the address spaces of all receiver processes
- This done by transmitting the data in the form of messages
- A *message* is a block of information
- Communication processes interact directly with each other

- Distributed system communicate by exchanging messages
- Message passing is the basic IPC mechanism in distributed system
- Message-passing system is a subsystem of a DOS that provides a set of message-based IPC protocols
- It enables processes to communicate by simple communication primitives *send* and *receive*

Desirable Features of a Good Message-Passing system

1. Simplicity

- MPS should be simple and easy to use
- Construction of new applications and to communicate with existing ones by using the primitives provided by the MPS
- Different modules of a distributed application use simple primitives without bothering the system or network
- Use of clean and simple semantics of IPC protocols

2. Uniform Semantics

- Uses two type of communication
  - ✓ Local communication – the communicating process are on the same node
  - ✓ Remote communication – the communicating processes are on different nodes
- The semantics of remotes communication should be close as possible to those of local communications

3. Efficiency

- If the MPS is not efficient , IPC may become so expensive
- Application users try to avoid its use in their applications
- An IPC protocol of a MPS can be made efficient by reducing the number of message exchanges during communication
- Some optimizations are
  - ✓ Avoiding cost of establishing and terminating connections between the same pair of processes of every exchange
  - ✓ Minimizing the cost of maintaining connections
  - ✓ Piggybacking of acknowledgement

4. Relaiability

- A reliable IPC protocol can cope up with failure problems and guarantees the delivery of a message
- Failure due to node crash or communication link failure
- Handling of lost messages usually involves acknowledgements and retransmissions on the basis of timeouts
- Another issues related to reliability is duplicate messages
- Duplicate messages because of event of failures or timeouts
- A reliable IPC protocol is also capable of detecting and handling duplicates
- Use sequence number to avoid duplicate messages

5. Correctness

- IPC system has group communication
- One sender to multiple receiver, multiple sender to one receiver
- Correctness related to IPC protocols group communication

- Issues related to correctness is
  - Atomicity
    - Ensures that every message sent to a group of receivers will be delivered to either all of them or none of them
  - Ordered delivery
    - Ensures that messages arrive at all receivers in an order acceptable to the application
  - Survivability
    - Guarantees that messages will be delivered despite of partial failure of processes, machines, or communication links

6. Flexibility

- Not all applications require the same degree of reliability and correctness of the IPC protocols
- Many applications do not require atomicity or ordered delivery of messages
- The IPC primitives should be such that users have the flexibility to choose and specify the types and levels of reliability and correctness requirements of applications
- Flexibility permit control flow as synchronous and asynchronous send/receive

7. Security

- A MPS be capable of providing a secure end-to-end communication
- A message in transit on the network should not be accessible to any user other than those to whom it is addressed and the sender
- Steps necessary for secure communication is
  - ✓ Authentication of the receiver(s) of a message by the sender
  - ✓ Authentication of the sender of a message by its receiver(s)
  - ✓ Encryption of a message before sending it over the network

8. Portability

- Two different aspects of portability
  - ✓ It should be easily construct new IPC facility on another system by reusing the basic design of existing MPS
  - ✓ Applications are also portable - heterogeneity must be considered while designing MPS

Issues in IPC by message passing

- A *message* is a block of information formatted by a sending process in such a manner that it is meaningful to the receiving process
- It consists of fixed-length header and a variable-size collection of typed data objects

- In a message-oriented IPC protocol, the sending process determines the actual contents of a message
- The receiving process to convert the contents
- Special primitives are explicitly used for sending and receiving the messages
- Following issues to be discussed in the design of an IPC protocol
  - Who is the sender?
  - Who is the receiver?
  - Is there one receiver or many receivers?
  - Is the message guaranteed to have been accepted by its receiver(s)?
  - Does the sender need to wait for a reply?
  - What should be done in case of failure (crash or communication)?
  - What should be done if the receiver is not accept the message?

Synchronization

- Major issue of communicating process is synchronization
- The semantics classified as *blocking* and *nonblocking* types
- *Nonblocking* semantics if it invocation does not block the execution of its invoker
- Otherwise a primitive is *blocking* (execution of invoker is blocked)
- Two types of semantics used for the *send* and *receive* primitives
- In case of *blocking send*, after the execution of send , the sending process is blocked until acknowledgement is received
- *Blocking receive*, after execution of receive statement , the receiving process is blocked until it receives message
- *Nonblocking send*, after sending process sending process is allowed to execute
- *Nonblocking receive*, the receiving process proceeds with its execution after execution the receive statement

- An important issue in a nonblocking receive primitive is how receiving process know that the message has arrived in the message buffer
- The following two methods used for this
  - Polling
    - A *test* primitive is allowed to the receiver to check the buffer status
    - Receiver periodically poll the kernel to check the buffer
  - Interrupt
    - When the message is filled in the buffer, *software interrupt* is used to notify the receiving process
    - This method permits the receiving process to continue without having unsucessful *test* requests
    - Its highly efficient and allows maximum parallelism
    - Drawback is user-level interrupts make programming difficult

- A variant of Nonblocking receive primitive is the *conditional receive primitive*
- It returns control immediately, either with a message or an indicator that no message
- Blocking send primitive uses the timeout values
- The value set by user or default value
- Timeout value used for blocking receive primitive to prevent the receiving process blocked indefinitely
- Both the send and receive primitives of a communication between two process use blocking semantics is said to be *synchronous*
- If its uses nonblocking primitives then communication is *asynchronous*
- Synchronous communication is simple and easy to implement
- Provide high reliability
- Drawbacks are
  - Limits the concurrency and is subject to communication deadlocks
  - Less flexible because sending process always has to wait for an acknowledgement, even it is nor required

Sender's execution
Receiver's execution
Send(message);
Execution suspended
Message
Execution resumed
Send(acknowledgment)
Acknowledgment
Execution resumed

Buffering

- Messages copying from the address space of the sending process to the address space of the receiving process
- If the receiving process is not ready to receive messages, then it should be save for later usage
- The message buffering is related to synchronization strategy
- The following are the buffering strategies
  1. Null buffer or no buffer
  2. Buffer with unbounded capacity
  3. Single-message buffer
  4. Finite-bound or multiple-message buffer

1. Null buffer (or no buffering)

- There is no place to temporarily store the message
- One of the following implementation strategies used
  - The message remains in sender address space and execution of *send* is delayed until the receiver executes *receive*
  - The message is simply discarded and the timeout mechanism is used to resend the message after a timeout period

2. Single-Message buffer

- A buffer capacity to store single message is used on the receiver's node
- An application module may have at most one message outstanding at a time
- Single-message buffer strategy is to keep the message ready for use at the location of the receiver
- The request message is buffered on the receiver's node if the receiver is not ready to receive the message
- The message buffer may either be located in the kernel's address space or in the receiver process's address space

3. Unbounded-capacity buffer

- A sender does not wait for the receiver to be ready
- An unbounded-capacity message buffer that can store all unreceived messages
- It assure that all the messages sent to the receiver will be delivered

4. Finite-bound (or multiple-message) buffer

- Asynchronous ode of communication use finite-bound buffers
- Need mechanism to handle the problem of buffer overflow
- Two ways to handle buffer overflow
  - Unsuccessful communication
    - Message transfers simply fail whenever there is no mode buffer space
    - The send normally returns an error message to the sending process
    - This method is less reliable

- Flow-controlled communication
  - The sender is blocked until the receiver accepts some messages
  - This method introduces a synchronization between sender and receiver
  - It result in unexpected deadlocks

- the amount of buffer space to be allocated depends on implementation
- A *create-buffer* system call is provided to the users
- The receiver mail box is located in the kernel address space or in the receiver process address space
- This buffering provides better concurrency and flexibility

Multiple-message buffer/mailbox/port

Multidatagram Messages

MULTIDATAGRAM MESSAGES

- Almost all networks have an upper bound on the size of data that can be transmitted at a time.
- This size is known as the maximum transfer unit (MTU) of a network.
- A message whose size is greater than the MTU has to be fragmented into multiples of the MTU, and then each fragment has to be sent separately.
- Each fragment is sent in a packet that has some control information in addition to the message data.
- Each packet is known as a datagram.
- Messages smaller than the MTU of the network can be sent in a single packet and are known as single-datagram messages.
- Messages larger than the MTU of the network have to be fragmented and sent in multiple packets, such messages are known as multidatagram messages.

ENCODING AND DECODING OF MESSAGE DATA

- A message data should be meaningful to the receiving process.
  - The structure of program objects should be preserved while they are being transmitted from the address space of the sending process to the address space of the receiving process.
- However, this is not possible in a heterogeneous system in which the sending and receiving processes are on computers of different architectures.
- Further, even in homogeneous systems, it is very difficult to achieve this goal.
  - The reason behind this are two.

ENCODING AND DECODING OF MESSAGE DATA

First:

- An absolute pointer value loses its meaning when transferred from one process address space to another.
- Therefore, such program objects that use absolute pointer values cannot be transferred in their original form, and some other form of representation must be used to transfer them.
- For example, to transmit a tree object, each element of the tree must be copied in a leaf record and properly aligned in some fixed order in a buffer before it can be sent to another process.
- The leaf records themselves have no meaning in the address space of the receiving process, but the tree can be regenerated easily from them by using object- typed information
  - object-type information must be passed between the sender and receiver, indicating not only that a tree object is being passed but also the order in which the leaf records are aligned. (e.g Linked Lists)

ENCODING AND DECODING OF MESSAGE DATA

Second:

- Different program objects occupy varying amount of storage space.
- To be meaningful, a message must normally contain several types of program objects, such as long integers, short integers, variable-length character strings, and so on.
- In this case, to make the message meaningful to the receiver, there must be some way for the receiver to identify which program object is stored where in the message buffer and how much space each program object occupies.

ENCODING AND DECODING OF MESSAGE DATA

- Program objects are first converted to a stream form that is suitable for transmission and placed into a message buffer.
- This conversion process takes place on the sender side and is known as encoding of a message data.
- The encoded message, when received by the receiver, must be converted back from the stream form to the original program objects before it can be used.
- The process of reconstruction of program objects from message data on the receiver side is known as decoding of the message data.

ENCODING AND DECODING OF MESSAGE DATA

Representations used for the encoding and decoding of a message data:

- tagged representation
- untagged representation

Tagged representation

- In tagged representation the type of each program object along with its value is encoded in the message.
- In this method, it is a simple matter for the receiving process to check the type of each program object in the message because of the self-describing nature of the coded data format.

untagged representation

- In untagged representation the message data only contains program objects.
- No information is included in the message data to specify the type of each program object.
- The receiving process must have a prior knowledge of how to decode the received data because the coded data format is not self-describing.

PROCESS ADDRESSING

- Important issue in message-based communication is addressing (or naming) of the parties involved in an interaction.
- A message-passing system usually supports two types of process addressing
  - Explicit addressing
  - Implicit addressing

Explicit addressing.

- The process with which communication is desired is explicitly named as a parameter in the communication primitive used.
- Primitive used in Explicit addressing are
  - send(process_id, message): Send a message to the process identified by "process_id".
  - receive(process_id, message): Receive a message from the process identified by "process_id".

PROCESS ADDRESSING

Implicit addressing.

- A process willing to communicate does not explicitly name a process for communication.
- Primitive used in Implicit addressing are
  - Send_any(service_id, message): Send a message to any process that provides the service of type “service_id”.
    - The sender names a service instead of a process.
    - This type of primitive is useful in client-server communications when the client is not concerned with which particular server out of a set of servers providing the service.
  - receive_any(process_id, message): Receive a message from any process and return the process identifier (“process_id”) of the process from which the message was received.
    - The receiver is willing to accept a message from any sender.
    - This type of primitive is useful in client-server communications when the server is meant to service requests of all clients that are authorized to use its service.

FAILURE HANDLING

- Distributed system may be prone to partial failures such as a node crash or a communication link failure.
- During interprocess communication, such failures may lead to the following problems:
  - Loss of request message: This may happen either due to the failure of communication link between the sender and receiver or because the receiver's node is down at the time the request message reaches there

Sender
Receiver
Send request
Request message
Lost

FAILURE HANDLING

- During interprocess communication, such failures may lead to the following problems:
  - Loss of response message. This may happen either due to the failure of communication link between the sender and receiver or because the sender's node is down at the time the response message reaches there.

Sender
Receiver
Send request
Request message
Successful request execution
Response message
Send response
Lost

FAILURE HANDLING

- During interprocess communication, such failures may lead to the following problems:
  - Unsuccessful execution of the request: This happens due to the receiver's node crashing while the request is being processed.

- To cope with these problems, a reliable IPC protocol of a message-passing system is normally designed based on the idea of:
  - Internal retransmissions of messages after timeouts and the return of an acknowledgment message to the sending machine's kernel by the receiving machine's kernel.

- The kernel of the sending machine is responsible for retransmitting the message after waiting for a timeout period
- If the no acknowledgement is received
- The kernel of the sending machine frees the sending process only when acknowledgement is received
- The timeout value is slightly more than the approximate round-trip time plus the average time required for executing request

- Four-message IPC protocol for client server communication between two processes

- In communication result of the processed request is sufficient acknowledgement
- Using this three-message IPC protocol defined

- In the above protocol a problem occurs if a request processing takes long time
- In this case the sender kernel timeouts and retransmit the message
- To avoid unnecessary retransmission use separate acknowledgement if processing takes long time
- Acknowledgement is not necessary because unnecessary retransmission of the messages and request processing
- Use two-message IPC protocol

- Based on 2-msg IPC failure handling in communication

Keeping Track of Lost and Out-of-Sequence Packets in Multigram Messages

- Message transmission is complete only all the packets of the message have been received
- Simple way to achieve is *stop-and-wait* protocol
- Separate acknowledgement leads to communication overhead
- Improve performance, better approach is to use a single acknowledgement packet for all the packets using *blast protocol*
- When this approach used it leads to the following problem due to node crash or link failure
  - One or more packets of the multidatagram message are lost in communication
  - The packets are received out of sequence by the receiver

- An efficient mechanism is to use a bitmap identify the packets of a message
- Header part contains two extra fields
  - One of which specifies total number of packets
  - Bitmap filed the specifies the position of this packet
- Receiving process set the buffer size according to the first field
- Packets stored in the corresponding buffer area
- After timeout, if all packet not received the bit map indicating unreceived packet send to sender
- This methods uses *selective repeat protocol*

GROUP COMMUNICATION

- The most elementary form of message-based interaction is one-to-one communication (also known as point-to-point, or unicast, communication)
  - In this a single-sender process sends a message to a single-receiver process.
- However, for performance and ease of programming,
  - several highly parallel distributed applications require that a message passing system should also provide group communication facility.
- Depending on single or multiple senders and receivers, the following three types of group communication are possible:
  - One to many (single sender and multiple receivers)
  - Many to one (multiple senders and single receiver)
  - Many to many (multiple senders and multiple receivers)

Broadcast Communication :

- In this scheme, multiple senders send messages to multiple receivers.
- When the host process tries to communicate with every process in a distributed system at same time.
- Broadcast communication comes in handy when a common stream of information is to be delivered to each and every process in most efficient manner possible. Since it does not require any processing whatsoever, communication is very fast in comparison to other modes of communication.

Many-to-Many Communication

Multicast Communication :

- When the host process tries to communicate with a designated group of processes in a distributed system at the same time.
- This technique is mainly used to find a way to address problem of a high workload on host system and redundant information from process in system.
- Multitasking can significantly decrease time taken for message handling.

P1
P2
P3
P4
P5
P6

Unicast Communication :

- When the host process tries to communicate with a single process in a distributed system at the same time.
- Although, same information may be passed to multiple processes.
- This works best for two processes communicating as only it has to treat a specific process only.

P2
P3
P1
P4
P5
P6

Distributed shared memory DSM

Distributed shared memory DSM

- Distributed shared memory DSM paradigm provides process with shared address space
- Primitives for shared memory: –
  - Read(address) –
  - Write(address , data)
- Shared memory paradigm gives the systems illusion of physically shared memory.
- DSM refers to shared memory paradigm applied to loosely coupled distributed memory systems

- Shared memory exists only virtually Similar concept to virtual memory DSM also known as DSVM
- DSM provides a virtual address space shared among processes on loosely coupled processors
- DSM is basically an abstraction that integrates the local memory of different machine into a single logical entity shared by cooperating processes

DSM architecture

DSM architecture

- Each node of the system consist of one or more CPUs and memory unit.
- Nodes are connected by high speed communication network
- Simple message passing system for nodes to exchange information
- Main memory of individual nodes is used to cache pieces of shared memory space

- Memory mapping manager routine maps local memory to shared virtual memory
- Shared memory of DSM exist only virtually Shared memory space is partitioned into blocks
- Data caching is used in DSM system to reduce network latency
- The basic unit of caching is a memory block
- The missing block is migrate from the remote node to the client processes node and operating system maps into the application's address space
- Data block keep migrating from one node to another on demand but no communication is visible to the user processes
- If data is not available in local memory network block fault is generated.

Advantages of Distributed shared memory

- **Simpler abstraction:** Programmer need not concern about data movement, As the address space is the same it is easier to implement than RPC.
- **Easier portability:** The access protocols used in DSM allow for a natural transition from sequential to distributed systems. DSM programs are portable as they use a common programming interface.
- **locality of data:** Data moved in large blocks i.e. data near to the current memory location that is being fetched, may be needed future so it will be also fetched.

Advantages of Distributed shared memory

- **on-demand data movement:** It provided by DSM will eliminate the data exchange phase.
- **larger memory space:** It provides large virtual memory space, the total memory size is the sum of the memory size of all the nodes, paging activities are reduced.
- **Better Performance:** DSM improve performance and efficiency by speeding up access to data.
- **Flexible communication environment:** They can join and leave DSM system without affecting the others as there is no need for sender and receiver to existing,
- **process migration simplified:** They all share the address space so one process can easily be moved to a different machine.

Disadvantages

- Generally slower to access than non-distributed shared memory.
- Must provide additional protection against simultaneous accesses to shared data.
- May incur a performance penalty.
- Little programmer control over actual messages being generated.

Design and implementation issues

- Granularity
- Structure of Shared memory
- Memory coherence and access synchronization
- Data location and access Replacement strategy
- Thrashing
- Heterogeneity

Granularity

- Granularity refers to the block size of DSM
- The unit of sharing and the unit of data transfer across the network when a network block fault occurs Possible unit are a few word , a page or a few pages

Structure of Shared memory:

- Structure refers to the layout of the shared data in memory Dependent on the type of applications that the DSM system is intended to support

Memory coherence and access synchronization:

- In a DSM system that allows replication of shared data item, copies of shared data item may simultaneously be available in the main memories of a number of nodes
- To solve the memory coherence problem that deal with the consistency of a piece of shared data lying in the main memories of two or more nodes

Data location and access:

- To share data in a DSM, should be possible to locate and retrieve the data accessed by a user process
- Replacement strategy:
- If the local memory of a node is full, a cache miss at that node implies not only a fetch of accessed data block from a remote node but also a replacement
- Data block must be replaced by the new data block

Thrashing:

- Data block migrate between nodes on demand.
- Therefore if two nodes compete for write access to a single data item the corresponding data block may be transferred back.

Heterogeneity:

- The DSM system built for homogeneous system need not address the heterogeneity issue

Granularity

- Most visible parameter in the design of DSM system is block size

Factors influencing block size selection:

- Sending large packet of data is not much more expensive than sending small ones

Paging overhead: A process is likely to access a large region of its shared address space in a small amount of time

- Therefore the paging overhead is less for large block size as compared to the paging overhead for small block size

Cont…

Directory size:

- The larger the block size, the smaller the directory
- Ultimately result in reduced directory management overhead for larger block size

Thrashing:

- The problem of thrashing may occur when data item in the same data block are being updated by multiple node at the same time
- Problem may occur with any block size, it is more likely with larger block size

Cont…

False sharing:

A data block

- Occur when two different processes access two unrelated variable that reside in the same data block

Cont…

- The larger is the block size the higher is the probability of false sharing
- False sharing of a block may lead to a thrashing problem

Cont…

Using page size as block size:

- Relative advantage and disadvantages of small and large block size make it difficult for DSM designer to decide on a proper block size

Following advantage:

- It allows the use of existing page fault schemes to trigger a DSM page fault
- It allows the access right control
- Page size do not impose undue communication overhead at the time of network page fault
- Page size is a suitable data entity unit with respect to

Structure of shared-memory space

- Structure defines the abstract view of the shared memory space
- The structure and granularity of a DSM system are closely related

Three approach:

- No structuring
- Structuring by data type
- Structuring as a database

Cont…

No structuring:

- The shared memory space is simply a linear array of words

Advantage:

- Choose any suitable page size as the unit of sharing and a fixed grain size may be used for all application
- Simple and easy to design such a DSM system

Cont….

Structuring by data type:

- The shared memory space is structured either as a collection of variables in the source language
- The granularity in such DSM system is an object or a variable
- DSM system use variable grain size to match the size of the object/variable being accessed by the application

Cont…

Structuring as a database:

- Structure the shared memory like a database
- Shared memory space is ordered as an associative memory called tuple space
- To perform update old data item in the DSM are replaced by new data item
- Processes select tuples by specifying the number of their fields and their values or type
- Access to shared data is nontransparent. Most system they are transparent

Consistency Models

- Consistency requirement vary from application to application
- A consistency model basically refers to the degree of consistency that has to be maintained for the shared memory data
- Defined as a set of rules that application must obey if they want the DSM system to provide the degree of consistency guaranteed by the consistency model
- If a system support the stronger consistency model then the weaker consistency model is automatically supported but the converse is not true

Consistency Models

Types:

- *Strict Consistency model*
- *Sequential Consistency model*
- *Causal consistency model*
- *Pipelined Random Access Memory consistency model(PRAM)*
- *Processor Consistency model*
- *Weak consistency model*
- *Release consistency model*

Strict consistency model

- This is the strongest form of memory coherence having the most stringent consistency requirement
- Value returned by a read operation on a memory address is always same as the value written by the most recent write operation to that address
- All writes instantaneously become visible to all processes
- Implementation of the strict consistency model requires the existence of an absolute global time

Cont….

- Absolute synchronization of clock of all the nodes of a distributed system is not possible
- Implementation of strict consistency model for a DSM system is practically impossible
- If the three operations read(r1), write(w1), read(r2) are performed on a memory location in that order
- Only acceptable ordering for a strictly consistency memory is (r1, w1, r2)

Sequential Consistency model

- Proposed by Lamport [1979]
- A shared memory system is said to support the sequential consistency model if all processes see the same order
- Exact order of access operations are interleaved does not matter
- If the three operations read(r1), write(w1), read(r2) are performed on a memory location in that order
- Any of the orderings (r1, w1, r2), (r1, r2, w1), (w1, r1, r2), (w1, r2, r1), (r2, r1, w1), (r2, w1, r1) is acceptable provided all processes see the same ordering

Cont…

- The consistency requirement of the sequential consistency model is weaker than that of the strict consistency model
- A sequentially consistency memory provide one-copy /single-copy semantics
- sequentially consistency is acceptable by most applications

Causal Consistency Model

- Proposed by Hutto and Ahamad (1990)
- All processes see only those memory reference operations in the correct order that are potentially causally related
- Memory reference operations not related may be seen by different processes in different order
- Memory reference operation is said to be related to another memory reference operation if one might have been influenced by the other
- Maintaining dependency graphs for memory access operations

Pipelined Random Access Memory Consistency model

- Proposed by Lipton and Sandberg (1988)
- Provides a weaker consistency semantics than the consistency model described so far
- Ensures that all write operations performed by a single process are seen by all other processes in the order in which they were performed
- All write operations performed by a single process are in a pipeline
- Write operations performed by different processes can be seen by different processes in different order

Cont…

- If w11 and w12 are two write operations performed by a process P1 in that order, and w21 and w22 are two write operations performed by a process P2 in that order
- A process P3 may see them in the order [(w11,w12), (w21,w22)] and another process P4 may see them in the order [(w21,w22), (w11,w12)]
- Simple and easy to implement and also has good performance
- PRAM consistency all processes do not agree on the same order of memory reference operations

Processor consistency model

- Proposed by Goodman [1989]
- Very similar to PRAM model with additional restriction of memory coherence
- Memory coherence means that for any memory location all processes agree on the same order of all write operations performed on the same memory location (no matter by which process they are performed) are seen by all processes in the same order
- If w12 and w22 are write operations for writing the same memory location x, all processes must see them in the same order- w12 before w22 or w22 before w12
- Processes P3 and P4 must see in the same order, which may be either [(w11,w12), (w21,w22)] or [(w21,w22), (w11,w12)]

Weak consistency model

- Proposed by Dubois [1988]
- Common characteristics to many application:

1 It is not necessary to show the change in memory done by every write operation to other processes eg. when a process executes in a critical section

2 Isolated accesses to shared variable are rare

- Better performance can be achieved if consistency is enforced on a group of memory reference operations rather than on individual memory reference operations

Implementing sequential consistency model

- Most commonly used model
- Protocols for implementing the sequential consistency model in the DSM system depend to a great extent on whether the DSM system allows replication and/or migration of shared memory data blocks

Strategies:

- *Nonreplicated, Nonmigrating blocks (NRNMB)*
- *Nonreplicated, migrating blocks (NRMB)*
- *Replicated, migrating blocks (RMB)*
- *Replicated, Nonmigrating blocks (RNMB)*

NRNMBs

- Simplest strategy for implementing a sequentially consistency DSM system
- Each block of the shared memory has a single copy whose location is always fixed

Fig 3 NRNMB strategy

Cont…

- Enforcing sequential consistency is simple in this case
- Method is simple and easy to implement and suffers following drawback:
  - Serializing data access creates a bottleneck
  - Parallelism, which is a major advantage of DSM is not possible with this method

Data locating in the NRNMB strategy :

- There is a single copy of each block in the entire system
- The location of a block never changes
- Hence use Mapping Function

NRMBs

- Each block of the shared memory has a single copy in the entire system
- In this strategy only the processes executing on one node can read or write a given data item at any one time and ensures sequential consistency

Fig 4. NRMB strategy

Cont…

Advantage :

- No communications cost are incurred when a process accesses data currently held locally
- It allows the applications to take advantage of data access locality

Drawbacks:

- It is prone to thrashing problem
- The advantage of parallelism can not be availed in this method also

Data locating in the NRMB strategy

- There is a single copy of each block, the location of a block keeps changing dynamically

Following method used :

1. Broadcasting
2. Centralized server algorithm
3. Fixed distributed server algorithm
4. Dynamic distributed server algorithm

Broadcasting

- Each node maintains an owned blocks table that contains an entry for each block for which the node is the current owner

Fig 5. structure and locations of owned bloks table in the broadcasting data locating mechanism for NRMB strategy

Cont….

- When a fault occurs , the fault handler of the faulting node broadcasts a read/write request on the network

Disadvantage:

- It does not scale well

Centralized server algorithm

Fig 6. structure and location of block table in the centralized server data locating mechanism for NRMB strategy

Cont…

- A centralized server maintains a block table that contains the location information for all block in the shared memory space

Drawback:

- A centralized server serializes location queries, reducing parallelism
- The failure of the centralized server will cause the DSM system to stop functioning

Fixed distributed server algorithm

Fig 7. structure and locations of block table in the fixed distributed server data locating mechanism for NRMB strategy

Cont..

- Scheme is a direct extension of the centralized server scheme
- It overcomes the problems of the centralized server scheme by distributing the role of the centralized server
- Whenever a fault occurs, the mapping functions is used by the fault handler of the faulting node to find out the node whose block manager is mapping the currently accessed block

Dynamic distributed-server algorithm

Fig 8. structure and locations of block table in the dynamic distibuted server data locating mechanism for NRMB strategy

Cont…

- Does not use any block manager and attempts to keep track of the ownership information of all block in each node
- Each node has a block table that contains the ownership information for all block
- A field gives the node a hint on the location of the owner of a block and hence is called the probable owner
- When fault occurs, the faulting node extracts from its block table the node information

Replicated, migrating blocks

- A major disadvantage of the non replication strategies is lack of parallelism
- To increase parallelism, virtually all DSM system replicate blocks
- Replication tends to increase the cost of write operation because for a write to a block all its replica must be invalidated or updated to maintain consistency
- If the read/write ratio is large, the extra expense for the write operation may be more than offset by the lower average cost of the read operation

Cont…

- Two basic protocols that may be used for ensuring sequential consistency in this case are:

write-invalidate:

- When a write fault occurs fault handler copies the accessed block
- If one of the nodes that had a copy of the block before invalidation tries to perform a memory access operation (read/write) on the block after invalidation, a cache miss will occur and the fault handler of that node will have to fetch the block again from a node having a valid copy of the block, therefore the scheme achieves sequential consistency

Cont…

Fig 9 Write invalidate memory coherence approach for replicated, migrating block (RMB) strategy

Write-update

- A write operation is carried out by updating all copies of the data on which the write is perform
- When write fault occurs the fault handler copies the accessed block from one of the block's current node to its owe node
- The write operation completes only after all the copies of the block have been successfully updated
- Sequentially consistency can be achieved by using a mechanism to totally order the write operations of all the node
- The intended modification of each write operation is first sent to the global sequencer

Cont…

Fig 10. write update memory coherence approach for replicated, migrating blocks strategy

Cont…

- Sequence number to the modification and multicasts the modification with this sequence number to all the nodes where a replica of the data block to be modified is located
- The write operations are processed at each node in sequence number order
- If the verification fails, node request the sequencer for a retransmission of the missing modification
- Write-update approach is very expensive
- In write-invalidate approach, updates are only propagated when data are read and several updates can take place before communication is necessary

Cont….

Fig 11. globle sequencing mechanism to sequence the write operations of all nodes

Cont…

Read request:

- If there is a local block containing the data and if it is valid, the request is satisfied by accessing the local copy of data
- Otherwise, the fault handler of the requesting node generates a read fault

Write request:

- If there is a local block containing the data and if it is valid and writable, the request is immediately satisfied by accessing the local copy of the data
- Otherwise, the fault handler of the requesting node generates a write fault and obtain a valid copy of the block

Data Locating in the RMB strategy

- Data-locating issues are involved in the write-invalidate protocol used with the RMB strategy:
  1. Locating the owner of a block, the most recent node to have write access to it
  2. Keeping track of the node that are currently have a valid copy of the block
- Following algorithms may be used:
  1. Broadcasting
  2. Centralized-server algorithm
  3. Fixed distributed-server algorithm
  4. Dynamic distributed-server algorithm

Replicated, Nonmigrating Block

- A shared memory block may be replicated at multiple node of the system but the location of each replica is fixed .
- All replicas of a block are kept consistent by updating them all in case of a write excess.
- Sequential consistency is ensured by using a global sequencer to sequence the write operation of all nodes.

Data locating in the RNMB strategy

- Following characteristics:
  1. The replica location of a block never change
  2. All replicas of a data block are kept consistent
  3. Only a read request can be directly send to one of the node having a replica.

Cont…

structure and location of block table and sequence table in the centralized sequencer data locating mechanism for RNMB strategy

Cont….

- A write operation on a block is sent to sequencer
- The sequencer assign the next sequence number to the requested modification
- It then multicast the modification with this sequence number to all the nodes listed in the replica set field.

Release consistency model

- Enhancement of weak consistency model
- Use of two synchronization variables
  - Acquire (used to tell the system it is entering CR)
  - Release (used to tell the system it has just exited CR)
- Acquire results in propagating changes made by other nodes to process's node
- Release results in propagating changes made by the process to other nodes

Implementation of Release Consistency

- Release consistency application must be modeled around critical sections.
- Munin provides two such synchronization mechanism.
  - A locking mechanism and a barrier mechanism.
- The locking mechanism uses lock synchronization variables with acquirelock and releasedlock primitives for accessing these variables.
- The acquirelock primitive with a lock variable as its parameter is executed by a process to enter a critical section .

Munin: A Release Consistent DSM System

Structure of Shared-Memory Space:

- The shared memory space of Munin is structured as a collection of shared variables.
- The shared variables are declared with the keywords shared so the compiler can recognized them.
- Each shared variable, by default, is placed by the compiler on a separate page.
- Unit of data transfer across the network by MMU hardware.

Cont…

- Releaselock primitive with the same lock variable parameter is executed by the process to exit from the critical section.
- When a process makes an acquirelock request the system first checks if the lock variable on the local node.
- If not, the probable owner mechanism is used to find the location owner of the lock variable and request is sent to that node.

Replacement strategy

- In DSM system that allow shared memory block to be dynamically migrated/replicated

- **Following issue:**
  1. Which block should be replaced to make space for a newly required block?
  2. Where should the replaced block be placed?

Which block to replace

- Classification of replacement algorithms:
  1. Usage based verses non-usage based
  2. Fixed space verses variable space

Where to place a replaced block

- Once a memory block has been selected for replacement, it should be ensured that if there is some useful information in the block , it should not be lost.
- The two commonly used approaches for storing a useful block as follow:
- **Using secondary store:**
  - The block is simply transferred on to a local disc.
  - **Advantages:** it does not waste any memory space

Cont…

- **Using the memory space of other nodes:**
  - It may be faster to transfer a block over the network than to transfer it to a local disc.
  - Methods require each node to maintain a table of free memory space in all other nodes.

Thrashing

- Thrashing is said to occur when the system spends a large amount of time transferring shared data blocks from one node to another
- Thrashing may occur in following situation:
  1. When interleaved data accesses made by processes on two or more nodes
  2. When blocks with read only permissions are repeatedly invalidated soon after they are replicated
- Thrashing degrades system performance considerably

Other approaches to DSM

- There are three main approaches for designing a DSM system
  1. Data caching managed by the operating system
  2. Data caching managed by MMU hardware
  3. Data caching managed by the language runtime system

- In the first approach each node has its own memory and access to a word in another nodes memory causes a trap to the operating system then fetches and acquires the page.

Heterogeneous DSM

- A heterogeneous computing environment allow the applications to exploit the best of all characteristics features of several different types of computers.
- Measurements made on their experimental prototype heterogeneous DSM, called Mermaid.
- Heterogeneous DSM is not only feasible but can also be comparable in performance to its homogeneous counterpart
- The two issues in building a DSM system:
  - Data conversion and selection of block size
  - Machines of different architecture may used different bytes ordering and floating point representation.

Structuring the DSM system as a collection of source language objects

- It is structured as a collection of variables or object so that the unit of data migration is an object instead of a block
- A suitable conversion runtime is used to translate the object before migrating its to requesting node.
- This method of data conversion is used in the Agora shared memory system.
- Array may easily be too large to be treated as unit of sharing and data migration.

Allowing only one type of data in a block

- This mechanism is used in Mermaid which uses a page size as its block size therefore , a page can contain only one type of data.
- Whenever a page is moved between two machines of different architecture, a routine converts the data in the page to the appropriate format

Limitations :

- Allowing a page to content data of only one type may lead to wastage of memory due to fragmentation, resulting increased paging activity

Cont…

- Compilers used of different types of machine must be compatible
- Another problem that entire pages are converted even thought only a small portion may be accessed before it is transferred away
- The mechanism is not fully transparent
- Another serious problem associated, accuracy of floating point value in numerical applications

Block size selection

- In the Homogeneous DSM system, the block size is usually the same size as a native virtual memory (VM)
- MMU hardware can be used to trigger a DSM block fault.
- In heterogeneous, the virtual memory page size may be different for machine of different types.
- Block size selection become a complicated task
- **Following algorithm for block size selection**:
  1. Largest page size algorithm
  2. Smallest page size algorithm
  3. Intermediate page size algorithm

Distributed File System (DFS)

Distributed File System (DFS)

- A Distributed File System (DFS) as the name suggests, is a file system that is distributed on multiple file servers or multiple locations.
- It allows programs to access or store isolated files as they do with the local ones, allowing programmers to access files from any network or computer.
- The main purpose of the Distributed File System (DFS) is to allows users of physically distributed systems to share their data and resources by using a Common File System.
- A collection of workstations and mainframes connected by a Local Area Network (LAN) is a configuration on Distributed File System.

In DFS, a namespace is created and this process is transparent for the clients.

DFS has two components:

Location Transparency –

Location Transparency achieves through the namespace component.

Redundancy –

Redundancy is done through a file replication component.

In the case of failure and heavy load, these components together improve data availability by allowing the sharing of data in different locations to be logically grouped under one folder, which is known as the “DFS root”.

It is not necessary to use both the two components of DFS together, it is possible to use the namespace component without using the file replication component and it is perfectly possible to use the file replication component without using the namespace component between servers.

Features of DFS :

1. **Transparency** :

Structure transparency – There is no need for the client to know about the number or locations of file servers and the storage devices. Multiple file servers should be provided for performance, adaptability, and dependability.

Access transparency –Both local and remote files should be accessible in the same manner. The file system should be automatically located on the accessed file and send it to the client's side.

Naming transparency –There should not be any hint in the name of the file to the location of the file. Once a name is given to the file, it should not be changed during transferring from one node to another.

Replication transparency – If a file is copied on multiple nodes, both the copies of the file and their locations should be hidden from one node to another.

User mobility : It will automatically bring the user's home directory to the node where the user logs in.

Performance : Performance is based on the average amount of time needed to convince the client requests. This time covers the CPU time + time taken to access secondary storage + network access time. It is advisable that the performance of the Distributed File System be similar to that of a centralized file system.

S**implicity and ease of use :** The user interface of a file system should be simple and the number of commands in the file should be small.

High availability :

A Distributed File System should be able to continue in case of any partial failures like a link failure, a node failure, or a storage drive crash. A high authentic and adaptable distributed file system should have different and independent file servers for controlling different and independent storage devices.

High reliability :

The likelihood of data loss should be minimized as much as feasible in a suitable distributed file system. That is, because of the system's unreliability, users should not feel forced to make backup copies of their files. Rather, a file system should create backup copies of key files that can be used if the originals are lost. Many file systems employ stable storage as a high-reliability strategy.

Data integrity : Multiple users frequently share a file system. The integrity of data saved in a shared file must be guaranteed by the file system. That is, concurrent access requests from many users who are competing for access to the same file must be correctly synchronized using a concurrency control method. Atomic transactions are a high-level concurrency management mechanism for data integrity that is frequently offered to users by a file system.

Heterogeneity: Heterogeneity in distributed systems is unavoidable as a result of huge scale. Users of heterogeneous distributed systems have the option of using multiple computer platforms for different purposes.

Security: A distributed file system should be secure so that its users may trust that their data will be kept private. To safeguard the information contained in the file system from unwanted & unauthorized access, security mechanisms must be implemented.

FILE MODELS

- Different file system use the diffent conceptual models.
- The two most commonly used for a file modeling are,
  - Structure
  - Mobility
- File model based on these criterias.

Unstructured and structured files

- It is the simplest model.
- A file is an unstructured sequence of data.
- There is no "Substructure"known to the file server.
- The contents of each file of the file system appers to the file server as an "Uninterpreted sequence of data".
- The os is not interest in the information stored in the files.
- Eg,UNIX and DOS use this file model.

- Another file model is rarely used the "Stucture file model".
- In this model ,a file server is an "Ordered sequence of records".
- Records of different file of the same file system and can be different size.
- Many type of files exist on file system each having different properties.
- A record is the smallest unit of data can be accessed,a file system read or write operations are carried out on a "Set of records".

- Structured files are two types
  - Indexed records.
  - Nonindexed records.
- A file record is accessed by specifying its positionWithin the file.
- A file system that allow a indexed records.
- A file is maintained as a B –tree or suitable data
- Structure .
- A hash table is used to locate records are quickly.

- Most modern systems use the "Unsturucred file model".
- This is mainly used for "Sharing of file" by different applicatin.
- A file has a no structured in the unstructured model,different applications can interprete a file in different ways.
- A file also normally have "Attributes".
- A file's attributes are information describing that file.

- Each attribute has a name and value.
- The types of attributes can be associated with a file are fixed by the file system.
- File attributes are normally maintained by the “Directory service”.
- They are stored with the corresponding file.
- A many directory system allow files to be referenced by more than one name.

MUTABLE FILES:

****

- Most existing os use the mutable file model.
- In this model,a update performed on a file overwrites.
- Its old contents produce to new contents.
- A file is represented as a single stored sequence that is already by each update operation.

IMMUTABLE FILES

****

- Some more file system ,such as (CFS)
- [Gifford et al.1988]use the immutable model.
- CFS-CEDAR FILE SYSTEM.
- A file can't be modified.
- Once the file is created,except to be deleted.
- To implement file updates,and each file is represented by a immutable versions.

- Giffort et al.[1988] emphasized that sharing only immutable files makes it easy to support "Consistent sharing".
- It is much easier to support file caching and replication in a distributed system.
- Immutable file model eliminates all problems associated with multible versions of a file consistent.
- The immutable file model suffers from two potential problem,

  * Incresed use of disk space.
  * Incresed a disk allocation activity.

- Prevent the disk space from filling instantaneously.
- CFS[Giffort et al.1988] uses a “Keep” parameter as the no of most current versions of a file retained.
- For eg,”demo.abc!6” is a name for version6.
- The base name of “demo.abc”.
- If the value of the keep parameter of a file is 1,the creation of new version of the file cause the file
- “Demo.abc!6 “to be deleted.
- And its diskspace to be reused for the new file named “demo.abc!7”.

- When the keep parameter value is greater than 1,for a file that is,when multible versions of a file exist.
- Users may use a particular version of the file by specifying its full name.
- When the version number is not specified,by default,CFS uses the lowest version number for
- Some operations such as “Delete”.
- And highest versin number for other operation such as “Open”.

File Accessing Models

File Accessing Models of DFS mainly depends on : Method used for accessing remote files and the unit of data access

Accessing Remote Files

- Remote Service Model:
  - Client's request processed at server's node
  - In this case Packing and communication overhead can be significant
- Data Caching Model:
  - Client's request processed on the client's node itself by using the cached data.
  - This model greatly reduces network traffic
  - Cache consistency problem may occur

LOCUS and NFS use the remote service model but add caching fro better performance

Sprite use data caching model but employs the remote service model under certain circumstances

File Accessing Models

Unit of Data Transfer

- File Level Transfer Model(Ex. Amoeba, AFS)
  - The whole file is moved when an operation requires file data
  - It is simple, It has better scalability
  - Disk access routines on the servers can be better optimized
  - But it requires sufficient storage space on client's node
- Block Level Transfer Model(Ex. LOCUS, Sprite)
  - Data transferred in units of file blocks
  - It does not require client node to have large storage space
  - It can be used in diskless workstations
  - Network traffic may be significant
- Byte Level Transfer Model(Cambridge file server)
  - Data transfers in units of bytes
  - Low Storage requires but difficulty in cache management
- Record Level Transfer Model(Research Storage System)
  - Suitable for Structured model

File Caching Schemes

File Caching Schemes

- File caching has been implemented in centralized time sharing system to improve I/O performance.
- The file is cached in the main memory so that repeated accesses to the same information can be handled without additional disk transfer.
- In distributed file system, it also helps in scalability and reliability. There are certain schemes for distributed file caching system.

i. Cache location

ii. Modification Propagation

iii. Cache Validation schemes

Cache location

- Cache location refers to the place where the cached data is stored.
- There are certain 3 possibility of cache location considering original file is at server disk.
  - **a. Server's main memory** – when no caching is used, generally the file is first transferred from the server's disk to the server's memory. Then file is transferred to client main memory.
  - **b. Client's disk –** the second option is to have cache in the client's disk. Keeping file cached at client’s location has many advantage s – it eliminates network access cost, improve reliability, doesn’t get effected by data lost in a crash, also has more memory as compared with the main memory cache. Disadvantages can be – it will not support diskless workstation, will cost disk access cost.
  - **c. Client’s main memory –** the third option is to have the cache in the client’s main memory. This option eliminate s both the network cost and disk access cost.

Client's Disk
Client's Main Memory
4
3
Server's Disk
Server's Main Memory
2
1

ii. Modification Propagation

- In distributed file system, when the cache is located at the client ‘s side, it may be cached on multiple nodes. In such situation, when the caches all the nodes needed to be consistent if any node modified the cached files.
- Then same needed to be replicated at all the nodes. Hence there are certain modification propagation followed by distributed file system. It has 2 schemes:
  - **a. Write- through schemes** – in this scheme, when a cache entry is modified, the new value is immediately send to the server to update the master copy of the file. This removes the lost updated file in crash case. But it does have poor write performance.
  - **b. Delayed-write schemes –** wrote through scheme helps in read but does increase network traffic. In this scheme, when cache entry is modified it is kept at client’s cache and when all the entries are modify then the entire cache is send together. It also certain way as write on ejection from cache, periodic write and write on close.

iii. Cache Validation schemes

- Now it’s clear that multiple copies if cached files are kept by on multiple nodes of clients.
- Modifications propagation specify when to 4 update the master copy at server side when cache is modified at clients side. This scheme tells when the other copies at multiple Nodes should be updated.
- It has 2 approaches:
  - a. Client- Initiated Approach – In this approach client contact the server whether it has same copy as the master one. It again follows 3 ways to check.
    - Checking before every access
    - Periodic checking at regular interval of time
    - Checking whenever clients open the file – it uses session semantics

iii. Cache Validation schemes

b. Server-Initiated Approach –

- Since client initiated approach results in traffic congestion, hence server initiated approach is used.
- In this approach client informs the file server that it is opening a file in which mode(reading, writing, or both).
- A server keeps monitoring the file usages .
- When a client closes the file, an intimation is send to server along with any modifications made to the file.
- On receiving the intimation it update the file at its end.

File Replication

- High availability is a desirable feature of a good distributed file system and *file replication* is the primary mechanism for improving file availability.

- A *replicated file* is a file that has multiple copies, with each file on a separate file server.

Difference Between Replication and Caching

- A replica of a file is associated with a server, whereas a cached copy is normally associated with a client.
- The existence of a cached copy is primarily dependent on the locality in file access patterns, whereas the existence of a replica normally depends on availability and performance requirements.
- As compared to a cached copy, a replica is more persistent, widely known, secure, available, complete, and accuracy.
- A cached copy is contingent upon a replica. Only by periodic revalidation with respect to a replica can a cached copy be useful.

Advantages of Replication

- Increased Availability.
  Alternate copies of a replicated data can be used when the primary copy is unavailable.
- Increased Reliability.
  Due to the presence of redundant data files in the system, recovery from catastrophic failures (e.g. hard drive crash) becomes possible.
- Improved response time.
  It enables data to be accessed either locally or from a node to which access time is lower than the primary copy access time.
- Reduced network traffic.
  If a files replica is available with a file server that resides on a clients node, the clients access request can be serviced locally, resulting in reduced network traffic.
- Improved system throughput.
  Several clients request for access to a file can be serviced in parallel by different servers, resulting in improved system throughput.
- Better scalability.
  Multiple file servers are available to service client requests since due to file replication. This improves scalability.

Replication Transparency

- Replication of files should be transparent to the users so that multiple copies of a replicated file appear as a single logical file to its users.
- This calls for the assignment of a single identifier/name to all replicas of a file.
- In addition, replication control should be transparent, i.e., the number and locations of replicas of a replicated file should be hidden from the user.
- Thus replication control must be handled automatically in a user-transparent manner.

Multicopy Update Problem

- Maintaining consistency among copies when a replicated file is updated is a major design issue of a distributed file system that supports file replication.

1. Read-only replication

   In this case the update problem does not arise. This method is too restrictive.

2. Read-Any-Write-All Protocol

   A read operation on a replicated file is performed by reading any copy of the file and a write operation by writing to all copies of the file. Before updating any copy, all copies need to be locked, then they are updated, and finally the locks are released to complete the write.

Disadvantage: A write operation cannot be performed if any of the servers having a copy of the replicated file is down at the time of the write operation.

3. Available-Copies Protocol

A read operation on a replicated file is performed by reading any copy of the file and a write operation by writing to all **available** copies of the file. Thus if a file server with a replica is down, its copy is not updated. When the server recovers after a failure, it brings itself up to date by copying from other servers before accepting any user request.

4. Primary-Copy Protocol

For each replicated file, one copy is designated as the primary copy and all the others are secondary copies. Read operations can be performed using any copy, primary or secondary. But write operations are performed only on the primary copy. Each server having a secondary copy updates its copy either by receiving notification of changes from the server having the primary copy or by requesting the updated copy from it.

E.g. for UNIX-like semantics, when the primary-copy server receives an update request, it immediately orders all the secondary-copy servers to update their copies. Some form of locking is used and the write operation completes only when all the copies have been updated. In this case, the primary-copy protocol is simply another method of implementing the read-any-write-all protocol.

Fault Tolerance

- Fault Tolerance simply means a system's ability to continue operating uninterrupted despite the failure of one or more of its components.
- This is true whether it is a computer system, a cloud cluster, a network, or something else.
- In other words, fault tolerance refers to how an operating system (OS) responds to and allows for software or hardware malfunctions and failures.
- Various types of fault could harm the integrity of the data stored by distributed file system
- Examples - a) In event of crash: processors loses the content of the Memory. Such failure s could result in logically complete but physically incomplete file operations. This results in file inconsistency.
- b) In case of request processing between client and server, loss of information state.

Fault Tolerance

The primary file properties that directly influence the ability of a distributed file system to tolerate faults are as follows :

1. Availability. Availability of a file refers to the fraction of time for which the file is available for use. Note that the availability property depends on the location of the file and the locations of its clients (users).

For example, if a network is partitioned due to a communication link failure, a file may be available to the clients of some nodes, but at the same time, it may not be available to the clients of other nodes. Replication is a primary mechanism for improving the availability of a file.

2. Robustness. Robustness of a file refers to its power to survive crashes of the storage device and decays of the storage medium on which it is stored. Storage devices that are implemented by using redundancy techniques, such as a stable storage device, are often used to store robust files.

3. Recoverability.

Recoverability of a file refers to its ability to be rolled back to an earlier, consistent state when an operation on the file fails or is aborted by the client. Notice that a robust file is not necessarily recoverable and vice versa. Atomic update techniques such as a transaction mechanism are used to implement recoverable files.

Storage

- In context to crash resistance capabilities, storage may be classified into 3 types:

- **Volatile Storage –** this kind of storage Cannot withstand any kind of failure , whether it is power failure or system failure. This means that the data stored in a volatile storage is lost in the event of any kind of failure.

  Example is RAM.

- **Nonvolatile Storage –** this kind of storage can withstand CPU Failure but cannot withstand transient I/O failure or decay of the storage media.

  Example is Disk.

- **Stable Storage** – this kind can withstand both the transient failure and decay of the storage media. The basic idea of stable storage is the use of duplicate storage devices. This way it keep the device stable.

Storage

- The main idea to keep disk 2 advanced exact copy of disk 1, that is each block on disk 2 is an exact copy of corresponding block on disk 1.
- To ensure that both the disk is not damage at the same time, some restrictions on how the two disks are accessed.
- Two basic operations – read & write is Synchronized. A read operation first attempts to Read from disk 1.If it fails, the read is done from disk 2.
- A write operation writes to both disks but write to disk 2 does not start until disk 1 has been successfully completed.

Effects of Service Paradigm on Fault Tolerance

- There are 2 main paradigms by which server may be implemented.

1. Stateful File Servers – In this paradigm, a stateful file server maintain s the client's state information. For subsequent requests made by the client to the stateful server for accessing the file, some state information stores the service performed for the client as a Result of first request execution.

- To allow the file server to decide how long to retain the state information till a session is established. To understand how Stateful file server works :-

- **Open(filename mode) :** this operation is used to open a file identified by Filename in the specified mode. When the server executes the operation, it creates an entry in the file-table. The file-table maintain the file state information of all the open file. This consist of the identifier of the file, the open mode and the read- write pointer which is set to 0.

Stateful File Servers

- **Read(fid, n, buffer) –** this operation is used to get n bytes of data from the file identified by fid into specified buffer. When server executes this operation, it returns to the client n bytes starting from read-write pointer and then increment by n.

- **Write(fid, n, buffer) -** this operation is used on execution of the operation, the server take n bytes of data from the specified buffer, writes it on the file identified by fid specified by currently addressed by read-write pointer and then increment by n.

- **Seek(fid, position) –** this operation cause the server to change the Value if the read-write pointer of the file identified by fid to the new value specified as position.

- **Close(fid) –** this statement cause the server to delete from its file-table the file state information of the file identified by fid.

Stateless File Servers

- A stateless file Server does not maintain any client state information.
- Therefore, every request from a client must be accompanied with all the necessary parameters to successfully carry out the desired operation.
- That is, each request identifies the file and the position in the file for read/write access.
- It has 2 operations :

1. Read(filename, position, n, buffer) – this operation specify that the server returns to the client n bytes of data of the file identified by filename. The returned data is placed in the specified buffer. The Position specify the from where the Reading begins.

2. Write(filename, position, n, buffer) – when the server executes this operation, it takes n bytes of data from the specified buffer. And writes into the file identified by filename. The position parameter specified the position within the file from where to start writing. The server returns the actual number of bytes written.

Stateful vs Stateless File Server

• There are certain points which shows significant difference between stateful and stateless file server. They are as follows:-

1. Parameter State –

• Stateful – this remember the client data (state) from one request to next.

• Stateless – this doesn't keep the client data information.

2. Programming • Stateful – it is harder to code.

• Stateless – it is straightforward to code.

3.. Crash recovery • Stateful – difficult due to loss of information.

• Stateless – can easily recover from failure as there no client state.

4. Information transfer

• Stateful – client send less data with each request.

• Stateless – client has to specify complete information with each request.

5. Extra Service • Stateful – it offers file locking mechanism.

• Stateless – no locking mechanism.

Atomic Transaction

- An atomic transaction (or just transaction for short) is a computation consisting of a collection of operations that take place indivisibly in the presence of failures and concurrent computations.
- That is, either all of the operations are performed successfully or none of their effects prevail, and other processes executing concurrently cannot modify or observe intermediate states of the computation.
- Transactions help to preserve the consistency of a set of shared data objects in the face of failures and concurrent access.
- They make crash recovery much easier, because a transaction can only end in two states transaction carried out completely or transaction failed completely.

ACID properties of transactions

In the context of transaction processing, the acronym ACID refers to the four key properties of a transaction: atomicity, consistency, isolation, and durability.

Atomicity

All changes to data are performed as if they are a single operation. That is, all the changes are performed, or none of them are.

For example, in an application that transfers funds from one account to another, the atomicity property ensures that, if a debit is made successfully from one account, the corresponding credit is made to the other account.

Consistency

Data is in a consistent state when a transaction starts and when it ends.

For example, in an application that transfers funds from one account to another, the consistency property ensures that the total value of funds in both the accounts is the same at the start and end of each transaction.

Isolation

The intermediate state of a transaction is invisible to other transactions. As a result, transactions that run concurrently appear to be serialized.

For example, in an application that transfers funds from one account to another, the isolation property ensures that another transaction sees the transferred funds in one account or the other, but not in both, nor in neither.

Durability

After a transaction successfully completes, changes to data persist and are not undone, even in the event of a system failure.

For example, in an application that transfers funds from one account to another, the durability property ensures that the changes made to each account will not be reversed.

Operations for Transaction Based File Service

In a file system that provides a transaction facility, a transaction consists of a sequence of elementary file access operations such as read and write. Therefore, the client interface to such a file system must include special operations for transaction service. The three essential operations for transaction service are as follows:

begin_transaction returns (TID): Begins a new transaction and returns a unique transaction identifier (TID). This identifier is used in other operations of this transaction. All operations within a begin transaction and an end_transaction form the body of the transaction.

end_transaction (TID) returns (status): This operation indicates that, from the viewpoint of the client, the transaction completed successfully. Therefore the transaction is terminated and an attempt is made to commit it. The returned status indicates whether the transaction has committed or is inactive because it was aborted by either the client or the server. If the transaction commits, all of its changes are made permanent and subsequent transactions will see the results of all the changes to files made by this transaction. On the other hand, if the transaction was aborted, none of the changes requested so far within the transaction will become visible to other transactions. A transaction is aborted either on explicit request by the client or in the event of system failures that disrupt the execution of the transaction.

abort_transaction (TID): Aborts the transaction, restores any changes made so far within the transaction to the original values, and changes its status to inactive. A transaction is normally aborted in the event of some system failure. However, a client may use this primitive to intentionally abort a transaction.

Concurrency Control Techniques

Concurrency Control is provided in a database to:

(i) enforce isolation among transactions.

(ii) preserve database consistency through consistency preserving execution of transactions.

(iii) resolve read-write and write-read conflicts.

- Various concurrency control techniques are:
- **1.** Two-phase locking Protocol
- **2.** Time stamp ordering Protocol
- **3.** Multi version concurrency control
- **4.** Validation concurrency control

These are briefly explained below.

1. Two-Phase Locking Protocol:

Locking is an operation which secures: permission to read, OR permission to write a data item. Two phase locking is a process used to gain ownership of shared resources without creating the possibility of deadlock.

The 3 activities taking place in the two phase update algorithm are:

(i). Lock Acquisition

(ii). Modification of Data

(iii). Release Lock

Two phase locking prevents deadlock from occurring in distributed systems by releasing all the resources it has acquired, if it is not possible to acquire all the resources required without waiting for another process to finish using a lock. This means that no process is ever in a state where it is holding some shared resources, and waiting for another process to release a shared resource which it requires. This means that deadlock cannot occur due to resource contention.

A transaction in the Two Phase Locking Protocol can assume one of the 2 phases:

- **(i) Growing Phase:**
  In this phase a transaction can only acquire locks but cannot release any lock. The point when a transaction acquires all the locks it needs is called the Lock Point.
- **(ii) Shrinking Phase:**
  In this phase a transaction can only release locks but cannot acquire any
- .**2.** Time Stamp Ordering Protocol**:**
- A timestamp is a tag that can be attached to any transaction or any data item, which denotes a specific time on which the transaction or the data item had been used in any way. A timestamp can be implemented in 2 ways. One is to directly assign the current value of the clock to the transaction or data item. The other is to attach the value of a logical counter that keeps increment as new timestamps are required.

The timestamp of a data item can be of 2 types:

- **(i) W-timestamp(X):**
  This means the latest time when the data item X has been written into.
- **(ii) R-timestamp(X):**
  This means the latest time when the data item X has been read from. These 2 timestamps are updated each time a successful read/write operation is performed on the data item X.

3. Multiversion Concurrency Control:

Multiversion schemes keep old versions of data item to increase concurrency.

Multiversion 2 phase locking:

Each successful write results in the creation of a new version of the data item written. Timestamps are used to label the versions. When a read(X) operation is issued, select an appropriate version of X based on the timestamp of the transaction.

4. Validation Concurrency Control**:**

The optimistic approach is based on the assumption that the majority of the database operations do not conflict. The optimistic approach requires neither locking nor time stamping techniques. Instead, a transaction is executed without restrictions until it is committed. Using an optimistic approach, each transaction moves through 2 or 3 phases, referred to as read, validation and write.

- **(i)** During read phase, the transaction reads the database, executes the needed computations and makes the updates to a private copy of the the database values. All update operations of the transactions are recorded in a temporary update file, which is not accessed by the remaining transactions.
- **(ii)** During the validation phase, the transaction is validated to ensure that the changes made will not affect the integrity and consistency of the database. If the validation test is positive, the transaction goes to a write phase. If the validation test is negative, he transaction is restarted and the changes are discarded.
- **(iii)** During the write phase, the changes are permanently applied to the database.

Resource Management

Resource Management

- A resource can be a logical, such as a shared file, or physical, such as a CPU (a node of the distributed system).

- One of the functions of a distributed operating system is to assign processes to the nodes (resources) of the distributed system such that the resource usage, response time, network congestion, and scheduling overhead are optimized.

Resource Management

- A resource manager of a distributed operating system schedules the processes in a distributed computing environment to a pool of free resources that can optimize a combination of resource usage, response time, network congestion and scheduling overhead.

- The process scheduling decisions are based on factors such as the resource requirements of the processes, the availability of resources on different nodes of the system and the static or dynamic state information of the resources on different nodes of the system.

Resource Management Scheduling Techniques in DS

- Task Assignment Approach
- Load Balancing Approach
- Load Sharing Approach

1. Task Assignment Approach,

- In which each process submitted by a user for processing is viewed as a collection of related tasks.
- These tasks are scheduled to suitable nodes so as to improve performance.

2. Load-balancing approach,

- In which all the processes submitted by the users are distributed among the nodes of the system.
- So as to equalize the workload among the nodes.

3. Load-sharing approach

- In which simply attempts to conserve the ability of the system to perform work by assuring that no node is idle while processes wait for being processed.

Desirable features of a good Scheduling Algorithms

- No prior knowledge about the processes.
- Dynamic in nature.
- Quick decision-making capability.
- Balanced system performance.
- Stability.
- Scalability.
- Fault tolerance.
- Fairness of service.

No prior knowledge about the processes

- Scheduling algorithms that operate based on the information about the characteristics and resource requirements of the processes pose an extra burden on the users who must provide this information while submitting their processes for execution.

Dynamic in nature

- Process assignment decisions should be dynamic, i.e., be based on the current load of the system and not on some static policy.
- It is recommended that the scheduling algorithm possess the flexibility to migrate a process more than once because the initial decision of placing a process on a particular node may have to be changed after some time to adapt to the new system load.

Quick decision making capability

- Heuristic methods requiring less computational efforts (and hence less time) while providing near-optimal results are preferable to exhaustive (optimal) solution methods.

Balanced system performance

- Scheduling overhead and Algorithms that provide near-optimal system performance with a minimum of global state information (such as CPU load) gathering overhead are desirable.
- This is because the overhead increases as the amount of global state information collected increases.
- This is because the usefulness of that information is decreased due to both the aging of the information being gathered and the low scheduling frequency as a result of the cost of gathering and processing the extra information.

Stability

- Fruitless migration of processes Known as processor thrashing, must be prevented. E.g. if nodes n1 and n2 observe that node n3 is idle and then offload a portion of their work to n3 without being aware of the offloading decision made by the other node.
- Now if n3 becomes overloaded due to this it may again start transferring its processes to other nodes.
- This is caused by scheduling decisions being made at each node independently of decisions made by other nodes.

Scalability

- A scheduling algorithm should scale well as the number of nodes increases.
- An algorithm that makes scheduling decisions by first inquiring the workload from all the nodes and then selecting the most lightly loaded node has poor scalability.
- This will work fine only when there are few nodes in the system.
- This is because the inquirer receives a flood of replies almost simultaneously, and the time required to process the reply messages for making a node selection is too long as the number of nodes (N) increase.
- Also the network traffic quickly consumes network bandwidth.
- A simple approach is to probe only m of N nodes for selecting a node.

Fault tolerance

- A good scheduling algorithm should not be disabled by the crash of one or more nodes of the system.
- Also, if the nodes are partitioned into two or more groups due to link failures, the algorithm should be capable of functioning properly for the nodes within a group.
- Algorithms that have decentralized decision making capability and consider only available nodes in their decision making have better fault tolerance capability.

Fairness of service

- Global scheduling policies that blindly attempt to balance the load on all the nodes of the system are not good from the point of view of fairness of service.
- This is because in any load-balancing scheme, heavily loaded nodes will obtain all the benefits while lightly loaded nodes will suffer poorer response time than in a stand-alone configuration.
- A fair strategy that improves response time of the former without unduly affecting the latter is desirable.
- Hence load-balancing has to be replaced by the concept of load sharing, that is, a node will share some of its resources as long as its users are not significantly affected.

1. Task Assignment Approach,

Assumptions:

1. A process has already been split up into pieces called *tasks*.
2. This split occurs along natural boundaries (such as a method), so that each task will have integrity in itself and data transfers among the tasks are minimized.
3. The amount of computation required by each task and the speed of each CPU are known.
4. The cost of processing each task on every node is known. This is derived from assumption 2.
5. The IPC costs between every pair of tasks is known. The IPC cost is 0 for tasks assigned to the same node. This is usually estimated by an analysis of the static program. If two tasks communicate n times and the average time for each inter-task communication is t, them IPC costs for the two tasks is $n * t$.
6. Precedence relationships among the tasks are known.
7. Reassignment of tasks is not possible.

Task Assignment Approach,

- Goal is to assign the tasks of a process to the nodes of a distributed system in such a manner as to achieve goals such as the following goals:
  - Minimization of IPC costs
  - Quick turnaround time for the complete process
  - A high degree of parallelism
  - Efficient utilization of system resources in general

These goals often conflict. E.g., while minimizing IPC costs tends to assign all tasks of a process to a single node, efficient utilization of system resources tries to distribute the tasks evenly among the nodes.

So also, quick turnaround time and a high degree of parallelism encourage parallel execution of the tasks, the precedence relationship among the tasks limits their parallel execution.

Also note that in case of m tasks and q nodes, there are m^q possible assignments of tasks to nodes . In practice, however, the actual number of possible assignments of tasks to nodes may be less than m^q due to the restriction that certain tasks cannot be assigned to certain nodes due to their specific requirements (e.g. need a certain amount of memory or a certain data file).

- There are two nodes, {n1, n2} and six tasks {t1, t2, t3, t4, t5, t6}. There are two task assignment parameters – the task execution cost (x_{ab} the cost of executing task a on node b) and the inter-task communication cost (c_{ij} the inter-task communication cost between tasks *i* and *j*).

Inter-task communication cost

	t1	t2	t3	t4	t5	t6
t1	0	6	4	0	0	12
t2	6	0	8	12	3	0
t3	4	8	0	0	11	0
t4	0	12	0	0	5	0
t5	0	3	11	5	0	0
t6	12	0	0	0	0	0

Execution costs

	Nodes	
	n1	n2
t1	5	10
t2	2	∞
t3	4	4
t4	6	3
t5	5	2
t6	∞	4

Serial assignment	
Task	Node
t_1	n_1
t_2	n_1
t_3	n_1
t_4	n_2
t_5	n_2
t_6	n_2

(c)

Task t6 cannot be executed on node n1 and task t2 cannot be executed on node n2 since the resources they need are not available on these nodes.

1) **<u>Serial assignment</u>**, where tasks t1, t2, t3 are assigned to node n1 and tasks t4, t5, t6 are assigned to node n2:

Execution cost, $x = x_{11} + x_{21} + x_{31} + x_{42} + x_{52} + x_{62}$

$= 5 + 2 + 4 + 3 + 2 + 4$

$= 20$

Communication cost, $c = c_{14} + c_{15} + c_{16} + c_{24} + c_{25} + c_{26} + c_{34} + c_{35} + c_{36}$

$= 0 + 0 + 12 + 12 + 3 + 0 + 0 + 11 + 0$

$= 38.$

Hence total cost = 58.

2) **<u>Optimal assignment</u>**, where tasks t1, t2, t3, t4, t5 are assigned to node n1 and task t6 is assigned to node n2.

Execution cost, $x = x_{11} + x_{21} + x_{31} + x_{41} + x_{51} + x_{62}$

$= 5 + 2 + 4 + 6 + 5 + 4 = 26$

Communication cost, $c = c_{16} + c_{26} + c_{36} + c_{46} + c_{56}$

$= 12 + 0 + 0 + 0 + 0 = 12$

Total cost = 38

Optimal assignments are found by first creating a static assignment graph. In this graph, the weights of the edges joining pairs of task nodes represent inter-task communication costs. The weight on the edge joining a task node to node n1 represents *the execution cost of that task on node n2 and vice-versa*. Then we determine a *minimum cutset* in this graph.

A ***cutset*** is defined to be a set of edges such that when these edges are removed, the nodes of the graph are partitioned into two disjoint subsets such that nodes in one subset are reachable from n1 and the nodes in the other are reachable from n2. Each task node is reachable from either n1 or n2. The weight of a cutset is the sum of the weights of the edges in the cutset. This sums up the execution and communication costs for that assignment. An optimal assignment is found by finding a minimum cutset.

Assignment Graph with Minimum Cost Cut

Min Cutset = 2 + 4 + 12 + 5 + 4 + 6 + 5

= 38

Load-balancing approach

- A Taxonomy of Load-Balancing Algorithms

Load-balancing approach

Type of load-balancing algorithms

- Static versus Dynamic
  - Static algorithms use only information about the average behavior of the system
  - Static algorithms ignore the current state or load of the nodes in the system
  - Dynamic algorithms collect state information and react to system state if it changed
  - Static algorithms are much more simpler
  - Dynamic algorithms are able to give significantly better performance

Deterministic versus Probabilistic

- Deterministic algorithms use the information about the properties of the nodes and the characteristic of processes to be scheduled.
- Probabilistic algorithms use information of static attributes of the system (e.g. number of nodes, processing capability, topology) to formulate simple process placement rules
- Deterministic approach is difficult to optimize Probabilistic approach has poor performance

Centralized versus Distributed

- Centralized approach collects information to server node and makes assignment decision
- Distributed approach contains entities to make decisions on a predefined set of nodes
- Centralized algorithms can make efficient decisions, have lower fault-tolerance
- Distributed algorithms avoid the bottleneck of collecting state information and react faster

Cooperative versus Non Cooperative

- In Noncooperative algorithms entities act as autonomous ones and make scheduling decisions independently from other entities
- In Cooperative algorithms distributed entities cooperate with each other
- Cooperative algorithms are more complex and involve larger overhead
- Stability of Cooperative algorithms are better

Issues in designing Load-balancing algorithms

- Load estimation policy
  - determines how to estimate the workload of a node
- Process transfer policy
  - determines whether to execute a process locally or remote
- State information exchange policy
  - determines how to exchange load information among nodes
- Location policy
  - determines to which node the transferable process should be sent
- Priority assignment policy
  - determines the priority of execution of local and remote processes
- Migration limiting policy
  - determines the total number of times a process can migrate

Load estimation policy I.

for Load-balancing algorithms

- To balance the workload on all the nodes of the system, it is necessary to decide how to measure the workload of a particular node
- Some measurable parameters (with time and node dependent factor) can be the following:
  - Total number of processes on the node
  - Resource demands of these processes
  - Instruction mixes of these processes
  - Architecture and speed of the node's processor
- Several load-balancing algorithms use the <u>total number of processes</u> to achieve big efficiency

Process transfer policy I.

for Load-balancing algorithms

- Most of the algorithms use the *threshold policy* to decide on whether the node is lightly-loaded or heavily-loaded
- Threshold value is a limiting value of the workload of node which can be determined by
  - Static policy: predefined threshold value for each node depending on processing capability
  - Dynamic policy: threshold value is calculated from average workload and a predefined constant
- Below threshold value node accepts processes to execute, above threshold value node tries to transfer processes to a lightly-loaded node

Process transfer policy II.

for Load-balancing algorithms

- Single-threshold policy may lead to unstable algorithm because underloaded node could turn to be overloaded right after a process migration

Single-threshold policy

Double-threshold policy

- To reduce instability double-threshold policy has been proposed which is also known as high-low policy

- Bidding method
  - Nodes contain managers (to send processes) and contractors (to receive processes)
  - Managers broadcast a request for bid, contractors respond with bids (prices based on capacity of the contractor node) and manager selects the best offer
  - Winning contractor is notified and asked whether it accepts the process for execution or not
  - Full autonomy for the nodes regarding scheduling
  - Big communication overhead
  - Difficult to decide a good pricing policy

- Pairing
  - Contrary to the former methods the pairing policy is to reduce the variance of load only between pairs
  - Each node asks some randomly chosen node to form a pair with it
  - If it receives a rejection it randomly selects another node and tries to pair again
  - Two nodes that differ greatly in load are temporarily paired with each other and migration starts
  - The pair is broken as soon as the migration is over
  - A node only tries to find a partner if it has at least two processes

State information exchange policy I.

for Load-balancing algorithms

- Dynamic policies require frequent exchange of state information, but these extra messages arise two opposite impacts:
  - Increasing the number of messages gives more accurate scheduling decision
  - Increasing the number of messages raises the queuing time of messages
- State information policies can be the following:
  - Periodic broadcast
  - Broadcast when state changes
  - On-demand exchange
  - Exchange by polling

State information exchange policy II.

for Load-balancing algorithms

- Periodic broadcast
  - Each node broadcasts its state information after the elapse of every T units of time
  - Problem: heavy traffic, fruitless messages, poor scalability since information exchange is too large for networks having many nodes
- Broadcast when state changes
  - Avoids fruitless messages by broadcasting the state only when a process arrives or departures
  - Further improvement is to broadcast only when state switches to another region (double-threshold policy)

State information exchange policy III.

for Load-balancing algorithms

- On-demand exchange
  - In this method a node broadcast a State-Information-Request message when its state switches from normal to either underloaded or overloaded region.
  - On receiving this message other nodes reply with their own state information to the requesting node
  - Further improvement can be that only those nodes reply which are useful to the requesting node
- Exchange by polling
  - To avoid poor scalability (coming from broadcast messages) the partner node is searched by polling the other nodes on by one, until poll limit is reached

Migration limiting policy

for Load-balancing algorithms

- This policy determines the total number of times a process can migrate
  - Uncontrolled
    - A remote process arriving at a node is treated just as a process originating at a node, so a process may be migrated any number of times
  - Controlled
    - Avoids the instability of the uncontrolled policy
    - Use a *migration count* parameter to fix a limit on the number of time a process can migrate
    - Irrevocable migration policy: *migration count* is fixed to 1
    - For long execution processes *migration count* must be greater than 1 to adapt for dynamically changing states

- Drawbacks of Load-balancing approach
  - Load balancing technique with attempting equalizing the workload on all the nodes is not an appropriate object since big overhead is generated by gathering exact state information
  - Load balancing is not achievable since number of processes in a node is always fluctuating and temporal unbalance among the nodes exists every moment
- Basic ideas for Load-sharing approach
  - It is necessary and sufficient to prevent nodes from being idle while some other nodes have more than two processes
  - Load-sharing is much simpler than load-balancing since it only attempts to ensure that no node is idle when heavily node exists
  - Priority assignment policy and migration limiting policy are the same as that for the load-balancing algorithms

Load estimation policies

for Load-sharing algorithms

- Since load-sharing algorithms simply attempt to avoid idle nodes, it is sufficient to know whether a node is busy or idle
- Thus these algorithms normally employ the simplest load estimation policy of counting the total number of processes
- In modern systems where permanent existence of several processes on an idle node is possible, algorithms measure CPU utilization to estimate the load of a node

Process transfer policies

for Load-sharing algorithms

- Algorithms normally use all-or-nothing strategy
- This strategy uses the threshold value of all the nodes fixed to 1
- Nodes become receiver node when it has no process, and become sender node when it has more than 1 process
- To avoid processing power on nodes having zero process load-sharing algorithms use a threshold value of 2 instead of 1
- When CPU utilization is used as the load estimation policy, the double-threshold policy should be used as the process transfer policy

Location policies I.

for Load-sharing algorithms

- Location policy decides whether the sender node or the receiver node of the process takes the initiative to search for suitable node in the system, and this policy can be the following:
  - Sender-initiated location policy
    - Sender node decides where to send the process
    - Heavily loaded nodes search for lightly loaded nodes
  - Receiver-initiated location policy
    - Receiver node decides from where to get the process
    - Lightly loaded nodes search for heavily loaded nodes

Location policies II.

for Load-sharing algorithms

- Sender-initiated location policy
  - Node becomes overloaded, it either broadcasts or randomly probes the other nodes one by one to find a node that is able to receive remote processes
  - When broadcasting, suitable node is known as soon as reply arrives
- Receiver-initiated location policy
  - Nodes becomes underloaded, it either broadcast or randomly probes the other nodes one by one to indicate its willingness to receive remote processes

SUMMARY

- Resource manager of a distributed system schedules the processes to optimize combination of resources usage, response time, network congestion, scheduling overhead
- Three different approaches has been discussed
  - Task assignment approach deals with the assignment of task in order to minimize inter process communication costs and improve turnaround time for the complete process, by taking some constraints into account
  - In load-balancing approach the process assignment decisions attempt to equalize the avarage workload on all the nodes of the system
  - In load-sharing approach the process assignment decisions attempt to keep all the nodes busy if there are sufficient processes in the system for all the nodes

Process Management

Introduction

- In conventional (centralized) operating system, process management deals with mechanisms and policies for sharing the processor of the system among all processes.
- In a distributed operating system, the main goal of process management is to make the best possible use of the processing resources of the entire system by sharing them among all processes.
- **Three important concepts are used to achieve this goal:**

  -Processor allocation

  -Process migration

  -Threads

Cont…

- **Process allocation** deals with the process of deciding which process should be assigned to which processor.
- **Process migration** deals with the movement of a process from its current location to the processor to which it has been assigned.
- **Threads** deals with fine-grained parallelism for better utilization of the processing capability of the system.

Process Migration

- Relocation of a process from its current location to another node.

- Process may be migrated
  - either before it start executing on its source node
    - known as **non-preemptive process migration.**
  - or during the course of its execution
    - Known as preemptive process migration.

Cont… : Flow of execution of a migration process

Desirable features of a good process migration mechanism

- Transparency
- Minimal Interference
- Minimal Residual Dependencies
- Efficiency
- Robustness
- Communication between coprocesses of a job

Transparency

Level of transparency:

- Object Access Level
- System Call & Interprocess Communication level

Cont… : Object Access level Transparency

- Minimum requirement for a system to support non-preemptive process migration facility.
- Access to objects such as files and devices can be location independent.
- Allows free initiation of program at arbitrary node.
- Requires transparent object naming and locating.

Cont… : System Call & IPC level

- For migrated process, system calls should be location independent.
- For transparent redirection of messages during the transient state of a process.
- Transparency must be provided to support preemptive process migration facility.

Minimal Interference

- Migration of a process should cause minimal interference of progress of the process involved.

- Achieve by minimizing the freezing time of the process being migrated.
  - Freezing time is the time period for which the execution of the process is stopped for transferring its information to the destination node.

Minimal Residual Dependencies

- No residual dependency should be left on previous node.

Efficiency

- Minimum time required for migrating a process.
- Minimum cost of locating an object.
- Minimum cost of supporting remote execution once the process is migrated.

Robustness

- The failure of a node other than the one on which a process is currently running should not in any way affect the accessibility or execution of that process.

Communication between coprocessors of a job

- To reduce communication cost, it is necessary that coprocesses are able to directly communicate with each other irrespective of their locations.

Process Migration Mechanisms

Four major activities

- Freezing the process on its source node and restarting it on its destination node.
- Transferring the process's address space from its source node to its destination node.
- Forwarding massages meant for the migrant process.
- Handling communication between cooperating processes that have been separated as a result of process migration.

Naming

Naming

- The naming facility of a distributed operating system enables users and programs to assign character-string names to objects and subsequently use these names to refer to those objects.
- The locating facility, which is an integral part of the naming facility, maps an object's name to the object's location in a distributed system.
- The naming and locating facilities jointly form a naming system that provides the users with an abstraction of an object that hides the details of how and where an object is actually located in the network.

Naming

- It provides a further level of abstraction when dealing with object replicas. Given an object name, it returns a set of the locations of the object's replicas.
- The naming system plays a very important role in achieving the goal of location transparency, facilitating transparent migration and replication of Objects, object sharing.

Desirable Features of a Good Naming System

1. Location transparency: Location transparency means that the name of an object should not reveal any hint as to the physical location of the object.

2. Location independency: For performance, reliability, availability and security reasons, distributed systems provide the facility of object migration that allows the movement and relocation of objects dynamically among the various nodes of a system.

3. Scalability: Distributed systems vary in size ranging from one with a few nodes to one with many nodes. Distributed systems are normally open systems, and their size changes dynamically.

4. Uniform naming convention: In many existing systems, different ways of naming objects called naming conventions, are used for naming different types of objects. For example, filenames typically differ from user names and process names.

5. Multiple user-defined names for the same object: For a shared object, it is desirable that different users of the object can use their own convenient names for accessing it. A naming system must provide the flexibility to assign multiple user-defined names to the same object.

6. Group naming: A naming system should allow many different objects to be identified by the same name. Such a facility is useful to support broadcast facility or to group objects for conferencing or other applications.

7. Meaningful names: A name can be simply any character string identifying some object. For users, meaningful names are preferred to lower level identifiers such as memory pointers, disk block numbers, or network address.

8. Performance: The most important performance measurement of a naming system is the amount of time needed to map an object's name to its attributes, such as its location. In a distributed environment, this performance is dominated by the number of messages exchanged during the name-mapping operation.

9. Fault tolerance: A naming system should be capable of tolerating, to some extent, faults that occur due to the failure of a node or a communication link in a distributed system network. The naming system should continue functioning, perhaps in a degraded form, in the event of these failures.

10. Replication transparency: In a distributed system, replicas of an object are generally created to improve performance and reliability.

11. Locating the nearest replica: When a naming system supports the use of multiple copies of the same object, it is important that the object-locating mechanism of the naming system should always supply the location of the nearest replica of the desired object.

FUNDAMENTAL TERMINOLOGIES AND CONCEPTS

The naming system is one of the most important components of a distributed operating system because it enables other services and objects to be identified and accessed in a uniform manner.

1. **Name**: A name in a distributed system is a string of bits or characters that is used to refer to an entity.

   For example, SINHA, #173#4879#5965, node-1!node-2!node-3!sinha, lalble, 25A2368DM197, etc. are all valid names composed of symbols from the ASCII character set.

   A name is also called an identifier because it is used to denote or identify an object.

Human-Oriented and System-Oriented Names

- A human-oriented name is generally a character-string that is meaningful to its users. For example, /user/sinha/project-file-1 is a human-oriented name. Human-oriented names are defined by their users.
- Human-oriented names are also known as high-level names because they can be easily remembered by their users.
- Therefore, in addition to human-oriented names, which are useful for users, system-oriented names are needed to be used efficiently by the system.
- These names generally are bit patterns of fixed size that can be easily manipulated and stored by machines.
- They are automatically generated by the system.
- They should be generated in a distributed manner to avoid the problems of efficiency and reliability of a centralized unique identifier generator. They are basically meant for use by the system but may also be used by the users.
- They are also known as unique identifiers and low-level names.

Name space

A naming system employs one or more naming conventions for name assignment to objects.

For example, a naming system may use one naming convention for assigning human-oriented names to objects and another naming convention for assigning system-oriented names to objects.

The syntactic representation of a name as well as its semantic interpretation depends on the naming convention used for that name.

The set of names complying with a given naming convention is said to form a name space

Flat Name Space

- The simplest name space is a flat name space where names are character strings exhibiting no structure.
- Names defined in a flat name space are called primitive or flat names. Since flat names do not have any structure, it is difficult to assign unambiguous meaningful names to a large set of objects.
- Therefore, flat names are suitable for use either for small name spaces having names for only a few objects or for system-oriented names that need not be meaningful to the users.

Partitioned Name Space

- When there is a need to assign unambiguous meaningful names to a large set of objects, a naming convention that partitions the name space into disjoint classes is normally used.
- When partitioning is done syntactically, which is generally the case, the name structure reflects physical or organizational associations.
- Each partition of a partitioned name space is called a domain of the name space.
- Each domain of a partitioned name space may be viewed as a flat name space by itself, and the names defined in a domain must be unique within that domain. However, two different domains may have a common name defined within them.
- A name defined in a domain is called a simple name.

- In a partitioned name space, all objects cannot be uniquely identified by simple names, and hence compound names are used for the purpose of unique identification

- A commonly used type of partitioned name space is the hierarchical name space, in which the name space is partitioned into multiple levels and is structured as an inverted tree.

Name Server

- Name spaces are managed by name servers.
- A name server is a process that maintains information about named objects and provides facilities that enable users to access that information.
- It acts to bind an object's name to some of its properties, including the object's location.
- In practice, several name servers are normally used for managing the name space of object names in a distributed system.

- Each name server normally has information about only a small subset af the set of objects in the distributed system.

- The name servers that store the informatlon about an object are called the authoritative name servers of that object.

- To determine the authoritative name servers for every named object, the name service maintains authority attributes that contain a list of the authoritative name servers for each object.

Name Agent

The distribution of the name service and the locations of the name servers should be transparent to the clients of a name service. This transparency is achieved through name agents that act between name servers and their clients. A name agent maintains a knowledge of existing name servers. When a client requests for a name service, the name agent uses the proper communication protocol to transfer the user's request to a proper name server. On receiving a reply from the server, it forwards it to the client. Name agents may be of two types:

(a) private, which work for a single client,

and (b)shared, which work for several clients.

Name agents are known by various names. For instance, in the CSNET Name Server they are called "name server agent programs," in the DARPA Internet Domain Name Service (DNS) they are called "resolvers," in the DCE Directory Service they are called "clerks," and in the COSIE Name Server they are called "user interfaces."

Context

- Names are always associated with some context.
- A context can be thought of as the environment in which a name is valid.
- Because all names are interpreted relative to some context, a context/name pair is said to form a qualified name that can be used for uniquely identifying an object.

System-oriented names

System-oriented names normally have the following characteristic features:

1. They are large integers or bit strings. For example, in a typical naming system,system-oriented names may be members of some large set of integers, such as the integers up to 2^128-1.

2. They are also referred to as unique identifiers because in most naming systems they are guaranteed to be unique in both space and time. That is, these names do not change during their lifetime, and once used, they are never reused. Therefore, in the naming system discussed above, a 128-bit pattern refers either to nothing or, if it refers to anything, to the same thing at all times. This is the main reason why unique identifiers are so large.

3. Unlike human-oriented names that are variable in length, all system-oriented names of a naming system are of the same size irrespective of the type or location of the object identified by these names. This allows the naming of all objects uniformly.

4. Since all the system-oriented names of a naming system are of uniform size and also are normally shorter than human-oriented names, manipulations like hashing, sorting, and so on, can be easily performed on them. Hence, they are suitable for efficient handling by machines.

5. They are hard to guess and hence are sometimes also used for security..related purposes. For example, they may be used as tickets of permission to access objects. A computation that presents a valid name may be assumed to have been given it deliberately rather than to have guessed or invented it. This is an example of use of protected names or capabilities, one of the classical approaches to access control management.

6. System-oriented names are automatically generated by the system.

Object locating Mechanism

- Object locating is the process of mapping an object's system-oriented unique identifier (UID for short) to the replica locations of the object.
- It may be noted here that the objectlocating operation is different and independent of the object-accessing operation. In a distributed system, object locating is only the process of knowing the object's location, that is, the node on which it is located.
- On the other hand, object accessing involves the process of carrying out the desired operation (e.g., read, write) on the object. Therefore, the object-accessing operation starts only after the object-locating operation has been carried out successfully.

Broadcasting

- In this method, an object is located by broadcasting a request for the object from a client node. The request is processed by all nodes and then the nodes currently having the object reply back to the client node. Amoeba uses this method for locating a remote port.
- The method is simple and enjoys a high degree of reliability because it supplies all replica locations of the target object.
- However, it suffers from the drawbacks of poor efficiency and scalability because the amount of network traffic generated for each request is directly proportional to the number of nodes in the system and is prohibitive for large networks.
- Therefore, this method is suitable only when the number of nodes is small, communication speed is high, and object-locating requests are not so frequent.

Expanding Ring Broadcast

- Pure broadcasting is expensive for large networks. Moreover, direct broadcasting to all nodes may not be supported by wide-area networks.
- Therefore, a modified form of broadcasting, called expanding ring broadcast is normally employed in an internetwork that consists of local area networks (LANs) connected by gateways.
- In this method, increasingly distant LANs are systematically searched until the object is found or until every LAN has been searched unsuccessfully.
- The distance metric used is a hop.
- A hop corresponds to a gateway between processors.

- **HUMAN-ORIENTED NAMES ·**
  - Human-Oriented Hierarchical Naming Schemes ¨
  - Combining an Object's Local Name with Its Host Name ¨
  - Interlinking Isolated Name Spaces into a Single Name Space ¨
  - Sharing Remote Name Spaces on Explicit Request ·
- **Context Binding ¨**
  - Table-Based Strategy ¨
  - Procedure-Based Strategy ·
- **Distribution of Contexts and Name Resolution Mechanisms ¨**
  - Centralized Approach ¨
  - Fully Replicated Approach ¨
  - Distribution Based on Physical Structure of Name Space ¨
  - Structure-Free Distribution of Contexts ¨
  - Interacting with Name Servers During Name Resolution.

NAME CACHES:
1. Client Side Caching

Caching is an important technique for building responsive, scalable distributed systems. A cache can be maintained either by the client or the server or by both.

Caching at the client is an effective way of pushing the processing workload from the server out to client devices,if a client has the capacity.

Server Based Cache

- If results data likely to be reused by multiple clients or if the client devices do not have the capacity then caching at the server is more effective.

Multi level Caching

- Caches can be maintained at multiple levels.
- For example, caches can be maintained at all clients and all servers.
- Use of a cache at one level reduces the number of requests handled at the levels below.

LAN Technology: Ethernet and IEEE Token Ring

Ethernet

- Ethernet is the most widely used LAN technology for building distributed systems because it is relatively fast and economical.
- It was introduced by DEC (Digital Equipment Corporation), Intel, and Xerox in 1980, and adopted by the IEEE as a standard LAN technology, known as the IEEE 802.3standard.
- It enables devices to communicate with each other via a protocol, which is a set of rules or common network language.
- Ethernet describes how network devices format and transmit data so other devices on the same LAN or campus network can recognize, receive and process the information.
- An Ethernet cable is the physical, encased wiring over which the data travels.

Data Flow
Terminator

Ethernet Frame Structure

6 bytes	6 bytes	2	46 - 1500 bytes	4 bytes
Destination address	Source address	Type	User data	FCS

Destination Address – This is 6-Byte field which contains the MAC address of machine for which data is destined.

Source Address – This is a 6-Byte field which contains the MAC address of source machine. As Source Address is always an individual address (Unicast), the least significant bit of first byte is always 0.

Type– Type is a 2-Byte field, which indicates the length of entire Ethernet frame. This 16-bit field can hold the length value between 0 to 65534, but length cannot be larger than 1500 because of some own limitations of Ethernet.

Data – This is the place where actual data is inserted, also known as Payload. Both IP header and data will be inserted here if Internet Protocol is used over Ethernet. The maximum data present may be as long as 1500 Bytes. In case data length is less than minimum length i.e. 46 bytes, then padding 0's is added to meet the minimum possible length.

Frame check sequence (FCS) is an error-detecting code added to a frame in a communication protocol. It is 4 Byte field. This field contains a 32-bits hash code of data, which is generated over the Destination Address, Source Address, Length, and Data field. If the checksum computed by destination is not the same as sent checksum value, data received is corrupted.

Token Ring

Token Ring

Token ring (IEEE 802.5) is a communication protocol in a local area network (LAN) where all stations are connected in a ring topology and pass one or more tokens for channel acquisition. A token is a special frame of 3 bytes that circulates along the ring of stations. A station can send data frames only if it holds a token. The tokens are released on successful receipt of the data frame

Ring Topology

ComputerHope.com

Frame Format

Start Deli miter	Access Control	Frame Control	Dest Address	Src Address	Data	FCS	End Deli miter	Frame Status
1 bytes	1	1	6	6	>=0	4	1	1

Data/Command

Start Deli miter	Access Control	End Deli miter
1	1	1

Token

- The first byte of the token contains a fixed bit pattern that enables sites to recognize the start of a packet and synchronize to the data transmission rate.
- The second byte contains priority and token status (free/busy) information.
- The third byte contains packet control information. The priority field can be used to implement a variety of methods for sharing the channel capacity among the sites on the network.
- In addition to the token, a packet in the IEEE Token Ring has fields for source and destination addresses (each 6 bytes long), message data (length 5000 bytes), checksum (4 bytes), end delimiter (1 byte), and packet status (1 byte).
- The source address and destination address fields respectively contain the addresses of the sending and receiving sites. The message data field contains the data to be transmitted. This field is of variable length and allows packets to be of almost any length.
- The upper bound of 5000 bytes for the length of this field is a default value for a parameter that can be configured on a per-installation basis.
- The checksum field contains a checksum generated by the sender and used by the receiver to check the validity of the received packet.
- The end-delimiter field contains a fixed bit pattern that enables sites to recognize the end of a packet.
- Finally, the packet status field specifies whether the packet was successfully received by the receiving site. This field helps the sending site in knowing whether its packet was received by the receiver.

WAN TECHNOLOGIES

- A WAN of computers is constructed by interconnecting computers that are separated by large distances; they may be located in different cities or even in different countries. In general, no fixed regular network topology is used for interconnecting the computers of a WAN. Moreover, different communication media may be used for different links of a WAN.
- For example, in a WAN, computers located in the same country may be interconnected by coaxial cables (telephone lines), but communications satellites may be used to interconnect two computers that are located in different countries.
- The computers of a WAN are not connected directly to the communication channels but are connected to hardware devices caned packet-switching exchanges (PSEs), which are special-purpose computers dedicated to the task of data communication.
- Therefore, the communication channels of the network interconnect the PSEs, which actually perform the task of data communication across the network (Fig. 2.6). A computer of a WAN only interacts with the PSE of the WAN to which it is connected for sending and receiving messages from other computers on the WAN.

Switching Techniques

Circuit Switching

When a dedicated path is established for data transmission between sender and receiver, it is called circuit switching. When any network node wants to send data, be it audio, video, text or any other type of information, a call request signal is sent to the receiver and acknowledged back to ensure availability of dedicated path. This dedicated path is then used to send data. ARPANET used circuit switching for communication over the network.

Advantages of Circuit Switching

Circuit switching provides these advantages over other switching techniques −

- Once path is set up, the only delay is in data transmission speed
- No problem of congestion or garbled message

Disadvantages of Circuit Switching

Circuit switching has its disadvantages too −

- Long set up time is required
- A request token must travel to the receiver and then acknowledged before any transmission can happen
- Line may be held up for a long time

Packet Switching

As we discussed, the major problem with circuit switching is that it needs a dedicated line for transmission. In packet switching, data is broken down into small packets with each packet having source and destination addresses, travelling from one router to the next router.

In this method, instead of establishing a dedicated communication path between a sender and receiver pair (of computers), the channels are shared for transmitting packets of different sender-receiver pairs. That is, a channel is occupied by a sender-receiver pair only while transmitting a single packet of the message of that pair; the channel may then be used for transmitting either another packet of the same sender-receiver pair or a packet of some other sender-receiver pair.

Synchronization

Introduction

- Distributed System is a collection of computers connected via the high speed communication network.
- In the distributed system, the hardware and software components communicate and coordinate their actions by message passing.
- Each node in distributed systems can share their resources with other nodes. So, there is need of proper allocation of resources to preserve the state of resources and help coordinate between the several processes.
- To resolve such conflicts, synchronization is used. Synchronization in distributed systems is achieved via clocks.

Issues in synchronization

- Clock synchronization
- Event Ordering
- Mutual Exclusion
- Deadlock
- Election algorithms

Clock Synchronization

- Every computer needs a computer clock to keep track of computer time and calculating the time spent by a process in execution on processor.
- An application may have many processes that concurrently running on different nodes of the system.
- Every node need clock that should be synchronized.

Example, Airline Reservation System

- If the clock of these nodes are not synchronized ,it is difficult to achieve the correct result.
- That problem is called the clock synchronization problem.
- Need some suitable ways to properly synchronizing the clocks.

How computer clocks are implemented

- A computer clock consists of three components.
- Quartz crystal (Oscillate 60 times in a second)
- Counter register
- Constant register (frequency of quartz crystal)

To make the computer clock function as an ordinary clock.

- The value in the constant register is chosen 60. (60 clock ticks in a second)
- The computer clock is synchronized with the real time.
- Two values stored in system , Date and Time, Number of Ticks

Drifting of Clocks

- Due to differences in crystals ,two clock rates different from each other.
- Difference is small but observable , no matter how accurately initialized.
- Therefore, Passage of time, a computer clock drifts from the real-time clock.
- After 11.6 days of 1 second.
- Hence, Computer clock must be periodically synchronized with the real-time clock called non-faulty.
- Even non-faulty clocks do not always maintain perfect time.
- A clock is called non-faulty if there is bound on the amount of drift.

Drifting of Clocks cont..

Slow, perfect and fast clocks

Types of Clock Synchronization

- External clocks
  - Required for real time applications
  - Allows the system to exchange information about the timing with other systems and users.
  - A external time source used UTC(Coordinated universal time).
- Internal clocks
  - Provide a constant view of time across all the nodes of distributed systems.

- Note
  - Externally synchronized clocks are also internally synchronized.

Clock Synchronization Issues

- Two clocks cannot be perfectly synchronized.
- In practically for the synchronization, the difference of time value of two clocks should less than Δ.
- The difference in time values of two clocks is called clock skew.
- It is necessary of synchronization that each node should read the clock values of other nodes. The mechanism of reading the clock values of other nodes varies in different algorithms.
- Un-predictable communication delays of clock message between two nodes.
- Time must never run backward because it causes repetition of certain operations, when fast clock is set to slow down.
- Time should not be slowed down or moved backward at once, it should be gradually updated with each interrupt.

Clock Synchronization algorithms

- Centralized Algorithms
  - Passive time server
  - Active time server
- Distributed Algorithms
  - Global Averaging Algorithms
  - Localized Averaging Algorithms

Centralized Algorithms

- **Centralized** is the one in which a time server is used as a reference.
- The single time server propagates its time to the nodes and all the nodes adjust the time accordingly.
- It is dependent on single time server so if that node fails, the whole system will lose synchronization.
- Examples of centralized are-
- Passive Time Server
- Active Time ServeR

Passive time server algorithms –

- Each node periodically sends a message called ‘time=?’ to the time server.
- When the time server receives the message, it responds with ‘time=T’ message.
- Assume that client node has a clock time of T0 when it sends ‘time=?’ and time T1 when it receives the ‘time=T’ message.
- T0 and T1 are measured using the same clock, thus the time needed in propagation of message from time server to client node would be (T1- T0)/2
- When client node receives the reply from the time server, client node is readjusted to Tserver+(T1- T0)/2.

Client sets time to: $T_{new} = T_{server} + \frac{T_1 - T_0}{2}$

Passive Time server Algorithm cont

This method assumes the availability of some additional information.

- Two things are known:
- Approximate time taken by the time server to handle the interrupt .
- Process time of request message of client (“ Time = ? “) .
- Sum up of both times is the I .
- Best estimate in this case would be (T1-To-I) / 2.
- So , Total time at the client node is readjusted to T+ (T1-To-I) / 2.
- This methods assumes the no additional information, several measurements of T1-To are made, and threshold is set.
- If the (T1-To) exceeds threshold then values are discarded.
- The average of remaining measurements is calculated and half of this value is added to T.
- Alternative way is, Measurements for which (T1-To) is minimum is considered to be most accurate one. And half of this added to T.

Active time server algorithms

It is also called Berkeley Algorithm.

- An algorithm for internal synchronization of a group of computers.
- A master polls to collect clock values from the others (slaves).
- The master uses round trip times to estimate the slaves' clock values.
- It obtains average from participating computers.
- It sends the required adjustment to the slaves.
- If master fails, can elect a new master to take over.
- It synchronizes all clocks to average.

Time daemon
3:00
3:00
3:00
3:00
3:25
2:50
(a)
3:00
0
+25
-10
3:25
2:50
(b)
3:05
+5
-20
+15
Network
3:05
3:05
(c)

- Here the time server (actually, a time daemon) is active, polling every machine periodically to ask what time it is there.
- Based on the answers, it computes an average time and tells all the other machines to advance their clocks to the new time or slow their clocks down until some specified reduction has been achieved.
- In Figure (a), at 3:00, the time daemon tells the other machines its time and asks for theirs.
- In Figure (b), they respond with how far ahead or behind the time daemon they are.
- Armed with these numbers, the time daemon computes the average and tells each machine how to adjust its clock Figure (c).

Disadvantages of Centralized Algorithms

- Single point of failure , if time server node failed , clock synchronization operation cannot be performed.
- From scalability point of view , adding more node will put more burden on the single central server.
- Solutions, of these problem are coming next...

Distributed Algorithms

- Externally synchronized clocks are also internally synchronized
- Each node's clock is independently synchronized with real time.
- All clocks of system remain mutually synchronized.
- Each node is equipped with real time receiver so that each node can be independently synchronized.
- Theoretically internal synchronization of clock is not required.
- in practice, due to inherent inaccuracy of real time clocks, different real time clocks produce different time.
- Internal synchronization is performed for better accuracy.
- Types of internal Synchronization:
    - Global averaging
    - Localized Averaging

Global Averaging

- Each node broadcasts its local clock time in the form of a special “resync” message.
- After broadcasting the clock value , the clock process of a node waits for time T (T is calculated by algorithm).
- During this period, clock process collects the resync messages broadcast by other nodes.
- Clock process estimates the skew of its clock with respect to other nodes (when message received) . (Difference in time value of two clocks is SKEW)
- It then computes a fault-tolerant average of the estimated skews and use it to correct the local clock.
- Two commonly algorithms use to calculate the fault-tolerant average.

Global Averaging cont..

- 1st simplest algorithm is to take the average of estimated skews and use it as the correction for the local clock. And only below a threshold are taken into account, and other are set to zero before computing the average.

- 2nd Algorithm, each node limits the impact of faulty clocks by discarding the highest and lowest skews. And then calculating the average of the remaining skews. And use it to the correction for the local clock.

Localized Averaging Algorithm

- Global algorithm was only Suitable for small networks due to broadcast facility.
- Where node has direct communication to every other node (fully connected topology) .
- This algorithm attempt to overcome the drawbacks of the global averaging algorithm.
- Nodes of a distributed system are logically arranged in some kind of pattern such as Ring.
- Periodically each node exchange its clock time with is neighbors in the ring.
- And then sets its clock time by taking Average.
- Average is calculated by , its own clock time and clock times of its neighbors.

Event Ordering

- Keeping the clock synchronized within 5,10 milisec is expensive task.
- Lamport observed, Its not necessary to keep the clocks synchronized in distributed system.
- It is sufficient to keep the all event be totally ordered.
- For Partial ordering of the events, Lamport defined a new relation called Happened-Before relation.

Happened Before Relation

- Happened Before Relation(Denoted by $\rightarrow$) on a set of events Satisfies Following Conditions.
- If a and b are two events in the same process and a occurs before b, then $a \rightarrow b$.
- If a is event of sending a message by one process and b is event of receipt of same message by another process, then $a \rightarrow b$. it follows law of causality because receiver can not receive message until sender sends it and propagation time from sender to receiver is always positive.
- If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$. happened before relation follows transitive relation
- $a \rightarrow a$ is not possible. Happened before relation is irreflexive partial ordering.

Diagram

- Causal Events:
- e1o → e11 e20 →e24 e11 → e23 e21 →e13
- e30 → e24
- e11 → e32
- Concurrent Events:
- e12 and e20 e21 and e30 e10 and e30 e11 and e31 e12 and e32 e13 and e22

Happened Before relation cont..

- Event types;
- Causal events
- Concurrent Events:

- Causal Events:
- Two events a →b is true if path exists between a and b by moving forward in time along process and message lines in the direction of arrows. These events are related by happened before relation.

- Concurrent events: two events are concurrent if no path exists between them and are not related by happened before relation.

Logical Clocks Concept

- For determining the event a occurs before b, need common clock, or set of perfectly synchronized clocks.
- Neither of these available, Lamport provided a solution of this problem. Local clock concept.
- The logical clock concept is to associate a timestamp with each system event.
- Each process Pi, has a clock Ci.
- And assigned a number Ci(a) to any event in that process.
- In fact, Logical clocks implemented by counters with no actual timing mechanism.
- Happened-before relation should be ordered, using these clocks.
- So that Logical clocks must satisfy the following condition:
- For any two events a and b, if $a \rightarrow b$, then $C(a) < C(b)$

Implementation of Logical Clocks

- Happened Before algorithm with logical clocks must follows the following conditions:
- C1: If a and b are two events within the same process Pi and a occurs before b , then Ci(a) < Cj(b).
- C2: If a is the sending message by process Pi, and b is the receipt of that message by process Pj, then Ci(a) < Cj(b)
- C3: A clock Ci associated with a process Pi, must always go forward , never backward, That is, corrections to time of a logical clock must always be made by adding a positive value to the clock , never by subtracting the value.

Implementation of Logical Clocks cont..

To meet conditions C1, C2, and C3 , Lamport used the following implementation rules:

IR1(implementation Rule): Each process Pi increments Ci between two successive events.

IR2(implementation Rule): If Pi sending a message m by an event a, and the message m contains a timestamp Tm= Ci(a) , and other process Pj receive this message Ci(a) . And if Ci(a) is greater than and equal to its present value then Ci(a) value updated on it with +1 increment, otherwise Cj present value continue.

Implementation of Logical Clocks by Using Counters

- Two processes P1 and P2 each have counters C1 and C2.
- C1 values increase C1=0,1,2,3,4,5...
- C2 values increase C2=0,1,2,3,4,5...
- Counters act like logical clocks.
- Message having the incremented value of counter, that is sent by P1.
- On the other side P2 receive this message, and instead of simple increment of its counter, a check is made .
- Check is , If the incremented counter value of P2 is less than or equal to the timestamp in the received message from P1 .
- if C2 <= C1 ,e.g 3 <= 4 , C2 value will be updated with +1.

Implementation of Logical Clocks by Using Counters cont…

Fig. 6.4 Example illustrating the implementation of logical clocks by using counters.

Implementation of Logical Clocks by Using Physical Clocks

- Each process has a physical clock associated with it.
- Each clock run at a constant rate.
- However, The rates at which different clocks run are different.
- When clock of process P1 has ticked 10 times, The clock of process P2 has ticked only 8 times.
- , If the incremented counter value of P2 is less than or equal to the timestamp in the received message.
- if C2 <= C1 ,e.g 56<= 60 , C2 value will be updated with +1.

Total Ordering of Events

- Happened-before has only partial ordering.
- Two events a or b not related by Happened-before , if same timestamps are associated with them.
- P1 and P2 clocks are same for example 100 .
- Nothing can be said about the order.
- Additional information required.
- Lamport purposed Total ordering of processes.
- He used process identity numbers.
- Timestamp associated with them will be 100.001 and 100.002.
- Here , process identity numbers of processes P1 and P2 are 001 and 002.

Mutual Exclusion

- A file must not be simultaneously updated by multiple processes.
- Such as printer and tape drives must be restricted to a single process at a time.
- Therefore , Exclusive access to such a shared resource by a process must b ensured.
- This exclusiveness of access is called mutual exclusion between processes.

Critical Section

- The sections of a program that need exclusive access to shared resources are referred to as critical sections.
- For mutual exclusion , Ways are introduced, to prevent processes from executing concurrently with their associated critical section.

- Following requirement must satisfy for mutual exclusion:
- 1. Mutual exclusion: At anytime only one process should access the resource. A process can get another resource first releasing its own.
- 2. No starvation: if every process that is granted the resource eventually release it, every request must be eventually granted.

Three approaches to implement mutual exclusion

- Centralized Approach
- Distributed Approach
- Token Ring

Centralized Approach

- One process in the system is selected as coordinator.
- That coordinates the entry to the critical sections.
- Each process that wants to enter a critical section must first seek permission from the coordinator.
- If two processes simultaneously ask for permission to enter the critical section , Coordinator grants permission according with some scheduling algorithms.
- When process exits, it notify the coordinator so that the coordinator can grant permission to another process.
- This algorithm (in centralized) ensures mutual exclusion, because at a time , the coordinator allows only one process to enter in critical section.
- This algorithm (in centralized) ensures no starvation, because use of FCFS (First come First served) .

P_2
③Request
⑥Reply
⑦Release
⑤ Release
②Reply
①Request
P_1
P_c
⑨Release
⑧Reply
④Request
P_3
Initial status
P_2
Status after ③
P_3
P_2
Status after ④
P_3
Status after ⑤
Status after ⑦
Status of
request queue

ed

Centralized Approach cont..

- Benefits:
  - easy to implement
  - Requires only three messages per critical section (request, reply, release).
- Drawbacks:
  - Single coordinator
  - Single point of failure
  - Performance down due to bottleneck in large systems.

Distributed Approach

- Decision making for mutual exclusion is distributed across entire system.
- All process that want to enter critical section co-operate with each other before taking decision.
- Ricart and Agarwala's algorithm is used, based upon Lamport's event ordering scheme to assign unique time stamp to each event.
- When a process wants to enter critical section it sends request message to all process. Message contains
- Process identifier of process
- Name of critical section that process wants to enter.
- Unique time stamp generated by process for request message.
- On receiving request message a process immediately sends reply message or defers reply based upon following rules.

Distributed Approach Cont..

- If receiver process is in critical section, it places request message in queue and defers sending a reply.
- If receiver process is not in critical section and waiting to enter, it compares it's own time stamp with received time stamp. If received time stamp is lower than it' own, it means that process made request earlier, than it sends reply to that process otherwise places it in queue and defers sending reply.
- If receiver process is neither in critical section nor waiting, it simply sends reply message.
- A process enters critical section if it receives message from all process, executes in critical section and than sends reply message to all processes in queue.

Distributed Approach Cont..

5.7 Example illustrating the distributed algorithm for mutual exclusion: (*a*) status when processes P_1 and P_2 send request messages to other processes while process P_4 is already in the critical section; (*b*) status while process P_4 is still in critical section; (*c*) status after process P_4 exits critical section; (*d*) status after process P_2 exits critical section.

Drawbacks and solutions

- System having n processes is liable to n points of failure and blocks in case of single process failure.
- Solution: receiver sends permission denied message when it is in critical section, else it sends OK. if receiver did not receive reply it keeps on trying after time out,until it gets reply, else it considers that process has crashed.
- Each process must know identity of process participating in group. When a process joins it must receive names of all other processes and name of new process must be distributes to all members of group. suitable only for small group due to large no of updates of creation and deletion.
- To much wait if there are large no of nodes. If there are n nodes and single message travels through network then each process will wait for 2(n-1) messages.
- Suitable for small group.
- Best solution: Majority consensus rather than consensus of all processes is proposed in literature.

Token passing Approach

- Mutual Exclusion is achieved by circulating single token among processes.
- Token is special type of message that entitles its holder to enter critical section.
- Process are arranged in logical ring structure.
- Token can be moved clockwise or anti-clockwise.
- When a process gets token it can enters critical section and after finishing work, exits from critical section and passes token to neighbor process.
- Process can enter critical section one time in each turn with token.
- If process is not interested to enter in critical section, it simply passes token to neighbor, if no one is interested token keeps on circulating in ring.
- Waiting time varies from 0 to n-1 messages depending upon receipt of token and interested to enter in critical section or interested just after the token passed to neighbor.

Drawbacks and Solutions:

- 1) **Process failure**: Single process failure causes logical ring to break.
- Solution: exchange of Ack(acknowledgement) after message Receipt with neighbor. If no Ack comes, means problem with process.
- Current ring information with each process, using this each process skips its faulty neighbor process and when that process recovers, it informs it's previous neighbor or process.
- 2**) Lost Token**: Token is lost, new token message must be generated.
- Solution: Designate one process as monitor process.
- Monitor periodically circulates "who has token" message. All the process passes message to neighbor, except the process which has token. It send its process identifier in special field with token to neighbor.
- When message returns to monitor after one circulation, if there is no entry in that field means token has been lost, generates new token and circulates in ring
- 3) **Monitor process and "who has token" message is lost.**
- Solution: more than one monitor process. And if one fails election has to be done for selection of next monitor who will generate token.

Diagram

Deadlock

- A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.
- Suppose process 1 holds resource and process 2 holds resource 2.
- Now process 2 requests resource 1 and process 1 requests resource 2 but none of process will get this resource because these resources are already hold by other process so both can be blocked, with neither one be able to proceed, this situation is called deadlock.

Resource 1
Assigned
to
Waiting
For
Process 1
Process 2
Waiting
For
Resource 2
Assigned
to

Deadlock example.

Deadlock example (Bridge Crossing)

- Traffic only in one direction.
- Each section of bridge can be viewed as a resource.

- If a deadlock occurs. It can be resolved if one car back up.
- Called preempt resources and rollback.
- Several cars may have to be backed up if a deadlock occurs.
- Starvation is possible.

Deadlock cont..

- Process use a resources in following sequence of events.
- Request: Process first makes a request for resources. If not available must wait. When available , request satisfied. But number of requests may not exceeded the total number of available units of resources.
- Allocate: System allocate the resources . It maintain a table. Where records of resources , and which resource is allocated to which process. Also having queue for waiting process.
- Release: After the process finished using the allocated resource , it release the resource to the system. System update the allocation.

Deadlock cont..

- If some of the processes entered in the waiting state, will never change state. Because requested resources are held by other waiting processes . This situation is called deadlock.

- Resource can be , physical objects (Disk drive , tape, CPU, RAM) and logical objects (files, tables, database) .

- **Preemptable**, meaning that the **resource** can be taken away from its current owner (and given back later). An example is memory.
- **Non-preemptable**, meaning that the **resource** cannot be taken away. An example is a printer.

Necessary Conditions for Deadlock

- Deadlock occurs if the following conditions hold simultaneously:

- Mutual exclusion condition
- Hold and wait condition
- No-Preemption condition
- Circular-wait condition

Necessary Conditions for Deadlock cont...

- Deadlock occurs if the following conditions hold simultaneously:

- **Mutual exclusion:** only one process can use a resource at a time.
- **Hold and wait:** Processes are allowed to request for new resources without releasing the resources that they are currently holding.
- **No-preemption:** If a process has a resource , cannot be preempted.
- **Circular-wait:** Two or more processes from a circular chain.

Circular wait

Handling Deadlocks

- Prevention
- Avoidance
- Detection and recovery

Deadlock Prevention

- Designing the system in a way that deadlock become impossible.
- Prevention is different from avoidance because no runtime testing (dynamically) of potential allocations need be performed.

- Prevent the one of the four necessary conditions:
  - Mutual exclusion condition
  - Hold and wait condition
  - No-Preemption condition
  - Circular-wait condition

Deadlock Prevention cont..

- **Mutual Exclusion**
- Unfortunately it is not possible to prevent deadlocks by denying mutual-exclusion.
- Cannot be prevented for all resources. Some resources are intrinsically non-sharable for example printer, and some are sharable like CPU.

Deadlock Prevention cont..

- **Hold and wait (Collective Requests)**
- This method denies the hold-and-wait condition.
- We must guarantee that whenever a process requests a resource, it does not hold any other resources.

- Following policies maybe used to ensure this:
- 1. A process must request all of its resources before it begins execution. If all the needed resources are available , they are allocated to it.
  If one or more resources are not available , none will be allocated and process would just wait.

Deadlock Prevention cont..

- **No-Preemption condition (resources will be preempted)**
- A preemptable resource is one whose state can be easily saved and restored later.
- A resource temporarily taken away from the process without harm to the computation performed by this process.
- CPU, Main Memory are preemptable resources.
- If the resources are preemptable, deadlocks can be prevented.
- Two policies:
- 1. when a process requests for a resource, and resource is not available. Its already held resources taken away from it. And the process is blocked. The process is unblocked when the resources (requested by it, and preempted from it) available and allocated to it.

Deadlock Prevention cont..

- **No-Preemption condition (resources will be preempted)**
- 2.When a process requests a resource that is not currently available, the system checks, if the requested resource is currently held by another process that is already blocked and waiting for other resources. If so , the requested resource is taken away from the waiting process and give to the requesting process, otherwise condition-1 applied.

Deadlock Prevention cont..

- **Circular wait condition (ordered requests)**
- Each resource (type) is assigned a global number to impose total ordering.
- Impose a total ordering of all resource types , and require that each process can only requests resources in an increasing order of enumeration.
- Process should not request a resource with a number lower than is already held resource.
- If process request = j and process already held resource = I=i
  then, j > I
- Single request will be enough for the same resource type.
- It has proven that with this ,graph can never have circular-wait.
- If tape drive =1 , disk drive = 5, printer = 12,
  after getting tape drive process can request for the disk drive, after disk drive process can request for the printer.
  But it a process have the number =5, it cannot request for the number=1 for tape drive.

Deadlock Avoidance

- The system dynamically considers every request and decides whether it is safe to grant it at this point.
- Deadlock avoidance use some advance knowledge of the resource usage of processes.

- Following steps are for deadlock avoidance:
- When a process request for resource, even if the resource is available for allocation. It is not immediately allocated to that process.
- Advance knowledge used to perform analysis to decide whether process's request is safe or unsafe.
- When request is safe, resource allocated, otherwise deferred.

Deadlock Avoidance cont..

- Safe State:
- A system is considered in safe state , if there exists some ordering of processes in which run all requests lead to completion .
- Safe Ordering (sequence):
- Any ordering of processes, that can guarantee the completion of all the process is called safe sequence.

Election Algorithms

- Election algorithms are meant for electing a coordinator process from among the currently running processes in such a manner that at any instant of time there is a single coordinator for all processes in the system.
- Election algorithms are based on following assumptions
- Each process in system has unique priority no.
- Whenever an election is held, process having highest priority no among the currently running processes is elected as coordinator.
- On recovery, a failed process can take appropriate actions to rejoin the set of active process.
- **Types:**
- Bully Algorithm
- Ring Algorithm

Bully Algorithm

- When a process(say Pi) sends a request message to to coordinator and does not receive a reply within a fixed time out period, it assumes that coordinator has failed.
- It initiates election by sending an election message to every process with higher priority no than itself.
- If Pi does not receive response in time out means it has highest priority.
- It will become coordinator and will inform all processes through message.
- If Pi receives reply means other processes with higher priority are available and does not take any decision and waits for message of coordinator.
- When Process Pj receives an election message. It informs sender that is alive and now Pj will do election and will become coordinator if it has highest priority no.
- Else Pj will initiate message and this will continue.

Bully algorithm

- Bully algorithm specifies the process with the highest identifier will be the coordinator of the group. It works as follows:
- When a process p detects that the coordinator is not responding to requests, it initiates an election:
- p sends an election message to all processes with higher numbers. o If nobody responds, then p wins and takes over.
- If one of the processes answers, then p's job is done. If a process receives an election message from a lower-numbered process at any time, it:
- sends an OK message back.
- holds an election (unless it is already holding one).
- A process announces its victory by sending all processes a message telling them that it is the new coordinator. If a process that has been down recovers, it holds an election.

The Bully Algorithm (4)

The bully election algorithm

a) Process 4 holds an election

b) Process 5 and 6 respond, telling 4 to stop

c) Now 5 and 6 each hold an election

Ring Algorithm

- The ring algorithm assumes that the processes are arranged in a logical ring and each process is knowing the order of the ring of processes.
- If any process detects failure, it constructs an election message with its process ID (e.g., its network address and local process ID) and sends it to its neighbor.
- If the neighbor is down,the process skips over it and sends the message to the next process in the ring.
- This process is repeated until a running process is located. .
- At each step, the process adds its own process ID to the list in the message and sends the message to its living neighbor.
- Eventually, the election message comes back to the process that started it.
- The process then picks either the highest or lowest process ID in the list and sends out a message to the group informing them of the new coordinator.

(a)
P5
DEAD
P4
P0
P3
P1
P2
Election: {P1}
(b)
P5
DEAD
P4
P0
P3
P1
P2
Election: {P1, P2}
(c)
Election: {P1, P2, P3, P4}
Fails: P5 is dead
P5
DEAD
P4
P0
P3
P1
P2
(d)
Election: {P1, P2, P3, P4}
Skip to P0
P5
DEAD
P4
P0
P3
P1
P2
(e)
P5
DEAD
P4
P0
P3
P1
P2
This is me!
Winner!
Election: {P1, P2, P3, P4, P0}
(f)
P5
DEAD
P4
P0
P3
P1
P2
P4
P4
P4
P4
P4

Discussion of Two Election Algorithms

- Bully Algorithm:
- When lowest priority no process detects coordinator failure in a system of n processes n-2 elections are performed, leaving failed coordinator and sender.
- In worst case complexity is O(n^2).
- best case n-2 messages are required. Complexity O(n)

Ring Algorithm:

election requires 2(n-1) messages.

n-1 messages for one round rotation of election message.

n-1 messages for one round rotation of coordinator.

Distributed Computing Environment

Introduction

- DCE is an architecture defined by the Open Software Foundation (OSF) to provide an Open Systems platform to address the challenges of distributed computing.
- The DCE supplies a framework and toolkit for developing client/server applications.
- A vendor-independent distributed computing environment, DCE was defined by the Open Software Foundation (OSF), a consortium of computer manufacturers, including IBM,DEC, and HP.

- It is not an operating system, nor is it an application.
- Rather, it is an integrated set of services and tools that can be installed as a coherent environment on top of existing operating systems and serve as a platform for building and running distributed applications.
- A primary goal of DCE is vendor independence. It runs on many different kinds of computers, operating systems, and networks produced by different vendors.

Position of DCE software in a DCE-based distributed system.

DCE is middleware software sandwiched between the application and the operating system layer.

DCE is middleware software sandwiched between the application and the operating system layer.

Goals of DCE:

- It should run on different computers, operating systems and networks in a distributed system.
- It should provide a coherent seamless platform for distributed applications.
- It should provide a mechanism for clock synchronization on different machines.
- It should provide tools which make it easier to write distributed applications where multiple users at multiple locations can work together.
- Provide extensive tools for authentication authorization.
- Authentication confirms that users are who they say they are. Authorization gives those users permission to access a resource.

DCE Architecture:

- DCE cell:

The DCE system is highly scalable.To accommodate such large systems DCE uses the concept of cells. A cell is the basic unit of operation in the DCE which breaks down a large system into smaller, manageable units.

- DCE uses various technologies that form the components such as:

DCE Components:

- Thread package
- Remote Procedure Call (RPC) mechanism known as DCE/RPC.
- Naming (directory) Service.
- Time Service.
- Authentication Service.
- Authorization Service.
- Distributed File System (DFS) known as DCE/DFS.

1.Thread package

- It is a package that provides a programming model for building concurrent applications. It is a collection of user-level library procedures that supports the creation, management, and synchronization of multiple threads.
- It was designed to minimize the impact on the existing software that could convert single threaded into multithreaded one.
- Threads facility is not provided by many operating systems and DCE components require threads, hence threads package is included in DCE.

2.Remote procedure call

- A procedure call is a method of implementing the Client/Server Communication.
- The procedure call is translated into network communications by the underlying RPC mechanism.
- In DCE RPC, one or more DCE RPC interfaces are defined using the DCE interface definition language (IDL). Each interface comprises a set of associated RPC calls each with their input and output parameters.
- RPC hides communication detail and removes system and hardware dependencies. It can automatically handle data type conversions between the client and the server without considering whether they run on the same or different architecture, or have same or different byte ordering.

3. Name service

The Name service of DCE include

a. Cell Directory Service (CDS):The CDS manages a database of information about the resources in a group of machines called a DCE cell.

b. Global Directory Service (GDS):The Global Directory Service implements an international, standard directory service and provides a global namespace that connects the local DCE cells into one worldwide hierarchy.

c. Global Directory Agent (GDA): The GDA acts as a go-between for cell and global directory services.

4. Distributed file service

- Distributed File service is a worldwide distributed file system which allows users to access and share files stored on a file server anywhere on the network without knowing the physical location of the file.
- It provides characteristics such as high performance, high availability and location transparency and is able to perform tasks like replicating data; log file system data, quick recovery during crash etc.

5. Time service

- The DCE Time Service (DTS) provides synchronized time on the computers.
- It enables distributed applications on different computers to determine event sequencing, duration, and scheduling participating in a Distributed Computing Environment.
- DTS also provides services which return a time range to an application, which compare time ranges from different machines.

6. Security service

- There are three aspects to DCE security which provide resource protection within a system against illegitimate access. They are:
- **Authentication:** This identifies that a DCE user or service is allowed to use a particular service.
- **Secure communications**: Communication over the network can be checked for tampering or encrypted for privacy.
- **Authorization:** The permission to access the service is given after authorization.
- These are services are provided using Registry Service, Privilege Service, Access Control List (ACL) Facility, and Login Facility etc.

DCE Cells

The DCE system is highly scalable.To accommodate such large systems DCE uses the concept of cells. A cell is the basic unit of operation in the DCE which breaks down a large system into smaller, manageable units.

In a DCE system, a cell is a group of users, machines, or other resources that typically have a common purpose and share common DCE services. The minimum cell configuration requires a cell directory server, a security server, a distributed time server, and one or more client machines.

An important decision to be made while setting up a DCE system is to decide the cell boundaries. The following four factors should be taken into consideration for making this decision

1. Purpose
2. Administration.
3. Security.
4. Overhead.

1. Purpose. The machines of users working on a common goal should be put in the same cell, as they need easy access to a common set of system resources. That is, users of machines in the same cell have closer interaction with each other than with users of machines in different cells.

2. Administration.

Each system needs an administrator to register new users in the system and to decide their access rights to the system's resources.To perform his or her job properly, an administrator must know the users and the resources of the system. Therefore, to simplify administration jobs, all the machines and their users that are known to and manageable by an administrator should be put in a single cell.

For example, all machines belonging to the same department of a company or a university can belong to a single cell. From an administration point of view,each cell has a different administrator.

3. Security.

Machines of those users who have greater trust in each other should be put in the same cell. That is, users of machines of a cell trust each other more than they trust the users of machines of other cells. In such a design, cell boundaries act like firewalls in the sense that accessing a resource that belongs to another cell requires more sophisticated authentication than accessing a resource that belongs to a user's own cell.

4. Overhead.

Several DCE operations, such as name resolution and user authentication, incur more overhead when they are performed between cells than when they are performed within the same cell. Therefore, machines of users who frequently interact with each other and the resources frequently accessed by them should be placed in the same cell. The need to access a resource of another cell should arise infrequently for better overall system performance.