Topper's Solutions

....In Search of Another Topper

--- Handcrafted by BackkBenchers Community ---

DISTRIBUTED COMPUTING

(BE - COMPUTER)

8 SEM

Aug 2020 Edition

As per Revised Syllabus w.e.f 2019-20

TOPPER'S SOLUTIONS

....In Search of Another Topper

There are many existing paper solution available in market, but Topper's Solution is the one which students will always prefer if they refer... ;) Topper's Solutions is not just paper solutions, it includes many other important questions which are important from examination point of view. Topper's Solutions are the solution written by the Toppers for the students to be the upcoming Topper of the Semester.

It has been said that "**Action Speaks Louder than Words**" So Topper's Solutions Team works on same principle. Diagrammatic representation of answer is considered to be easy & quicker to understand. So our major focus is on diagrams & representation how answers should be answered in examinations.

Why Topper's Solutions:

- Point wise answers which are easy to understand & remember.
- Diagrammatic representation for better understanding.
- Additional important questions from university exams point of view.
- Covers almost every important question.
- In search of another topper.

"Education is Free.... But its Technology used & Efforts utilized which we charge"

It takes lot of efforts for searching out each & every question and transforming it into Short & Simple Language. Entire Community is working out for betterment of students, do help us.

Thanks for Purchasing & Best Luck for Exams

BACKK BENCHERS

❤ Handcrafted by BackkBenchers Community ❤

If you don’t have **BIG DREAMS** and **GOALS**, you’ll end up working for someone who does.

---- Anonymous.

Syllabus:

Exam	TT-1	TT-2	AVG	Term Work	Oral/Practical	End of Exam	Total
Marks	20	20	20	25	25	80	150

#	Module	Details Contents	Page No.
1.	Introduction to Distributed Systems	1. Characterization of Distributed Systems: Issues, Goals, and Types of distributed systems, Distributed System Models, Hardware concepts, Software Concept. 2. Middleware: Models of Middleware, Services offered by middleware, Client Server model	01
2.	Communication	1. Layered Protocols, Interprocess communication (IPC): MPI, Remote Procedure Call (RPC), Remote Object Invocation, Remote Method Invocation (RMI) 2. Message Oriented Communication, Stream Oriented Communication, Group Communication.	20
3.	Synchronization	1. Clock Synchronization, Logical Clocks, Election Algorithms, Mutual Exclusion, Distributed Mutual Exclusion-Classification of mutual Exclusion Algorithm, Requirements of Mutual Exclusion Algorithms, Performance measure. 2. Non Token based Algorithms: Lamport Algorithm, Ricart–Agrawala‘s Algorithm, Maekawa‘s Algorithm 3. Token Based Algorithms: Suzuki-Kasami‘s Broardcast Algorithms, Singhal‘s Heurastic Algorithm, Raymond‘s Tree based Algorithm, Comparative Performance Analysis	36
4.	Resource and Process Management	1. Desirable Features of global Scheduling algorithm, Task assignment approach, Load balancing approach, load sharing approach 2. Introduction to process management, process migration, Threads, Virtualization, Clients, Servers, Code Migration	54
5.	Consistency, Replication and Fault Tolerance	1. Introduction to replication and consistency, Data-Centric and Client-Centric Consistency Models, Replica Management. 2. Fault Tolerance: Introduction, Process resilience, Reliable client-server and group communication, Recovery.	69
6.	Distributed File Systems and Name Services	1. Introduction and features of DFS, File models, File Accessing models, File-Caching Schemes, File Replication, Case Study: Distributed File Systems (DSF), Network File System (NFS), Andrew File System (AFS) 2. Introduction to Name services and Domain Name System, Directory Services, Case Study: The Global Name Service, The X.500 Directory Service 3. Designing Distributed Systems: Google Case Study	77

Note: We have tried to cover almost every important question(s) listed in syllabus. If you feel any other question is important and it is not cover in this solution then do mail the question on Support@BackkBenchers.com or Whatsapp us on +91-9930038388 / +91-7507531198

Contact No: 7507531198
Email ID: Support@ToppersSolutions.com
Website: www.ToppersSolutions.com

CHAP - 1: INTRODUCTION TO DISTRIBUTED SYSTEMS

Q1. Explain distributed system and list the advantages and disadvantages.

Ans:

DISTRIBUTED SYSTEM:

1. A distributed system is also known as **distributed computing.**
2. It is a collection of independent computers that appears to it users as a single coherent system.
3. Distributed system deals with two aspects i.e. hardware and software.
4. One important characteristics of distributed systems is that differences between the various computers and the ways in which they communicate are hidden from users.
5. **Example:** Internet and Intranet.

CHARACTERISTICS OF DISTRIBUTED SYSTEMS:

1. **Resource Sharing:** Connecting Resources and Users.
2. **Transparency:** Communication is hidden from users.
3. **Openness:** Applications can interact with uniform and consistent way.
4. **Scalability:** It will remain effective when there is a significant increase in the number of users and the number of resources.
5. **Heterogeneity:** Building blocks could be from different makes and models.
6. **Security:** How secure is the system against malicious attacks.
7. **Fault Tolerance:** How it deals with failures like message loss, network partitioning, etc.
8. **Concurrency:** How concurrent shared resources are being used.
9. **Quality of Service:** Adaptability, availability, reliability, etc.

ADVANTAGES OF DISTRIBUTED SYSTEMS:

1. All the nodes in the distributed system are connected to each other. So nodes can easily share data with other nodes.
2. More nodes can easily be added to the distributed system i.e. it can be scaled as required.
3. Failure of one node does not lead to the failure of the entire distributed system. Other nodes can still communicate with each other.
4. Resources like printers can be shared with multiple nodes rather than being restricted to just one.

DISADVANTAGES OF DISTRIBUTED SYSTEMS:

1. It is difficult to provide adequate security in distributed systems because the nodes as well as the connections need to be secured.
2. Some messages and data can be lost in the network while moving from one node to another.
3. The database connected to the distributed systems is quite complicated and difficult to handle as compared to a single user system.
4. Overloading may occur in the network if all the nodes of the distributed system try to send data at once.

Q2. Explain Architectures styles in distributed System

Ans:

DISTRIBUTED SYSTEM:

1. A distributed system is a collection of independent computers that appear to the users of the system as a single computer.
2. Example is **World Wide Web**.

ARCHITECTURAL STYLES:

I) Layered Architecture:

1. The basic idea for the layered architecture is that all the components are organized in a layered manner.
2. Where component at any layer is allowed to call components at its underlying layer.
3. A fundamental observation is that control generally flows from layer to layer, i.e., requests go down the hierarchy whereas the results flow upward.
4. Figure 1.1 represents layered architecture.

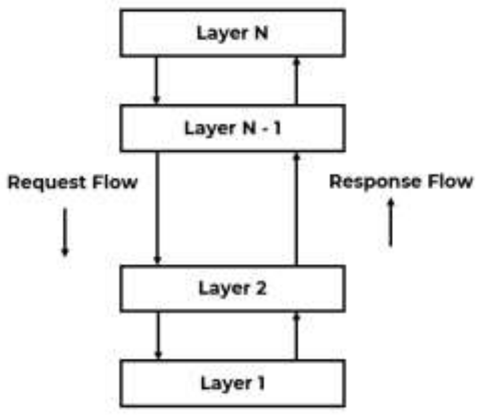


Figure 1.1: Layered Architecture.

II) Object-based Architecture:

1. Object based architecture uses **Remote procedure call** mechanism for communication.
2. In the object based architecture, each component or object is connected through a remote procedure call mechanism.
3. Thus, any object can call to any other object in the system and hence the called object can return data or information to the calling object.
4. Figure 1.2 represents object-based architecture.

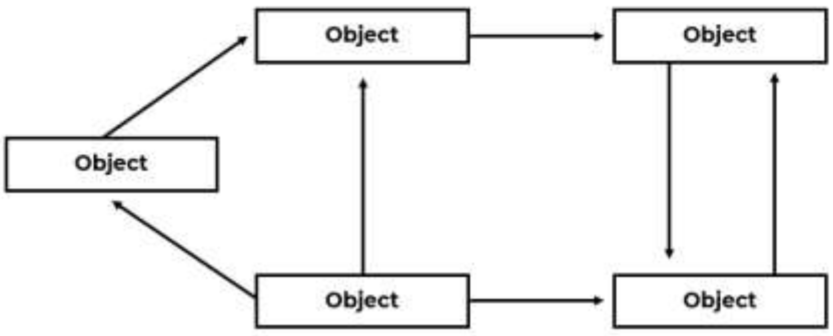


Figure 1.2: Object-Based Architecture.

III) Data-Centered Architecture:

1. In data-centered architecture, server or database lies at the center of the architecture.
2. Clients are placed around the server.
3. Thus centered server provides data or information to different clients of the system as shown in figure 1.3.

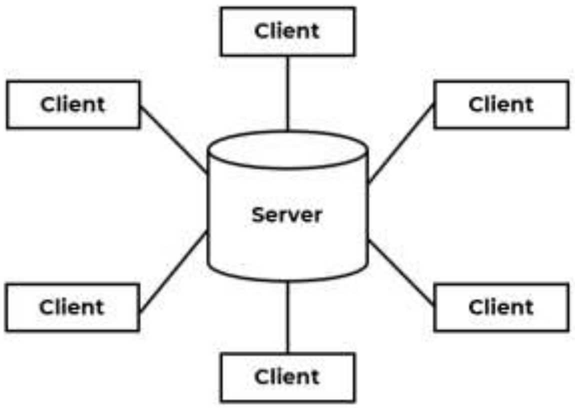


Figure 1.3: Data-Centered Architecture.

IV) Event-Based Architecture:

1. The entire communication in this kind of a system happens through events.
2. When an event is generated, it will be sent to the bus system.
3. With this, everyone else will be notified telling that such an event has occurred.
4. So, if anyone is interested, that node can pull the event from the bus and use it.
5. Sometimes these events could be data, or even URLs to resources.
6. So the receiver can access whatever the information is given in the event and process accordingly.
7. Processes communicate through the propagation of events.
8. These events occasionally carry data.
9. The main advantage of event-based distributed system is that, processes are loosely coupled or they are simply distributed.
10. Figure 1.4 represents event based architecture.

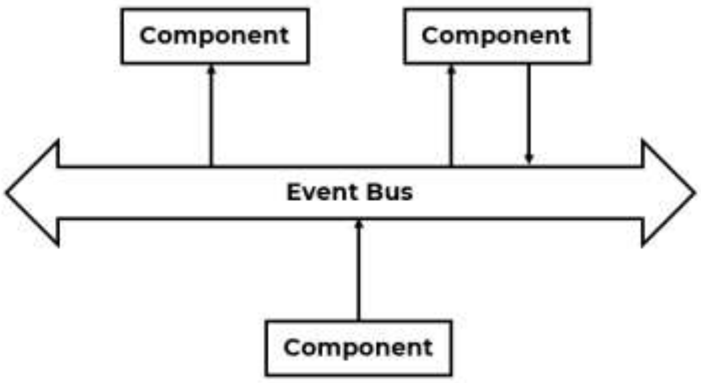
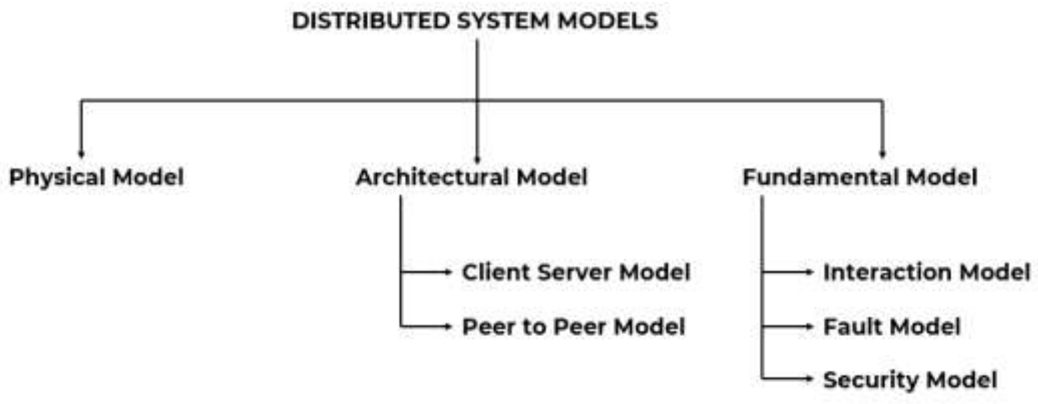


Figure 1.4: Event-Based Architecture.

Q3. Explain Distributed System Architectures

Ans:

DISTRIBUTED SYSTEM MODELS:



PHYSICAL MODEL:

1. A physical model is a representation of the underlying hardware elements of a distributed system.
2. It captures the hardware composition of a system in terms of computers and other devices and their interconnecting network.
3. Physical Model can be categories into three generations of distributed systems i.e. early distributed systems, Internet-scale distributed systems and Contemporary distributed systems.

I) Early distributed systems:

- Emerged in the late 1970s and early 1980s because of the usage of local area networking technologies.
- System typically consisted of 10 to 100 nodes connected by a LAN, with limited Internet connectivity and supported services.
- Example: shared local printer, file servers

II) Internet-scale distributed systems:

- Emerged in the 1990s because of the growth of the Internet.
- It incorporates a large number of nodes, across organizations.
- It has increased heterogeneity.

III) Contemporary distributed systems:

- Emergence of mobile computing leads to nodes that are location-independent.
- Emergence of cloud computing and ubiquitous computing
- Scale is ultra large.

ARCHITECTURAL MODEL:

1. Architectural model defines the way in which the components of the system interact with each other and the way in which they are mapped onto an underlying network of computers.
2. It **simplifies and abstracts the functionality** of the individual components of a distributed system.
3. It describes responsibilities distributed between system components and how these components are placed.
4. **Examples: Client Server Model and Peer to Peer Model**.

I) Client Server Model:

- In the basic client-server model, processes in a distributed system are divided into two groups i.e. **server and client**.
- A server is a process implementing a specific service, for example, a file system service or a database service.
- A client is a process that requests a service from a server by sending it a request and subsequently waiting for the server's reply.
- The client-server model is usually based on a **simple request/reply protocol**.
- Figure 1.5 represents client server model in distributed system.
- A server can itself request services from other servers; thus, in this new relation, the server itself acts like a client.

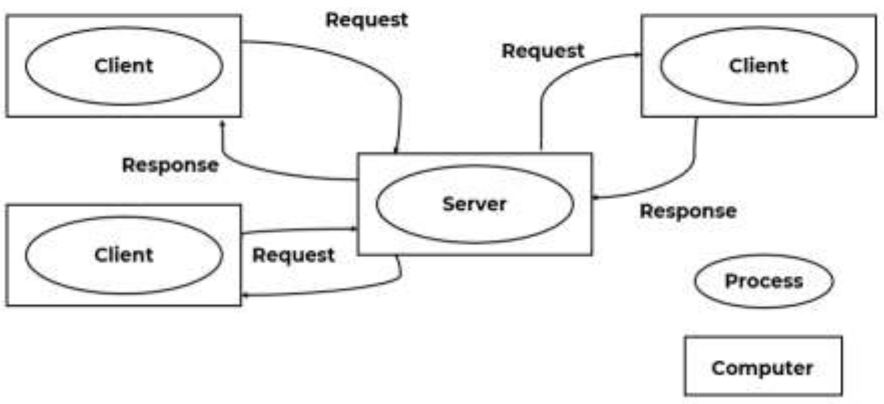


Figure 1.5: Client Server Model

II) Peer to Peer Model:

- The general idea behind peer to peer is where there is no central control in a distributed system.
- The basic idea is that, each node can either be a client or a server at a given time.
- If the node is requesting something, it can be known as a client, and if some node is providing something, it can be known as a server. In general, each node is referred to as a Peer.
- This is the most general and flexible model.
- In this network, any new node has to first join the network.
- After joining in, they can either request a service or provide a service.
- In this model, all the processes has the **same capabilities and responsibilities**.
- It tries to solve some problems which are associate with client server model.
- Figure 1.6 represents peer to peer model in distributed system.

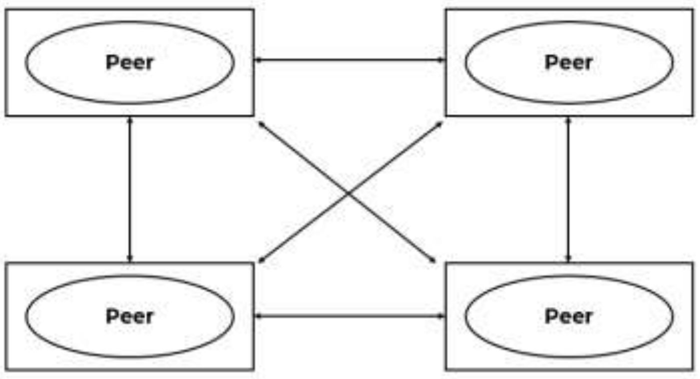


Figure 1.6: Peer to Peer Model

FUNDAMENTAL MODEL:

1. The purpose of fundamental model is to make explicit all relevant assumptions about the system we are modeling.
2. It includes interaction, fault and security model.

I) Interaction Model:

- Interaction model **deals with performance and are used for handling time** in distributed systems i.e. for process execution, message delivery, clock drifts etc.
- The two variants of the interaction model are **synchronous and asynchronous distributed systems**.
- **Synchronous distributed systems:**
    - A synchronous distributed system comes with strong guarantees about properties and nature of the system.
    - Because the system makes strong guarantees, it usually comes with **strong assumptions and certain constraints**.
    - In synchronous distributed system, following bounds are defined:
        - **Upper Bound on Message Delivery:**
          There is a known upper bound on message transmission delay from one process to another process. Messages are not expected to be delayed for arbitrary time periods between any given set of participating nodes.
        - **Ordered Message Delivery:**
          The communication channels between two machines are expected to deliver the messages in **FIFO order**. It means that the network will never deliver messages in an arbitrary or random order that can't be predicted by the participating processes.
        - **Notion of Globally Synchronized Clocks:**
          Each node has a local clock, and the clocks of all nodes are always in sync with each other. This makes it trivial to establish a global real time ordering of events not only on a particular node, but also across the nodes.
        - **Lock Step Based Execution:**
          The participating processes execute in lock-step.
- **Asynchronous distributed systems:**
    - The most important thing about an asynchronous distributed system is that it is more suitable for real world scenarios since it does not make any strong assumptions about time and order of events in a distributed system.
    - There are no bounds on:
        - Process execution speed.
        - Message transmission delays.
        - Clock drift rate
    - In asynchronous distributed system, there is **no global physical time**.
    - Reasoning can be only in terms of logical time.
    - Asynchronous distributed systems are unpredictable in terms of timing.
    - No timeouts can be used.

II) Fault Model:

- Failures can occur both in processes and communication channels.
- The reason can be both software and hardware faults.
- The failure model defines how the failure occurs in the system and what its effects are.
- Fault models are needed in order to build systems with predictable behavior in case of faults (systems which are fault tolerant).
- There are three categories of fault – **omission fault, arbitrary fault and timing fault**.
- **Omission Fault:**
    - When a process or channel fails to do something that it is expected to it is termed an **omission failure**.
    - There are two categories of omission fault i.e. process and communication omission fault.
    - **Process omission failures:**
        - A process makes an omission failure when it **crashes** — it is assumed that a crashed process will make no further progress on its program.
        - A crash is considered to be clean if the process either functions correctly or has halted.
        - A crash is termed a **fail-stop** if other processes can detect with certainty that the process has crashed.
    - **Communication omission failures:**
        - Communication omission failures may occur in the sending process (send-omission failures), the receiving process (receive omission failures) or the channel (channel omission failures).

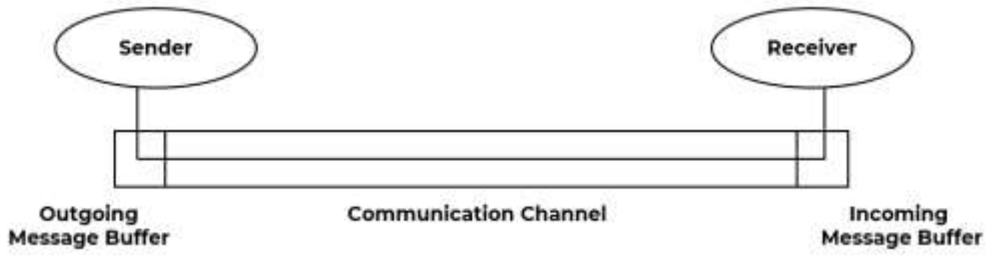


Figure 1.7

- **Arbitrary Fault:**
    - It is also known as **Byzantine Fault**.
    - The term “arbitrary” or “byzantine failure” is used to refer to the type of failure in which any error may occur.
    - In a process, arbitrary behaviour may include setting incorrect data values, returning a value of incorrect type, stopping or taking incorrect steps.
    - In a channel, arbitrary behaviour may include duplication or corruption of messages.
    - Most protocols include mechanisms to overcome arbitrary failures in a channel.
    - For example, checksums to detect corruption and sequence numbers to detect duplication.
    - Arbitrary failures in a process are less easy to detect and can have a profound impact on a system in which several processes need to cooperate.
    - For example, consider the behaviour of the leader election algorithms if one of the processes behaved erratically and did not follow the protocol of the algorithm.

- **Timing Fault:**
  - Synchronous distributed system is one in which each of the following are defined:
    - Upper and lower bounds on the time to execute each step of a process;
    - A bound on the transmission time of each message over a channel;
    - A bound on the drift rate of the local clock of each process.
  - If one of these bounds is not satisfied in a synchronous system then a **timing failure** is said to have occurred.
  - Few real systems are synchronous (they can be constructed if there is guaranteed access to resources) but they provide a useful model for reasoning about algorithms — timeouts can be used to detect failed processes.
  - In an asynchronous system a timeout can only indicate that a process is not responding.

III) Security Model:

- Security model is based on establishing the trustworthiness and role of each component such as trusted users, trusted servers, trusted administrators and client.
- Security model is used for:
  - Protecting access to objects - Access rights and authentication of clients
  - Protecting processes and interactions.
  - Protecting communication channel.
- Figure 1.8 shows represents protecting objects and figure 1.9 shows protecting process.

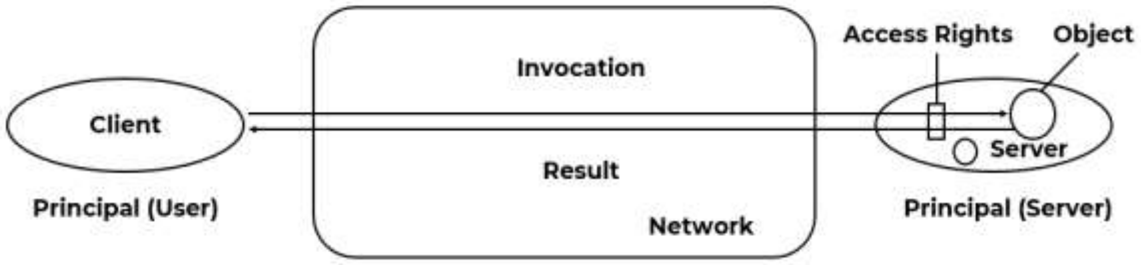


Figure 1.8: Projecting Objects.

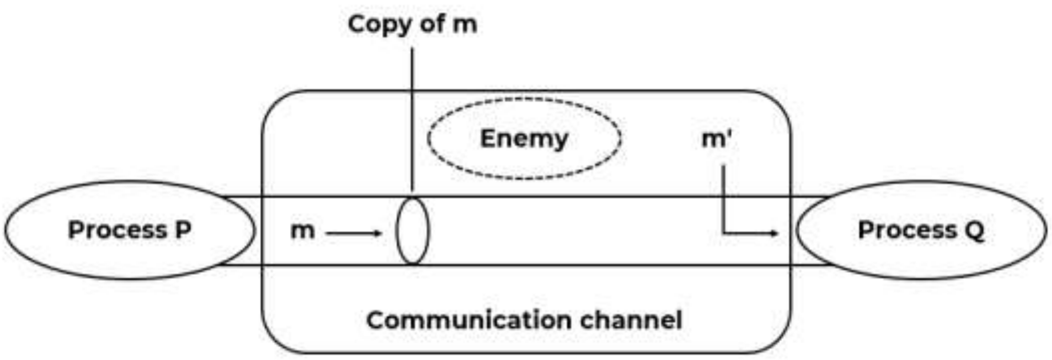


Figure 1.9: Protecting process

- Protecting communication channel can be done using:
  - Client and server knows the identity of each other (Authentication Process)
  - Secure channel ensures privacy and integrity of data (Cryptography Process)
  - Each message includes a logical timestamp to prevent messages from being reordered.

- Threats to processes are the problem of unauthenticated requests / replies.
- Example: “man in the middle"
- Threats to communication channels are the problem such as enemy may copy, alter or inject messages as they travel across network.
- Use of “secure” channels, based on cryptographic methods solve such threats.

Q4. Explain designing issues of distributed systems.

Ans:

ISSUES OF DISTRIBUTED SYSTEMS:

I) Transparency:

1. Transparency ensures that the distributed system should be perceived as the single entity by the users.
2. The existence of multiple computers should be invisible to the user.
3. There are 8 types of transparency that has to be achieved:
   a. **Access Transparency:** The user should not know if the resource (Hardware or Software) is remote or local.
   b. **Location Transparency:** There are two aspects to location transparency:
      i. **Name Transparency:** The name of the resource should not reveal the location of the server.
      ii. **User Mobility:** The user should be able to login and access a resource with the same name, no matter the machine.
   c. **Replication Transparency:** The existence of multiple copies of a file should be transparent to the user.
   d. **Failure Transparency:** Any types of failures including communication failure should be masked from the user and the system should be able to continue performing as if no failure has occurred.
   e. **Migration Transparency:** For better performance, reliability and security reasons an object is often migrated from one node to another in a distributed system. The user should be transparent to this migration.
   f. **Concurrency Transparency:** The user should feel that he is the sole owner of the system.
   g. **Performance Transparency:** The system should automatically reconfigure itself to improve performance transparent from the user.
   h. **Scaling Transparency:** The system should be able to automatically expand its scale without disrupting the activities of the users.

II) Reliability:

1. One of the original goals of building distributed systems was to make them more reliable than single-processor systems.
2. The idea is that if a machine goes down, some other machine takes over the job.
3. A highly reliable system must be highly available, but that is not enough.
4. 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.

5. 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.

III) Failure Handling:

1. When some faults occur in hardware and the software program, it may produce incorrect results or they may stop before they have completed the intended computation.
2. So corrective measures should to implement to handle this case.
3. Failure handling is difficult in distributed systems because the failure is partial i.e. some components fail while others continue to function.

IV) Flexibility:

1. Flexibility is considered as the ability which can be easily modified.
2. It includes:
   a. **Ease of modification:** It should be easy to incorporate changes in the system.
   b. **Ease of enhancement:** It should be easy to add new functionality into the system.

V) Performance:

1. Always the hidden data in the background is the issue of performance.
2. Building a transparent, flexible, reliable distributed system, more important lies in its performance.
3. Running a particular application on a distributed system, should not be appreciably worse than running the same application on a single processor.
4. The performance should be at least as good as a centralized system.

VI) Scalability:

1. Distributed systems operate effectively and efficiently at many different scales, ranging from a small intranet to the Internet.
2. 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.

VII) Heterogeneity:

1. A heterogeneous system consists of interconnected sets of dissimilar hardware or software or both.
2. Heterogeneity is applied to the network, computer hardware, operating system and implementation of different developers.
3. The system should be designed in such a way that it should cope with the heterogeneous environment.

VIII) Security:

1. Security is one of the most difficult jobs in a distributed environment. But it has to be achieved.
2. It includes:
   a. Confidentiality: Protection against disclosure to unauthorized person.
   b. Integrity: Protection against corruption and alteration.
   c. Availability: Keep the resources accessible.

IX) Concurrency:

1. There is a possibility that several clients will attempt to access a shared resource at the same time.

2. Multiple users make requests on the same resources, i.e. read, write, and update.
3. Each resource must be safe in a concurrent environment.
4. Any object that represents a shared resource a distributed system must ensure that it operates correctly in a concurrent environment.

X) Openness:

1. It is also known as **Directness**.
2. The distributed system should be open.
3. An open distributed system is a system that offers services according to standard rules.
4. It is used to describe syntax and semantics of services used.

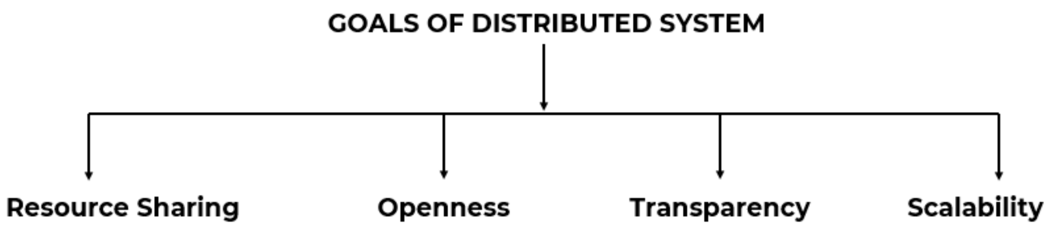
Q5. Explain goals of distributed systems.

Ans:

DISTRIBUTED SYSTEM:

1. A distributed system is a collection of independent computers that appear to the users of the system as a single computer.
2. Example is World Wide Web.

GOALS OF DISTRIBUTED SYSTEM:



I) Resource Sharing:

1. The main goal of a distributed system is to make it easy for users to access remote resources and to share them with others in a controlled way.
2. Examples: Printers, Files, Web Pages etc.
3. A distributed system should also make it easier for users to exchange information.
4. Easier resource and data exchange could cause security problems – a distributed system should deal with this problem.

II) Openness:

1. The openness of distributed system 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.
2. It is an important goal of distributed system.
3. It offers services according to standard rules that describe the syntax and semantics of those services.
4. Open distributed system must be flexible making it easy to configure and add new components without affecting existing components.

5. An open distributed system must also be extensible.
6. Achieving openness by at least make the distributed system independent from heterogeneity of the underlying environment:
    a. Hardware.
    b. Platforms.
    c. Languages.

III) Transparency:

1. It is important for a distributed system to hide the fact that the processes and resources are physically distributed across multiple computers.
2. A distributed system that can portray itself as a single system is said to be transparent.
3. Transparency is of various forms as follows

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

IV) Scalability:

1. Scalability is one of the most important goals which are measured along three different dimensions.
2. First, a system can be scalable with **respect to its size** i.e. how effective the system is when there is a significant increase in the number of resources and the number of users.
3. Second, users and resources can be **geographically apart**.
4. Third, it is possible to manage even if many **administrative organizations are spanned**.

Q6. What are the different models of middleware?

Ans:

Middleware is computer software that provides services to software applications beyond those available from the operating system.

MODELS OF MIDDLEWARE:

I) Client Server Model:

1. Client Server Model is most widely used Model in Middleware.
2. It is used for communication between processors.
3. In this model, one process acts as a server while all other processes acts as a clients.
4. Clients request for the services.
5. Server provides the services to clients.
6. Figure 1.10 shows client server model.

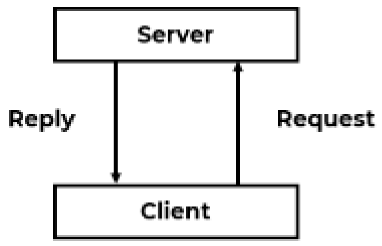


Figure 1.10: Client Server Model.

II) Vertical Distribution Model:

1. It is also known as **Multi-Tier Model**.
2. It is an extension of client server model.
3. This model includes multiple servers.
4. The client request is directed to first server.
5. This request is proceed to next server, until the final server is reached.
6. Figure 1.11 shows Vertical Distribution Model.

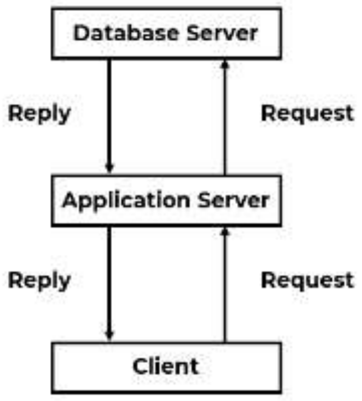


Figure 1.11: Vertical Distribution Model.

III) Horizontal Distribution Model:

1. Horizontal Distribution Model is used to improve **scalability and reliability**.
2. It involves replicating a server's functionality over multiple computers.
3. In this model, each server machine contains a complete copy of all hosted web pages and client's requests are passed on to the servers.
4. Figure 1.12 shows horizontal distribution model.

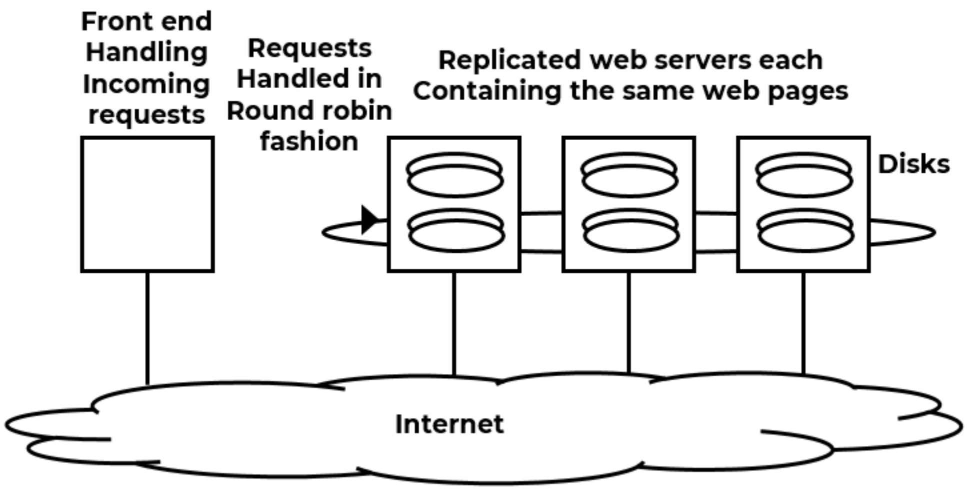

Front end Handling Incoming requests
Requests Handled in Round robin fashion
Replicated web servers each Containing the same web pages
Disks
Internet

Figure 1.12: Horizontal Distribution Model.

IV) Peer-to-Peer Model:

1. Peer-to-Peer Model is a **decentralized communication model**.
2. In this model, each node functions as both a client and server.
3. It can request the services as well as send a respond to the request.
4. Example of Peer-to-Peer Model is Collection of PCs in an office or home.
5. Figure 1.13 shows the Peer-to-Peer Model.

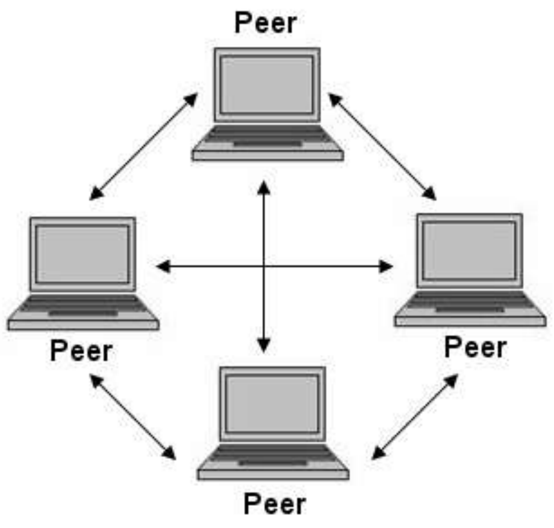


Figure 1.13: Peer-to-Peer Model.

V) Hybrid Model:

1. Hybrid Model can be combination of any of the above models.
2. Some of the examples of Hybrid Models are:
   a. **Super-peer networks:** It is combination of client server and peer-to-peer model. Example is BitTorrent.
   b. **Edge-server networks:** In edge server networks, servers are placed at the edge of the internet. Example is at Internet Service Provider (ISP), the home users access the nearby edge servers instead of the original servers.

Q7. Explain in brief the software concept of distributed systems

Ans:

SOFTWARE CONCEPT:

Software concept in distributed systems includes:

I) Distributed Operating System:

1. Distributed Operating System are referred as **Loosely Coupled Systems**.
2. Distributed Operating System is a model where distributed applications are running on multiple computers linked by communications.
3. A distributed operating system is an **extension of the network operating system**.
4. It supports higher levels of communication and integration of the machines on the network.
5. Figure 1.14 shows the example of Distributed Operating System.

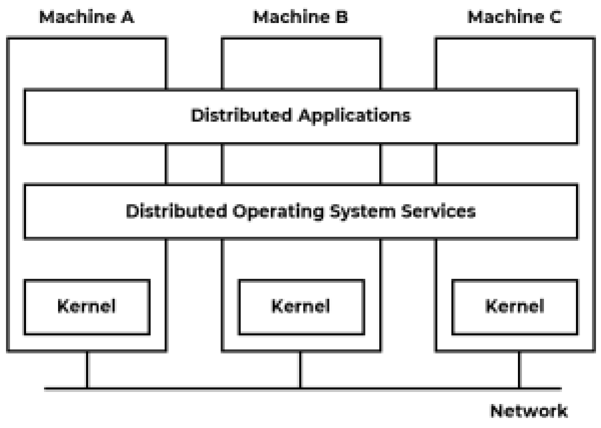


Figure 1.14: Distributed Operating System.

Goal: To hide and to manage hardware resources.

<u>Distributed Operating Systems provide the following advantages:</u>

1. Sharing of resources.
2. Reliability.
3. Communication.
4. Computation speedup.

II) <u>Network Operating System:</u>

1. Network Operating System are referred as **Tightly Coupled Systems**.
2. An operating system that provides the connectivity among a number of autonomous computers is called a Network Operating System.
3. It includes special functions for connecting computers and devices into a local-area network (LAN) or Inter-network.
4. Some popular network operating systems are Windows NT/2000, Linux, Sun Solaris, UNIX, and IBM OS/2.
5. Figure 1.15 shows the example of Network Operating System.

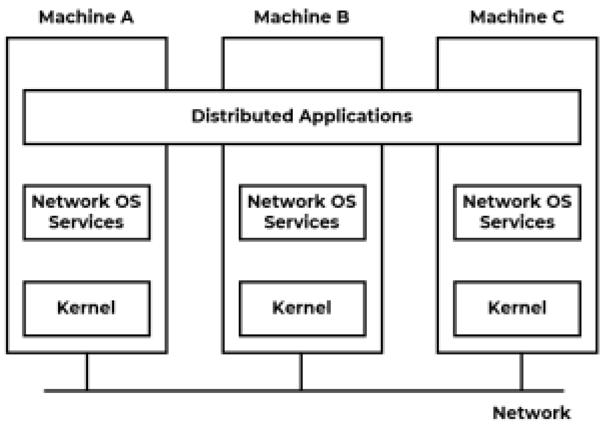


Figure 1.15: Network Operating System.

Goal: To offer local services to remote clients.

<u>Network Operating Systems provide the following advantages:</u>

1. Centralized servers are highly stable.
2. Security is server managed.

3. Communication.
4. Remote access to servers is possible from different locations.

III) Middleware:

1. Middleware is an additional layer at the top of Network Operating System.
2. It is used to implement general purpose services.
3. Figure 1.16 shows the example of Middleware Operating System.

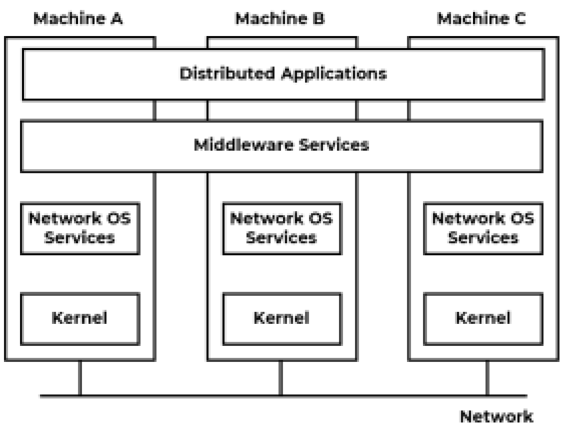


Figure 1.16: Middleware Operating System.

Goal: To provide distribution transparency.

Middleware Services:

1. Communication Services.
2. Information System Services.
3. Control Services.
4. Security Services.

Q8. Explain in difference between network operating and distributed operating system

Ans:

COMPARISON BETWEEN NOS & DOS:

Table 1.1: Comparison between NOS & DOS

Parameters	NOS	DOS
Definition:	It is a computer operating system that is designed primarily to support workstation, personal computer which is connected on a LAN.	It is an operating system which manages a number of computers and hardware devices which make up a distributed system.
Goal:	Goal is to offer local service to remote client.	Goal is to hide and manage hardware resources.
OS type:	It is tightly coupled operating system.	It is loosely coupled operating system.
Used for:	Used for Heterogeneous multi computers.	Used for multi-processor and homogeneous multi computers.

Architecture:	Follows '2' tier client server architecture.	Follows 'n' tier client server architecture.
Types:	Multicomputer and multiprocessor operating system.	Peer to peer and client server architecture.
Communication mode:	It uses file for communication.	It uses messages for communication.
Transparency:	Degree of transparency is low.	Degree of transparency is high.
Reliability:	Low.	High.
Example:	Novell NetWare.	Microsoft distributed component object model.
Diagram:	Refer Figure 1.15	Refer Figure 1.14

Q9. Distributed Computing System & Models.

Ans: **[Optional]**

DISTRIBUTED COMPUTING SYSTEM:

Features of Distributed Computing System:

1. Multiprocessor System.
2. Processors are geographically apart.
3. Communication is by message passing.
4. Coordination among the processors to the specific task.

MODELS:

I) Mini Computer Model:

1. Mini computer model is simple extension of centralized time sharing system.
2. In this model, several mini computers are interconnected by a communication network.
3. Several interactive terminals are connected to each minicomputer.
4. This network allows a user to access remote resources.
5. In this model, the terminals which are interconnected to minicomputer shares the load of resources.
6. ARPAnet is the example of a minicomputer model.
7. Figure 1.17 represents Mini Computer Model.

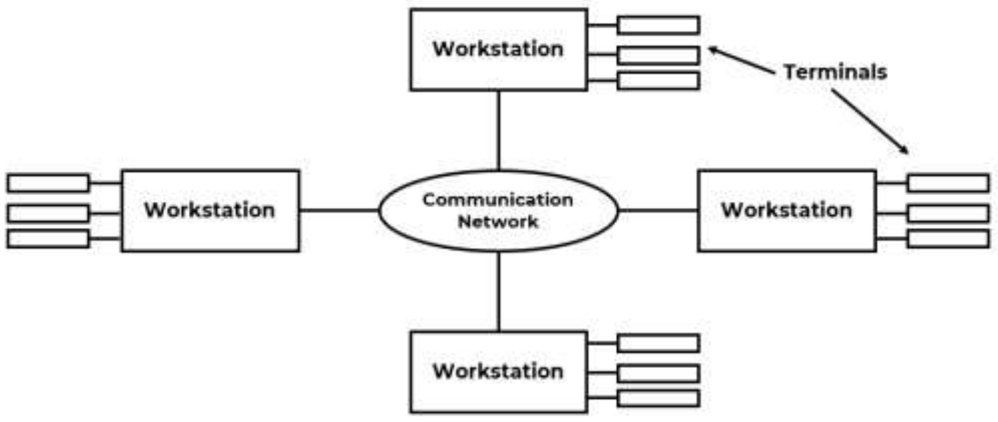


Figure 1.17: Minicomputer Model.

II) Workstation Model:

1. In this model, several workstations are interconnected to a communication network.
2. In this the terminals does not shares the load of resources.
3. The idea of this model is to interconnect all the workstations by high speed LAN.
4. Example is a company's office may have several workstation scattered throughout an organization.
5. Figure 1.18 represents Workstation Model.

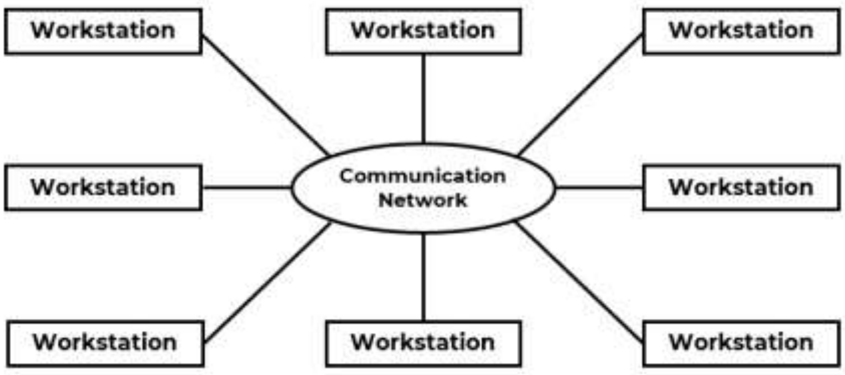


Figure 1.18: Workstation Model.

III) Workstation Server Model:

1. This model is combination of minicomputer model and workstation model.
2. In this model, several workstations and mini computers are interconnected to a communication network.
3. A workstation model is a network of personal workstation each with its own disk and a local file system.
4. Workstation with its own local disk is called as diskful workstation.
5. Workstation without a local disk is called as diskless workstation.
6. Minicomputer offers services such as database service and print service.
7. Example: The V system.
8. Figure 1.19 represents Workstation Server Model.

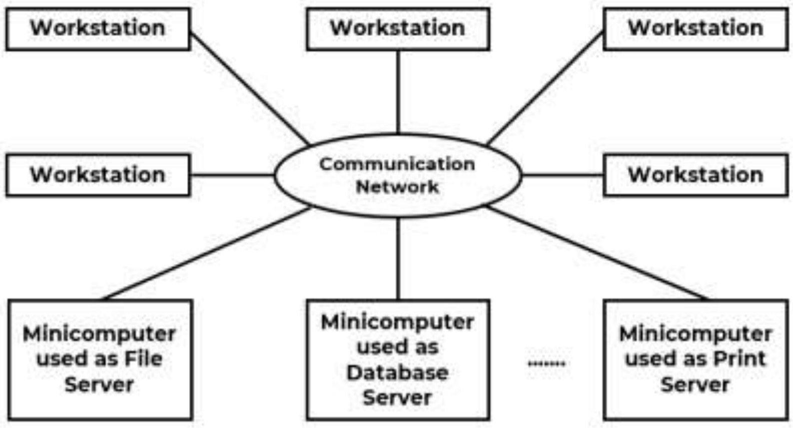

Workstation
Workstation
Workstation
Workstation
Communication Network
Workstation
Minicomputer used as File Server
Minicomputer used as Database Server
....
Minicomputer used as Print Server

Figure 1.19: Workstation Server Model.

Advantages of Workstation-Server model to Workstation model:

1. Cheaper.
2. Backup and hardware maintenance is easier.
3. Flexibility to use any workstation.

IV) Processor Pool Model:

1. It is better than workstation model.
2. In this model, several terminals and pool of processor are interconnected with a communication network.
3. Processor pool model is used when large amount of computing power is required for a short time.
4. It has better utilization of processes.
5. The pool of processer consists of large number of microcomputers and minicomputers attached to the network.
6. Each processor in the pool has its own memory to load and run a system program.
7. Example: Cambridge distributed computing system.
8. Figure 1.20 represents Processor Pool Model.

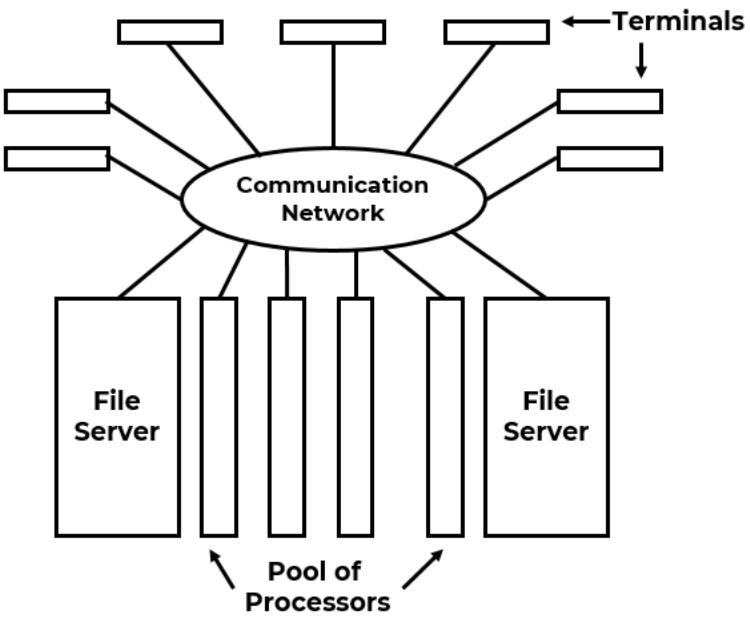


Figure 1.20: Processor Pool Model.

V) Hybrid Model:

1. The workstation-server model is the most widely used model.
2. The processor-pool model is used for when you have large number of users and often used to perform jobs needing massive computation.
3. The hybrid model is based on workstation-server model but with the addition of a pool of processors, which can be allocated for computation dynamically.
4. The advantages of both workstation model and processor pool model are combined and thus formed a hybrid model.
5. This model is very expensive.

CHAP - 2: COMMUNICATION

Q1. Explain the concept of Remote Procedure Call

Ans:

REMOTE PROCEDURE CALL:

1. Remote Procedure Call is as known as **Remote Function Call or a Remote Subroutine Call.**
2. A remote procedure call is an Interprocess communication technique that is used for client-server based applications.
3. RPC provides a general IPC protocol that can be **used for designing several distributed applications**.
4. Features of a RPC for distributed applications are:
   a. Simple call syntax.
   b. Familiar semantics.
   c. Ease of use.
   d. Generality.
   e. Efficiency.

GENERIC RPC MODEL:

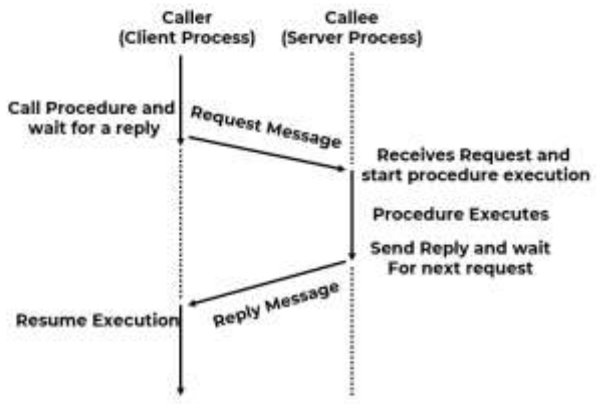

Caller
(Client Process)
Callee
(Server Process)
Call Procedure and wait for a reply
Request Message
Receives Request and start procedure execution
Procedure Executes
Send Reply and wait For next request
Reply Message
Resume Execution

Figure 2.1: Generic RPC Model.

1. Figure 2.1 shows the generic RPC Model.
2. The basic idea of RPC is to make a **remote procedure call look transparent**.
3. The calling procedure should not be aware that the called procedure is executing on a different machine.
4. RPC achieve the **transparency in analogous way**.
5. The caller (client process) sends a call (request) message to the callee (server process) and waits for the reply.
6. The request message contains the remote procedures parameters, among other things.
7. The server process executes the procedure and then returns the result of procedure execution in a reply message to the client process.
8. Once the reply message is received, the result of procedure execution is extracted, and the caller's execution is resumed.
9. It has no idea that work was done remotely instead of the local operating system.
10. In this way transparency is achieved.

RPC MODEL:

Figure 2.2 shows the RPC Model.

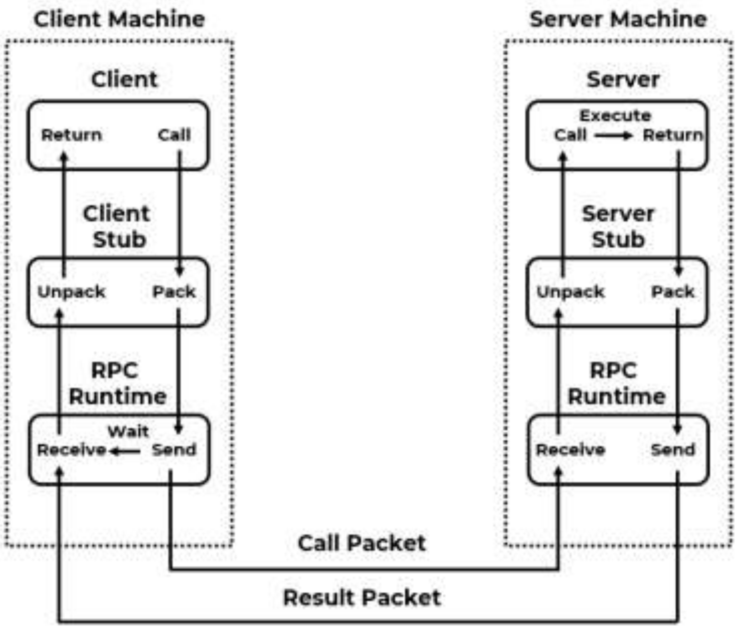


Figure 2.2: RPC Model.

IMPLEMENTATION OF RPC:

1. To achieve the goal of semantic transparency the implementation of an RPC mechanism is based on the concept of **STUBS**.
2. Stubs provide a **local procedure call abstraction.**
3. The implementation of an RPC mechanism usually involves the following five elements.
4. **Client:** The user process that initiates a remote procedure call.
5. **Client Stub:** It is responsible for:
   a. On receipt of a call request from the client, it packs a specification of the target procedure and the arguments into the message.
   b. On receipt of the result of procedure execution, it unpacks the result and passes it to the client.
6. **RPC Runtime:** It handles transmission of messages across the network between client and server machines including encryption, routing acknowledgement.
7. **Server Stub:** The job of the server stub is very similar to client stubs.
8. **Server:** The server executes the appropriate procedure and returns the result.

ADVANTAGES OF REMOTE PROCEDURE CALL

1. Remote procedure calls support process oriented and thread oriented models.
2. The internal message passing mechanism of RPC is hidden from the user.
3. The effort to re-write and re-develop the code is minimum in remote procedure calls.
4. Remote procedure calls can be used in distributed environment as well as the local environment.

DISADVANTAGES OF REMOTE PROCEDURE CALL

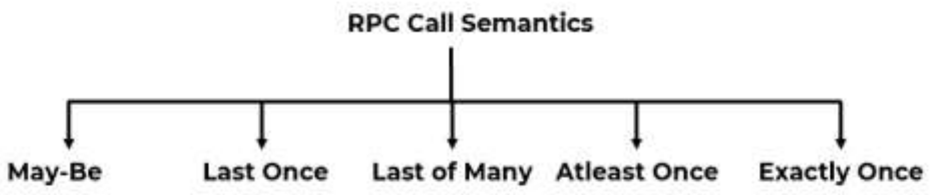
1. The remote procedure call is a concept that can be implemented in different ways. It is not a standard.
2. There is no flexibility in RPC for hardware architecture. It is only interaction based.
3. There is an increase in costs because of remote procedure call.

Q2. Explain Call Semantics of RPC

Ans: **[Optional]**

RPC CALL SEMANTICS:

1. In RPC, the caller and callee processes can be situated on different nodes.
2. The normal functioning of an RPC may get disrupted due to one or more reasons mentioned below:
   a. Call message is lost or response message is lost.
   b. The callee node crashes and is restarted.
   c. The caller node crashes and is restarted.
3. In RPC system the call semantics determines how often the remote procedure may be executed under fault conditions.
4. The different types of RPC call semantics are shown below.



I) Possibly or May-Be Call Semantics:

1. This is the weakest semantics.
2. It is a **Request Protocol**.
3. It is an Asynchronous RPC
4. In May-be call semantics, a timeout mechanism is used that prevents the caller from waiting indefinitely for a response from the callee.
5. This means that the caller waits until a pre-determined timeout period and then continues to execute.
6. Hence this semantics does not guarantee the receipt of call message nor the execution.
7. This semantics is applicable where the response message is less important.

II) Last-Once Call Semantics:

1. It is Request / Reply protocol.
2. Retransmission based on timeout.
3. After time out, result of last execution is used by the caller.
4. It issues Orphan Call (crash of caller).
5. It is useful in designing simple RPC.

III) Last-of-Many Call Semantics:

1. It is similar to last-one semantic except that the orphan calls are neglected using call identifiers.
2. Orphan call is one whose caller has expired due to node crash.
3. To identify each call, unique call identifiers are used which to neglect orphan calls.
4. When a call is repeated, it is assigned to a new call identifier and each response message has a corresponding call identifier.
5. A response is accepted only if the call identifier associated with it matches the identifier of the most recent call else it is ignored.

IV) At-Least-Once Call Semantics:

1. This semantics guarantees that the call is executed one or more times but does not specify which results are returned to the caller.
2. It can be implemented using timeout based retransmission without considering the orphan calls.

V) Exactly-Once Call Semantics:

1. This is the strongest and the most desirable call semantics.
2. It is Request / Reply / ACK protocol.
3. No matter how many calls are made, a procedure is executed only once
4. It eliminates the possibility of a procedure being executed more than once irrespective of the number of retransmitted call.
5. The implementation of exactly-once call semantics is based on the use of timeouts, retransmission, call identifiers with the same identifier for repeated calls and a reply cache associated with the callee.

Q3. Explain parameters passing in RPC:

Ans:

PARAMTERS PASSING in RPC:

1. The function of client stub is to take its parameters, pack them into message and send them to server stub.
2. This is the difficult process.
3. Passing parameters includes:

I) Call by Value:

1. Packing parameters into a message is called as parameter marshaling.
2. In this, all the parameters are copied into a message that is transmitted from the client to the server.
3. Consider a remote procedure, sum (i, j) which takes two integers parameters and returns their arithmetic sum.
4. The client stubs take these two parameters and puts them into message and also puts the name or number of the procedure to be called in the message.
5. When the message arrives at the server, the stub examines the message to see which procedure is needed, and then makes the appropriate call.
6. When the server has finished its execution, it takes the result provided by the server and packs it into a message.
7. This message is sent back to the client stub, which unpacks it and returns the value to the client procedure.

Example:

1. IBM mainframes use the EBCDIC character code, whereas IBM personal computers use ASCII.
2. It is not possible to pass a character parameter from an IBM PC client to an IBM mainframes server using the simple scheme as explained above.
3. Similar problems can occur with the representation of integers and floating point number.

Drawback:

Drawback of this type of parameter passing semantics is that they are not desirable for large data.

II) Call by Reference:

1. Some RPC mechanisms allow passing of parameters by reference in which pointers to the parameters are passed from client to server.
2. In this, only the reference to the data is passed.
3. For example distributed systems with distributed shared memory mechanisms can allow passing of parameters by reference.
4. A pointer is meaningful only within the address space of the process in which it is being used.
5. Suppose there are two parameters to be passed, if the second parameter is the address of the buffer which is 1000 on the client, one cannot just pass the number 1000 to the server and expect it to work.
6. Address 1000 on the server might be in the middle of the program text.

Drawback:

1. Drawback of this type of parameter passing semantics is that they pointers lose their value when the message is passed from client to a server in another geographically located Server.
2. They are usually used in a closed system.

Q4. Explain message oriented communication with suitable example.

Ans:

MESSAGE ORIENTED COMMUNICATION:

1. Message-oriented communication is a way of communicating between processes.
2. Messages correspond to events and are considered as the basic units of data delivered.
3. Two aspects of communication are persistent communication and transient communication.

PERSISTENT COMMUNICATION:

1. In this communication, transferred message is stored at a communication server until it is delivered to the receiver.
2. Example: Email system - message is stored at the communication server until it is delivered.
3. It includes synchronous and a synchronous communication.

I) Persistent Synchronous Communication:

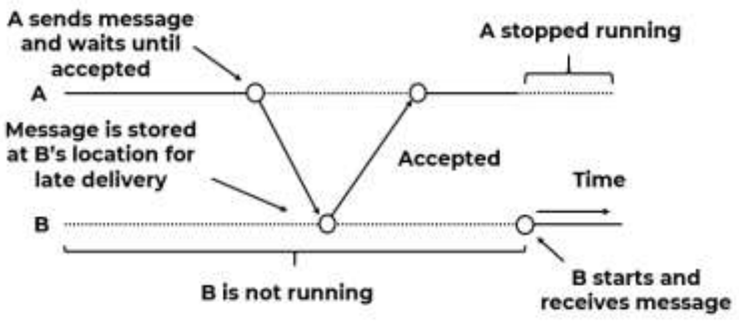

A sends message and waits until accepted
A stopped running
A
Message is stored at B's location for late delivery
Accepted
Time
B
B is not running
B starts and receives message

Figure 2.3: Persistent synchronous communication

1. Here messages are stored at the receiving host.
2. A sender is blocked until its message is stored in receiver's host.
3. It is not necessary that receiving application should be running.
4. Figure 2.3 shows the example of persistent synchronous communication.

II) Persistent Asynchronous Communication:

1. Here the message is either stored in buffer at the local host or communication server.
2. Example: Email.
3. Figure 2.4 shows the example of persistent asynchronous communication.

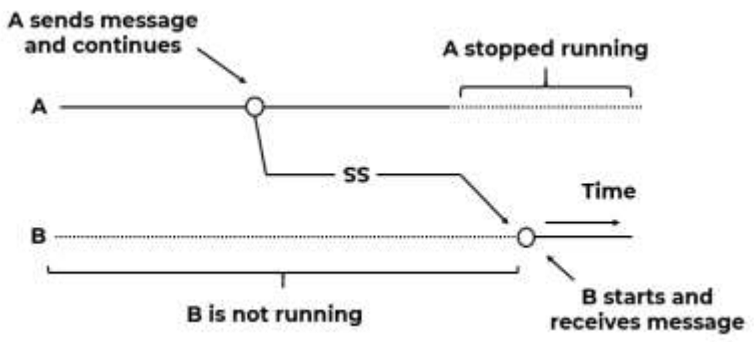


Figure 2.4: Persistent asynchronous communication

TRANSIENT COMMUNICATION

1. In this communication, message can be stored only if the sender and receiver applications are executing
2. A message is discarded by a communication server as soon as it cannot be delivered to the receiver.
3. Example: Router- if the router cannot deliver the message to the next router it drops the message
4. It includes Asynchronous and synchronous communication.
   a. **Asynchronous communication:** Sender continues immediately after the message has been sent.
   b. **Synchronous communication:** Sender is blocked until the request is known to be accepted.

I) Transient asynchronous communication:

1. Figure 2.5 explains this form of communication.
2. A sends the message and continues execution (non blocking).
3. B has to be running, because if it is not running the message will be discarded.
4. Even if any router along the way is down, the message will be discarded.
5. UDP communication is an example of transient asynchronous communication

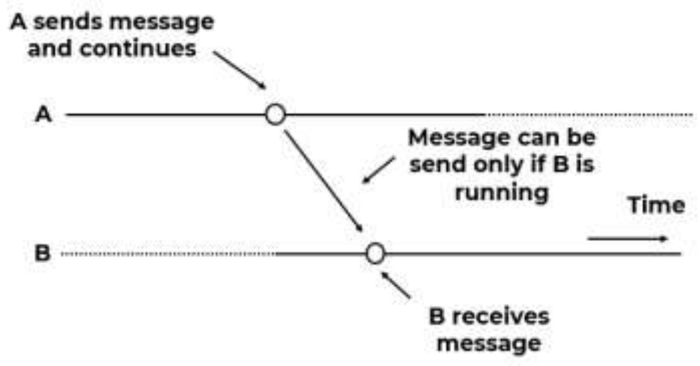

A sends message and continues
A
Message can be send only if B is running
Time
B
B receives message

Figure 2.5: Transient Asynchronous Communication

II) Receipt based Transient Synchronous Communication:

1. Figure 2.6 explains receipt-based transient synchronous communication.
2. A's message to B is blocking (synchronous) until an ack is received.
3. This ack simply tells us that the message was received at the other end.
4. It does not tell us anything about whether the process has started.

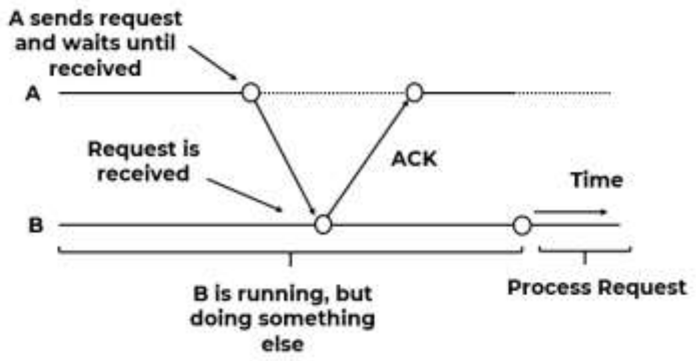

A sends request and waits until received
A
Request is received
ACK
Time
B
B is running, but doing something else
Process Request

Figure 2.6: Receipt based Transient Synchronous Communication

III) Delivery based Transient Synchronous Communication:

1. This is an extension of receipt-based transient synchronous communication.
2. As shown in figure 2.7, A will resume running when B takes the delivery of the message.
3. The ack comes a little bit later than the previous method.
4. This is essentially asynchronous RPC because from the perspective of an RPC, we are not blocking for the reply.

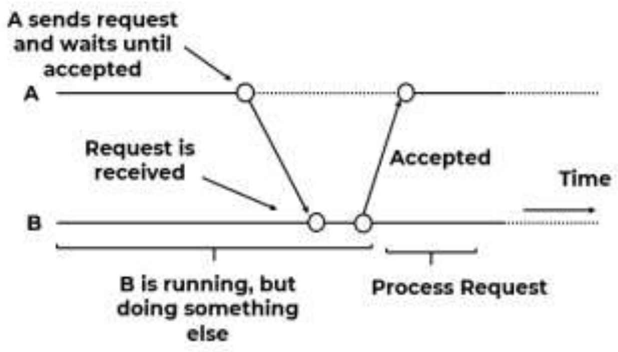

A sends request and waits until accepted
A
Request is received
Accepted
Time
B
B is running, but doing something else
Process Request

Figure 2.7: Delivery based Transient Synchronous Communication.

IV) Response based Transient Synchronous Communication:

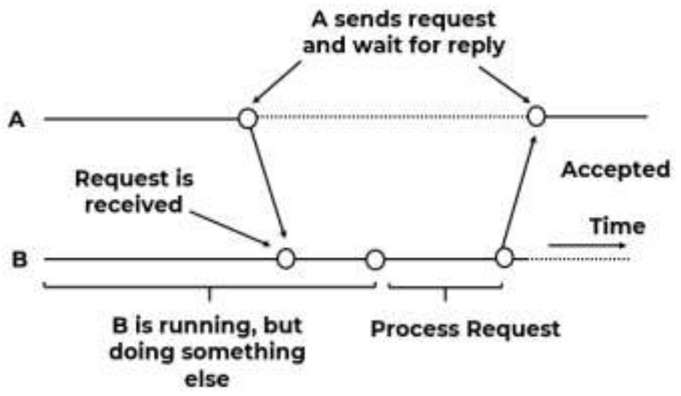

A sends request and wait for reply
A
Request is received
Accepted
Time
B
B is running, but doing something else
Process Request

Figure 2.8: Response based Transient Synchronous Communication

1. Figure 2.8 explains this type of communication.
2. A resumes execution upon receiving a response.
3. That is, not only has the message been delivered, but it has been processed and the ack comes back in the form of a reply.
4. This is traditional RPC. The client is blocked for the entire duration the reply comes back.

Q5. Explain stream oriented communication with suitable example.

Ans:

STREAM ORIENTED COMMUNICATION:

1. RPC and Message Oriented Communication are based on the **exchange of discrete messages**.
2. Timing might affect performance, but not correctness.
3. In Stream Oriented Communication the message content must be delivered at a certain rate, as well as correctly.
4. **Example:** Music or video.
5. Stream Oriented Communication supports **continuous media communication.**

TRANSMISSION MODES IN STREAM ORIENTED COMMUNICATION:

1. **Asynchronous Mode:** No restrictions with respect to when data is to be delivered
2. **Synchronous Mode:** Define a maximum end-to-end delay for individual data packets.
3. **Isochronous Mode:** Define a maximum end-to-end delay and maximum delay variance.

STREAM:

1. A data stream is a **connection-oriented communication facility**.
2. It supports **isochronous data transmission**.
3. Some common stream characteristics:
   a. Streams are **unidirectional**.
   b. There is generally a single **source.**
4. Stream types:
   a. **Simple**: It consists of a single flow of data (e.g., audio or video)
   b. **Complex**: It consists of multiple data flows (e.g., stereo audio or combination audio/video)

EXAMPLE:

Token Bucket Algorithm:

1. Figure 2.9 shows the implementation of Token Bucket.
2. The token bucket can be easily implemented with a counter.
3. The token is initialized to zero.
4. Each time a token is added, the counter is incremented by 1.
5. Each time a unit of data is dispatched, the counter is decremented by 1.
6. If the counter contains zero, the host cannot send any data.

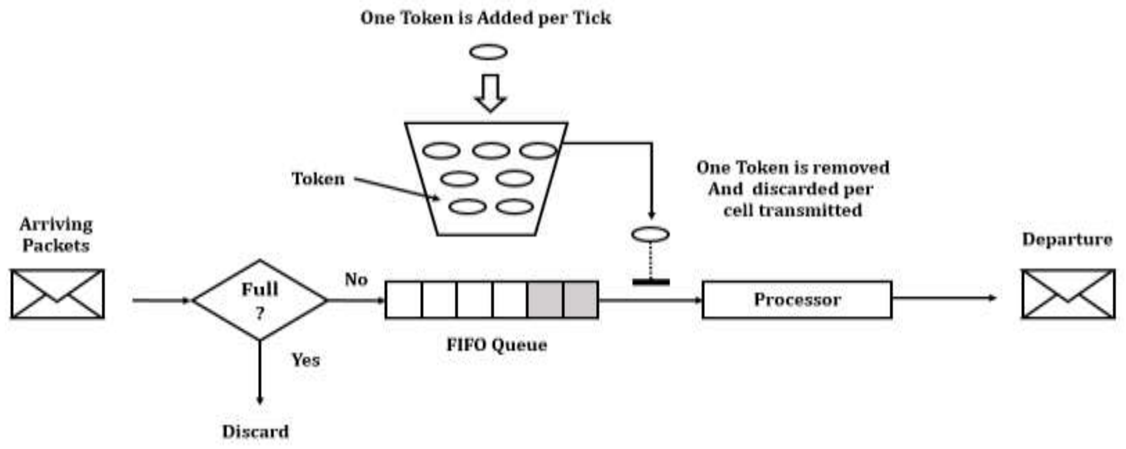

One Token is Added per Tick
Token
One Token is removed And discarded per cell transmitted
Arriving Packets
Full ?
No
Yes
Discard
FIFO Queue
Processor
Departure

Figure 2.9: Implementation of Token Bucket.

Q6. Differentiate between Message oriented & Stream oriented communications

Ans:

Table 2.1: Difference between Stream oriented & Message oriented communications

Stream Oriented Communications	Message Oriented Communications
It is Connection – Oriented Communication.	It is Connection - less Communication.
1 – to – 1 Communication.	Many – to – Many Communication.
Sender transfers a sequence of individual bytes.	Sender transfers a sequence of discrete messages.
It send data byte-by-byte.	It sends data as input stream
It does not keep track of the message boundaries.	It keep track of the message boundaries.
Arbitrary length transfer.	Each message limited to 64 KB.
It is used by most applications.	It is used for multimedia applications.
It runs over TCP.	It runs over UDP.
Timing might affect performance.	Timing does not affect performance.
Example of the byte stream downloading a song or a movie.	Example Of message stream is sequence of pages

---

Q7. Explain Group Communication

Ans:

GROUP COMMUNICATION:

1. A group is the collection of users sharing some common interest.
2. In group communication, Messages sent to a group of processes will deliver to all members of the group.
3. There are three types of group communication i.e. one to many, many to one and many to many communication.

ONE – TO – MANY COMMUNICATIONS:

1. There exist single sender and multiple receivers.
2. It is also called as **multicast communication**.
3. There exists a special case of, multicast communication as broadcast communication in which message is sent to all processors connected to the network.
4. One to many communication includes:
   - **a. <u>Group Management:</u>** There two types of groups open group and closed group.
     - A closed group is the one in which only the members of the group can send a message to the group as a whole, although it may send a message to an individual.
     - An Open Group is the one in which any processes in the system can send a message to the group as a whole.
   - **b. <u>Group Addressing:</u>**
     - It is possible to create a special network address to which multiple machines can listen.
     - This network address is called **multicast address**.
     - Networks with broadcasting facility declare a certain address, such as zero, as Broadcast Address.
   - **c. <u>Buffered and Unbuffered Multicast:</u>**
     - For Unbuffered Multicast the message is not buffered for the receiving process and it is lost if the receiving process is not in the state ready to receive it.
     - For Buffered Multicast, the message is buffered for receiving processes so each process of the multicast group will eventually receive the message.
   - **d. <u>Send to all and Bulletin Semantics:</u>**
     - **<u>Send-to-all semantics:</u>** A copy of message is sent to each process for multicast group and the message is buffered until it is accepted by the process.
     - **<u>Bulletin-board Semantics:</u>** A Message to be multicast is addressed to a channel instead of being sent to every individual process of multicast group.
   - **e. <u>Flexible Reliability in Multicast Communication:</u>**
     - **<u>0 reliable:</u>** No response is expected by the sender.
     - **<u>1 reliable:</u>** The sender expects at least one acknowledgement.
     - **<u>M out of N reliable:</u>** The sender decides as to how many acknowledgement it expects out of N.

MANY – TO – ONE COMMUNICATION:

1. There is only one receiver at any given time.
2. The receiver can be selective or nonselective.
3. **<u>Selective Receiver:</u>** Specifies the unique sender. A message exchange takes place only if that sender sends a message.
4. **<u>Nonselective Receiver:</u>** The receiver is ready to accept a message from any sender that who is present in the system.

MANY – TO – MANY COMMUNICATIONS:

1. In this scheme multiple senders send message to multiple receivers.
2. An important issue with many to many communications is ordered message delivery.
3. It includes:

a. <u>No ordering:</u>

- In many-to-many communication, message sent from the sender may arrive at the receiver's destination before the arrival of the message from another sender.
- But this may be reversed at another receiver's destination.
- The commonly used semantics for ordered delivery of Multicast messages is absolute ordering, consistent ordering and casual ordering.
- Figure 2.10 represents the example of no ordering.

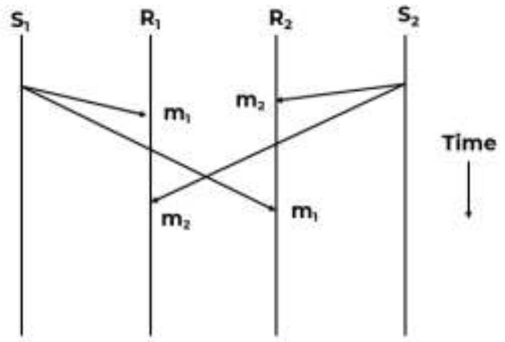


Figure 2.10: No Ordering

b. <u>Absolute ordering:</u>

- This semantics ensures that all messages are delivered to all receiver processes in the exact order in which they were sent.
- Figure 2.11 represents the example of absolute ordering.

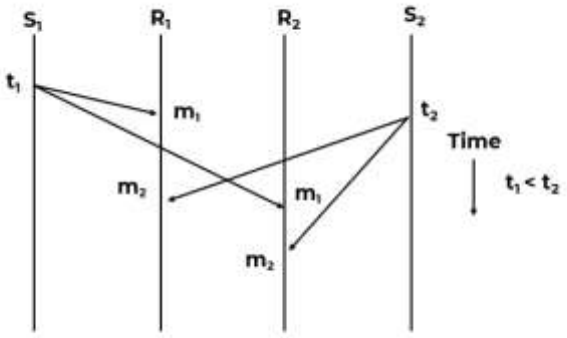


Figure 2.11: Absolute Ordering

c. <u>Consistent ordering:</u>

- This semantics ensures that all messages are delivered to all receiver processes in the same order. However, this order may be different from the order in which messages were sent.
- Figure 2.12 represents the example of consistent ordering.

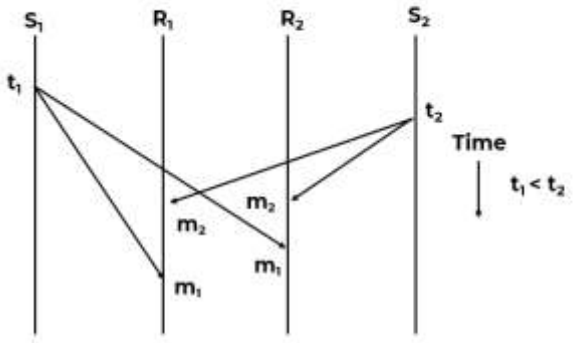


Figure 2.12: Consistent Ordering

d. **Casual Ordering:**

- This semantics ensures that if the event of the sending one message is casually related to the event of sending another message, the two messages are delivered to all receivers in the correct order.
- Figure 2.13 represents the example of casual ordering.

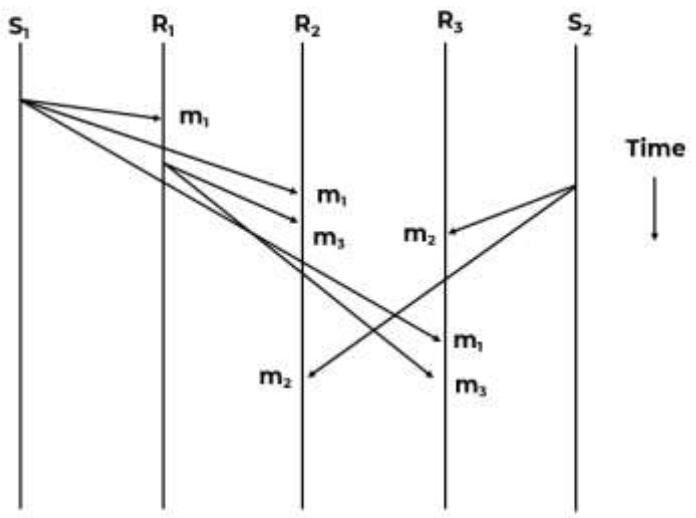


Figure 2.13: Casual Ordering

Q8. Explain Remote Method Invocation (RMI)

Ans:

REMOTE METHOD INVOCATION (RMI):

1. Remote Method Invocation (RMI) is an API which allows an object to invoke a method on an object that exists in another address space, which could be on the same machine or on a remote machine.
2. Through RMI, object running in a client side can invoke methods on an object present in server side.
3. RMI creates a public remote server object that enables client and server side communications through simple method calls on the server object.

RMI ARCHITECTURE:

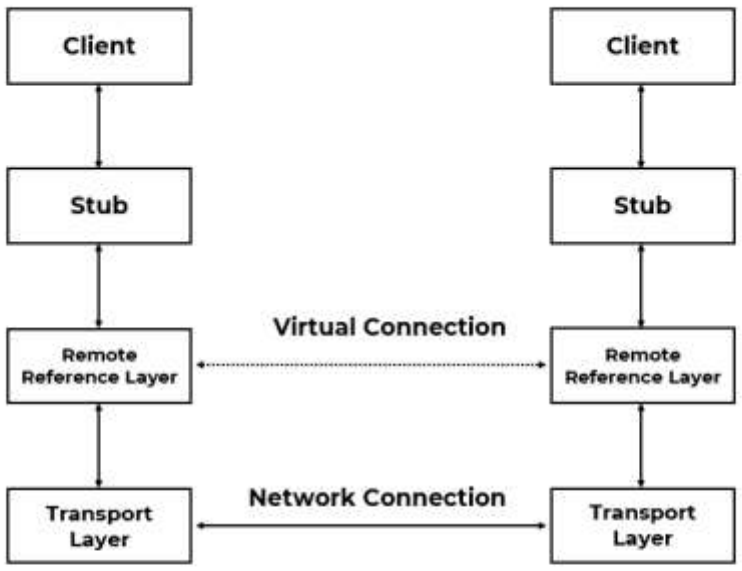


Figure 2.14: RMI Architecture

1. In an RMI application, we write two programs, a server program (resides on the server) and a client program (resides on the client).
2. Inside the server program, a remote object is created and reference of that object is made available for the client (using the registry).
3. The client program requests the remote objects on the server and tries to invoke its methods.
4. The following figure 2.14 shows the architecture of an RMI.
5. It includes:
   a. **Transport Layer:** This layer connects the client and the server. It manages the existing connection and also sets up new connections.
   b. **Stub:** A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.
   c. **Skeleton:** This is the object which resides on the server side. Stub communicates with this skeleton to pass request to the remote object.
   d. **Remote Reference Layer:** It is the layer which manages the references made by the client to the remote object.

WORKING OF RMI:

1. When the client makes a call to the remote object, it is received by the stub which eventually passes this request to the Remote Reference Layer (RRL).
2. When the client-side RRL receives the request, it invokes a method of the object. It passes the request to the RRL on the server side.
3. The RRL on the server side passes the request to the Skeleton (proxy on the server) which finally invokes the required object on the server.
4. The result is passed all the way back to the client.

MARSHALLING AND UNMARSHALLING

1. Whenever a client invokes a method that accepts parameters on a remote object, the parameters are bundled into a message before being sent over the network.
2. These parameters may be of primitive type or objects.
3. In case of primitive type, the parameters are put together and a header is attached to it.
4. In case the parameters are objects, then they are serialized.
5. This process is known as **marshalling**.
6. At the server side, the packed parameters are unbundled and then the required method is invoked.
7. This process is known as **unmarshalling**.

GOALS OF RMI

1. To minimize the complexity of the application.
2. To preserve type safety.
3. Distributed garbage collection.
4. Minimize the difference between working with local and remote objects.

Q9. Explain ISO OSI reference model with diagram?

Ans: **[Optional]**

ISO OSI REFERENCE MODEL:

1. The users of a computer network are located all over the world.
2. Therefore International group of standards has been developed to ensure that nationwide and worldwide data communication systems can be developed and are compatible to each other.
3. This standards will fit into a framework which has been developed by the "**International Organization of Standardization (ISO)**".
4. This framework is called as **OSI Model**.
5. OSI Model Stands for Open Systems Interconnection Model.
6. OSI Model has **7 Layers**.

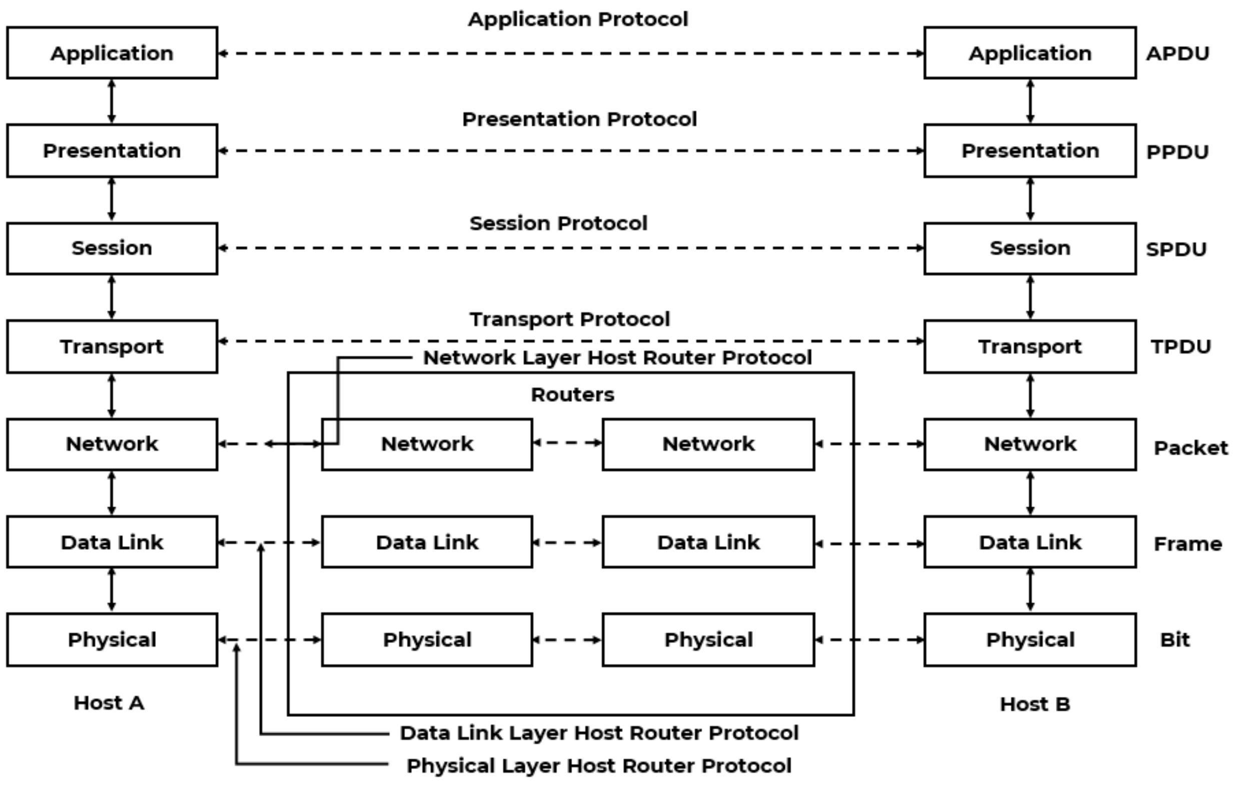


Figure 2.15: OSI Reference Model.

I) Physical Layer:

1. It is the bottom layer of OSI Model.
2. It is responsible for the actual physical connection between the devices.
3. Such physical connection may be made by **using twisted pair cable**.
4. It is concerned with transmitting bits over a communication channel.
5. **Functions of Physical Layer:**
   a. It defines the transmission rate.
   b. Transforming bits into signals.
   c. Provides synchronization of bits by a clock.
   d. It defines the way in which the devices are connected to the medium.
   e. It can use different techniques of multiplexing.

II) Data Link Layer:

1. It is responsible for **node-to-node delivery of data**.
2. It receives the data from network layer and creates **FRAMES**, add physical address to these frames & pass them to physical layer.
3. It consist of 2 layers:
   a. Logical Link Layer (LLC).
   b. Medium Access Control (MAC).
4. **Functions of Data Link Layer:**
   a. It is used for synchronization and error control for the information.
   b. It divides the bits received from Network layer into frames.
   c. It provides Flow Control.
   d. To enable the error detection, it adds error detection bits to the data which is to be transmitted.

III) Network Layer:

1. It is responsible for the **source to destination delivery of a packet** across multiple networks.
2. If two systems are attached to different networks with devices like routers, then Network layer is used.
3. Thus Data Link Layer overseas the delivery of the packet between the two systems on same network and the network layer ensures that the packet gets its point of origin to its final destination.
4. **Functions of Network Layer:**
   a. It provides Internetworking.
   b. Network Layer route the signals through various channels to the other end.
   c. It is used in Logical Addressing.
   d. It acts as a network controller for routing data.

IV) Transport Layer:

1. It is responsible for **process-to-process delivery of the entire message**.
2. Transport Layer looks after the delivery of entire message considering all its packets & make sure that all packets are in order.
3. At the receiver side, Transport Layer provides services to application layer & takes services form network layer.
4. At the source side, Transport Layer receives message from upper layer into packets and reassembles these packets again into message at the destination.
5. **Functions of Transport Layer:**
   a. It provides Connection Less & Connection Oriented Transmission.
   b. It does the functions such as multiplexing, splitting or segmentation on the data.
   c. It provides Error Control & Flow Control.
   d. It is used for Port Addressing.

V) Session Layer:

1. Session layer is the **fifth layer of OSI Model**.
2. It has the responsibility of beginning, maintaining and ending the communication between two devices, called session.

3. It also provides for orderly communication between devices by regulating the flow of data.
4. **Functions of Session Layer:**
    a. It manages & synchronizes the conversations between two different applications.
    b. It controls logging on and off.
    c. It is used for billing, user identification & session management.
    d. It provides dialog control & dialog separation.

VI) Presentation Layer:

1. Presentation layer is the **sixth layer of OSI Model**.
2. It is concerned with the syntax & semantics of the information exchanged between the two devices.
3. It works as translating layer.
4. **Functions of Presentation Layer:**
    a. It is used for data encryption, decryption and compression.
    b. It ensures that the information delivered at receiver end is understandable & usable.
    c. It is used for data presentation or translation of form and syntax.
    d. **Example:** ASCII to EBCDIC and vice versa.

VII) Application Layer:

1. It is the topmost i.e. **seventh layer of OSI Model**.
2. It enables the user to access the network.
3. It provides user interface & supports for services such as e-mail, file transfer, access to the World Wide Web.
4. So it provides services to different user applications.
5. **Functions of Application Layer:**
    a. Application Layer provides various E-mail Services.
    b. It allows users to access files in a remote host, to retrieve files from remote computer for use etc.
    c. It is provides Remote Login.

CHAP - 3: SYNCHRONIZATION

Q1. Explain Clock Synchronization and physical clock synchronization algorithms.

Ans:

CLOCK SYNCHRONIZATION:

1. Clock synchronization is a problem from computer science and engineering which deals with the idea that **internal clocks of several computers may differ**.
2. Every computer needs a timer mechanism to keep track of current time and also for various accounting purpose.
3. The variance in time is called as **clock skew**.
4. One major clock synchronization issue in distributed system is that multiple clocks have to be synchronized from time to time.
5. Physical clock synchronization algorithms can be classified as **centralized and distributed**.

CENTRALIZED CLOCK SYNCHRONIZATION ALGORITHMS:

1. In centralized system, a node called as time server node has the real time and the other nodes in the system has to update their time according to the time server node.
2. **Goal:** To keep the clocks of all other nodes synchronized with time server node.
3. **Drawback:**
   a. Single point failure.
   b. Heavy burden on a single process/node.
4. It includes Cristian's and Berkley Algorithm.

I) Cristian's Algorithm:

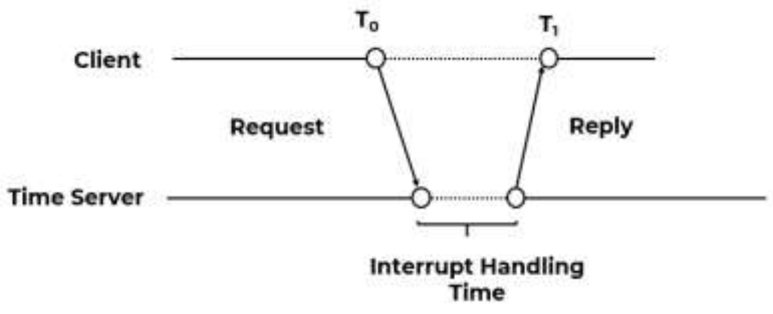


Figure 3.1: Cristian's Algorithm

1. Cristian's Algorithm is a clock synchronization algorithm is used to synchronize time with a time server by client processes.
2. This algorithm works well with low-latency networks where Round Trip Time is short as compared to accuracy.
3. Here Round Trip Time refers to the time duration between start of a Request and end of corresponding Response.
4. In this method, each node periodically sends a message to server.
5. When the time server receives the message it responds with a message T, where T is the current time of server node.

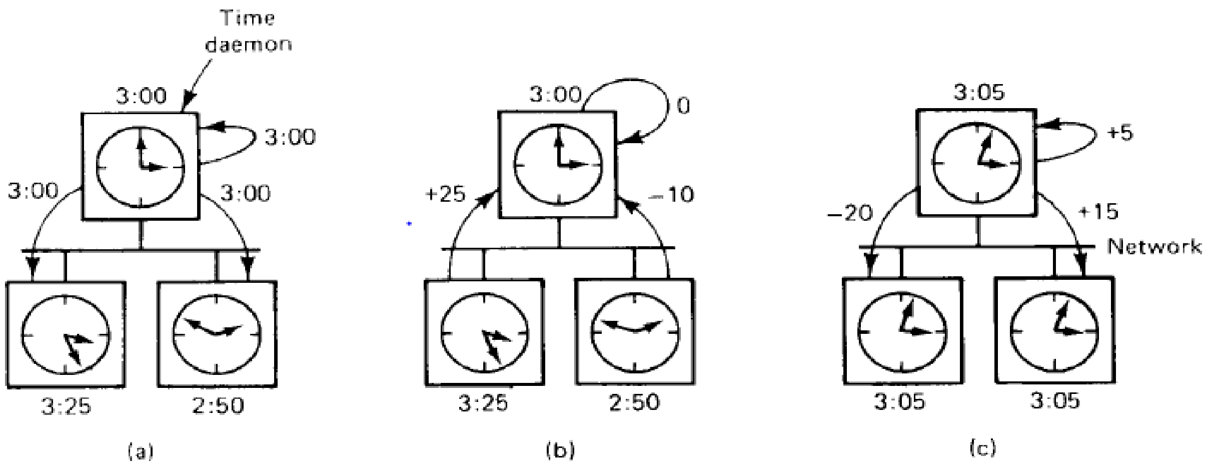
6. Assume the clock time of client be T_0 when it sends the message and T_1 when it receives the message from server.
7. T_0 and T_1 are measured using same clock so best estimate of time for propagation is $(T_1-T_0)/2$.
8. When the reply is received at clients node, its clock is readjusted to $T + (T_1-T_0)/2$.
9. Figure 3.1 represents cristian's algorithm.

Advantage: It assumes that no additional information is available.

Disadvantage: It restricts the number of measurements for estimating the value.

II) Berkley Algorithm:

1. The Berkeley algorithm is a method of clock synchronization in distributed computing which assumes no machine has an accurate time source.
2. It was developed by Gusella and Zatti at the University of California, Berkeley in 1989
3. This is an **active time server approach**.
4. In this approach, the time server periodically sends a message to all the computers in the group of computers.
5. When this message is received each computer send backs its own clock value to the time server.
6. The time server has a prior knowledge of the approximate time required for propagation of a message which is used to readjust the clock values.
7. It then takes average of clock values of all the computers.
8. The calculated average is the current time to which all clocks should be readjusted.
9. The time server readjusts its own clock to this value and instead of sending the current time to other computers it sends the amount of time each computer needs for readjustment.
10. This can be positive or negative value.
11. Figure 3.2 represents berkley algorithm.



(a) The time daemon asks all the other machines for their clock values. (b) The machines answer. (c) The time daemon tells everyone how to adjust their clock.

Figure 3.2: Berkley Algorithm

DISTRIBUTED CLOCK SYNCHRONIZATION ALGORITHMS:

1. Distributed algorithm overcomes the problem of centralized algorithm by internally synchronizing for better accuracy.
2. In this all the nodes in the system contribute in fixing the time.
3. Examples of Distributed algorithms are – Global Averaging Algorithm, Localized Averaging Algorithm, NTP (Network time protocol) etc.

I) Network Time Protocol (NTP):

1. NTP is an application layer protocol.
2. Default port no for NTP is 123.
3. It uses standard UDP internet transport protocol.
4. NTP is an Internet protocol used to synchronize the clocks of computers with some time reference.
5. It uses UTC (Coordinated Universal Time) as time reference.
6. NTP is developed by David. Mills.
7. NTP is a fault tolerant protocol that will automatically select the best of several available time sources to synchronize.
8. It is highly scalable.
9. The distributed system may consist of several reference clocks.
10. Each node of such network can exchange information either bidirectional or unidirectional.
11. A client node can send request message to its directly connected time server node or nearly connected node so the propagation time from one node to another node forms hierarchical graph with reference clocks at the top.
12. This hierarchical structure is known as strata synchronization subnet where each level is referred as strata.
13. It is shown as below in figure 3.3

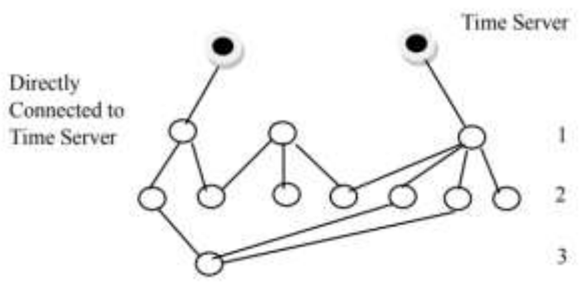


Figure 3.3: Strata Synchronization Subnet

Simple Network Time Protocol (SNTP):

1. This is a simplified way of clock synchronization method.
2. SNTP is based on unicast mode of Network Time Protocol (NTP) and also operates in multicast and procedure call mode.
3. It is recommended in network environment where server is root and client is leaf node.
4. Client node send request message at time t_1 to server node and server send current time value in reply message at time t_3 as shown below in figure 3.4.

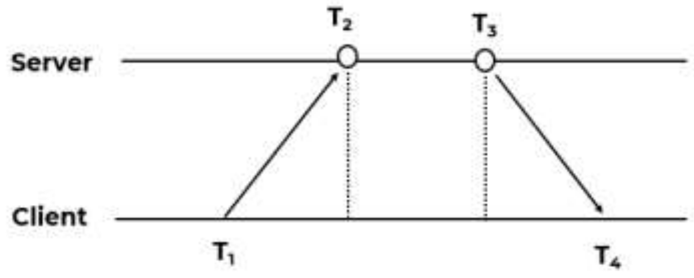


Figure 3.4: SNTP Synchronization

5. NTP protocol gives more accuracy in time than SNTP protocol.
6. It is useful for simple applications where more accurate time is not necessary.

Limitations of NTP protocol as follows:

1. NTP supports UNIX operating systems only.
2. For windows there are problems with time resolution, reference clock drives, authentication and name resolution.

II) Global Averaging Distributed Algorithm: (Optional)

1. In this algorithm, each node broadcasts its local clock time in a fixed time interval.
2. This is done when its local time is equal to To + iR for some integer i.
3. Where 'To' is a fixed time agreed by all nodes and R is a system parameter which depends on total nodes in a system.
4. After broadcasting the clock value, the clock process of a node waits for time T which is determined by the algorithm.
5. During this waiting, the clock process collects the resync message and records the time when the message is received which estimates the skew after waiting is done.
6. To find the correct time, need to estimate skew value as follows:
   a. Take the average of estimated skew and use it as correction of local clock. To limit the impact of faulty clock skews greater than threshold are set to zero before computing average of estimated skew.
   b. Each node limits the impact of faulty clock by discarding highest and lowest estimated skews and then calculates average of estimated skew.

III) Localized Averaging Distributed Algorithm: (Optional)

1. The global averaging algorithms do not scale as they need a network to support broadcast facility and a lot of traffic is generated.
2. It overcomes the drawbacks of global averaging algorithm as the nodes in distributed systems are logically arranged in a pattern or ring.
3. In this algorithm, each node exchanges its local time with its neighbors.
4. It then sets its clock time to the average of its own clock time and of its neighbors.

Q2. What is a logic clock? Why are logic clocks required in distributed systems? How does Lamport synchronize logical clocks? Which events are said to be concurrent in Lamport timestamps

Q3. Describe any one method of Logical Clock synchronization with the help of an example

Ans:

LOGIC CLOCK:

1. A logical clock is a **mechanism for capturing chronological** and **causal relationships** in a distributed system.
2. Distributed systems may have no physically synchronous global clock.
3. So a logical clock is used.
4. It allows **global ordering** on events from different processes in such systems.
5. The first implementation, the Lamport timestamps, was proposed by Leslie Lamport in 1978.
6. In logical clock systems each process has two data structures: logical local time and logical global time.
7. Logical local time is used by the process to mark its own events, and logical global time is the local information about global time.
8. A special protocol is used to update logical local time after each local event, and logical global time when processes exchange data.
9. Logical clocks are useful in **computation analysis, distributed algorithm design, individual event tracking, and exploring computational progress**.

LAMPORT'S LOGICAL CLOCKS:

1. In a distributed system, clocks need not be synchronized absolutely.
2. If two processes do not interact, it is not necessary that their clocks be synchronized because the lack of synchronization would not be observable and thus it does not cause problem.
3. It is not important that all processes agree on what the actual time is, but that they agree on the order in which events occur.
4. Lamport clocks are a simple technique used for determining the order of events in a distributed system.
5. Lamport clocks provide a partial ordering of events – specifically "happened-before" ordering.
6. If there is no "happened-before" relationship, then the **events are considered concurrent**.
7. Rules of Lamport's Logical Clocks:
   - **a. Happened Before Relation:**
     - If a & b are events in the same process and a occurs before b, then a → b.
     - If a & b belong to two different processes and a sends message to b, then a → b.
     - If a → b and b → c, then a → c.
   - **b. Logical Clocks Concept:**
     - Maintaining a common clock or set of perfectly synchronized clock is difficult.
     - So, local timestamp is given for happened before relation.

EXAMPLE:

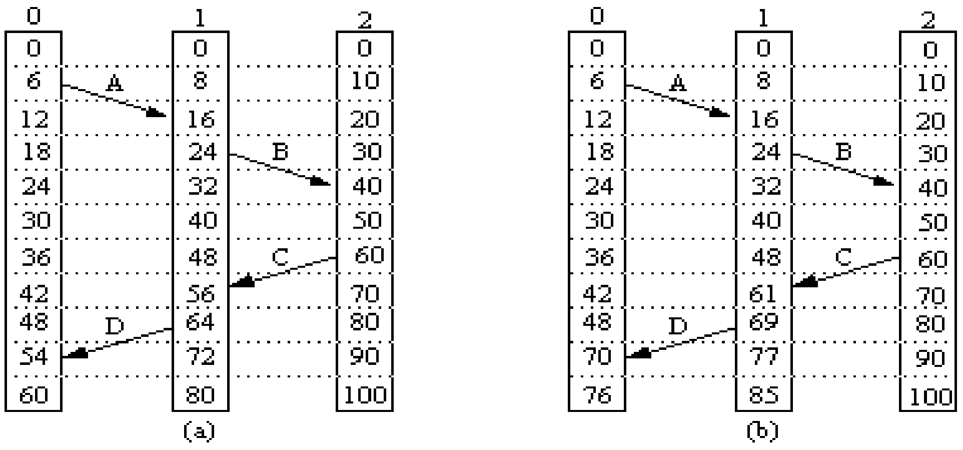


Figure 3.5: Event Ordering Example.

1. Consider three processes 0, 1 & 2.
2. For process 0 the time ticked is 6, for 1 its 8 and for 2 its 10.
3. The clocks run at constant rate but different due to difference in the crystals.
4. At time 6, process 0 sends message A to process 1 and is received at time 16.
5. Time taken for transfer is 10.
6. Similarly process 1 sends message B to process 2 and is received at time 40.
7. Time taken for transfer is 16.
8. Now message C is send from process 2 to process 1 it takes 16 ticks to reach and similarly message D takes 10 ticks to reach.
9. This value is certainly impossible and such a situation must be prevented.
10. Solution for this is Lamport happen before relation.
11. Since message C left at 64 so it must arrive at 65 or later and similarly, message D left at 69 so it must be arrive at 70 or later.
12. This is shown in figure 3.5.

Q4. What is the requirement of Election algorithm in Distributed Systems? Describe any one Election algorithm in detail with an example

Ans:

ELECTION ALGORITHM:

1. Many algorithms used in distributed systems require a **coordinator**.
2. In general, all processes in the distributed system are equally suitable for the role.
3. Election algorithms are designed to **choose a coordinator**.
4. Any process can serve as coordinator.
5. Any process can “call an election”.
6. There is no harm in having multiple concurrent elections.
7. Elections may be needed when the system is initialized, or if the coordinator crashes or retires.
8. Example of election algorithm is **Bully Algorithm**.

Assumptions and Requirement of Election Algorithm:

1. Every process/site has a unique ID.
2. **Example:** The network address or a process number.
3. Every process in the system should know the values in the set of ID numbers, although not which processors are up or down.
4. The process with the highest ID number will be the new coordinator.
5. Process groups (as with ISIS toolkit or MPI) satisfy these requirements.
6. When the election algorithm terminates a single process has been selected and every process knows its identity.
7. Formalize: every process pi has a variable e, to hold the coordinator's process number.
   - $\forall i$, e_i = undefined or $e_i = P$, where P is the non-crashed process with highest id.
   - All processes (that have not crashed) eventually set $e_i = P$.

BULLY ALGORITHM:

1. Bully Algorithm was proposed by **Garcia-Molina**.
2. This algorithm is planned on assumptions that:
   a. Each process involved within a scenario has a process number that can be used for unique identification.
   b. Each process should know the process numbers of all the remaining processes.
   c. Scenario is synchronous in nature.
   d. A process with highest process number is elected as a coordinator.
3. Process ‘p’ calls an election when it notices that the coordinator is no longer responding.
4. Process ‘p’ sends an election message to all higher-numbered processes in the system.
5. If no process responds, then p becomes the coordinator.
6. If a higher-level process (q) responds, it sends ‘p’ a message that terminates p’s role in the algorithm.
7. The process ‘q’ now calls an election (if it has not already done so).
8. Repeat until no higher-level process responds.
9. The last process to call an election “wins” the election.
10. The winner sends a message to other processes announcing itself as the new coordinator.

Example:

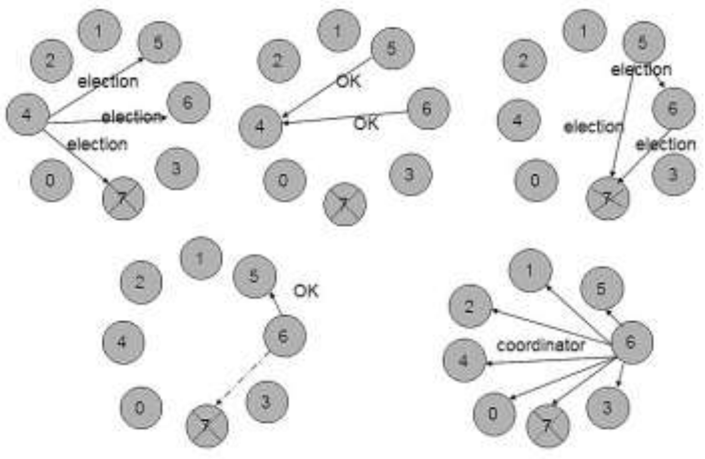


Figure 3.6: Example of Election Algorithm

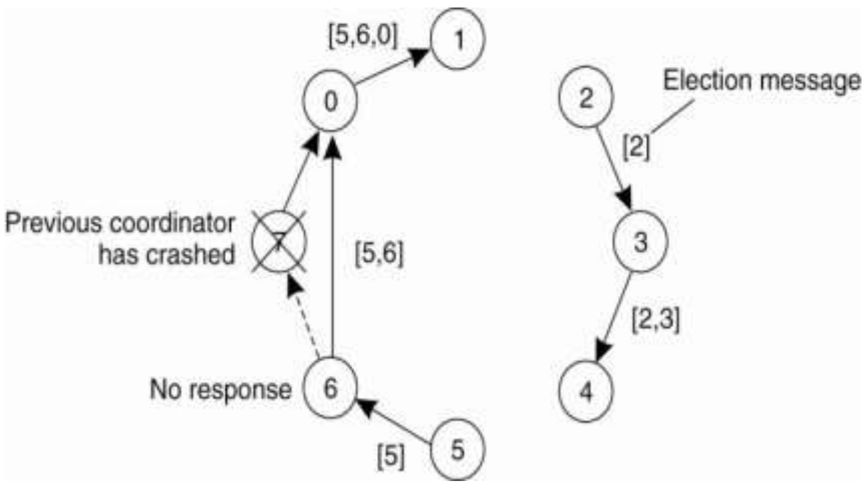
1. Process (4) calls an election when it notices that the coordinator is no longer responding.
2. Process (4) sends an election message to all higher-numbered processes in the system.
3. If no process responds, then Process (4) becomes the coordinator.
4. If a higher-level process (5, 6) responds, it sends process (4) a message that terminates (4)'s role in the algorithm.
5. Now process (5) now calls an election (if it has not already done so).
6. Repeat until no higher-level process responds.
7. The last process to call an election "wins" the election.
8. The winner sends a message to other processes announcing itself as the new coordinator.
9. Example is shown in figure 3.6.

Q5. Explain Ring Algorithm and Token ring algorithm.

Ans:

RING ALGORITHM:

1. The ring algorithm assumes that the processes are arranged in a logical ring and each process is knows the order of the ring of processes.
2. Processes are able to "skip" faulty systems: Instead of sending to process j, send to j + 1.
3. Faulty systems are those that don't respond in a fixed amount of time.
4. Process 'P' thinks the coordinator has crashed; builds an ELECTION message which contains its own ID number.
5. Sends to first live successor
6. Each process adds its own number and forwards to next.
7. OK to have two elections at once.
8. When the message returns to p, it sees its own process ID in the list and knows that the circuit is complete.
9. P circulates a COORDINATOR message with the new high number.



10. Here, both 2 and 5 elect 6:
    [5,6,0,1,2,3,4]
    [2,3,4,5,6,0,1]

TOKEN RING ALGORITHM:

1. Token ring approach is used to **achieve mutual exclusion** in a distributed system.
2. Here we have a bus network (for e.g. Ethernet).
3. A logical ring is constructed in which each process is assigned a position in the ring.
4. The ring positions may be allocated in numerical order of network addresses or some other means.
5. It does not matter what the ordering is.
6. All that matters is that each process knows who is next in line after itself.
7. When the ring is initialized, process P1 (say) is given a token.
8. The token circulates around the ring.
9. It is passed from process k to process k+1 in point-to-point messages.
10. When a process acquires the token from its neighbor, it checks to see if it is attempting to enter a Critical Section (CS).
11. If so, the process enters the CS, does all the work it needs to, and leaves the section.
12. After it has exited, it passes the token along the ring. It is not permitted to enter a second CS using the same token.
13. If a process is handed the token by its neighbor and is not interested in entering a CS, it just passes it along.
14. As a consequence, when no processes want to enter any CS, the token just circulates at high speed around the ring.

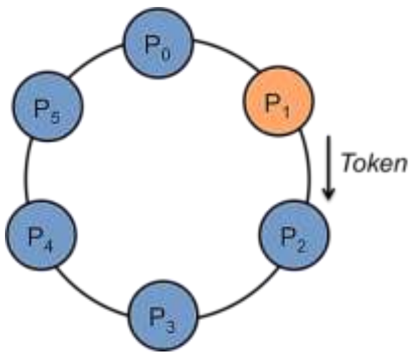


Figure 3.7: Token Ring Algorithm.

Advantage:

No starvation problem.

Disadvantage:

Detecting the lost token and regeneration is difficult.

Q6. Explain Ricart-Agrawala algorithm for Mutual Exclusion.

Q7. Explain the centralized algorithms for Mutual Exclusion?

Q8. Explain the distributed algorithms for Mutual Exclusion? What are the advantages and disadvantages of it over centralized algorithms?

Ans:

MUTUAL EXCLUSION:

1. There may be some resources like printers that must be restricted to single process at a time.

2. Therefore, exclusive accesses to such shared resources by a process have to be ensured.
3. This exclusiveness of access is called mutual exclusion.
4. The algorithm implementing mutual exclusion must satisfy two conditions:
   a. **Mutual Exclusion:** Only one process at a time can access a particular resource.
   b. **No starvation:** Every request must be eventually granted.
5. It includes centralized and distributed approach.

CENTRALIZED APPROACH:

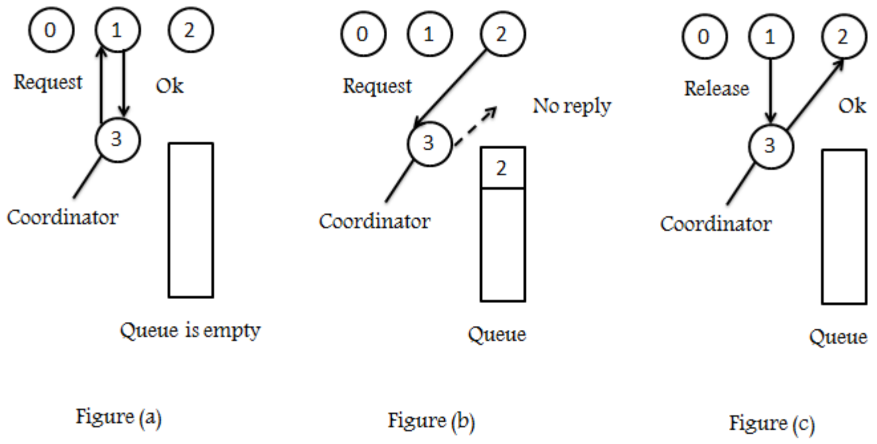


Figure 3.8: Centralized Approach for Mutual Exclusion

1. In centralized algorithm one process is elected as the **coordinator**.
2. Coordinator may be the machine with the highest network address.
3. Whenever a process wants to enter a critical region, it sends a request message to the coordinator stating which critical region it wants to enter and asking for permission.
4. If no other process is currently in that critical region, the coordinator sends back a reply granting permission, as shown in figure 3.8 (a).
5. When the reply arrives, the requesting process enters the critical region.
6. Suppose another process 2 shown in figure 3.8 (b), asks for permission to enter the same critical region.
7. Now the coordinator knows that a different process is already in the critical region, so it cannot grant permission.
8. The coordinator just refrains from replying, thus blocking process 2, which is waiting for a reply or it could send a reply saying ‘permission denied’.
9. When process 1 exits the critical region, it sends a message to the coordinator releasing its exclusive access as shown in figure 3.8 (c).
10. The coordinator takes the first item off the queue of deferred requests and sends that process a grant message.
11. If the process was still blocked it unblocks and enters the critical region.
12. If an explicit message has already been sent denying permission, the process will have to poll for incoming traffic or block later.
13. When it sees the grant, it can enter the critical region.

Advantages:

1. No starvation problem.
2. Easy to implement & use for general resources allocation.

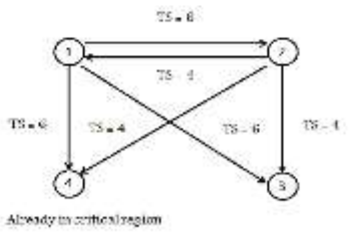
Disadvantages:

1. If coordinator crashes, the entire system will fail.
2. In large system a single coordinator can become a performance bottleneck.

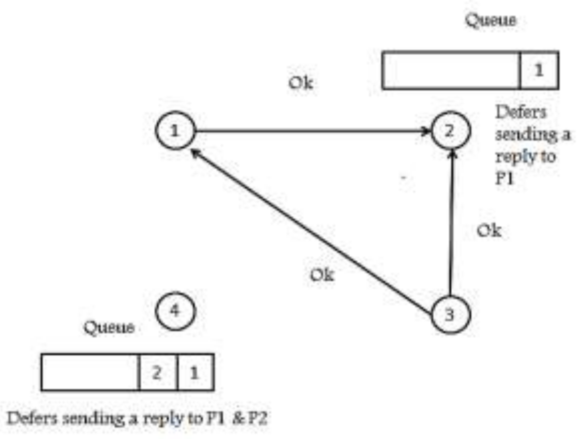
DISTRIBUTED APPROACH:

1. It is also known as **Ricart-Agrawala Algorithm**.
2. Whenever a process wants to enter the critical section it broadcasts a message that it is entering the critical section.
3. This message contains:
   a. Process identifier.
   b. Name of the critical section.
   c. A unique timestamp generated by the process for the request message.
4. On receiving a request message the process will do one of the following:
   a. If it is currently executing in the critical section it will not reply back.
   b. If a process is waiting for the said critical section it compares the time stamp of the request message and its own time stamp. If own timestamp is greater (later) it will reply back, else it won't.
   c. If neither of the above, it will immediately reply back.

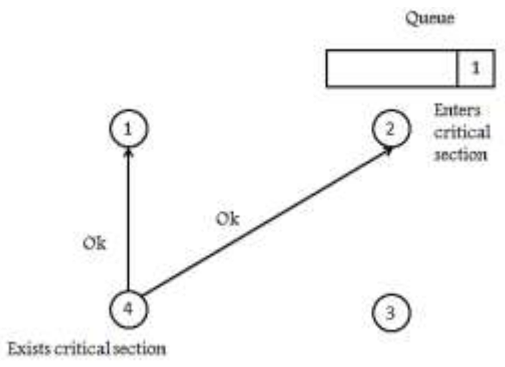
Example:



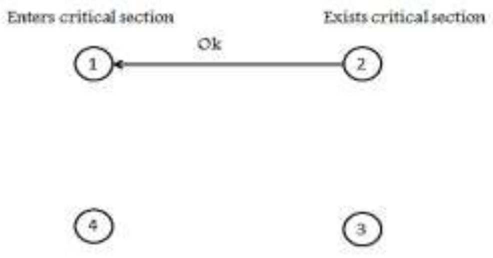
1. In above figure there are four processes 1, 2, 3 & 4.
2. While process 4 is in critical section, processes 1 and 2 want to enter the same critical section.
3. To get permission from other processes, processes 1 and 2 send request messages with the timestamps 6 and 4 respectively to other processes.


Queue
1
Ok
Defers sending a reply to P1
1
2
Ok
Ok
3
4
Queue
2
1
Defers sending a reply to P1 & P2

4. In above figure, since process 4 is in the critical section, it does not send a reply message to 1 and 2 and enters them in its queue.
5. Process 3 is currently not interested in the critical section, so it sends OK messages to both 1 and 2.
6. Process 2 defers sending a reply message to 1 and enters 1 in its queue because the timestamp (4) in its own request message is less than the timestamp (6) in 1's request message.
7. But 1 replies to 2 as the timestamp (6) in its own request message is greater than the timestamp (4) in 2's request message.



8. In above figure, when the process 4 exits the critical section, it sends an OK message to all processes in its queue.
9. Since process 2 has received OK message from all other processes, it enters the critical section.
10. However, process 1 continues to wait since it has not yet received a reply message from the process 2.



11. In above figure, when process 2 exits the critical section, it sends an OK message to 1.
12. Since process 1 has received a reply message from all other processes, it enters the critical section.

Advantage:

No starvation problem.

Disadvantages:

Message broadcasting is required when network bandwidth is a luxury.

ADVANTAGES AND DISADVANTAGES OF IT OVER CENTRALIZED ALGORITHMS

Advantage:

1. Distribution Algorithm guarantees for Mutual Exclusion.
2. **No Starvation**.
3. Request are granted in the order in which they are received.
4. Easy to implement.

Disadvantages:

1. If the Coordinator crashes then the whole system may go down.
2. Message broadcasting is required when network bandwidth is a luxury.
3. In a large system a single coordinator can become a **performance bottleneck**.

Q9. Explain Lamport's Distributed Mutual Algorithm

Ans:

LAMPORT'S DISTRIBUTED MUTUAL ALGORITHM:

1. Lamport's Distributed Mutual Exclusion Algorithm is a **contention-based algorithm** for mutual exclusion on a distributed system.
2. It is a computer algorithm.
3. It is **non token based algorithm**.
4. Lamport was the first to give a distributed mutual exclusion algorithm as an illustration of his clock synchronization scheme
5. It is intended to improve the safety in the usage of shared resources.
6. For synchronization of logical clocks, Lamport established a relation termed as happen-before relation.
7. Let Ri be the request set of site Si , i.e. the set of sites from which Si needs permission when it wants to enter critical section (CS).
8. In Lamport's algorithm, ∀i: 1 ≤ i ≤ N :: Ri= {S1, S2,...,SN}.
9. Every site Si keeps a queue, request_queue i, which contains mutual exclusion requests ordered by their timestamps.
10. This algorithm requires messages to be delivered in the FIFO order between every pair of sites.
11. Each process maintains request queue.
12. Queue contains mutual exclusion requests.

Requesting the critical section:

1. Process Pi sends request (i, Ti) to all nodes.
2. It then places request on its own queue.
3. When a process Pj receives a request, it returns a timestamped **ack.**

Entering the critical section:

1. Pi received a message (**ack** or release) from every other process with a timestamp larger than Ti.
2. Pi's request has the earliest timestamp in its queue

Releasing the critical section:

1. It removes the request from its own queue.
2. It than send a timestamped release message.
3. When a process receives a release message:
    a. It removes the request for that process from its queue.

b. This may cause its own entry have the earliest timestamp in the queue, enabling it to access the critical section.

Q10. Raymond's Tree based algorithm

Ans:

RAYMOND'S TREE BASED ALGORITHM:

1. Raymond's Tree Based Algorithm is a **token based algorithm**.
2. In Raymond's Tree Based Algorithm, a **directed tree is formed**.
3. Nodes are arranged as a tree.
4. Every node has a parent pointer.
5. Each node has a **FIFO queue** of requests.
6. There is one token with the root and the owner of the token can enter the critical section.
7. Figure 3.9 shows the example of Raymond's Tree Based Algorithm.

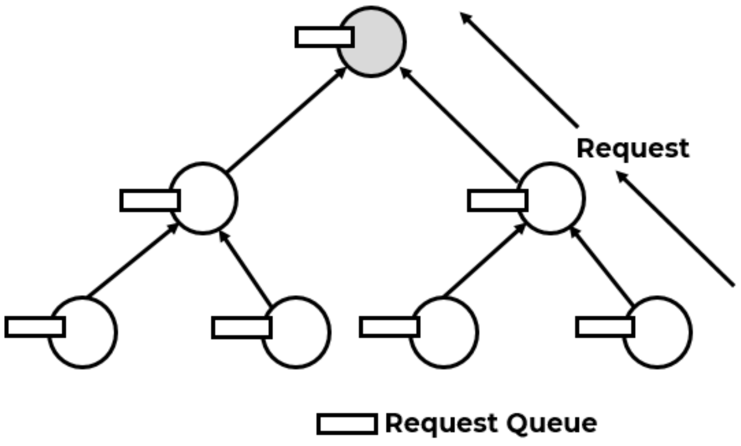


Figure 3.9: Raymond's Tree Based Algorithm.

8. As the token moves across nodes, the parent pointers change.
9. They always point towards the holder of the token as shown in figure 3.10.
10. It is thus possible to reach the token by following parent pointers.

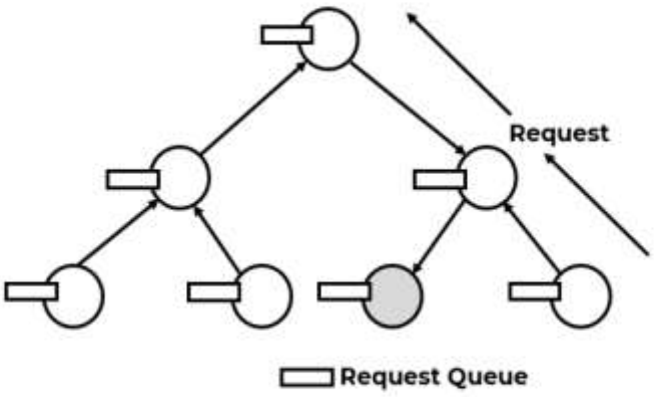


Figure 3.10: Raymond's Tree Based Algorithm.

Requesting a Token:

1. The node adds "self" in its request queue.
2. Forwards the request to the parent.

3. The parents adds the request to its request queue.
4. If the parent does not hold the token and it has not sent any requests to get the token, it sends a request to its parent for the request.
5. This process continues till we reach the root (holder of the token).

Releasing a Token:

1. Ultimately a request will reach the token holder.
2. The token holder will wait till it is done with the critical section.
3. It will forward the token to the node at the head of its request queue.
   a. It removes the entry.
   b. It updates its parent pointer.
4. Any subsequent node will do the following:
   c. Dequeue the head of the queue.
   d. If "self" was at the head of its request queue, then it will enter the critical section.
   e. Otherwise, it forwards the token to the dequeued entry.
5. After forwarding the entry, a process needs to make a fresh request for the token, if it has outstanding entries in its request queue.

Q11. Write a Suzuki-Kasami's Broadcast Algorithm. Explain with example.

Ans:

SUZUKI-KASAMI'S BROADCAST ALGORITHM:

1. Suzuki-Kasami's Broadcast Algorithm is a **token based algorithm**.
2. It is used for **achieving mutual exclusion** in distributed systems.
3. This is a modification to **Ricart–Agrawala algorithm**.
4. In this algorithm a **REQUEST and REPLY** message are used for attaining the critical section.
5. The process holding the token is the only process able to enter its critical section.
6. Each process maintains an array request.

   Req [j] denotes the sequence no of the **latest request** from process j
7. Additionally, the holder of the token maintains an array last.

   Last [j] denotes the sequence number of the latest visit to Critical Section for process j.
8. Figure 3.11 shows example of Suzuki-Kasami's Broadcast Algorithm.

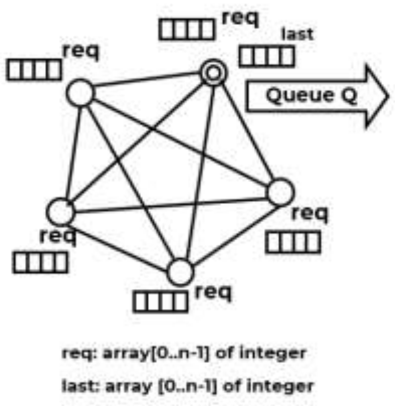

req
last
Queue Q
req
req
req
req
req: array[0..n-1] of integer
last: array [0..n-1] of integer

Figure 3.11: Example of Suzuki-Kasami's Broadcast Algorithm.

ALGORITHM:

Requesting the critical section (CS):

1. If the site does not have the token, then it increases its sequence number Req_i [i] and sends a request (i, sn) message to all other sites (sn= Req_i [i])
2. When a site S_j receives this message, it sets Req_j [i] to Max (Req_j [i], sn). If S_j has the idle token, then it sends the token to Si, if Req_j [i] = Last [i] + 1

Releasing the CS:

1. When done with the CS, site S_i sets Last [i] = Req_i [i]
2. For every site S_j whose ID is not in the token queue, it appends its ID to the token queue if Req_i [j] =Last [j] + 1
3. If the queue is not empty, it extracts the ID at the head of the queue and sends the token to that site.

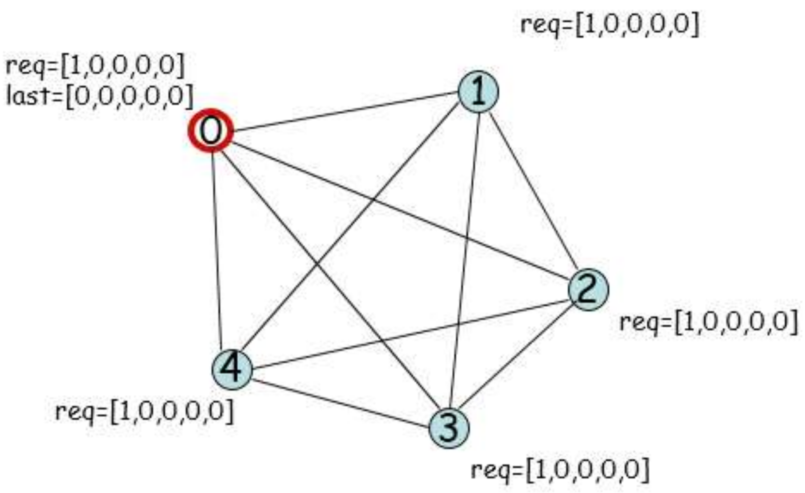
Executing the CS:

1. Site Si executes the CS when it has received the token

EXAMPLE:

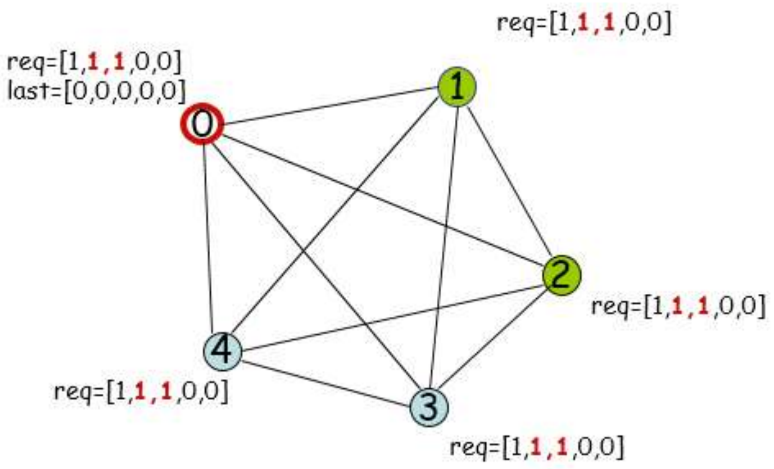
Step 1:

Initial state: Process 0 has sent a request to all, and grabbed the token.



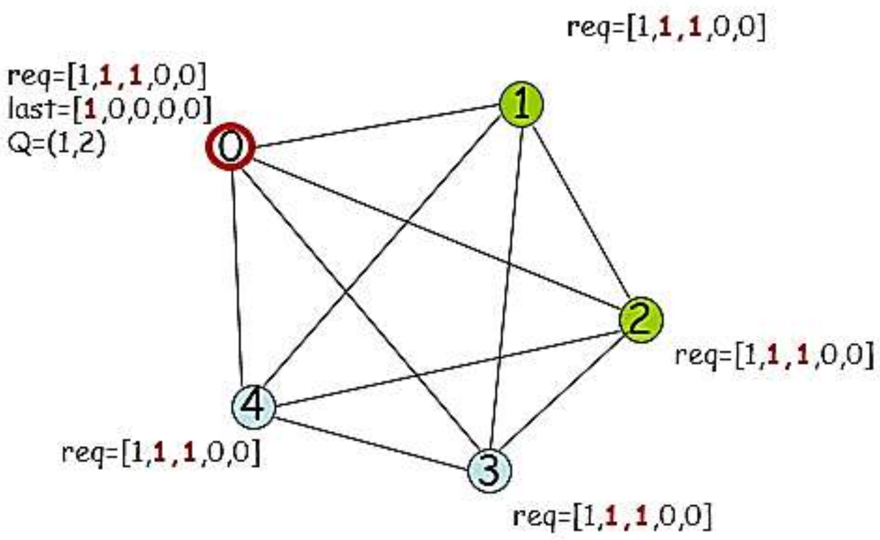
Step 2:

Now 1 & 2 send requests to enter CS



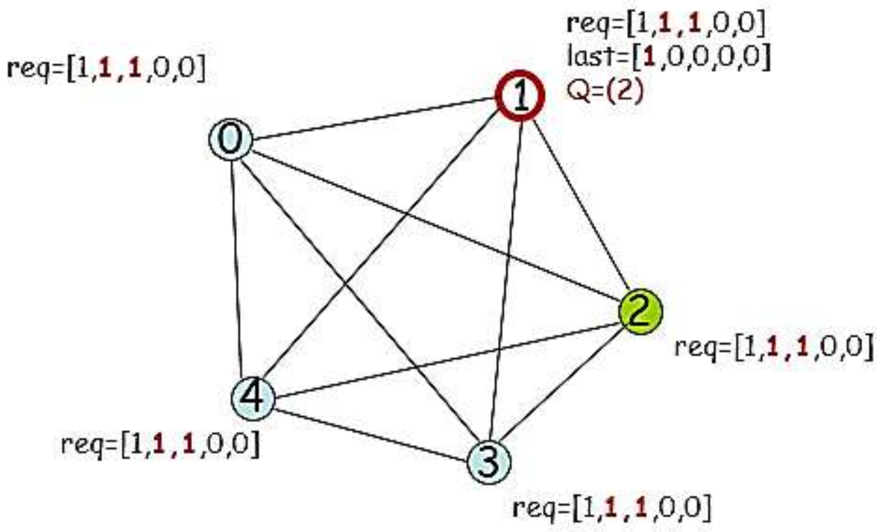
Step 3:

Now 0 prepares to exit CS



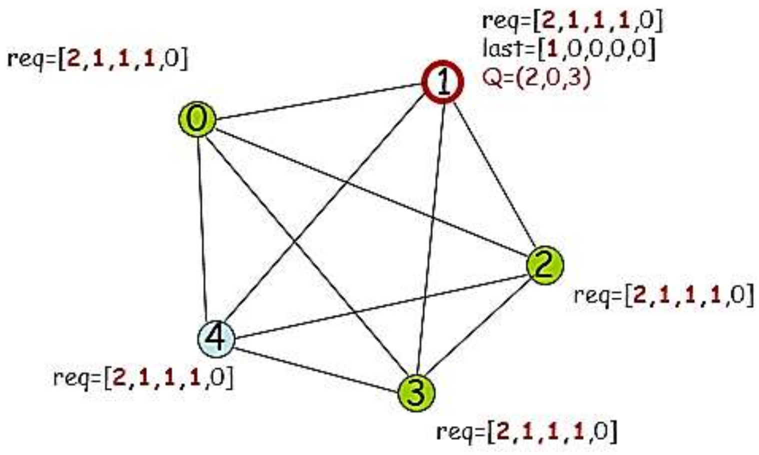
Step 4:

Now 0 passes token (Q and last) to 1



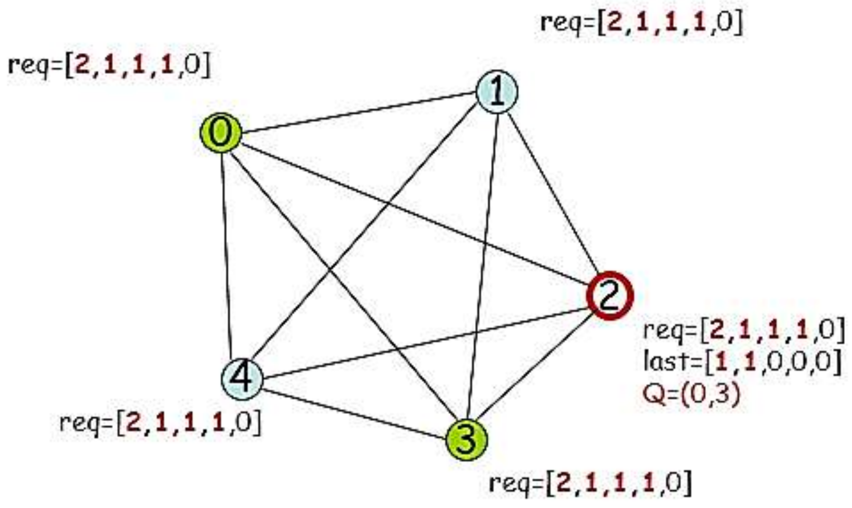
Step 5:

Now 0 and 3 send requests



Step 6:

Finally 1 sends token to 2.



Q12. Compare and contrast mutual exclusion algorithms

Ans:

Table 3.1: Comparison between mutual exclusion algorithms

Parameters	Centralized	Distributed	Token ring
Election	One process is elected as coordinator.	Total ordering of all events in the system.	Uses token for entering critical section.
Message per entry/exit	Requires three messages to enter and exit a critical region.	Requires 2(n-1) messages.	Variable number of messages required.
Delay in message times	Delay for messages is 2 messages.	Delay for messages is 2(n-1).	The time varies from 0 to n-1 tokens.
Mutual exclusion	Guarantees mutual exclusion.	Mutual exclusion guaranteed without deadlock.	Mutual exclusion is guaranteed.
Starvation	No starvation.	No starvation.	No starvation.
Complexity	Easy to implement	Complicated process.	Implementation is easy.
Used for	Used for general allocation.	Used for small group processes that does not change group membership.	Used processes in ring configuration.
Problems	Entire system can go down due to single point of failure, bottleneck.	N points of failure.	Detecting the lost token and regeneration is difficult.
Expenses	Less	More	Less
Robustness	More	Less	more

CHAP - 4: RESOURCE AND PROCESS MANAGEMENT

Q1. Explain resource management in distributed system.

Ans:

RESOURCE MANAGEMENT:

1. Resource management is the efficient and effective deployment of an organization's resources when they are needed.
2. Such resources may include financial resources, inventory, human skills, production resources, or information technology.
3. A resource manager schedules the processes in a distributed system.
4. This is done to make use of the system resources in such a manner that resource usage, response time, network congestion are optimized.
5. The techniques are:

I) Task Assignment Approach:

Each process submitted by a user for processing is viewed as a collection of related tasks and these tasks are scheduled to suitable nodes so as to improve performance.

Task assignment approach should fulfill 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.

II) Load Balancing Approach:

All the processes submitted by the users are distributed among the nodes of the system so as to equalize the workload among the nodes.

III) Load Sharing Approach:

Make sure that no node is idle and share the resources accordingly.

Q2. State the desirable features of global scheduling algorithm.

Ans:

FEATURES OF GLOBAL SCHEDULING ALGORITHM:

I) No a priori knowledge about the processes:

1. Most of the time, the prior knowledge of any process is not available.
2. Thus a good scheduling algorithm must be capable of working without any knowledge of the characteristics of processes.

II) Dynamic in nature:

1. In distributed system the load assigned to any node may change dynamically.
2. Thus the scheduling algorithm should be dynamic in nature.

III) Quick decision making capability:

1. In distributed system, the situation keeps on changing.

2. Thus scheduling algorithm must be capable of making quick decisions about the allocation by analyzing the situation.

IV) Balanced system performance and scheduling overhead:

1. Many global scheduling algorithms use the global state information to make process assignment decisions.
2. Thus the scheduling algorithm should provide a near optimal system performance with a minimum of global state information gathering overhead.

V) Stability:

1. Unstable process migration results in a process thrashing.
2. So the scheduling algorithm should have stable process migration.

VI) Scalability:

1. A scheduling algorithm must be capable of handling a network with any size.
2. Thus the scheduling algorithm should be scalable to adopt any change.

VII) Fault Tolerance:

1. In a distributed system, when the node crashes, it should not affect the performance of the overall system.
2. The scheduling algorithm should be able to handle such situation of fault tolerance.

VIII) Fairness of Service:

1. In a distributed system, there is chance that two users may initiate equivalent processes simultaneously.
2. The scheduling algorithm should take care of such fair assignment of services.

Q3. Explain designing issues in load balancing approach

Ans:

LOAD BALANCING APPROACH:

1. Load Balancing is used in **Process Management**.
2. Scheduling Algorithm that uses Load Balancing Approach are called as **Load Balancing Algorithms**.
3. The main goal of load balancing algorithms is to **balance the workload** on all the nodes of the system.
4. Load balancing aims to:
   a. Optimize resource use.
   b. Maximize throughput.
   c. Minimize response time.
   d. Avoid overload of any single resource.

DESIGNING ISSUES:

Designing a load balancing algorithm is a critical task. It includes

I) Load Estimation Policy:

1. The first issue in a load balancing algorithm is to decide which method to use to estimate the workload of a particular node.

2. Estimation of the workload of a particular node is a difficult problem for which no completely satisfactory solution exists.
3. A nodes workload can be estimated based on some measurable parameters below:
    a. Total number of processes on the node.
    b. Resource demands of these processes.
    c. Instruction mixes of these processes.
    d. Architecture and speed of the node's processor.

II) Process Transfer Policy:

1. The idea of using this policy is to transfer processes from heavily loaded nodes to lightly loaded nodes.
2. Most of the algorithms use the threshold policy to decide whether the node is lightly loaded or heavily loaded.
3. Threshold value is a limiting value of the workload of node which can be determined by:
    a. **Static Policy:** Each node has a predefined threshold value.
    b. **Dynamic Policy:** The threshold value for a node is dynamically decided.
4. Below threshold value, node accepts processes to execute.
5. Above threshold value node tries to transfer processes to a lightly loaded node.
6. Single threshold policy may lead to unstable algorithm.
7. To reduce instability double threshold policy has been proposed which is also known as high low policy.
8. Figure 4.1 (a) shows single threshold policy and (b) shows double threshold policy.

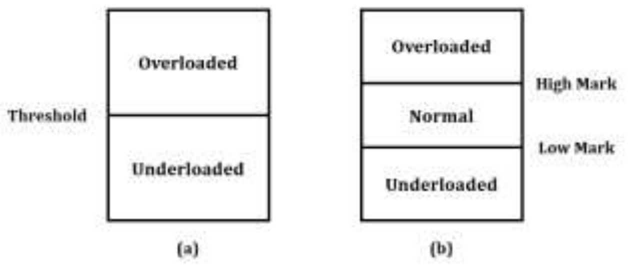


Figure 4.1: Threshold Policy.

III) Location Policy:

1. When a transfer policy decides that a process is to be transferred from one node to any other lightly loaded node, the challenge is to find the destination node.
2. Location policy determines this destination node.
3. It includes following:
    a. **Threshold:** This policy selects a random node and checks whether the node is able to receive the process then it transfers the process. If the node rejects the transfer then another node is selected randomly. This continues until the probe limit is reached.
    b. **Shortest method:** In this 'm' distinct nodes are chosen at random and each is polled to determine its load with minimum value.
    c. **Bidding method:** In this method, nodes contain **managers** to send processes and **contractors** to receive processes.
    d. **Pairing:** It is used to reduce the variance of load only between nodes of the system.

IV) State Information Exchange Policy:

1. It is used for frequent exchange of state information within the nodes.
2. It includes:
   a. **Periodic Broadcast:** Each node broadcasts its state information after the elapse of T units of time.
   b. **Broadcast when state changes:** Each node broadcasts its state information only when a process arrives or departures.
   c. **On demand exchanges:** Each node broadcasts its state information request message when its state switches from normal to either under load or over load.
   d. **Exchanges by polling:** The partner node is searched by polling the other nodes on by one until poll limit is reached.

V) Priority Assignment Policy:

1. The priority of the execution of local and remote processes at a particular node should be planned.
2. It includes:
   a. **Selfish priority assignment rule:** In this local process are given higher priority than remote process.
   b. **Altruistic priority assignment rule:** Remote processes are given higher priority than local process.
   c. **Intermediate priority assignment rule:** Priority in this rule is decided depending upon the number of local and remote processes on a node.

VI) Migration Limiting Policy:

1. This policy determines the total number of times a process can migrate from one node to another.
2. It includes:
   a. **Uncontrolled Policy:** The remote process is treated as local process. Hence, it can be migrated any number of times.
   b. **Controlled Policy:** Migration count is used for remote processes.

Q4. Explain Load balancing approach in distributed system.

Ans:

LOAD BALANCING APPROACH:

1. Load Balancing is used in **Process Management**.
2. Scheduling Algorithm that uses Load Balancing Approach are called as **Load Balancing Algorithms**.
3. The main goal of load balancing algorithms is to **balance the workload** on all the nodes of the system.
4. Load balancing aims to:
   a. Optimize resource use.
   b. Maximize throughput.
   c. Minimize response time.
   d. Avoid overload of any single resource.
5. In this, the processes are distributed among nodes to equalize the load among all nodes.
6. Load balancing algorithm tries to balance the total system load by transparently transferring the workload from heavily loaded nodes to lightly loaded nodes in an attempt to ensure good overall performance relative to some specific metric of system performance.

7. We can have the following categories of load balancing algorithms as shown in figure 4.2.

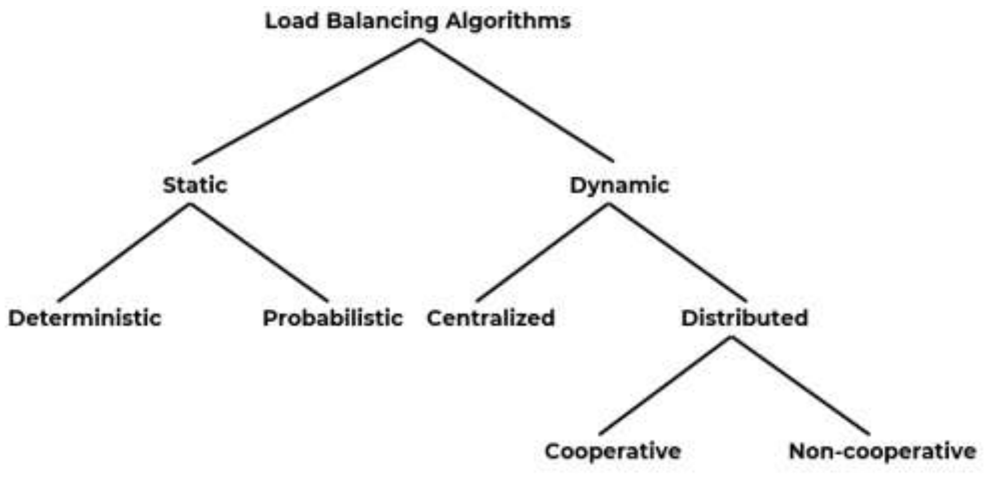


Figure 4.2: Taxonomy of Load Balancing Algorithm.

I) Static:

1. Static Algorithm ignores the current state of the system.
2. Example: If a node is heavily loaded, it picks up a task randomly and transfers it to a random node.
3. These algorithms are simpler to implement but performance may not be good.
   - **a. Deterministic:** Algorithms in this class use the processor and process characteristics to allocate processes to nodes.
   - **b. Probabilistic:** Algorithms in this class use information regarding static attributes of the system such as number of nodes, processing capability, etc.

II) Dynamic:

1. Use the current state information for load balancing.
2. There is an overhead involved in collecting state information periodically; they perform better than static algorithms.
   - **a. Centralized:**
     - System state information is collected by a single node.
     - This node makes all scheduling decisions.
   - **b. Distributed:**
     - Most desired approach.
     - Each node is equally responsible for making scheduling decisions based on the local state and the state information received from other sites.
     - **Cooperative:**
       - In these algorithms, the distributed entities cooperate with each other to make scheduling decisions.
       - Therefore they are more complex and involve larger overhead than non-cooperative ones.
       - But the stability of a cooperative algorithm is better than of a non-cooperative one.
     - **Non-Cooperative:**
       - A distributed dynamic scheduling algorithm.
       - In these algorithms, individual entities act as autonomous entities and make scheduling decisions independently of the action of other entities.

Q5. Explain Load sharing approach in distributed system.

Ans:

LOAD SHARING:

1. In a distributed system, proper utilization of resources is not required to balance load on all nodes.
2. But it is necessary and sufficient to prevent the nodes from being idle, this is known as Load Sharing.
3. For the proper utilization of resources of a distributed system, it is not required to balance the load on all the nodes.
4. It is necessary and sufficient to prevent the nodes from being idle while some other nodes have more than two processes.
5. This rectification is called the **Dynamic Load Sharing** instead of **Dynamic Load Balancing**.
6. So instead of load balancing, load sharing is better.
7. The design of a load sharing algorithms require that proper decisions be made regarding load estimation policy, process transfer policy, state information exchange policy, priority assignment policy, and migration limiting policy.
8. It is simpler to decide about most of these policies in case of load sharing, because load sharing algorithms do not attempt to balance the average workload of all the nodes of the system.
9. Rather, they only attempt to ensure that no node is idle when a node is heavily loaded.
10. The priority assignments policies and the migration limiting policies for load-sharing algorithms are the same as that of load-balancing algorithms.

ISSUES:

I) Load Estimation:

1. Load sharing algorithms attempt to avoid nodes from being idle.
2. But it is sufficient to know whether a node is busy or idle.
3. Thus these algorithms normally employ the simplest load estimation policy of counting the total number of processes on a node.

II) Process Transfer Policies:

1. Load sharing algorithms normally use all-or-nothing strategy.
2. This strategy uses the threshold value of all the nodes fixed at one.
3. A node becomes a receiver node when it has no process, and becomes a sender node when it has more than one process.
4. To avoid processing power on nodes having zero process, load sharing algorithms uses two threshold values instead of one.

III) Location policies:

1. It decides whether the sender node or the receiver node of the process takes the initiative to search for suitable node in the system.
2. It includes:

   a. Sender Initiated Location Policy:

   - In this policy heavily loaded nodes search for lightly loaded nodes.

- When the nodes 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.

b. **Receiver Initiated Location Policy:**

- In this policy lightly loaded nodes search for heavily loaded nodes.
- When the nodes becomes under loaded, it either broadcasts or randomly probes the other nodes one by one to find a node that is able to receive remote processes.

IV) State information exchange policies:

1. In load sharing algorithms it is not necessary for the nodes to periodically exchange state information.
2. But needs to know the state of the other nodes when it is either under loaded or overloaded.
3. It includes:

   a. **Broadcast when the state changes:**

   - In sender initiated/receiver initiated location policy a node broadcasts state information request when it becomes overloaded/under loaded.
   - It is called broadcast-when-idle policy when receiver-initiated policy is used with fixed threshold value of one.

   b. **Poll when the state changes:**

   - In large networks polling mechanism is used.
   - Polling mechanism randomly asks different nodes for state information until find an appropriate one or probe limit is reached.
   - It is called pool-when-idle policy when receiver-initiated policy is used with fixed threshold value of one.

Q6. Explain Task Assignment and Job Scheduling

Ans:

TASK ASSIGNMENT:

1. In task assignment, each process is viewed as a collection of tasks.
2. These tasks are scheduled to suitable processor to improve performance.
3. This is not a widely used approach because:

   a. It requires characteristics of all the processes to be known in advance.

   b. This approach does not take into consideration the dynamically changing state of the system.
4. In this approach, a process is considered to be composed of multiple tasks and the goal is to find an optimal assignment policy for the tasks of an individual process.
5. The following are typical assumptions for the task assignment approach:

   a. Minimize IPC cost (this problem can be modeled using network flow model)

   b. Efficient resource utilization.

   c. Quick turnaround time.

   d. A high degree of parallelism

JOB SCHEDULING:

1. Job scheduling is the process of allocating system resources to many different tasks.

2. The system handles prioritized job queues that are awaiting CPU time and it should determine which job to be taken from which queue and the amount of time to be allocated for the job.
3. This type of scheduling makes sure that all jobs are carried out fairly and on time.
4. Most OSs like Unix, Windows, etc., include standard job-scheduling abilities.
5. Job scheduling is performed using **job schedulers.**
6. Job schedulers are programs that enable scheduling and, at times, track computer "batch" jobs, or units of work like the operation of a payroll program.
7. Job schedulers have the ability to start and control jobs automatically by running prepared job-control-language statements or by means of similar communication with a human operator.
8. In scheduling, many different schemes are used to determine which specific job to run.
9. Some parameters that may be considered are as follows:
   a. Job priority.
   b. Availability of computing resource.
   c. Execution time assigned to the user.
   d. Number of parallel jobs permitted for a user.
   e. Projected execution time.
   f. Elapsed execution time.
   g. Number of cases of prescribed events

Q7. What is the need for process migration and explain the role of resource to process and process to resource binding in process migration

Ans:

PROCESS MIGRATION:

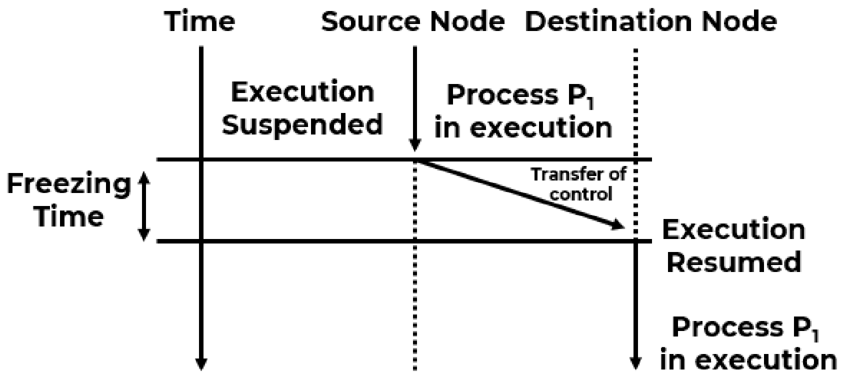


Figure 4.3: Process Migration.

1. To Move processes from one computing environment to another, process migration is required.
2. Process migration is a **specialized form of process management**.
3. The most common application of process migration is in computer clusters where processes are moved from machine to machine.
4. Process migration is the relocation of a process from its source node to another destination node.
5. The way of a process migrates from one node to another is shown in the following figure 4.3.

6. A process can either be migrated before it starts execution on its source node which is called as non-preemptive process or during the course of its execution that is known as preemptive process migration.
7. Preemptive process migration is more costly compared to non-preemptive because the process environment must accompany the process to its new node.
8. **Steps:**
   a. Process is selected for migration.
   b. Selection of destination node for the process to be migrated.
   c. Transfer of selected process to the destination node.
9. Migration policy is responsible for first two steps while third step is handles by migration mechanism.
10. Migration of a process is complex that involves handling various activities to meet the requirements for a good process migration.
11. **The sub activities involved are:**
    a. Freezing the process on its source node and restarting it on its destination node.
    b. Transferring the process's address space from its source node to its destination node.
    c. Forwarding messages meant for the migrant process.
    d. Handling communication between cooperating processes that have been separated as a result of process migration.
12. A preemptive process migration facility allows the transfer of an executing process from one node to another.
13. On the other hand, in system supporting only non-preemptive migration facility, a process can only be transferred prior to beginning its execution.
14. Preemptive process migration is costlier than non-preemptive process migration since the process state, which must accompany the process to its new node, becomes much more complex after execution begins.

FEATURES OF PROCESS MIGRATION:

1. Transparency.
2. Minimal Interference.
3. Efficiency.
4. Robustness.

Q8. Compare Load sharing to task assignment and Load balancing strategies for scheduling processes in a distributed system.

Ans:

COMPARISON BETWEEN LOAD SHARING AND LOAD BALANCING:

Table 4.1: Comparison between Load Sharing and Load Balancing.

Load Sharing	Load Balancing
Load sharing is the ability to distribute outgoing traffic over multiple paths	Load balancing is the ability to split the load toward the same destination over multiple paths.
Traffic is only shared across different links.	Traffic is balanced across different links.
Load Sharing is more efficient.	Load Balancing is less efficient as compared to load sharing.
Proper utilization of resources is required.	Proper utilization of resources is not required.
It is necessary to prevent the nodes from being idle.	It is not necessary to prevent the nodes from being idle.
It does not uses round robin fashion.	It uses round robin fashion.
It cannot performed on packet-by-packet basis.	It can performed on packet-by-packet basis.
It is a static concept.	It is a dynamic concept.
It is difficult to setup and maintain.	It is easy to setup and maintain.
Example of Protocol: eBGP.	**Example of Protocol:** IGP.

Q9. Code Migration

Q10. Describe code migration issues in detail

Ans:

CODE MIGRATION:

1. In process migration, an entire process has to be moved from one machine to another.
2. But this may be a **crucial task**, but it has good overall performance.
3. In some cases, a code needs to be migrated rather than the process.
4. Code Migration refers to transfer of program from one node to another.
5. For **example**: Consider a client server system, in which the server has to manage a big database.
6. The client application has to perform many database operations.
7. In such situation, it is better to move part of the client application to the server and the result is send across the network.
8. Thus code migration is a better option.
9. Code Migration is used to improve overall performance of the system by exploiting parallelism.
10. Example of code migration is shown below in figure 4.4.
11. Here the server is responsible to provide the client's implementation, when the client binds to the server.
12. The advantage of this approach is that, the client does not need to install all the required software. The software can be moved in as when necessary and discarded when it is not needed.

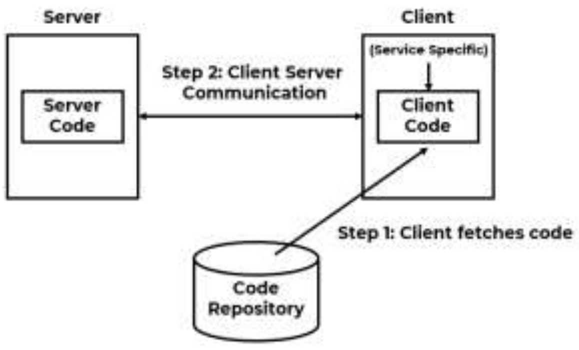


Figure 4.4: Example of Code Migration.

CODE MIGRATION ISSUES:

1. Communication in distributed systems is concerned with exchanging data between processes.
2. Code migration in the broadest sense which deals with moving programs between machines, with the intention to have those programs be executed at the target.
3. In code migration framework, a process consists of 3 segments i.e. **code segment, resource segment and execution segment**.
4. Code segment contains the actual code.
5. Resource segment contains references to resources needed by the process.
6. Execution segment stores the execution state of a process.
7. Consider the figure 4.5 of code migration.
8. Where,
    a. CS: Client-Server,
    b. REV: Remote evaluation.
    c. CoD: Code-on-demand.
    d. MA: Mobile agents.
9. Mobile agents moves from site to site.
10. Code-on-demand is used to migrate part of server to client.
11. Remote evaluation is used to perform database operations involving large amount of data.

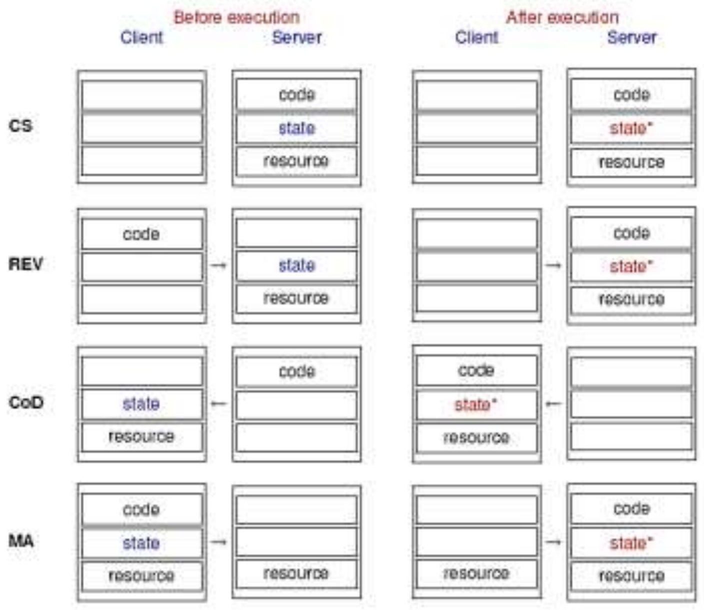


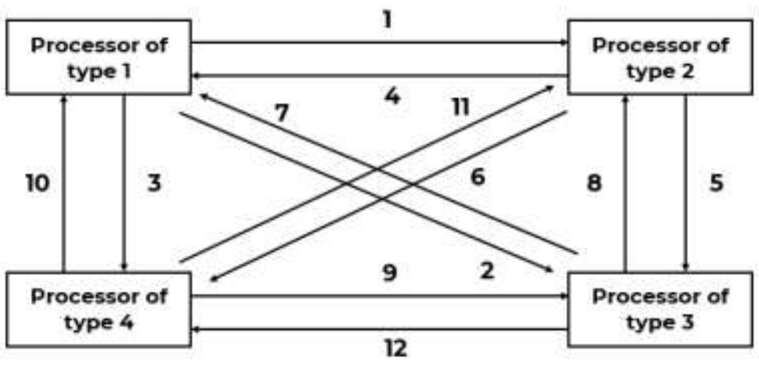
Figure 4.5: Code Migration.

Q11. Explain process migration in heterogeneous system

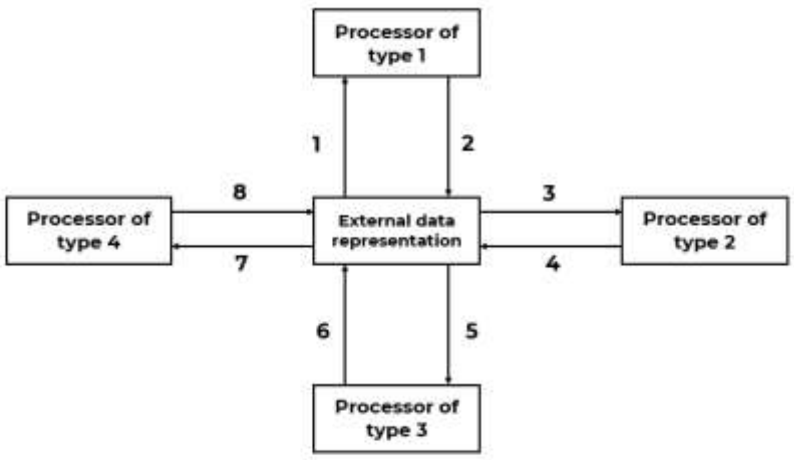
Ans:

PROCESS MIGRATION IN HETEROGENEOUS SYSTEM:

1. In homogeneous environment, for process migration the interpretation of data is consistent on the source and destination nodes.
2. No data translation is needed.
3. But in heterogeneous environment data needs to be translated from the source CPU format to the destination CPU format before it can be executed.
4. If a system consists of two CPU types then each processor should be able to convert foreign data its own understandable format.
5. If a third CPU is added then there is another overhead of converting the data format.
6. Hence a heterogeneous system that contains of 'n' CPU types should have n (n-1) pieces of translation software so that the translation can be done for all nodes.
7. The figure below explains the overhead due to translation of data during process migration.
8. Processor 1 has to convert 3 times if it sends to any of the other process and converts to its own format another 3 times.



9. To reduce software complexity external data representation mechanism can be used.
10. In this mechanism a standard representation is used for transport of data and each process needs to be converted to and from the standard form.



11. Issues during the design of external data representation are signed infinity and signed zero representation.

Q12. Explain threads

Ans:

THREADS:

1. Thread is considered as light weight process.
2. Thread is flow of control through an address space.
3. Each address space can have multiple concurrent control flows.
4. Each thread has access to entire address space.

MOTIVATION FOR USING THREADS:

1. Creating threads is easier than multiple processes.
2. Switching between threads is easier than switching than processes.
3. Allows parallelism.
4. Resources are better.

MODELS:

I) Dispatcher Workers Model:

1. This model is used in designing a server process.
2. A process consists of a single dispatcher thread and multiple worker threads.
3. The dispatcher thread accepts client requests, examines them then dispatches the request to one of the free worker thread for processing.
4. Each worker thread works on a different client request hence multiple client requests can be processes in parallel.

Example:

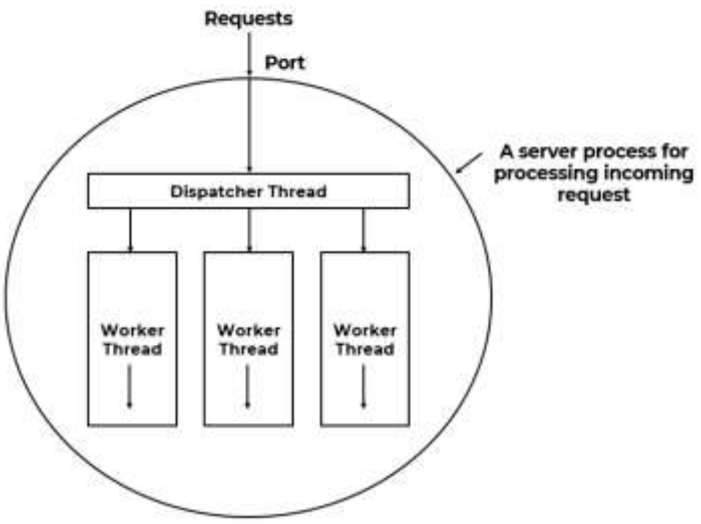


Figure 4.6: Dispatcher Worker Model

II) Team Model:

1. All threads in this model behave such that there is no dispatcher-worker relationship for processing requests.
2. Reach threads gets and processes client requests on its own.
3. This model is mostly used for implementing special threads within a process where each thread of the process is specialized to service a specific request.

4. Multiple types of requests can be simultaneously handles by the process.

Example:

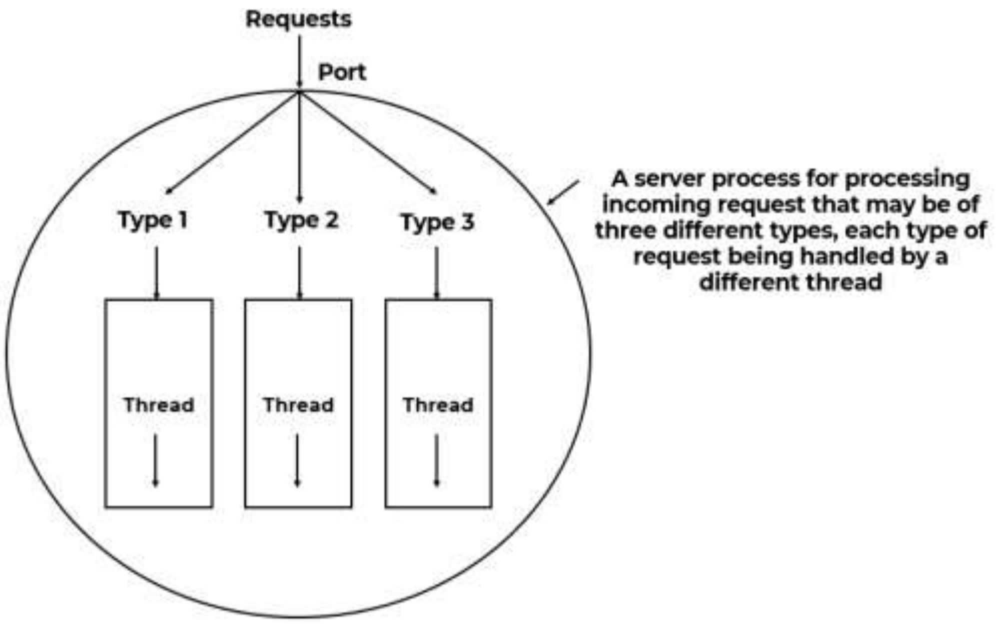

Requests
Port
Type 1
Type 2
Type 3
Thread
Thread
Thread
A server process for processing incoming request that may be of three different types, each type of request being handled by a different thread

Figure 4.7: Team Model

III) Pipeline Model:

1. Applications based on producer consumer model uses this type.
2. The output generated by one part of the application is used as input to another part of the application.
3. The threads in this model are organized as a process by the second thread and the output by second thread is used for processing the third thread and so on.
4. The final output of the last thread in the pipeline is the final output of the process to which the thread belongs.

Example:

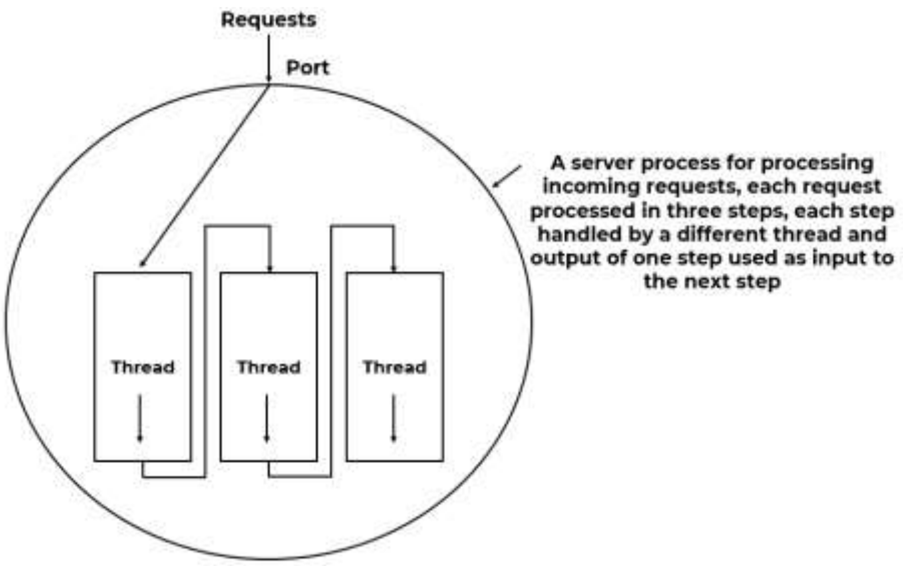

Requests
Port
Thread
Thread
Thread
A server process for processing incoming requests, each request processed in three steps, each step handled by a different thread and output of one step used as input to the next step

Figure 4.8: Pipeline Model

Q13. Compare Process and threads

Ans:

DIFFERENCE BETWEEN PROCESS AND THREADS:

Table 4.2: Comparison between Process and threads

Parameters	Process	Threads
Definition	In traditional operating system, the basic unit of CPU utilization is a process.	In operating system with facility, the basic unit of CPU utilization is a thread.
Parameters owned	Each process has its own program counter, its own register states, its own stack and its own address space.	Each thread of the process has its own program counter, its own register states, its own stack
Protection	Protection is needed.	Protection is not needed.
Process type	Heavyweight processes.	Light weighted processes.
Resources sharing	Less efficient.	More efficient.
Switching cost	Switching between processes is costlier.	Switching between threads is cheaper.
Overhead	Creating a new process causes more overhead.	Creating a new thread causes less overhead then for processes.

CHAP - 5: CONSISTENCY, REPLICATION AND FAULT TOLERANCE

Q1. Explain the need of client centric consistency models as compared to data centric consistency model. Explain any two client centric consistency model.

Q2. Discuss and differentiate various client-centric consistency models

Ans:

NEED OF CLIENT CENTRIC CONSISTENCY MODELS:

1. It is one of the type of **consistency model** in distributed system.
2. System wide consistency is hard, so usually client centric consistency model is more preferable.
3. It is **easier to deal with inconsistencies**.
4. Using client centric consistency model many inconsistencies can be **hidden in cheap way**.
5. It aims at providing a **consistent view on a database.**

CLIENT CENTRIC CONSISTENCY MODELS:

I) Eventual Consistency:

1. An eventual consistency is a **weak consistency model**.
2. It lacks in simultaneous updates.
3. It defines that if updates do not occur for a long period of time, all replicas will gradually become consistent
4. Eventual Consistency is a lot inexpensive to implement.
5. Requirements for Eventual Consistency are:
    a. Few read/write conflicts.
    b. No write/write conflicts.
    c. Clients can accept temporary inconsistency.
6. Example: WWW.

II) Monotonic Reads:

1. A data store is said to offer monotonic read consistency if the following condition holds:
   "If process P reads the value of data item x, any successive read operation on x by that process will always return the same value or a more recent one."
2. Consider a Distributed e-mail database.
3. It has distributed and replicated user-mailboxes.
4. Emails can be inserted at any location.
5. However, updates are propagated in a lazy (i.e. on demand) fashion.
6. Suppose the end user reads his mail in Mumbai. Assume that only reading mail does not affect the mailbox.
7. Later when the end user moves to Pune and opens his mail box again, monotonic read consistency assurances that the messages that were in the mailbox in Mumbai will also be in the mailbox when it is opened in Pune.
8. Figure 5.1 shows the read operations performed by a single process 'P' at two different local copies of the same data store.

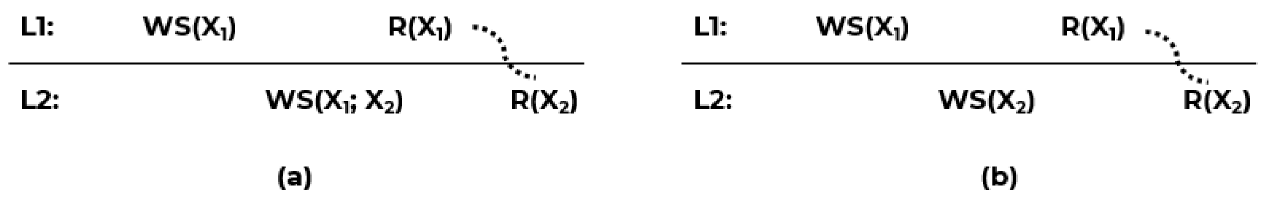


Figure 5.1

9. Figure 5.1 (a) Demonstrate a monotonic-read consistent data store.
10. Figure 5.2 (b) Demonstrate a data store that does not provide monotonic reads.

III) Monotonic Writes:

1. A data store is said to offer monotonic write consistency if the following condition holds:

   "A write operation by process P on data item x is completed before any successive write operation on x by the same process P can take place".
2. Consider a software library example.
3. In several cases, updating a library is accomplished by replacing one or more functions.
4. With monotonic write consistency, assurance is given that if an update is performed on a copy of the library, all previous or earlier updates will be accomplished primarily.
5. Figure 5.2 shows the write operations performed by a single process 'P' at two different local copies of the same data store.

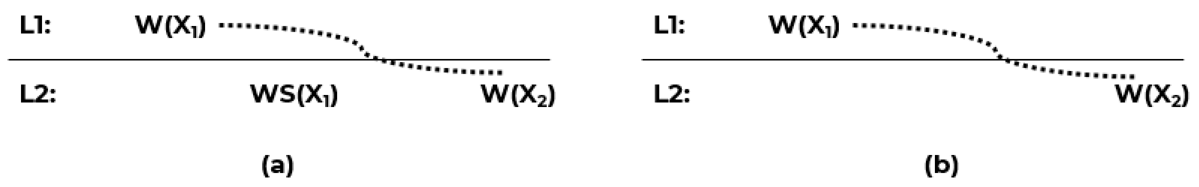


Figure 5.2

6. Figure 5.2 (a) Demonstrate a monotonic-write consistent data store.
7. Figure 5.2 (b) Demonstrate a data store that does not provide monotonic-write consistency.

IV) Read Your Writes:

1. A data store is said to offer read your write consistency if the following condition holds:

   "The effect of a write operation by a process P on a data item x at a location L will always be seen by a successive read operation by the same process."
2. **Example:** Updating a Web page and guaranteeing that the Web browser shows the newest version instead of its cached copy.
3. Figure 5.3 (a) Demonstrate a data store that delivers read your writes consistency.
4. Figure 5.3 (b) Demonstrate a data store that does not delivers read your writes consistency.

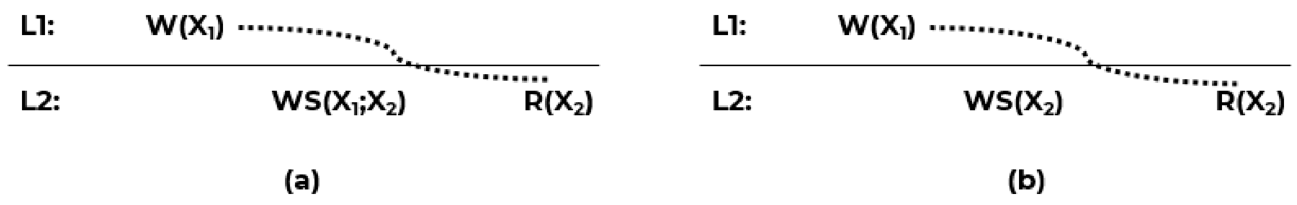


Figure 5.3

V) Writes Follow Reads:

1. A data store is said to offer write follow read consistency if the following condition holds:

"A write operation by a process P on a data item x following a previous read by the same process, is guaranteed to take place on the same or even a more recent value of x, than the one having been read before."

2. **Example:** Suppose a user first reads an article A then posts a response B. By requiring writes-follow-reads consistency, B will be written to any copy only after A has been written.
3. Figure 5.4 (a) Demonstrate a data store that delivers writes follow reads consistency.
4. Figure 5.4 (b) Demonstrate a data store that does not delivers writes follow reads consistency.

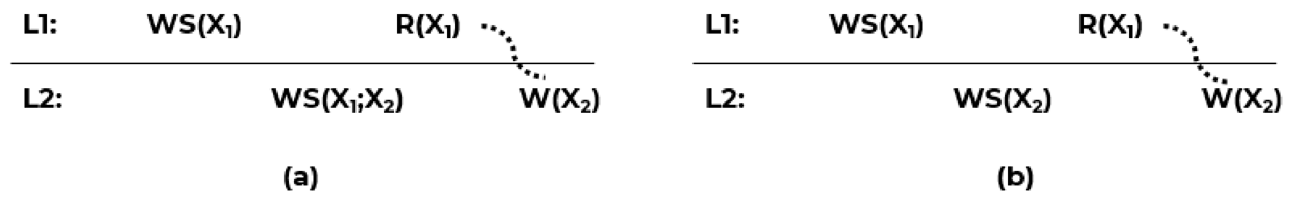


Figure 5.4

Q3. What are different data centric consistency model

Ans:

DATA CENTRIC CONSISTENCY MODEL:

1. Data-centric consistency models aim to provide system wide consistent view of the data store.
2. A data store may be physically distributed across multiple machines.
3. Each process that can access data from the store is assumed to have a local or nearby copy available of the entire store.
4. Figure 5.5 shows the example of data centric consistency model.

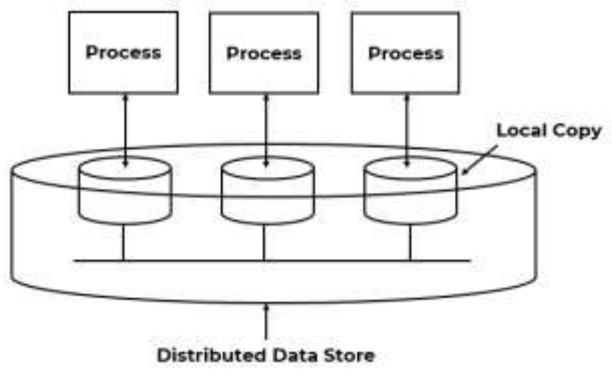
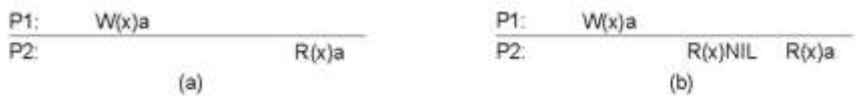


Figure 5.5: Example of Data Centric Consistency Model.

I) Strict Consistency Model:

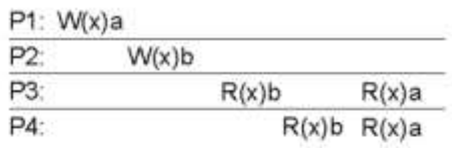
1. Any read on a data item X returns a value corresponding to the result of the most recent write on X.
2. This is the strongest form of memory coherence which has the most stringent consistency requirement.



3. Behavior of two processes, operating on the same data item.
    a. A strictly consistent store.
    b. A store that is not strictly consistent.

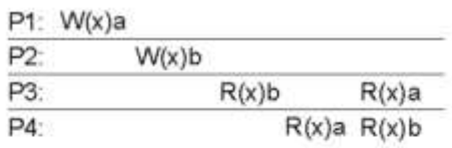
II) Sequential Consistency:

1. Sequential consistency is an important data-centric consistency model which is a slightly weaker consistency model than strict consistency.
2. A data store is said to be sequentially consistent if the result of any execution is the same as if the (read and write) operations by all processes on the data store were executed in some sequential order and the operations of each individual process should appear in this sequence in a specified order.



(a)

A sequentially consistent data store.



(b)

A data store that is not sequentially consistent.

III) Linearizability:

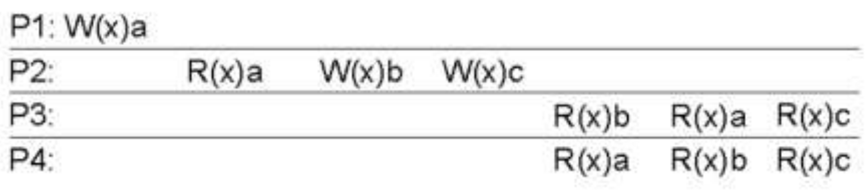
1. It that is weaker than strict consistency, but stronger than sequential consistency.
2. A data store is said to be linearizable when each operation is timestamped and the result of any execution is the same as if the (read and write) operations by all processes on the data store were executed in some sequential order.
3. The operations of each individual process appear in sequence order specified by its program.
4. If $ts_{OP1}(x) < ts_{OP2}(y)$, then operation OP1(x) should precede OP2(y) in this sequence.

IV) Causal Consistency:

1. It is a **weaker model than sequential consistency**.
2. In Casual Consistency all processes see only those memory reference operations in the correct order that are potentially causally related.
3. Memory reference operations which are not related may be seen by different processes in different order.
4. A memory reference operation is said to be casually related to another memory reference operation if the first operation is influenced by the second operation.
5. If a write (w2) operation is casually related to another write (w1) the acceptable order is (w1, w2).

V) FIFO Consistency:

1. It is **weaker than causal consistency**.
2. This model ensures that all write operations performed by a single process are seen by all other processes in the order in which they were performed like a single process in a pipeline.
3. This model is simple and easy to implement having good performance because processes are ready in the pipeline.
4. Implementation is done by sequencing write operations performed at each node independently of the operations performed on other nodes.



VI) Weak Consistency:

1. The basic idea behind the weak consistency model is enforcing consistency on a group of memory reference operations rather than individual operations.
2. A Distributed Shared Memory system that supports the weak consistency model uses a special variable called a synchronization variable which is used to synchronize memory.
3. When a process accesses a synchronization variable, the entire memory is synchronized by making visible the changes made to the memory to all other processes.

VII) Release Consistency:

1. Release consistency model tells whether a process is entering or exiting from a critical section so that the system performs either of the operations when a synchronization variable is accessed by a process.
2. Two synchronization variables acquire and release are used instead of single synchronization variable. Acquire is used when process enters critical section and release is when it exits a critical section.
3. Release consistency can be viewed as synchronization mechanism based on barriers instead of critical sections.

VIII) Entry Consistency:

1. In entry consistency every shared data item is associated with a synchronization variable.
2. In order to access consistent data, each synchronization variable must be explicitly acquired.
3. Release consistency affects all shared data but entry consistency affects only those shared data associated with a synchronization variable.

Q4. Explain the difference between Data Centric and Client Centric Consistency Models.

Ans:

DIFFERENCE BETWEEN DATA CENTRIC AND CLIENT CENTRIC CONSISTENCY:

Table 5.1: Difference between Data Centric and Client Centric Consistency.

Data Centric Consistency	Client Centric Consistency
It is used for all client in a system.	It is used for individual client.
It does not have lack of simultaneous updates.	It has lack of simultaneous updates.
It is data specific.	It is client specific.
Globally accessible.	Not globally accessible.
It aims to provide system wide consistent view on a database	It aims to provide client specific consistent view on a database.

Q5. Explain the difference between strict consistency and sequential consistency model

Ans:

DIFFERENCE BETWEEN STRICT CONSISTENCY MODEL AND SEQUENTIAL CONSISTENCY MODEL:

Table 5.2: Difference between Strict Consistency Model and Sequential Consistency Model

Strict Consistency Model	Sequential Consistency Model
It was proposed by R. Denvik.	It was proposed by Lamport.
In this the value returned by a read operation on a memory address is always the same as the value written by the most recent write operation to that address.	In this all the processes see the same order of all memory access operations on the shared memory.
Consistency requirement is stronger than sequential model.	Consistency requirement is weaker than strict model.
Implementation of the strict consistency model requires the existence of an absolute global time.	Implementation of the sequential consistency model does not require the existence of an absolute global time.
In the absence of explicit synchronization operations, running a program twice gives same results.	In the absence of explicit synchronization operations, running a program twice may not give same results.

Q6. Explain fault tolerance and file properties which influence the ability of DS to tolerate the faults

Ans:

FAULT TOLERANCE:

1. A system fails when it does not match its promises.
2. An error in a system can led to a failure.
3. The cause of an error is called a **fault**.
4. Faults are generally **transient, intermittent or permanent**.
5. The ability of the system to continue functioning in the event of partial failure is known as fault tolerance.
6. Fault tolerance is the important issue in designing a Distributed file system.
7. There are various types of fault which harms the integrity of a system.
8. For example: If the processor loses the contents of its main memory in the event of a crash it leads to logically complete but physically incomplete operations, making the data inconsistent.
9. Decay of disk storage devices may result in the loss or corruption of data stored by a file system.
10. Decay is when the portion of device is irretrievable.

FILE PROPERTIES WHICH INFLUENCE THE ABILITY OF DFS TO TOLERATE THE FAULTS ARE:

I) Availability:

1. This refers to the fraction of time for which the file is available for use.
2. This property depends on location of the clients and location of files.
3. Example: If a network is partitioned due to a communication failure, a link may be available to the clients on some nodes but may not be available to clients on other nodes.
4. Replication is the primary mechanism for improving the availability.

II) Robustness:

1. This refers to power to survive crashes of the storage files and storage decays of the storage medium on which it is stored.
2. Storage devices implemented using redundancy techniques are often used to store robust files.
3. A robust file may not be available until the faulty component has been removed.
4. Robustness is independent of either the location of the file or the location of clients.

III) Recoverability:

1. This refers to the ability to roll back to the previous stable and consistent state when an operation on a file is aborted by the client.
2. In this atomic update techniques such as transaction mechanisms are used.

Q7. Explain replication models.

Ans:

REPLICATION MODELS:

I) Master-Salve Model:

1. In this model, one of the copy is the master replica and all the other copies are slaves.
2. In this model, the functionality of the slaves are very limited, thus the configuration is very simple.
3. The slaves essentially are read-only.
4. Most of the master-slaves services ignore all the updates or modifications performed at the slave, and "undo" the update during synchronization, making the slave identical to the master.
5. Figure 5.6 shows the example of master slave model.

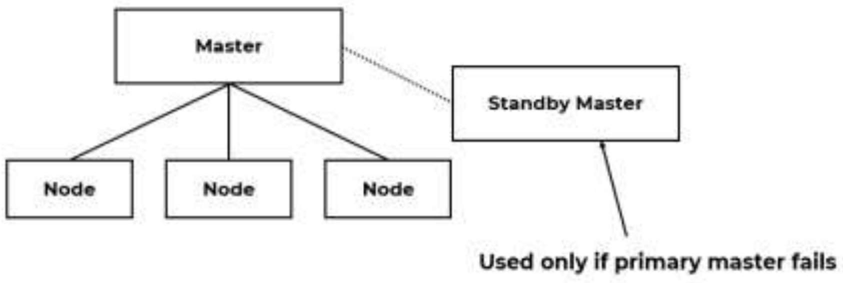


Figure 5.6: Master Salve Model

II) Client – Server Model:

1. The client-server model like the master slave designates one server, which serves multiple clients.
2. In Client- server replication all the updates must be propagated first to the server, which then updates all the other clients.

3. Since all updates must go through the server, the server acts as a physical synchronization point.
4. In this model the conflicts which occur are always be detected only at the server and only the server needs to handle them.
5. Figure 5.7 shows the example of client server model.

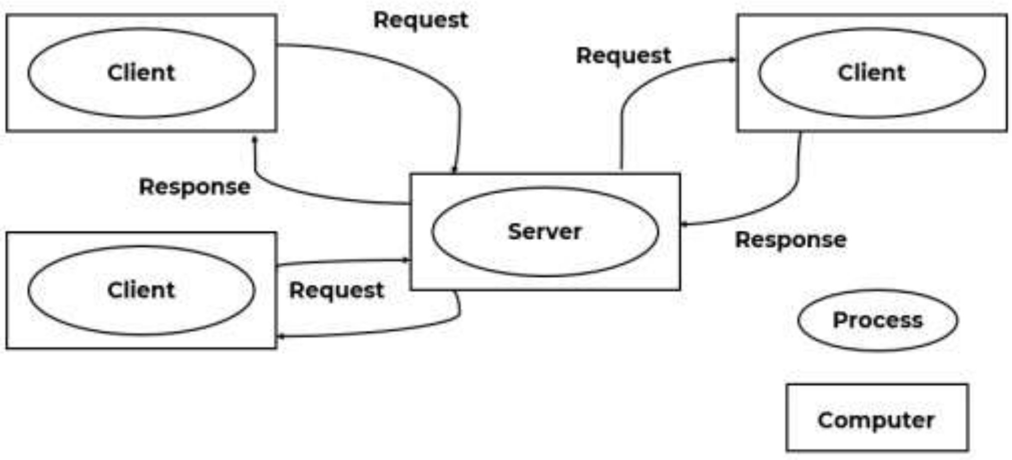


Figure 5.7: Client Server Model

III) Peer–to–Peer Model:

1. Here all the replicas or the copies are of equal importance or they are all peers.
2. In this model any replica can synchronize with any other replica, and any file system modification or update can be applied at any replica.
3. Peer-to-peer systems can propagate updates faster by making use of any available connectivity.
4. They provide a very rich and robust communication framework.
5. They are more complex in implementation and in the states they can achieve. One more problem with this model is scalability.
6. Figure 5.8 shows the example of peer to peer model.

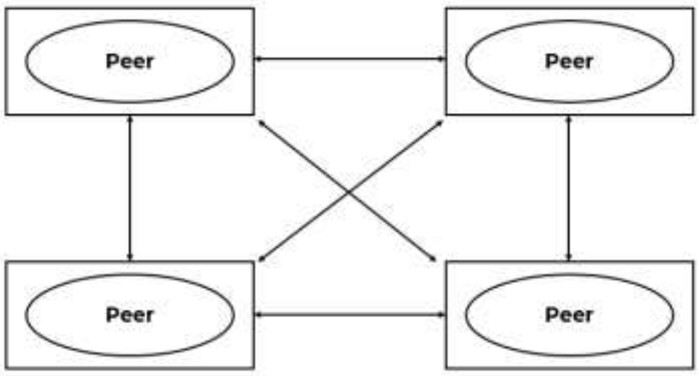


Figure 5.8: Peer to Peer Model

CHAP - 6: DISTRIBUTED FILE SYSTEMS AND NAME SERVICES

Q1. What are the desirable features of good distributed file systems? Explain file sharing semantic of it.

Ans:

DESIRABLE FEATURES OF A GOOD DISTRIBUTED FILE SYSTEM:

I) Transparency:

1. Transparency refers to hiding details from the user.
2. It consists of following types:
   a. **Structure Transparency:**
      - In multiple file servers the multiplicity of the file servers should be transparent to the clients of DFS.
      - Clients should not know the number of locations of the file servers and the storage devices.

   b. **Access Transparency:**
      - The local and remote files should be accessible in the same way.
      - File system interface should not be able to distinguish between the local and remote files.

   c. **Naming Transparency:**
      - Name of the file should give no hint of the location of the file.
      - Without changing the filename, the file should be allowed to move from one node to another.

   d. **Replication Transparency:**
      - If a file is located in multiple nodes, its existing multiple copies should be hidden from the client.

II) User Mobility:

1. User should not be forced to work on specific node.
2. User should have to flexibility to work on different nodes at different times.

III) Performance:

1. Performance is the average amount of time required to satisfy the client requests.
2. This time also includes network communication overhead when the accesses file is remote.
3. Performance of DFS should be comparable to the performance of Centralized File System.

IV) Simplicity and ease of use:

1. Semantics of the DFS should be easy to understand.
2. User interface of the file system must be simple and the number of commands should be as small as possible.

V) Scalability:

1. DFS grows with time due to expansion of networks by adding two or more networks at a common place.
2. DFS should be designed to easily cope up with the growth of nodes and users of the System.

VI) High availability:

1. System should continue to function even when partial failures occur.

2. Failure causes temporary loss of service to small groups of users.
3. High available DFS should have multiple independent file servers.

VII) High reliability:

1. Loss of stored data probability should be minimized.
2. User should not face problems on keeping a backup of their files.

VIII) Data integrity: Concurrent access of data must be properly synchronized using concurrency control mechanism.

IX) Security:

1. DFS should be secure so as to provide privacy to the users for their data.
2. Necessary Security mechanism must be implemented to keep the data secure.

X) Heterogeneity: Heterogeneous DFS provide the flexibility to the users to use different computer platforms for different applications.

FILE SHARING SEMANTICS:

I) UNIX Semantics:

1. In this semantics, absolute time ordering is performed on all operations.
2. UNIX semantic is easy to serialize read/write operation.

II) Session Semantics:

1. It consists of following access pattern:
   a. Client opens a file.
   b. Performs read/write operations on file.
   c. Closes the file.
2. A session is a series of accessed files between open and close operations.
3. In session semantics all changes made to the file are initially visible only to the client and invisible to other remote processes.

III) Immutable shared files semantics:

1. It is based on the **use of immutable file model**.
2. An immutable file cannot be modified once it has been created.
3. The creator declares the file as sharable so as to make it immutable file.

IV) Transaction like semantics:

1. It is based on the **transaction mechanism**.
2. A transaction is a setoff operations between a pair of begin-transaction and end-transaction.
3. It is a high-level mechanism for controlling concurrent access to shared mutable data.
4. It allows partial modification made to the shared data.

Q2. Describe File-Caching schemes

Q3. Discuss File caching for Distributed Algorithm

Ans:

FILE CACHING:

1. In order to improve the file I/O performance in centralized time sharing systems, file caching has been implemented and executed in numerous distributed file systems.
2. Main goal of file caching scheme is to retain recently accessed data in main memory.
3. So that repeated access to the same information can be efficiently handled.
4. In implementing file-caching scheme, one has to make several key decisions like granularity of cached data.
5. Three design issues are involved in file caching which are:

I) Cache location:

1. Cache location refers to the place where the cached data is stored.
2. There exists three possible cache locations in servers DFS i.e. server's main memory, client's disk and client's main memory as shown in figure 6.1.

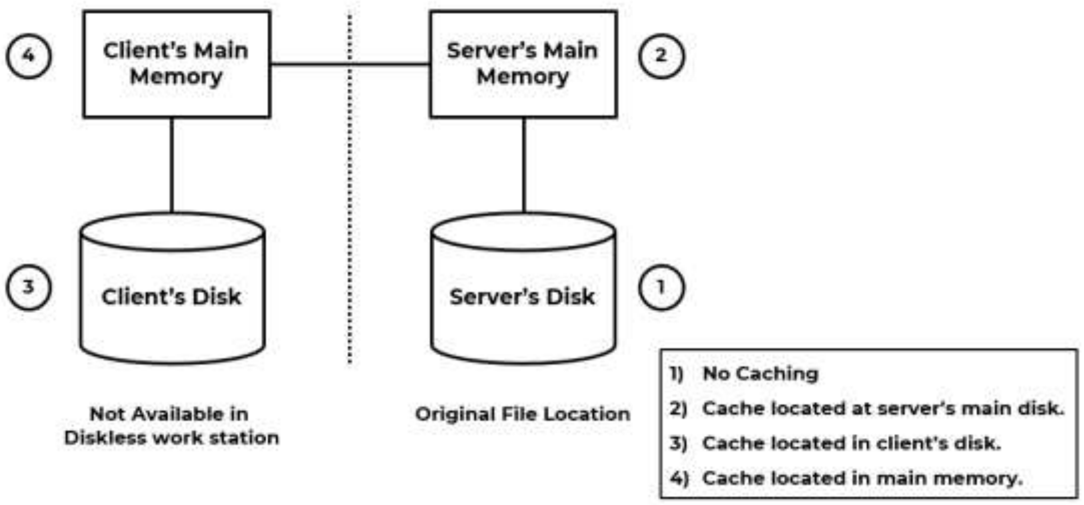

4
Client's Main Memory
Server's Main Memory
2
3
Client's Disk
Server's Disk
1
Not Available in Diskless work station
Original File Location
1) No Caching
2) Cache located at server's main disk.
3) Cache located in client's disk.
4) Cache located in main memory.

Figure 6.1: Cache Location.

Server's main memory:

1. In this the file must be first transferred to the server's main memory and then across the network
2. Hence the disks access cost is reduced.

Client's disks: The cache located in the client's disk eliminates network access cost but requires disk access cost.

Client's main memory: The cache located in a client's main memory eliminated both network access cost and disk access cost.

II) Modification propagation:

1. When the cache is located on clients' node; a file's data may simultaneously be cached on multiple nodes.

2. It is possible for caches to become inconsistent when the file data is changed by one of the clients and the corresponding data cached at other nodes are not changed or discarded.
3. A variety of approaches is used to handle these issues:
   a. When to propagate modifications made to a cached data to corresponding file server?
   b. How to verify the validity of cached data?
4. It consists of the following techniques:

Write through scheme:

When cache entry is modified new value is immediately sent to the server for updating the master copy of the file.

Delayed-write scheme:

1. In this scheme when a cache entry is modified the new value is written only to the cache.
2. The client just makes a note f the cache entry that has been updated.
3. All the cache entries are collected together and sent to the server later on.
4. Its approaches are as follows:
   a. Write on ejection from cache:
      - Modified data in the cache is sent to the server when the cache replacement policy has decided to eject it from the client's cache.

   b. Periodic write:
      - Here cache is scanned at regular intervals.
      - And cached data is been modified right from the last scan is sent to the server.

   c. Write on close:
      - Modification made to a cached data by a client is sent to the server when the corresponding file is closed by the client.

III) Cache Validation Schemes:

1. A file data resides in the cache of multiple nodes.
2. It becomes necessary to check if the data cached at the client node is consistent with the master node.
3. If it is not then the cached data must be invalidated and the updated version must be fetched from the server.
4. Validation is done in two ways:

Client initiated approach:

1. In this method the client contacts the server to check if locally cached data is consistent with the master copy.
2. File sharing semantics depends on the frequency of the validity check and can be use below approaches:
   a. **Check before every access:** The main purpose of caching is defeated in this process because the server has to be contacted on every access.
   b. **Periodic check:** For this method a check is initialized after fixed interval of time.
   c. **Check on file open:** A client cache entry is validated only when the client open a corresponding file for use.

Server initiated approach:

1. In this method a client informs the file server when it opens a file indicating whether the file is being opened for reading, writing or both.
2. The file server keeps a record of which client has which file opened and in what mode.
3. Thus server keeps a track of the file usage modes being used by different clients and reacts in case of inconsistency.
4. Inconsistency occurs when more clients try to open a file in conflicting modes.
5. **Example:** If a file is open for reading, other clients may be able to open it to read a file without any problem, but not for writing.
6. Similarly, a new client must not be allowed to open a file which is already open for writing.
7. When client closes a file it sends intimation to the server along with any modification made to the file.
8. On receiving intimation the server updates its record of which client has which file open in what mode.

Q4. Explain File Accessing Models

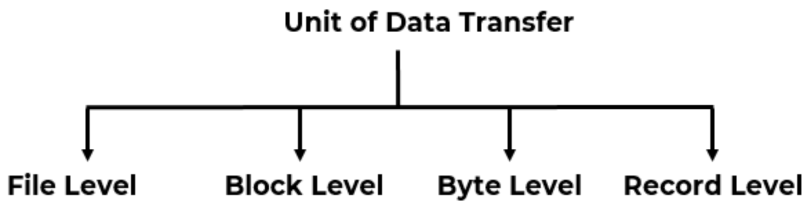
Ans:

FILE ACCESSING MODELS:

1. Specific client's request for accessing a particular file is serviced on the basis of the file accessing model used by the distributed file system.
2. File accessing model depends on the unit of data access and the method used for accessing remote files.

I) Unit of Data Access:

1. Unit of data access refers to fraction of file data that is transferred to and from client as a result of single read write operation.
2. It includes four data transfer models:



File level transfer model:

1. When the operation required file data, the whole file is moved
2. Advantages:
    a. Efficient because network protocol overhead is required only once.
    b. Better scalability because it requires fewer access to file server and reduce server load and network traffic.
3. Drawbacks is it requires sufficient storage space.
4. **Example** are amoeba, CFS, Andrew file system.

Block level transfer model:

1. File data transfer take place in units of data blocks.
2. A file block is contiguous portion of file and fixed in length.

3. Advantage is does not required large storage space.
4. Drawback is more network traffic and more network protocol overhead.
5. Poor performance.
6. **Example** are Sun microsystem's NFS, Apollo domain file system.

Byte level transfer model:

1. File data transfer take place in units of bytes.
2. Provides maximum flexibility.
3. Difficulty in cache management.

Record level transfer model:

1. Suitable with structured files.
2. File data transfer take place in unit of records.
3. **Example:** RSS(research storage system)

II) Accessing Remote Files:

One of the following model is used when request to access remote file.

Remote service model:

1. Processing of client request is performed at server's node.
2. Client request is delivered to server and server machine performs on it and returns replies to client.
3. Request and replies transferred across network as message.
4. File server interface and communication protocol must be designed carefully so as to minimize the overhead of generating the messages.
5. Every remote file access results in traffic.

Data catching model:

1. Reduced the amount of network traffic by taking advantage of locality feature
2. If requested data is not present locally then copied it from server's node to client node and catching there.
3. LRU is used to keep the cache size bounded.
4. Cache Consistency problem.

Q5. Andrew file system (AFS)

Ans:

ANDREW FILE SYSTEM (AFS):

1. The Andrew File System (AFS) is a distributed file system.
2. It was developed by Carnegie Mellon University as part of the Andrew Project.
3. Originally named "Vice", AFS is named after Andrew Carnegie and Andrew Mellon.
4. Its primary use is in **distributed computing**.
5. It uses a **session semantics.**
6. AFS supports information sharing on large scale.

FEATURES:

Andrew File System is designed to:

1. Handle terabytes of data.
2. Handle thousands of users.
3. Working in WAN Environment.

AFS ARCHITECTURE:

1. Figure 6.2 shows the AFS Architecture.
2. AFS is divided into two types of nodes:
   a. **Vice Node:** It is dedicated file server.
   b. **Virtue Node:** It is a Client Machines.
3. In VFS, the entire file is copied to the Virtue Node (local machine) from the Vice Node (Server) when opened.
4. When the file is changed – it is copied to the server when closed.

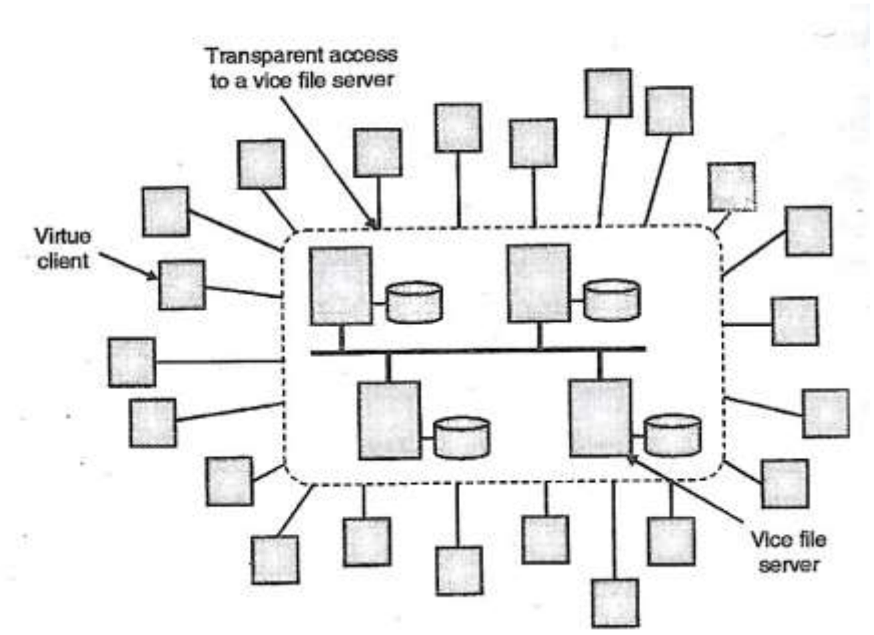


Figure 6.2: AFS Architecture.

LIMITATION OF AFS:

1. Non-availability of services when servers and network components fail.
2. Scaling problem.

Q6. Explain Naming and explain system oriented names in details

Ans:

NAMING:

1. Distributed system supports several types of objects such as processes, files, I/O devices, mail boxes and nodes.
2. The naming facility of a distributed operating system enables the user and programs to assign character string names to objects.

3. The naming system plays a very important role in achieving the goal of location transparency in a distributed system.

SYSTEM ORIENTED NAMES:

1. System oriented names are also known as unique identifiers and low level names.
2. These names are bit patterns of fixed size that can be easily changed and stored by the machines.
3. System oriented names uniquely identify every object for efficiency in the system.
4. System oriented names can be either structured or unstructured.
5. **Structured:** It contains more than one component and might provide information about object identifier.

A single field of large integers or bit string

6. **Unstructured:** It has only a single field of large integer or bit strings and it do not provide any information about object identifier.

Node Identifier	Local Unique Identifier

Characteristics:

1. They are large integers or bit strings.
2. They are unique identifiers.
3. All the system oriented names have the same size.
4. They are normally shorter in size compared to human oriented names.
5. They are hard to guess hence are sometimes also used for security related purpose.
6. They are automatically generated.

APPROACHES:

I) Centralized Approach:

1. This approach generates both structured as well as unstructured system oriented names.
2. Naming system mostly makes use of unstructured names.
3. Here uniform global identifier is generated for each object in the system.
4. Each node either directly binds to the global identifiers for locally created objects or maps it to the local identifiers for locally created objects.
5. This binding can be temporary or permanent.

Advantages:

1. It is simple and easy to implement
2. It is the only method for generating global unstructured identifier

Disadvantages:

1. Poor reliability.
2. Poor efficiency.

II) Distributed Approach:

1. To overcome poor reliability and efficiency of centralized naming system, distributed naming system was introduced.
2. Distributed naming system uses structured object identifiers.
3. In this, hierarchical concatenation strategy is used to create global unique identifiers.
4. A unique identifier is assigned to each identification domain.
5. A global unique identifier is created by concatenating a unique identifier of a domain with an identifier used within the domain.

Issues:

1. Main problem is to generate unique identifiers during a system crash.
2. A crash can lead to loss of state information of unique identifier generator.
3. After recovery also unique identifier generator may not function properly. Hence two approaches are used:
    a. **Use of clock that operates across failures:** Here clock is used instead of unique identifier generator that continues its operation across failures.
    b. **Using two more levels of storage:** Here unique identifiers are structured in hierarchical fashion with one field for each level.

Q7. Explain human oriented names in details

Ans:

HUMAN ORIENTED NAMES:

1. Human oriented names are character string that is meaningful to the users.
2. They are defined and used by the users.
3. Different users can define and use their own names for a shared object.
4. They provide the facility of aliasing.
5. Hence same name can be used for by two different users at the same time to refer two different objects.
6. It provides easy and efficient management of namespace.

Characteristics:

1. They are character strings that are meaningful too the user.
2. They are defined and used by the users.
3. Different names to the same object.
4. They are of variable length.
5. They are not unique

Issues:

1. Selecting proper scheme for global object naming.
2. Selecting proper scheme for partitioning namespace into contexts.
3. Selecting proper scheme for implementing context bindings.

4. Selecting proper scheme for name resolution.

The Figure 6.3 describes Human oriented names for objects that refer them easily i.e. the first level mapping. The second level mapping is done for physical locations of the object's replicas which has physical address for the objects.

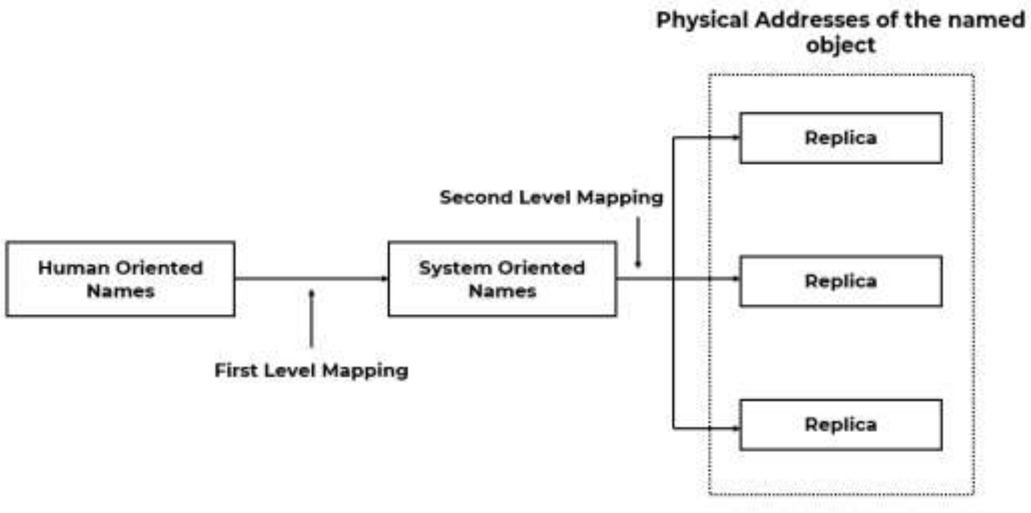


Figure 6.3

Q8. Explain name resolution in DNS

Ans:

NAME RESOLUTION IN DNS:

1. DNS stands for **Domain Name System**.
2. DNS is used for searching host addresses and mail servers.
3. The string representation of a path name consists of listing its label starting with the rightmost one having '.' (dot) as the separator.
4. Example: aa.bb.cc.dd. were the rightmost node is the root.

DNS NAME RESOLUTION:

1. Naming resolution is the process of mapping object's name to the object's properties such as its location.
2. It is basically the process of mapping object's name to the object's authorative name servers.
3. Each name agent in a distributed system knows about at least one name server.
4. The name servers are treated as objects.
5. Name resolution tries to convert some of the numerical address values into a human readable format.
6. A label is a case-insensitive string made up of alphanumeric characters.
7. A label has a maximum length of 63 characters.
8. The root is represented by a dot.
9. A sub tree is called a domain;
10. A path name to its root node is called a domain name.
11. The length of a complete path name is restricted to 255 characters.
12. Just like a path name, a domain name can be either absolute or relative.

13. The DNS name space is hierarchically organized as a rooted tree.

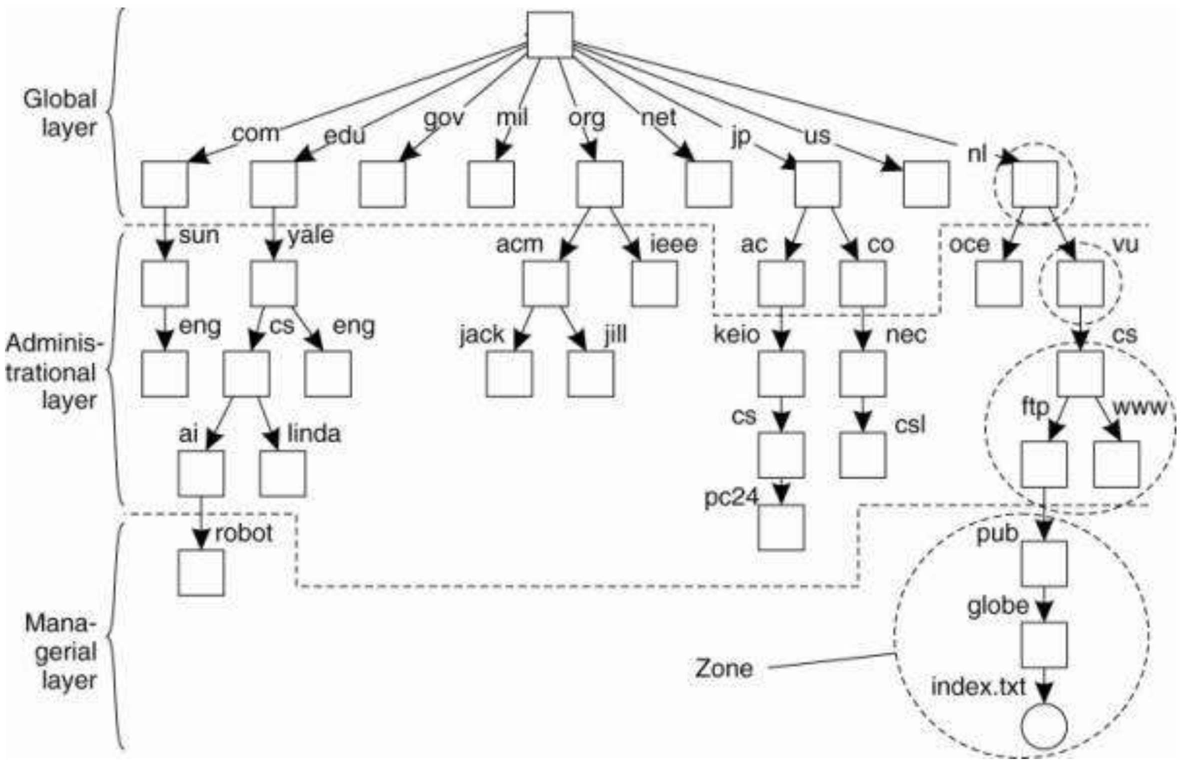


Figure 6.4: DNS Namespace

DNS NAME RESOLUTION PROCESS:

1. When a DNS name resolution request is forwarded to a DNS server, the DNS server examines its local DNS cache for the IP address.
2. If the IP address is not in the DNS server's cache, it checks its host file.
3. Then DNS server forwards the request to a higher level DNS server.
4. If the DNS server cannot forward the request, the DNS server uses its root hints file that lists the 13 roots DNS servers.
5. The root DNS server responds with the address of a .com, .edu, .net or other DNS server type depending upon the request.
6. DNS server forwards the request to a high level DNS server, which can respond with a variety of IP addresses.

DNS HIERARCHY COMPONENTS:

1. Top level domains.
2. Domains.
3. Hosts.

Q9. Explain x.500 directory service in detail

Ans:

X.500:

1. X.500 is a series of computer networking standards covering electronic directory services.
2. The X.500 series was developed by the Telecommunication Standardization Sector of the International Telecommunications Union (ITU-T)
3. It specifies a global, hierarchical directory service.
4. Figure 6.5 represents X.500

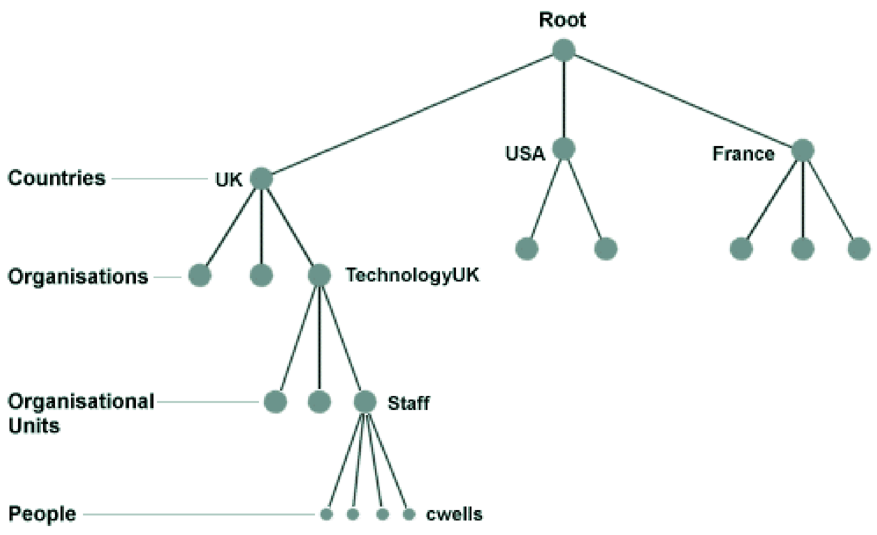


Figure 6.5: X.500

FEATURES OF X.500:

1. A standards-based directory service for those applications that require directory information
2. A single, global, hierarchical namespace of objects and their attributes
3. Data management functions for viewing, adding, modifying, and deleting directory objects
4. Search capabilities for customizing complete data queries

WORKING OF X.500:

1. X.500 defines a global directory service that consists of several components.
2. From an administrative point of view, the building blocks of the X.500 directory service are Directory Management Domains (DMDs).
3. An X.500 DMD is a collection of X.500 components that includes at least one Directory System Agent (DSA) and is managed by a Domain Management Organization (DMO).
4. There are two types of DMDs:
   a. **Administrative Directory Management Domains (ADDMDs):** Directory services managed by a registered private agency that provide public directory services. Examples of ADDMDs are Four11 and Bigfoot, which provide public X.500 directory services over the Internet.
   b. **Private Directory Management Domains (PRDMDs):** Directory services that provide private directory access. An example is a domain controller hosting Active Directory on a network running Microsoft Windows Server.

X.500 COMPONENTS:

Three main components are involved in maintaining and accessing X.500 directory services:

1. **Directory Information Base (DIB):** The actual hierarchical database that contains all the information in the directory. X.500 uses a distributed directory hierarchy in which different subsets of the DIB are found on different servers at different locations.
2. **Directory System Agent (DSA):** A particular server that maintains a subset of the DIB and provides an access point to the directory for DUAs to connect.
3. **Directory User Agents (DUAs):** The client software that accesses the X.500 directory on behalf of the user.

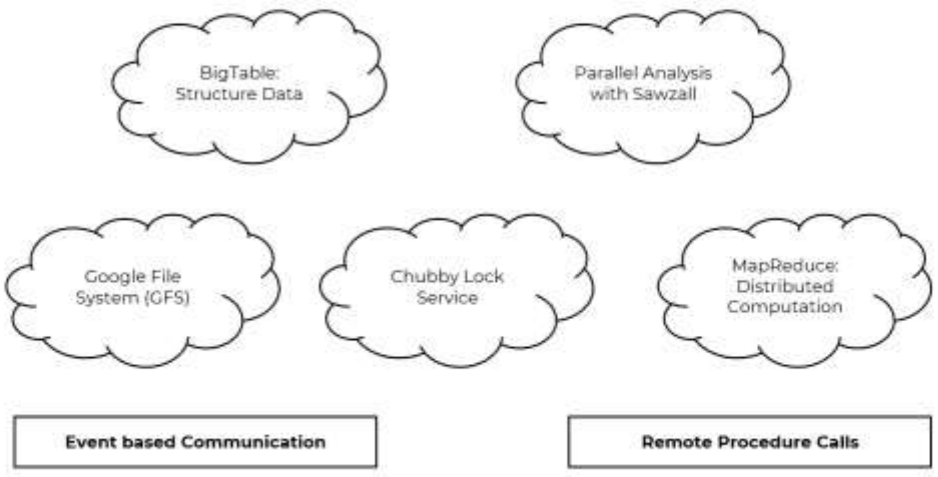
Q10. Explain components of Google infrastructure

Ans:

Google visualizes their infrastructure as a three layer stack i.e. products, distributed systems infrastructure and computing platforms.

a. **Products:** search, advertising, email, maps, video, chat, blogger
b. **Distributed Systems Infrastructure:** GFS, MapReduce, and BigTable.
c. **Computing Platforms:** a bunch of machines in a bunch of different data centers

COMPONENTS OF GOOGLE INFRASTRUCURE:


BigTable: Structure Data
Parallel Analysis with Sawzall
Google File System (GFS)
Chubby Lock Service
MapReduce: Distributed Computation
Event based Communication
Remote Procedure Calls

I) GFS:

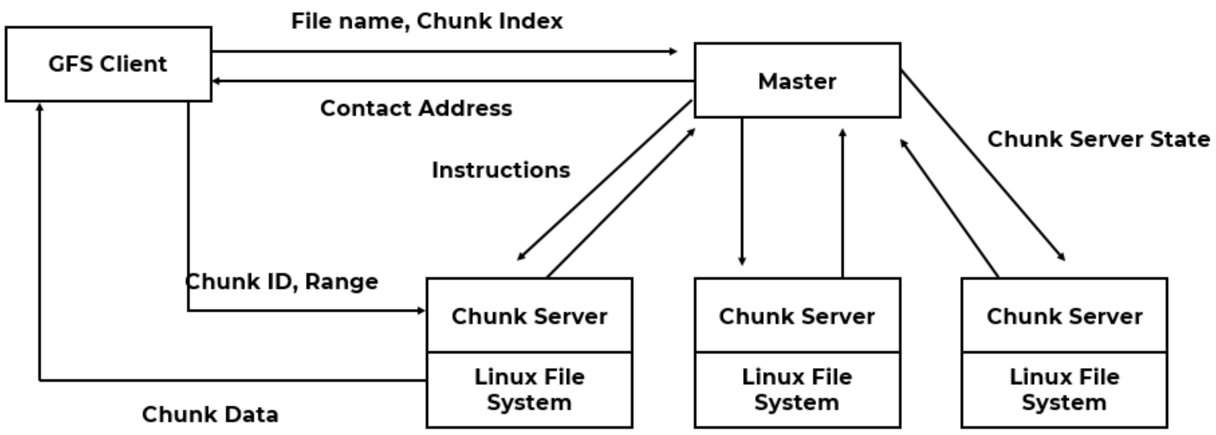


Figure 6.6: Organization of a google cluster of servers

1. Google has developed its own Google file system (GFS).
2. Google files tend to be very large, commonly ranging up to multiple gigabytes, where each one contains lots of smaller objects.
3. Moreover, updates to files usually take place by appending data rather than overwriting parts of a file.
4. These observations, along with the fact that server failures are the norm rather than the exception, lead to constructing clusters of servers as shown in figure 6.6
5. Each cluster has one master and multiple chunkservers.
6. Files replicated on by default three chunkservers.
7. System manages fixed sized chunks 64 MB (very large)
8. Over 200 clusters some with up to 5000 machines offering 5 petabytes of data and 40 gigabytes/sec throughput (old data)

II) <u>Chubby Lock Service:</u>

1. Google has developed Chubby, a lock service for loosely coupled distributed systems.
2. It is designed for coarse-grained locking and also provides a limited but reliable distributed file system.
3. It provides simple file system API.
4. Used for many purposes such as:
   a. As a small scale filing system for, e.g., meta-data.
   b. As a name-server
   c. To achieve distributed consensus/ locking.
   d. Use of Lamport's Paxos Algorithm
5. Figure 6.7 represents chubby lock service

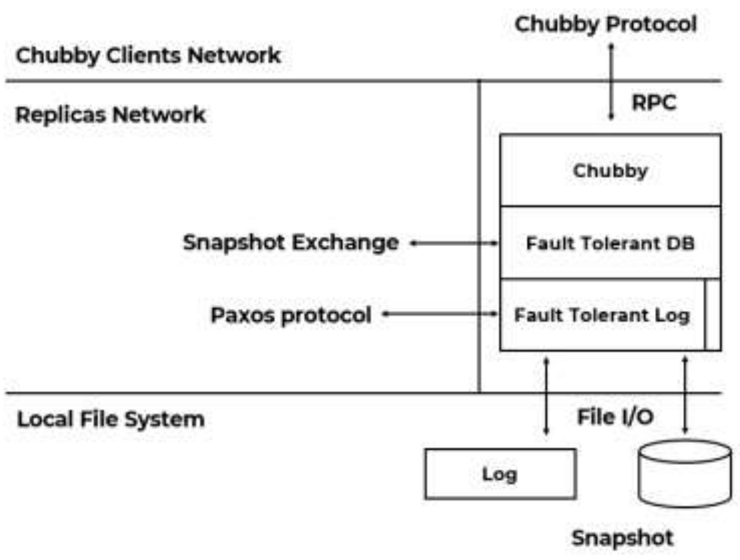


Figure 6.7: Chubby Lock Service

III) <u>BigTable:</u>

1. Bigtable is a very large scale distributed table indexed by row, column and timestamp (built on top of GFS and Chubby).
2. It supports semi-structured and structured data on top of GFS (but without full relational operators, e.g. join).
3. It is used by over 60 Google products including Google Earth.

IV) MapReduce:

1. It is a service offering a simple programming model for large scale computation over very large data-sets, e.g. a distributed grep.
2. It hides parallelization, load balancing, optimization, failure/ recovery strategies

---

Q11. Explain NFS in Distributed Systems

Ans:

NFS:

1. NFS stands for **Network File System**.
2. NFS is a **platform independent remote file system** technology created by Sun Microsystems (Sun) in 1984.
3. It is a **client/server application** that **provides shared file storage** for clients across a network.
4. It was designed to simplify the sharing of file systems resources in a network of non-homogeneous machines.
5. It is implemented using the RPC protocol and the files are available through the network via a **Virtual File System (VFS)**, an interface that runs on top of the TCP/IP layer.
6. Allows an application to access files on remote hosts in the same way it access local files.
7. NFS is generally implemented following the layered architecture shown in figure 6.8 below.

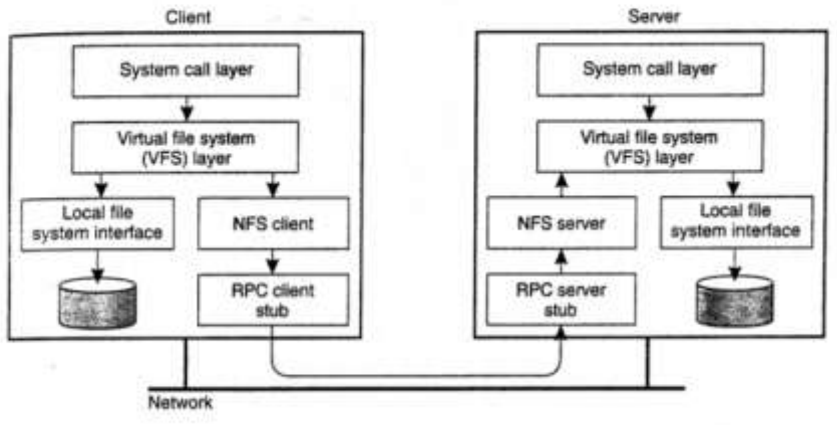

Client
Server
System call layer
Virtual file system (VFS) layer
Local file system interface
NFS client
RPC client stub
System call layer
Virtual file system (VFS) layer
NFS server
Local file system interface
RPC server stub
Network

Figure 6.8: NFS Architecture

NFS SERVERS: COMPUTERS THAT SHARE FILES

1. During the late 1980s and 1990s, a common configuration was to configure a powerful workstation with lots of local disks and often without a graphical display to be a NFS Server.
2. "Thin," diskless workstations would then mount the remote file systems provided by the NFS Servers and transparently use them as if they were local files.

NFS Simplifies management:

1. Instead of duplicating common directories such as /usr/local on every system, NFS provides a single copy of the directory that is shared by all systems on the network.
2. Simplify backup procedures - Instead of setting up backup for the local contents of each workstation (of /home for exmaple), with NFS a sysadm needs to backup only the server's disks.

NFS Clients: Computers that access shared files

1. NFS uses a mixture of kernel support and user-space daemons on the client side.
2. Multiple clients can mount the same remote file system so that users can share files.
3. Mounting can be done at boot time. (i.e. /home could be a shared directory mounted by each client when user logs in).
4. An NFS client:

   (a) Mounts a **remote file system** onto the client's **local file system** name space and

   (b) Provides an interface so that access to the files in the remote file system is done as if they were local files.

GOALS OF NFS DESIGN:

1. **Compatibility:** NFS should provide the same semantics as a local unix file system. Programs should not need or be able to tell whether a file is remote or local.
2. **Easy deployable:** Implementation should be easily incorporated into existing systems remote files should be made available for local programs without these having to be modified or relinked.
3. **Machine and OS independence:** NFS Clients should run in non-unix platforms Simple protocols that could be easily implemented in other platforms.
4. **Efficiently:** NFS should be good enough to satisfy users, but did not have to be as fast as local FS. Clients and Servers should be able to easily recover from machine crashes and network problems.

NFS PROTOCOL

1. Uses Remote Procedure Call (RPC) mechanisms
2. RPCs are synchronous (client application blocks while waits for the server response)
3. NFS uses a stateless protocol (server do not keep track of past requests) - This simplify crash recovery. All that is needed to resubmit the last request.
4. In this way, the client cannot differentiate between a server that crashed and recovered and one that is just slow.