Short answer Questions

1) Define DS. Examples of DS.

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

- A distributed system consists of components (i.e.,) computers that are autonomous, each of which has its own local memory.
- Users (people and programs) think they are dealing with a single system.
- Autonomous components need to collaborate.
- The entities communicate with each other by message passing.
- Each computer has only a limited, incomplete view of the system. Each computer may know only one part of the input.

In distributed computing, each processor has its own private memory (distributed memory) and information is exchanged by passing message between the processors unlike the way parallel computing systems work using shared memory.

Goals of Distributed Systems:

Four goals that should be met to make building of a distributed system worth the effort are:

- Making resources easily accessible
- Reasonably hide the fact that resources are distributed across a network.
- Openness
- Scalable

Types of Distributed Systems

1. Distributed Computing Systems

Distributed systems used for high-performance computing task

- Cluster computing
- Grid computing systems

2. Distributed Information Systems

- Transaction processing systems

3. Distributed pervasive system

Examples of DS:

1. Intranets, Internet, WWW, email.
2. Telecommunication networks: Telephone networks and Cellular networks.
3. Network of branch office computers

4. Real-time process control: Aircraft control systems, Electronic banking, Airline reservation systems, Sensor networks, Mobile and Pervasive Computing systems.

2) What is the role of middleware in DS?

- Middleware is a software that provides services beyond those provided by the operating system to enable the various components of a distributed system to communicate and manage data.
- Middleware supports and simplifies complex distributed applications.
- Middleware often enables interoperability between applications that run on different operating systems, by supplying services so the application can exchange data in a standards-based way
- Middleware forms a layer between applications and distributed platforms.
- Provide a degree of distribution transparency, hiding the distribution of data, processing, and control from applications.
- Systems are now developed with a stricter separation between policies and mechanisms.

  Led to mechanisms by which the behavior of middleware can be modified

- Commonly followed approach:

  Interceptors:

  An interceptor is a software construct that will break the usual flow of control and allow other (application specific) code to be executed.

  Adaptive software:

  1. Separation of concerns
  2. Computational reflect
  3. Component based design

3) Differentiate between

a) Structured & unstructured peer- to -peer architecture.

STRUCTURED P-2 -P	UNSTRUCTURED P-2-P
Some well known structured p2p networks are Chord,Pastry,Tapestry,CAN	Each node maintains a list of neighbors ,but that this list is constructed in a random way
A Structured peer-to-peer overlay network is constructed using a deterministic procedure	Rely on randomized algorithms for constructing overlay network that resembles a random graph
Most-used procedure - organize the process through a distributed hash table (DTH) .(Hash Table description)	Data items are assumed to be randomly placed on nodes Goal is that each node constructs a partial view of the graph

b) Stateful server and stateless server

Parameters	Stateful	Stateless
1. State	A Stateful server remembers client data (state) from one request to the next.	A Stateless server keeps no state information
2. Programming	Stateful Server is harder to code	Stateless Server is straightforward to code
3. Efficiency	More Because clients do not have to provide full file information every time they perform an operation	Less because information needs to be provided
4. Crash Recovery	Difficult due to loss of information	Can easily recover from failure. Because there is no state that must be restored
5. Information transfer	Using a Stateful,file server, the client can send less data with each request	Using a stateless file server, the client must,specify complete file names in each request specify location for reading or writing re authenticate for each request.

6. Extra services	Stateful servers can also offer clients extra services such as file locking, and remember read and write positions	It does not have to implement the state accounting associated with opening, closing, and locking of files.
7. Operations	Open, Read, Write, Seek, Close	Read, Write

c) Persistent and transient communication

Transient communication: middleware discards a message if it cannot be delivered to receiver immediately (sender and receiver run at the same time based on request/response protocol, the message is expected otherwise it will be discarded)

Example: applications using TCP (FTP),UDP(Video streaming)
Persistent communication: messages are stored by middleware until the receiver can accept it. Receiving applications need not be executed when the message is submitted.

Example: Email

d) Flat naming and structured naming.

Flat Naming : In some systems, each entity is assigned an identifier that uniquely identifies the entity • Identifiers are fixed-size bit strings, which we refer to as flat names (or unstructured names) – Flat names can be efficiently handled by machines
E.g., IP addresses, Ethernet addresses

Structured Naming : Flat names are difficult for humans to remember. Structured names are composed of several parts; these names are human-friendly

Example: file names, Internet host names

4) Explain how the interceptor is used to handle RMI.

5) Explain about processes and threads.

The major differences between a process and a thread are given as follows −

Comparison Basis	Process	Thread
Definition	A process is a program under execution i.e an active program.	A thread is a lightweight process that can be managed independently by a scheduler.
Context switching time	Processes require more time for context switching as they are more heavy.	Threads require less time for context switching as they are lighter than processes.
Memory Sharing	Processes are totally independent and don’t share memory.	A thread may share some memory with its peer threads.

Communication	Communication between processes requires more time than between threads.	Communication between threads requires less time than between processes .
Blocked	If a process gets blocked, remaining processes can continue execution.	If a user level thread gets blocked, all of its peer threads also get blocked.
Resource Consumption	Processes require more resources than threads.	Threads generally need less resources than processes.
Dependency	Individual processes are independent of each other.	Threads are parts of a process and so are dependent.
Data and Code sharing	Processes have independent data and code segments.	A thread shares the data segment, code segment, files etc. with its peer threads.
Treatment by OS	All the different processes are treated	All user level peer threads are treated as a single

	separately by the operating system.	task by the operating system.
Time for creation	Processes require more time for creation.	Threads require less time for creation.
Time for termination	Processes require more time for termination.	Threads require less time for termination.

6) What is a super server?

7) Define Message passing interface .List the primitive of MPI.

Message Passing Interface (MPI) is a standardized and portable **message-passing** system developed for distributed and parallel computing. MPI provides parallel hardware vendors with a clearly defined base set of routines that can be efficiently implemented. As a result, hardware vendors can build upon this collection of standard low-level routines to create higher-level routines for the distributed-memory communication environment supplied with their parallel machines.

8) Write about layered network communication protocols.

9) What is Namespace and Name Resolution

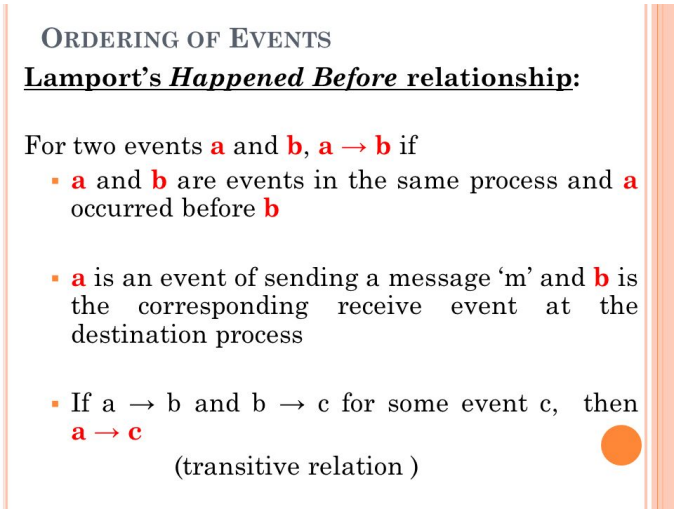
10) What are the reasons for Replication?

Ans:

Data Replication is a common technique in Distributed Systems.Copying the same data over multiple nodes in distributed systems is critical to keep the database up and keep on serving queries even during faults. Some of the reasons for data replication are as follows :

1) Higher Availability : To ensure the availability of the distributed system (System keeps on working even if one or fewer nodes fail).
2) Reduced Latency : Replication assists in reducing the latency of data queries.Eg:- CDN (Content Delivery Networks) keeps a copy of replicated data closer to the user.
3) Read Scalability : Read queries can be served from replicated copies of the same data.
4) Network Interruption : System works even under network faults.

11) What is Happened - Before relation



12) Define Sequential and Release consistency

Sequential consistency can be achieved simply by **hardware implementation**, while release consistency is also based on an i are properly synchronized.

Long answer Questions

13) What are the goals of DS?

Four goals that should be met to make building a distributed system worth the effort:

- should make resources easily accessible
- should reasonably hide the fact that resources are distributed across a network
- should be open
- should be scalable

1. Making Resources Accessible

- Main goal of a distributed system –
    - make it easy for the users (and applications) to access remote resources
    - to share them in a controlled and efficient way.
- Resources - anything: printers, computers, storage facilities, data, files, Webpages, and networks, etc.
- Accessibility Issues
    - Security
    - unwanted communication

2. Distribution Transparency

- Goal - hide the fact that its processes and resources are physically distributed across multiple computers – systems should be transparent
- Different forms of transparency in a distributed system

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

- **Degree of Transparency Issues:**
    - Timing
    - Synchronization
    - Performance
    - Consistency
    - Context Awareness
    - Limits of Possibility.

3. Openness

Goal: offer services according to standard rules that describe the syntax and semantics of those services.
e.g.

- computer networks - standard rules govern the format, contents, and meaning of messages sent and received.
- distributed systems - services are specified through interfaces, which are often described in an Interface Definition Language (IDL).
    - Interface definitions written in an IDL nearly always capture only the syntax of services
    - specify names of the available functions with types of parameters, return values, possible exceptions that can be raised, etc.
    - allows an arbitrary process that needs a certain interface to talk to another process that provides that interface
    - allows two independent parties to build completely different implementations of those interfaces, leading to two separate distributed systems that operate in exactly the same way.

Properties of specifications:

- Complete - everything that is necessary to make an implementation has been specified.
- Neutral -specifications do not prescribe what an implementation should look like

Lead to:

- Interoperability - characterizes the extent by which two implementations of systems or components from different manufacturers can co-exist and work together by merely relying on each other's services as specified by a common standard.
- Portability - characterizes to what extent an application developed for a distributed system A can be executed, without modification, on a different distributed system B that implements the same interfaces as A.

Goals: an open distributed system should also be extensible. i.e.

- be easy to configure the system out of different components (possibly from different developers).
- be easy to add new components or replace existing ones without affecting those components that stay in place.

4. Scalability

Scalability of a system is measured with respect to:

- Size - can easily add more users and resources to the system.
- Geographic extent - a geographically scalable system is one in which the users and resources may lie far apart.
- Administrative scalability - can be easy to manage even if it spans many independent administrative organizations.

Scalability Limitations of Size

Concept	**Example**
Centralized services	A single server for all users
Centralized data	A single on-line telephone book
Centralized algorithms	Doing routing based on complete information

Geographical scalability Limitations

- Synchronization
- Unreliability of communication

Administrative scalability:

- how to scale a distributed system across multiple, independent administrative domains.
- major problem - conflicting policies with respect to resource usage (and payment), management, and security.

14) Illustrate various types of DS with a neat diagram.

Different types of distributed systems exist which can be classified as being oriented toward supporting computations, information processing, and pervasiveness. Distributed computing systems are typically deployed for high-performance applications often originating from the field of parallel computing. A huge class of distributed can be found in traditional office environments where we see databases playing an important role. Typically, transaction processing systems are deployed in these environments. Finally, an emerging class of distributed systems is where components are small and the system is composed in an ad hoc fashion, but most of all is no longer managed through a system administrator. This last class is typically

represented by ubiquitous computing environments. In the following we make a distinction between distributed computing systems, distributed information systems, and distributed embedded systems.

A. Distributed Computing Systems

An important class of distributed systems is the one used for high-performance computing tasks. one can make a distinction between two subgroups.

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

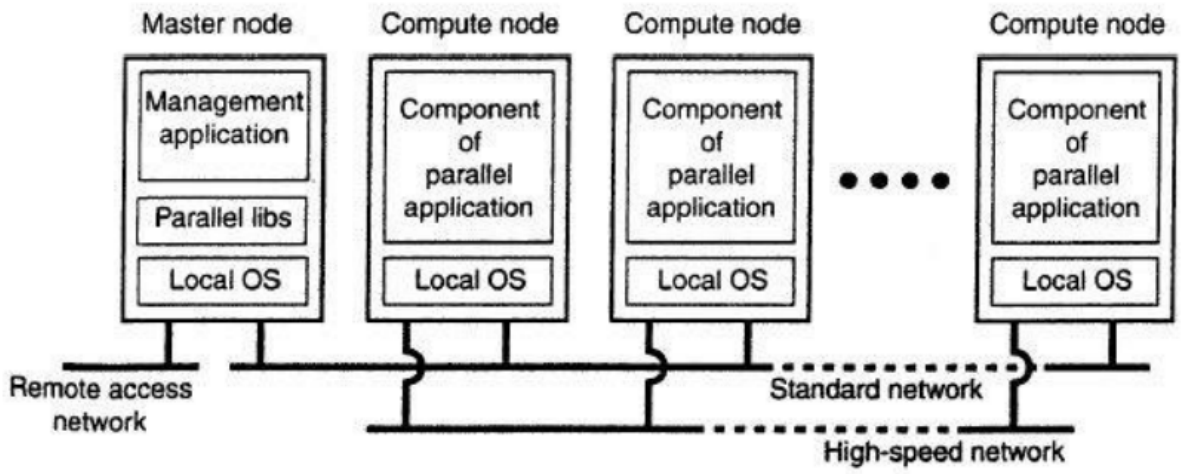


Figure 1-6. An example of a cluster computing system.

2.Grid Computing Systems: The situation becomes quite different in the case of grid computing. This subgroup consists of distributed systems that are often constructed as a federation of computer systems, where each system may fall under a different administrative domain, and may be very different when it comes to hardware, software, and deployed network technology. A characteristic feature of cluster computing is its homogeneity. In most cases, the computers in a cluster are largely the same, they all have the same operating system, and are all connected through the same network. In contrast, grid computing systems have a high degree of heterogeneity: no assumptions are made concerning hardware, operating systems, networks, administrative domains, security policies, etc.

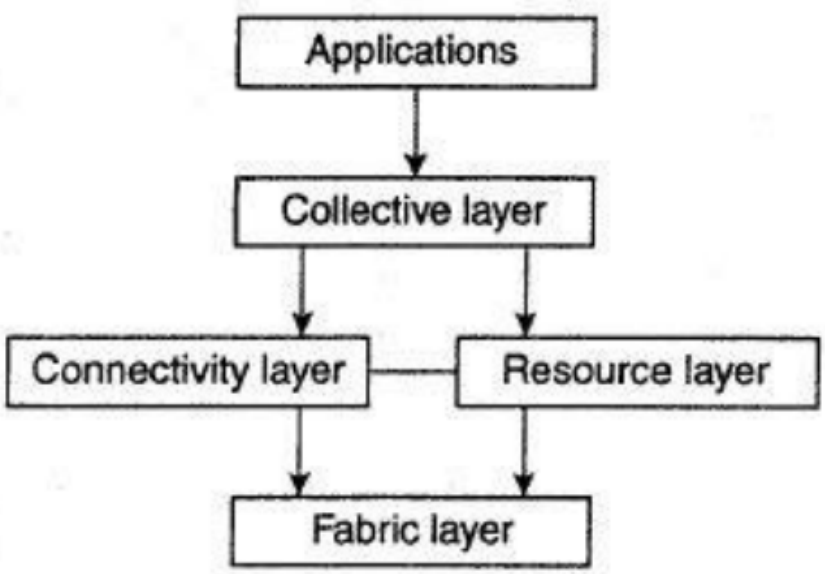


Figure 1-7. A layered architecture for grid computing systems.

B. Distributed Information Systems

Another important class of distributed systems is found in organizations that were confronted with a wealth of networked applications, but for which interoperability turned out to be a painful experience. Many of the existing middleware solutions are the result of working with an infrastructure in which it was easier to integrate applications into an enterprise-wide information system.

1.Transaction Processing Systems: operations on a database are usually carried out in the form of transactions. Programming using transactions requires special primitives that must either be supplied by the underlying distributed system or by the language runtime system This all-or-nothing property of transactions is one of the four characteristic properties that transactions have. More specifically, transactions are:

1. Atomic: To the outside world, the transaction happens indivisibly.

2. Consistent: The transaction does not violate system invariants.

3. Isolated: Concurrent transactions do not interfere with each other.

4.. Durable: Once a transaction commits, the changes are permanent.These properties are often referred to by their initial letters: ACID.

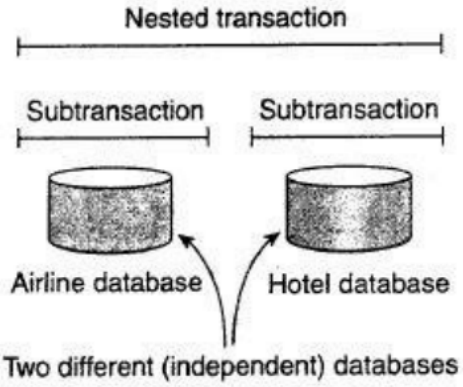


Figure 1-9. A nested transaction.

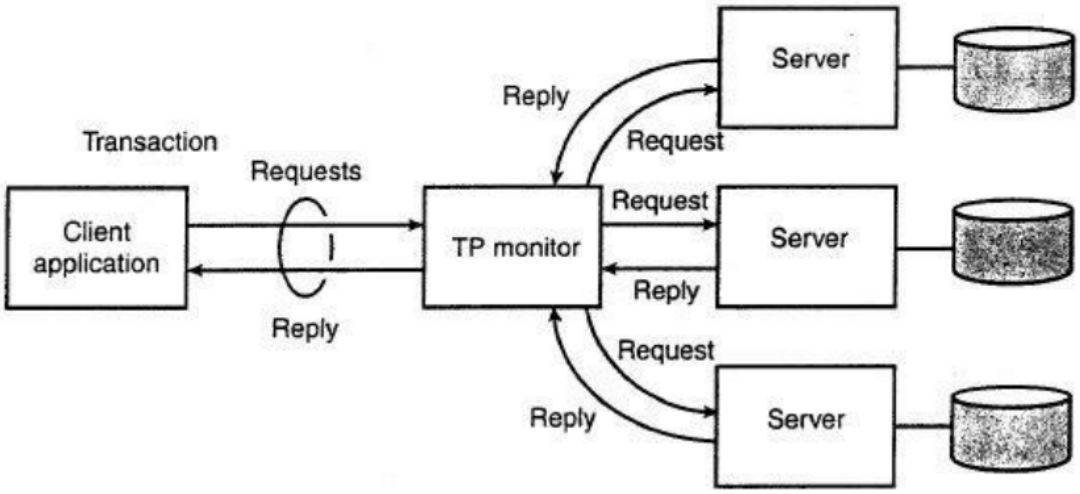


Figure 1-10. The role of a TP monitor in distributed systems.

2.Enterprise Application Integration: As mentioned, the more applications became decoupled from the databases they were built upon, the more evident it became that facilities were needed to integrate applications independent from their databases. In particular, application components should be able to communicate directly with each other and not merely by means of the request/reply behavior that was supported by transaction processing Systems. This need for interapplication communication led to many different communication Models, The main idea was that existing applications could directly exchange information,Several types of communication middleware exist.

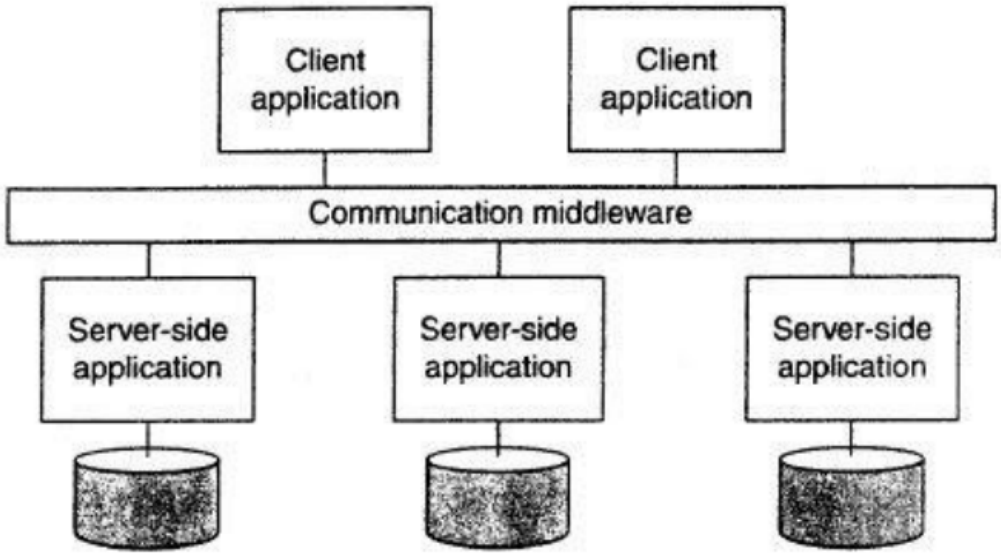


Figure 1-11. MiddJeware as a communication facilitator in enterprise application integration.

C. Distributed Pervasive Systems

The distributed systems we have been discussing so far are largely characterized by their stability: nodes are fixed and have a more or less permanent and high-quality connection to a network. To a certain extent, this stability has been realized through the various techniques that are discussed in this book and which aim at achieving distribution transparency However, matters have become very different with the introduction of mobile and embedded computing devices. We are now confronted with distributed systems in which instability is the default behavior. The devices in these, what we refer to as distributed pervasive systems, are often characterized by being small, battery-powered, mobile, and having only a wireless connection, although not all these characteristics apply to all devices.As its name suggests, a distributed pervasive system is part of our surroundings (and as such, is generally inherently distributed). An important feature is the general lack of human administrative control. At best, devices can be configured by their owners, but otherwise they need to automatically discover their environment and "nestle in" as best as possible. This nestling in has been made more precise by Grimm et al. (2004) by formulating the following three requirements for pervasive applications:

1. Embrace contextual changes.

2. Encourage ad hoc composition.

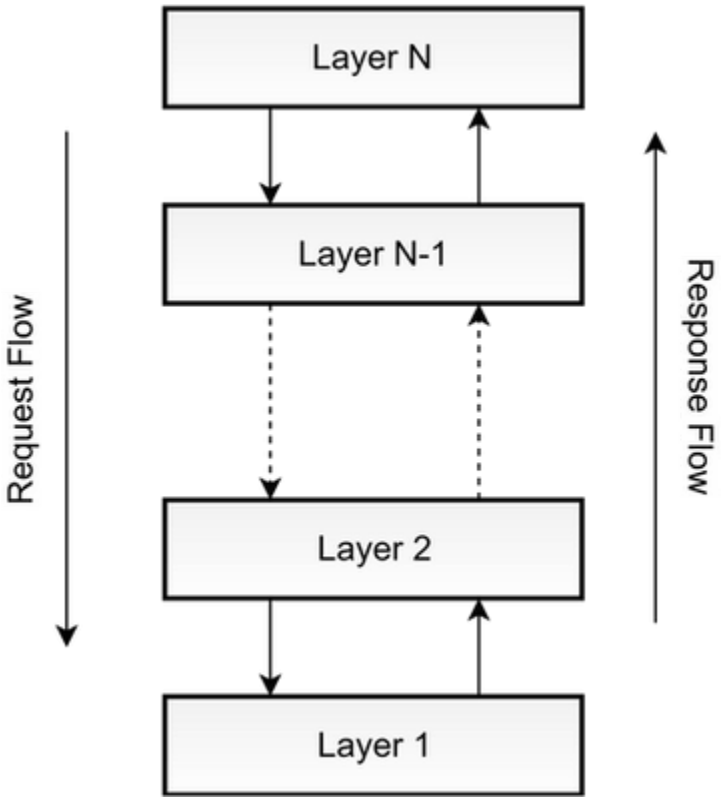
3. Recognize sharing as the default.

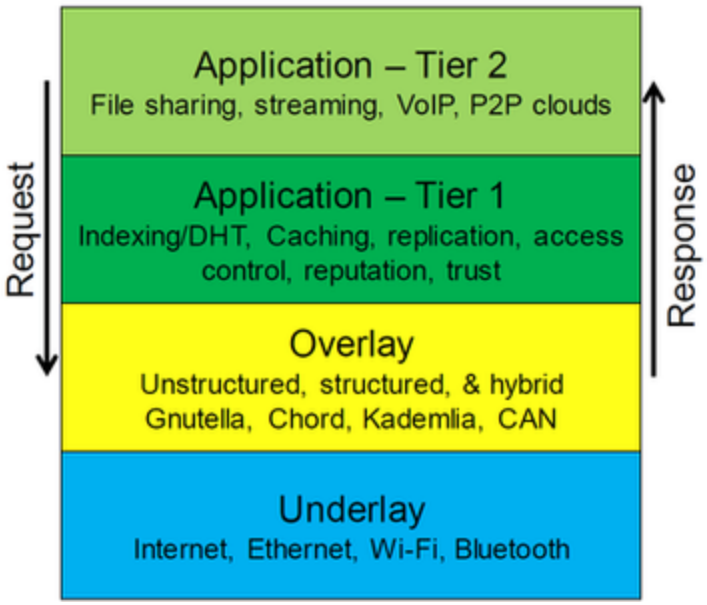
15) Explain about the following architectural styles in DS with a neat diagram

a. Layered architecture b. Object based c.Data centered d. Event based

Ans)

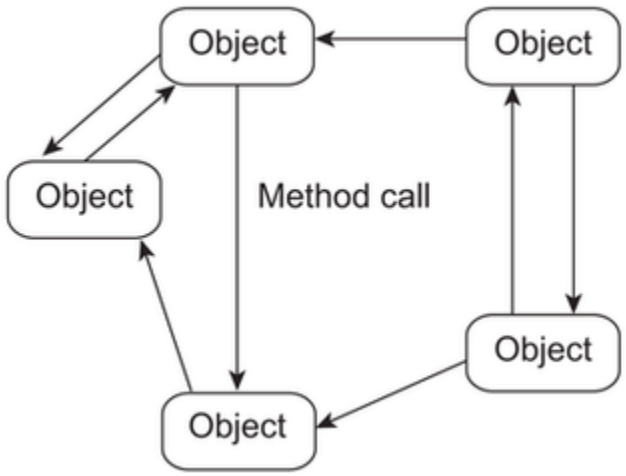
a. The layered architecture separates layers of components from each other, giving it a much more modular approach. A well known example for this is the OSI model that incorporates a layered architecture when interacting with each of the components. Each interaction is sequential where a layer will contact the adjacent layer and this process continues, until the request is catered to.But in certain cases, the implementation can be made so that some layers will be skipped, which is called cross-layer coordination. Through cross-layer coordination, one can obtain better results due to performance increase.The layers on the bottom provide a service to the layers on the top. The request flows from top to bottom, whereas the response is sent from bottom to top.





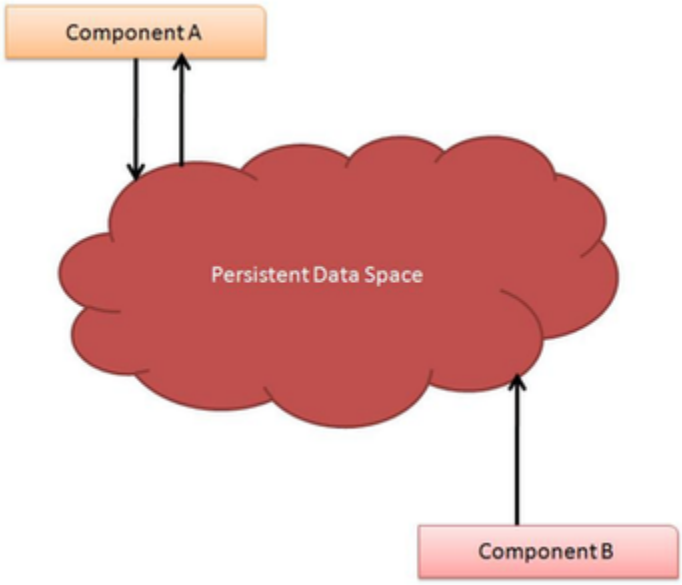
Layered Architecture

b.This architecture style is based on loosely coupled arrangement of objects. This has no specific architecture like layers. Like in layers, this does not have a sequential set of steps that needs to be carried out for a given call. Each of the components are referred to as objects, where each object can interact with other objects through a given connector or interface. These are much more direct where all the different components can interact directly with other components through a direct method call.



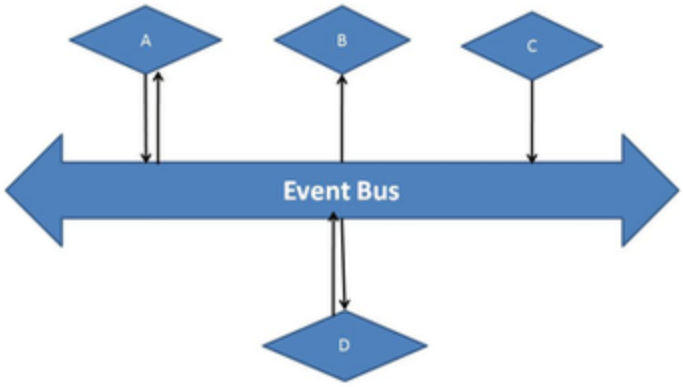
Object Based Architecture

c. this architecture is based on a data center, where the primary communication happens via a central data repository. This common repository can be either active or passive. This is more like a producer consumer problem. The producers produce items to a common data store, and the consumers can request data from it. This common repository could even be a simple database. But the idea is that, the communication between objects happening through this shared common storage.All the information related to the nodes in the system are stored in this persistent storage.



Data Centered Architecture

d.The entire communication in this kind of a system happens through events. When an event is generated, it will be sent to the bus system. With this, everyone else will be notified telling that such an event has occurred. So, if anyone is interested, that node can pull the event from the bus and use it. Sometimes these events could be data, or even URLs to resources. So the receiver can access whatever the information is given in the event and process accordingly. processes communicate through the propagation of events.These events occasionally carry data. An advantage in this architectural style is that, components are loosely coupled. So it is easy to add, remove and modify components in the system.



Event Based Architecture

15

16) Explain the approach of hybrid architecture?

16

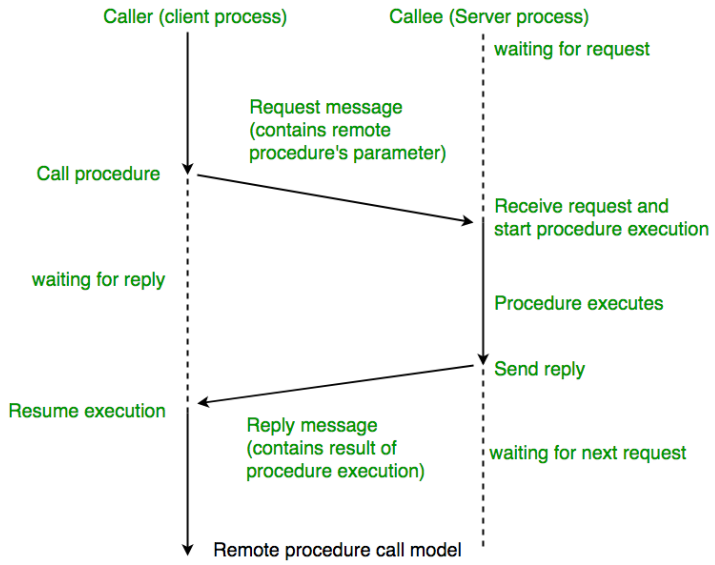
17) Explain different types of code migration.

17

18) Explain the basic operations of RPC With a Neat Diagram?

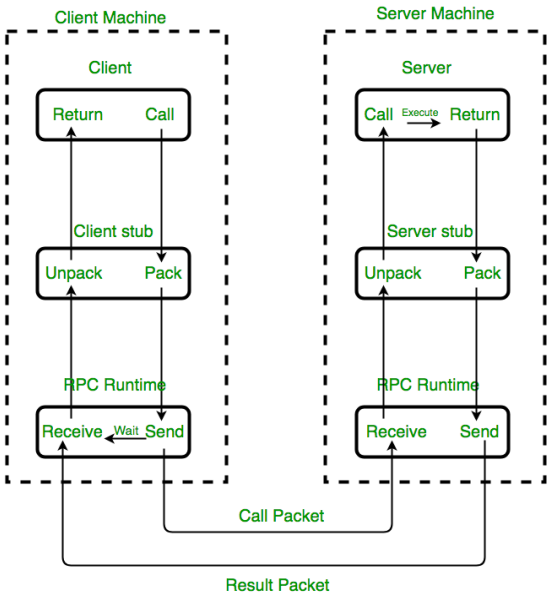
A remote procedure call is an interprocess communication technique that is used for client-server based applications. It is also known as a subroutine call or a function call.

A client has a request message that the RPC translates and sends to the server. This request may be a procedure or a function call to a remote server. When the server receives the request, it sends the required response back to the client. The client is blocked while the server is processing the call and only resumes execution after the server is finished.



The sequence of events in a remote procedure call are given as follows −

- The client stub is called by the client.
- The client stub makes a system call to send the message to the server and puts the parameters in the message.
- The message is sent from the client to the server by the client's operating system.
- The message is passed to the server stub by the server operating system.
- The parameters are removed from the message by the server stub.
- Then, the server procedure is called by the server stub.



Implementation of RPC mechanism

19) Explain Asynchronous RPC with examples?

20) Explain about Clock-Synchronization algorithms. (Christian's alg. Berkeley Unix, Multicast protocol)?

Berkeley's Algorithm:

1) An individual node is chosen as the master node from pool nodes in the network. This node is the main node in the network which acts as a master and rest of the nodes act as slaves. Master node is chosen using an election process/leader election algorithm.

2) Master node periodically pings slaves nodes and fetches clock time at them Using cristian algorithm.
Diagram below illustrates how the master sends request to slave nodes

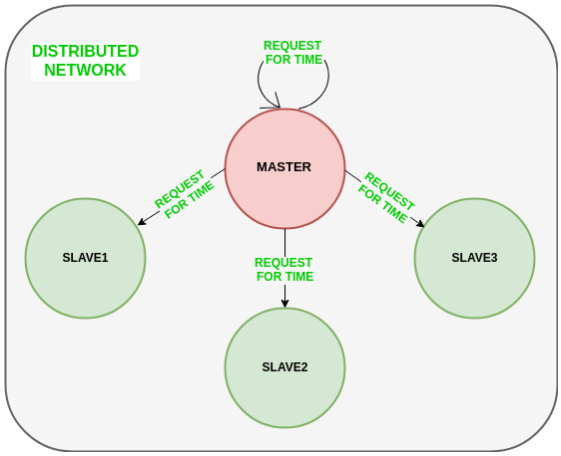
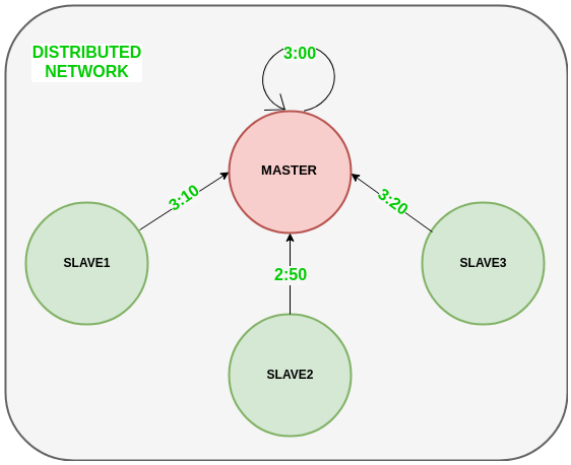


Diagram below illustrates how slave nodes send back time given by their system clock.



3) Master node calculates average time difference between all the clock times received and the clock time given by the master's system clock itself. This average time difference is added to the current time at master's system clock and broadcasted over the network.

Cristian's Algorithm:

1) The process on the client machine sends the request for fetching clock time(time at the server) to the Clock Server at time T0.

2) The Clock Server listens to the request made by the client process and returns the response in the form of clock server time.

3) The client process fetches the response from the Clock Server at timeT1 and calculates the synchronized client clock time using the formula given below.

$$T_{CLIENT} = T_{SERVER} + (T_1 - T_0)/2$$

where T_{CLIENT} refers to the synchronized clock time,

T_{SERVER} refers to the clock time returned by the server,

T_0 refers to the time at which request was sent by the client process,

T_1 refers to the time at which response was received by the client process

21) What is Mutual Exclusion? Explain about mutual-exclusion algorithms?

Mutual exclusion is a concurrency control property which is introduced to prevent race conditions. It is the requirement that a process can not enter its critical section while another concurrent process is currently present or executing in its critical section i.e only one process is allowed to execute the critical section at any given instance of time.

Mutual exclusion in single computer system Vs. distributed system:

In a single computer system, memory and other resources are shared between different processes. The status of shared resources and the status of users is easily available in the shared memory so with the help of shared variables (For example: Semaphores) mutual exclusion problem can be easily solved.

In Distributed systems, we neither have shared memory nor a common physical clock and there for we can not solve mutual exclusion problem using shared variables. To eliminate the mutual exclusion problem in distributed system approach based on message passing is used.

A site in distributed system do not have complete information of state of the system due to lack of shared memory and a common physical clock.