Unit 1 Chapter-2 (Short Notes)

Q.1) What is the difference between parallel and distributed computing?

>> There are mainly two computation types, including parallel computing and distributed computing. A computer system may perform tasks according to human instructions. A single processor executes only one task in the computer system, which is not an effective way. Parallel computing solves this problem by allowing numerous processors to accomplish tasks simultaneously. Modern computers support parallel processing to improve system performance. In contrast, distributed computing enables several computers to communicate with one another and achieve a goal. All of these computers communicate and collaborate over the network. Distributed computing is commonly used by organizations such as Facebook and Google that allow people to share resources.

1) Parallel computing is a sort of computation in which various tasks or processes are run at the same time. In contrast, distributed computing is that type of computing in which the components are located on various networked systems that interact and coordinate their actions by passing messages to one another.
2) In parallel computing, processors communicate with another processor via a bus. On the other hand, computer systems in distributed computing connect with one another via a network.
3) Parallel computing takes place on a single computer. In contrast, distributed computing takes place on several computers.
4) Parallel computing aids in improving system performance. On the other hand, distributed computing allows for scalability, resource sharing, and the efficient completion of computation tasks.
5) The computer in parallel computing can have shared or distributed memory. In contrast, every system in distributed computing has its memory.
6) Multiple processors execute multiple tasks simultaneously in parallel computing. In contrast, many computer systems execute tasks simultaneously in distributed computing.

Features	**Parallel Computing**	**Distributed Computing**
Definition	It is a type of computation in which various processes run simultaneously.	It is that type of computing in which the components are located on various networked systems that interact and coordinate their actions by passing messages to one another.
Communication	The processors communicate with one another via a bus.	The computer systems connect with one another via a network.
Functionality	Several processors execute various tasks simultaneously in parallel computing.	Several computers execute tasks simultaneously.
No. of computers	It occurs in a single computer system.	It involves various computers.
Memory	The system may have distributed or shared memory.	Each computer system in distributed computing has its own memory.
Usage	It helps to improve the system performance.	It allows for scalability, resource sharing, and the efficient completion of computation tasks.

Q.2) Identify the reason that parallel processing constituted an interesting option for computing?

>> Processing of multiple tasks simultaneously on multiple processors is called parallel processing. Parallel is also known as parallel processing. It utilizes several processors. Each of the processors completes the tasks that have been allocated to them. In other words, parallel computing involves performing numerous tasks simultaneously.

Parallel computing provides numerous advantages. Parallel computing helps to increase the CPU utilization and improve the performance because several processors work simultaneously. Moreover, the failure of one CPU has no impact on the other CPUs' functionality.

Applications of Parallel Computing –

- Databases and Data mining.
- Real-time simulation of systems.
- Science and Engineering.
- Advanced graphics, augmented reality, and virtual reality.

Why parallel computing? –

- The whole real-world runs in dynamic nature i.e. many things happen at a certain time but at different places concurrently. This data is extensively huge to manage.
- Computational requirements are ever increasing.
- Sequential architectures are reaching physical limitations.
- Hardware improvements are in the pipeline and require superscaling.
- Real-world data needs more dynamic simulation and modeling, and for achieving the same, parallel computing is the key.
- Parallel computing provides concurrency and saves time and money.
- Complex, large datasets, and their management can be organized only and only using parallel computing's approach.
- Ensures the effective utilization of the resources. The hardware is guaranteed to be used effectively whereas in serial computation only some part of the hardware was used and the rest rendered idle.
- Also, it is impractical to implement real-time systems using serial computing.

Future of Parallel Computing –

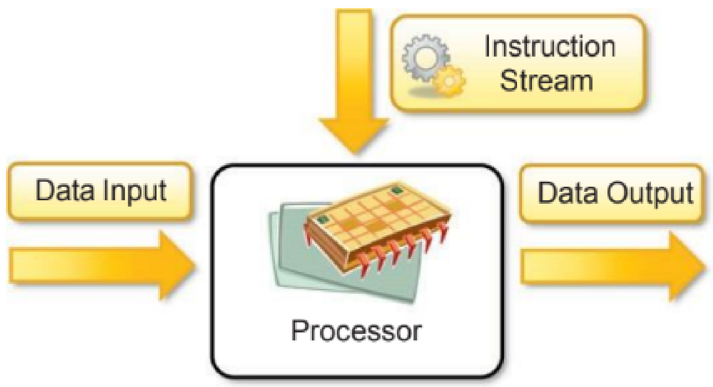
The computational graph has undergone a great transition from serial computing to parallel computing. Tech giants such as Intel have already taken a step towards parallel computing by employing multicore processors. Parallel computation will revolutionize the way computers work in the future, for the better good. With all the world connecting to each other even more than before, Parallel Computing does a better role in helping us stay that way. With faster networks, distributed systems, and multi-processor computers, it becomes even more necessary.

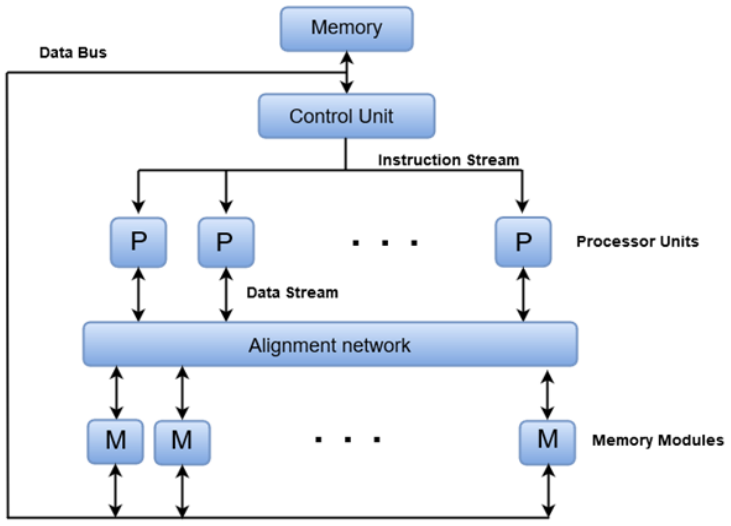
Q.3) What is a SIMD architecture? >>

1) SIMD stands for 'Single Instruction and Multiple Data Stream'.
2) It represents an organization that includes many processing units under the supervision of a common control unit.
3) A SIMD computing system is a multiprocessor machine capable of executing the same instruction on all the CPUs but operating on different data streams.
4) The shared memory unit must contain multiple modules so that it can communicate with all the processors simultaneously.
5) SIMD is mainly dedicated to array processing machines.

However, vector processors can also be seen as a part of this group.

6) The processor array is a collection of identical synchronized processing elements adequate for simultaneously implementing the same operation on various data. Each processor in the array has a small amount of local memory where distributed data resides while it is being processed in parallel.
7) The processor array is linked to the memory bus of the front end so that the front end can randomly create the local processor memories as if it were another memory.





Advantages –

- SIMD's primary advantage is its ability to conduct mathematical operations that are applied to large sets of data points in a much quicker manner than non-SIMD machines.
- In 3D graphics for example, when a player moves through a scene and the light source changes, the pixel of each visible image's brightness value has to be changed, a task that an SIMD very efficiently performs.
- Unlike other computer processors, SIMD does not conduct these operations in rapid sequences, but loads as many as possible into memory to execute the command(s) in parallel.

Disadvantages –

- SIMD technology cannot be applied to all algorithms. For example, parsing applications cannot be vectorized easily and do not gain from SIMD.
- The technology also consumes quite a bit of power and occupies a great deal of space on the microprocessor chip.
- SIMD instructions are very architecture specific and require re-coding to extend the application(s) that is using it across platforms.

Q.4) List the major categories of Parallel Computing System?

>> Major Categories of Parallel Computing System comes under Hardware architectures for parallel processing. The core elements of parallel processing are CPUs. It is Based on the number of instruction and data streams that can be processed simultaneously.

The Parallel Computing systems are classified into four categories –

1. SISD (Single-instruction, single-data systems) –

- An SISD computing system is a uniprocessor machine capable of executing a single instruction, which operates on a single data stream.
- Most conventional computers are built using the SISD model.
- This model is popularly called sequential computers.

- All the instructions and data to be processed have to be stored in primary memory. The speed of the processing element in the SISD model is limited.
- Domina representative SISD systems are IBM PC, Macintosh, and workstations.

2. SIMD. (Single-instruction, multiple-data systems) –

- A SIMD computing system is a multiprocessor machine capable of executing the same instruction on all the CPUs but operating on different data streams.
- Machines based on the SIMD model are well suited to scientific computing.
- These systems are Cray's vector processing.

3. MISD. (Multiple-instruction, single-data systems) –

- An MISD computing system is a multiprocessor machine capable of executing different instructions on different PEs but all of them operating on the same data set
- Machines built using the MISD model are not useful in most of the applications.

4. MIMD. (Multiple-instruction, multiple-data systems) –

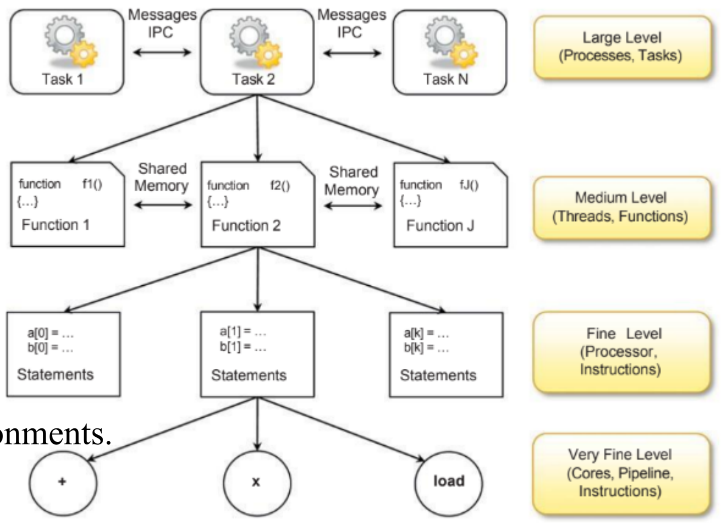
- An MIMD computing system is a multiprocessor machine capable of executing multiple instructions on multiple data sets; the MIMD model has separate instruction and data streams.
- The machines built using this model are well suited to any kind of application.
- Unlike SIMD and MISD machines, Pes in MIMD machines work asynchronously.
- MIMD machines are broadly categorized into shared-memory MIMD and distributed-memory MIMD.

Q.5) Describe the different levels of parallelism that can be obtained in a computing system.

>> Levels of parallelism are decided based on the lumps of code (grain size) that can be a potential candidate for parallelism. Following Table lists categories of code granularity for parallelism. All these approaches have a common goal to boost processor efficiency by hiding latency.

Grain Size	Code Item	Parallelized By
Large	Separate heavyweight process	Programmer
Medium	Function or procedure	Programmer
Fine	Loop or instruction block	Parallelizing compiler
Very Fine	Instruction	Processor

To conceal latency, there must be another thread ready to run whenever a lengthy operation occurs. The idea is to execute concurrently two or more single-threaded applications, such as compiling, text formatting, database searching, and device simulation.


Messages IPC
Messages IPC
Task 1
Task 2
Task N
Large Level (Processes, Tasks)
function f1() {...} Function 1
Shared Memory
function f2() {...} Function 2
Shared Memory
function fJ() {...} Function J
Medium Level (Threads, Functions)
a[0] = ... b[0] = ... Statements
a[1] = ... b[1] = ... Statements
a[k] = ... b[k] = ... Statements
Fine Level (Processor, Instructions)
+
x
load
Very Fine Level (Cores, Pipeline, Instructions)

As shown in the table and depicted in Figure, parallelism within an application can be detected at several levels –

1) Large grain (or task level) –

- Task parallelism (also known as function parallelism and control parallelism) is a form of parallelization of computer code across multiple processors in parallel computing environments.

- Task parallelism focuses on distributing tasks concurrently performed by processes or threads across different processors.
- In contrast to data parallelism which involves running same task on different components of data, task parallelism is distinguished by running many different tasks at the same time on the same data.
- A common type of task parallelism is pipelining, which consists of moving a single set of data through a series of separate tasks where each task can execute independently of the others.

2) Medium grain (or control level) –

- Medium-grained parallelism is used relatively to fine-grained and coarse-grained parallelism.
- Medium-grained parallelism is a compromise between fine-grained and coarse-grained parallelism, where we have task size and communication time greater than fine-grained parallelism and lower than coarse-grained parallelism.
- Most general-purpose parallel computers fall in this category.
- Intel iPSC is example of medium-grained parallel computer which has a grain size of about 10ms.

3) Fine grain (data level) –

- In fine-grained parallelism, a program is broken down to a large number of small tasks. These tasks are assigned individually to many processors.
- The amount of work associated with a parallel task is low and the work is evenly distributed among the processors. Hence, fine-grained parallelism facilitates load balancing.
- As each task processes less data, the number of processors required to perform the complete processing is high. This in turn, increases the communication and synchronization overhead.
- Fine-grained parallelism is best exploited in architectures which support fast communication.
- Shared memory architecture which has a low communication overhead is most suitable for fine-grained parallelism.

4) Very fine grain (multiple-instruction issue) –

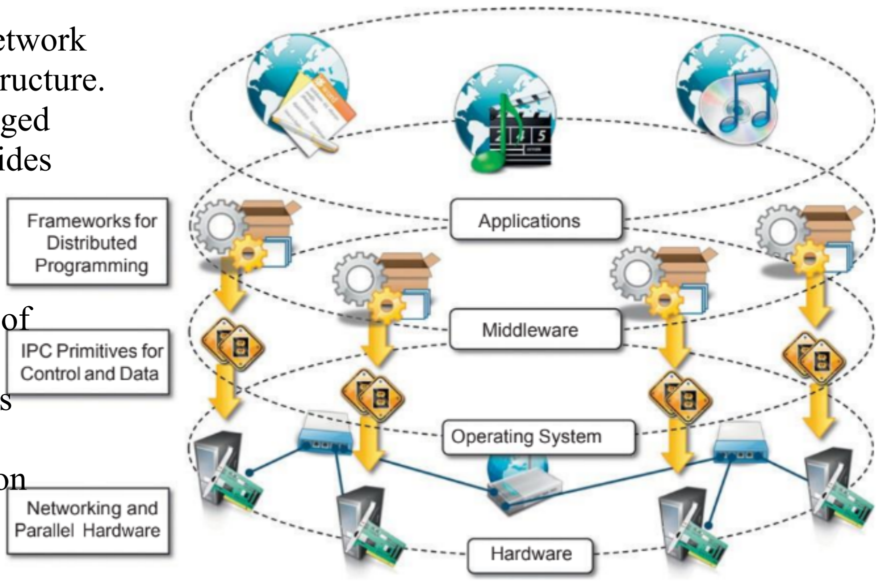
- Loop-level parallelism in computer architecture helps us with taking out parallel tasks within the loops in order to speed up the process.
- The utility for parallelism arises where data is stored in random access data structures like arrays.
- A program that runs in sequence will iterate over the array and perform operations on indices at a time, a program that has loop-level parallelism will use multi-threads/ multi-processes that operate on the indices at the same time or at different times.

Q.6) What is a distributed system? What are the components that characterize it? >>

1) A distributed system is a collection of independent computers that appears to its users as a single coherent system.
2) The architectural models that are used to harness independent computers and present them as a whole coherent system.
3) Communication is another fundamental aspect of distributed computing. Since distributed systems are composed of more than one computer that collaborate together, it is necessary to provide some sort of data and information exchange between them, which generally occurs through the network.
4) A distributed system is one in which components located at networked computers communicate and coordinate their actions only by-passing messages.
5) The components of a distributed system communicate with some sort of message passing.
6) A distributed system is the result of the interaction of several components that traverse the entire computing stack from hardware to software.
7) Figure provides an overview of the different layers that are involved in providing the services of a distributed system.

A) Hardware >>

- At a very low level, computer and network hardware constitutes physical infrastructure.
- These components are directly managed by the operating system, which provides basic services for interprocess communication (IPC), process determination and management, and resource management in the context of file systems and local devices.
- When taken together these two layers become a plat-form, above which special software is deployed to turn on a set of networked computers in a distributed system.


Frameworks for Distributed Programming
IPC Primitives for Control and Data
Networking and Parallel Hardware
Applications
Middleware
Operating System
Hardware

B) Operating System >>

- The use of even more known standards at the operating system level and hardware and network levels allows easy exploitation of heterogeneous components and their organization into a consistent and uniform system. For example, network connectivity between different devices is controlled by standards, which allows them to interact natively.
- At the operating system level, IPC services are applied to standardized communication protocols such as Transmission Control Protocol / Internet Protocol (TCP/IP), User Datagram Protocol (UDP), or others. Note that the hardware and operating system layers form a bare-bone infrastructure of one or more data centers, where racks of servers are deployed and connected together via high-speed connectivity. This infrastructure is managed by the operating system, which provides basic capabilities of machine and network management.

C) Middleware >>

- The Middleware layer leverages such services to create a uniform environment for the development and deployment of distributed services.
- This layer supports programming paradigms for distributed systems. By relying on the services offered by the operating system, Middleware develops its own protocols, data formats, and programming language or framework for developing distributed applications.
- All of them have formed a uniform interface for distributed developers that is completely independent from the underlying operating system and hides all the oddities of the bottom layers.
- Core logic is then implemented in middle-ware that manages the virtualization layer, which is deployed on the physical infrastructure to maximize its use and provide a customizable runtime environment for applications.
- Middleware provides different features to application developers according to the type of services sold to customers. These features, offered through the Web 2.0-compliant interfaces, range from virtual infrastructure building and deployment to application development and runtime environments.

D) Applications >>

- The top of the distributed system stack is represented by applications and services designed and developed to use middleware.

- These can serve multiple purposes and often highlight their features as a graphical user interface (GUI) accessible via the Internet, locally or through a web browser.
- For example, in the case of cloud computing systems, the use of web technologies is strongly preferred, not only to interface distributed applications with the end-user, but to provide platform services aimed at building distributed systems.

Q.7) What is an architectural style, and what is its role in the context of a distributed system? >>

1) An architectural style is a set of principles. You can think of it as a coarse-grained pattern that provides an abstract framework for a family of systems.
2) An architectural style improves partitioning and promotes design reuse by providing solutions to frequently recurring problems.
3) Architectural styles are mainly used to determine the vocabulary of components and connectors that are used as instances of the style together with a set of constraints on how they can be combined.
4) Design patterns help in creating a common knowledge within the community of software engineers and developers as to how to structure the relations of components within an application and understand the internal organization of software applications.
5) Architectural styles do the same for the overall architecture of software systems.
6) Architectural styles for distributed systems are helpful in understanding the different roles of components in the system and how they are distributed across multiple machines.
7) Architectural styles into two major classes –
    a) Software architectural styles.
    b) System architectural styles.
8) The first class relates to the logical organization of the software; the second class includes all those styles that describe the physical organization of distributed software systems in terms of their major components.
9) A component represents a unit of software that encapsulates a function or a feature of the system.
10) The example of components can be programs, objects, processes, pipes, and filters.
11) A connector is a communication mechanism that allows cooperation and coordination among components.
12) Differently from components, connectors are not encapsulated in a single entity, but they are implemented in a distributed manner over many system components.

Q.8) List the most important software architectural styles. >>

- Software architectural styles are based on the logical arrangement of software components.
- They are helpful because they provide an intuitive view of the whole system, despite its physical deployment.
- They also identify the main abstractions that are used to shape the components of the system and the expected interaction patterns between them.

Category	**Most Common Architectural Styles**
Data Centered	Repository, Blackboard
Data Flow	Pipe and Filter, Batch Sequential
Virtual Machine	Rule-based system, Interpreter

Call and Return	Main program and subroutine call, top-down systems, object-oriented systems, Layered Style
Independent components	Communicating process, Event Systems

Data Centered Architectures –

These architectures identify the data as the fundamental element of the software system, and access to shared data is the core characteristic of the data-centered architectures.

- Data Centered Architecture is a layered process which provides architectural guidelines in data center development.
- Data Centered Architecture is also known as Database Centric Architecture.
- This architecture is the physical and logical layout of the resources and equipment within a data center facility.
- Data Centered Architecture serves as a blueprint for designing and deploying a data center facility.
- It is usually created in the data center design and constructing phase.
- This architecture specifies how these devices will be interconnected and how physical and logical security workflows are arranged.

There are two categories which differentiates the architecture flow of control -

1) Repository Architecture Style –

- Repository architecture is a collection of independent components which operate on central data structure.
- Information System, Programming Environments, Graphical Editors, AI Knowledge Bases, Reverse Engineering Systems are examples of Repository Architecture Style.
- It includes central data structure.
- Repository architecture style is very important for data integration introduced in a variety of applications including software development, CAD etc.
- Repository architecture style makes changes to the data structure trigger computations.
- This architecture is suitable for applications in which the central issues are establishing and maintaining a complex central body of information.

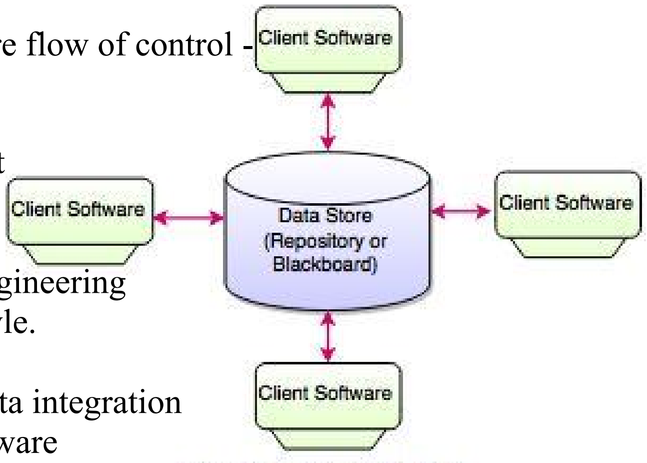


Fig. Data Centered Architecture

Advantages of Repository Architecture Style

- Repository Architecture Style provides data integrity, backup and restore features.
- It reduces overhead of transient data between software components.
- It has an efficient way to store large amounts of data.
- It has a centralized management which consists of backup, security and concurrency control.

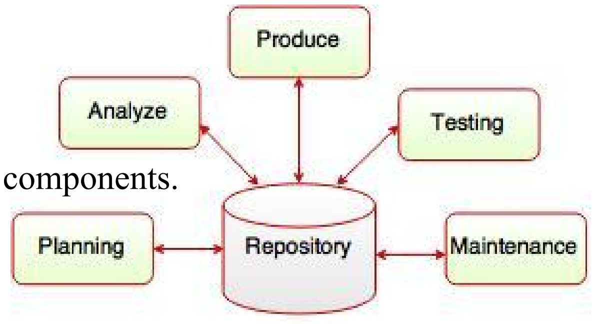


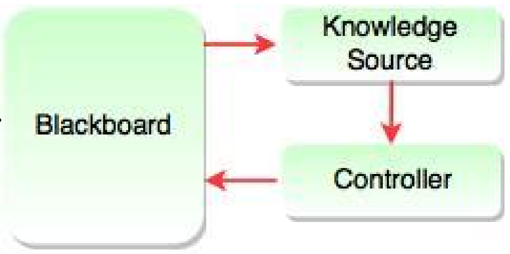
Fig. Repository Architecture Style

Disadvantages of Repository Architecture Style

- In the repository architecture style, evolution of data is difficult and expensive.
- It has a high dependency between the data structure of the data store and its software components or clients.

2) Blackboard Architecture Style –

- Blackboard architecture style is an artificial intelligence approach which handles complex problems, where the solution is the sum of its parts.
- In this architecture, the logical flow is determined by the current data status in the data store because the data store is active and its clients are passive.
- Blackboard architecture style has a blackboard component which acts as a central data repository.
- It is used in location-locomotion, data interpretation and environmental changes for solving the problem.
- It is an approach to processing agent communication centrally.
- There are major three components –
    - <u>Knowledge Source</u> -
        - KS is also known as Listeners or Subscribers. They are distinct and independent units.
        - Knowledge source solves the problem and aggregate partial results.
        - It provides specific expertise needed by the application.
        - The interaction between the knowledge source takes place uniquely through the blackboard.



    - <u>Blackboard</u> -
        - It is a shared repository of problems, solution, suggestion & contributed information.
        - It can be thought of as a dynamic library of information to the current problem which has been published by other knowledge sources.
        - In blackboard data structure, the problem solving state is organized into an application-dependent hierarchy and then knowledge source makes changes to the blackboard which leads incrementally to a solution to the problem.
    - <u>Control Shell</u> -
        - In the Control shell, it controls the flow of problem-solving activity in the system.
        - It manages the task and checks the work state.

Advantages of Blackboard Architecture Style

- Blackboard architecture style provides concurrency which allows knowledge sources to work in parallel.
- This architecture supports experimentation for hypotheses and reusability of knowledge source components.
- It allows blackboard applications to adapt to changing requirements.
- It allows new knowledge sources which can be developed and applied to the system without affecting the existing system.
- It performs more than one knowledge source in the same function and therefore it helps to improve both problem solving efficiency and the quality of the eventual solution.
- New applications can easily be constructed using existing knowledge sources.

Disadvantages of Blackboard Architecture Style

- Blackboard architecture style has the provision of tight dependency between the blackboard and knowledge source.
- It has difficulty in making a decision for reasoning termination.
- It has an issue in synchronization of multiple agents.

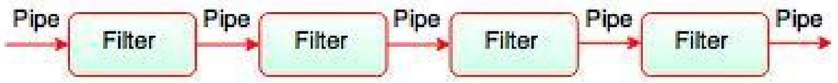
Data Flow –

- Data Flow Architecture is transformed input data by a series of computational or manipulative components into output data.
- It is a computer architecture which does not have a program counter and therefore the execution is unpredictable which means behavior is indeterministic.
- Data flow architecture is a part of Von-neumann model of computation which consists of a single program counter, sequential execution and control flow which determines fetch, execution, commit order.
- This architecture has been successfully implemented.
- Data flow architecture reduces development time and can move easily between design and implementation.
- Its main objective is to achieve the qualities of reuse and modifiability.
- In data flow architecture, the data can be flow in the graph topology with cycles or in a linear structure without cycles.

There are three types of execution sequences between modules:

1) Batch Sequential –

- Batch sequential compilation was regarded as a sequential process in 1970.
- In Batch sequential, separate programs are executed in order and the data is passed as an aggregate from one program to the next.
- It is a classical data processing model.



- The above diagram shows the flow of batch sequential architecture. It provides simpler divisions on subsystems and each subsystem can be an independent program working on input data and produces output data.
- The main disadvantage of batch sequential architecture is that it does not provide concurrency and interactive interface. It provides high latency and low throughput.

2) Pipe and Filter –

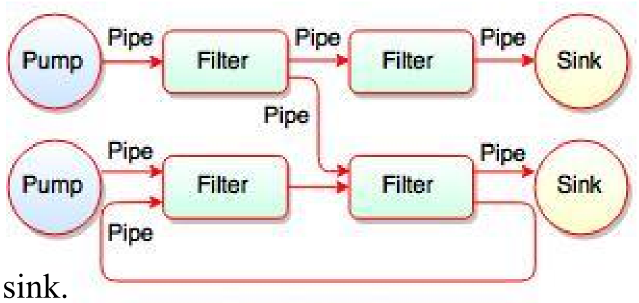


Fig. Pipes and Filters

- Pipe is a connector which passes the data from one filter to the next.
- Pipe is a directional stream of data implemented by a data buffer to store all data, until the next filter has time to process it.
- It transfers the data from one data source to one data sink.
- Pipes are the stateless data stream.
- The above figure shows the pipe-filter sequence. All filters are the processes that run at the same time, it means that they can run as different threads, coroutines or be located on different machines entirely.
- Each pipe is connected to a filter and has its own role in the function of the filter. The filters are robust where pipes can be added and removed at runtime.
- Filter reads the data from its input pipes and performs its function on this data and places the result on all output pipes. If there is insufficient data in the input pipes, the filter simply waits.

What are The Filters –

- Filter is a component.
- It has interfaces from which a set of inputs can flow in and a set of outputs can flow out.

- It transforms and refines input data.
- Filters are the independent entities.
- There are two strategies to construct a filter:
    - **Active filter** derives the data flow on the pipes.
    - **Passive filter** is driven by the data flow on the pipes.
- Filter does not share state with other filters.
- They don't know the identity of upstream and downstream filters.
- Filters are implemented by separate threads. These may be either hardware or software threads or coroutines.

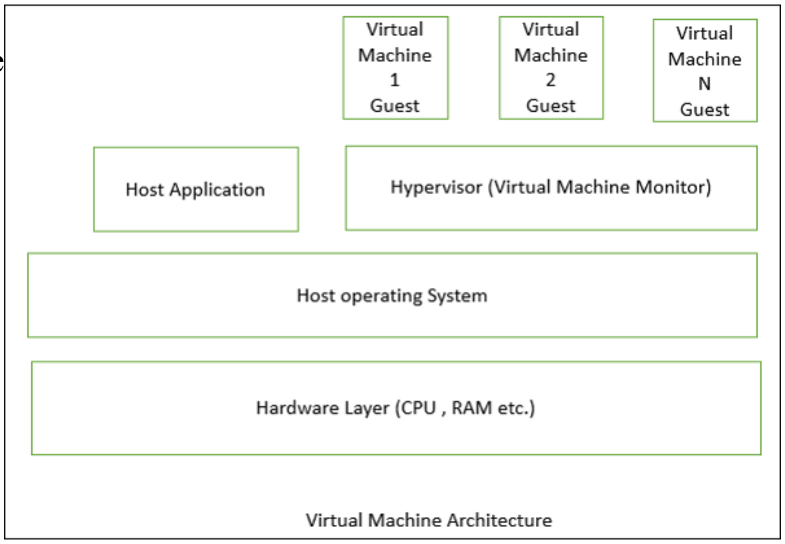
Advantages of Pipes and Filters

- Pipe-filter provides concurrency and high throughput for excessive data processing.
- It simplifies the system maintenance and provides reusability.
- It has low coupling between filters and flexibility by supporting both sequential and parallel execution.

Disadvantages of Pipes and Filters

- Pipe and Filter are not suitable for dynamic interactions.
- It needs a low common denominator for transmission of data in ASCII format.
- It is difficult to configure Pipe-filter architecture dynamically.

Virtual Machine –



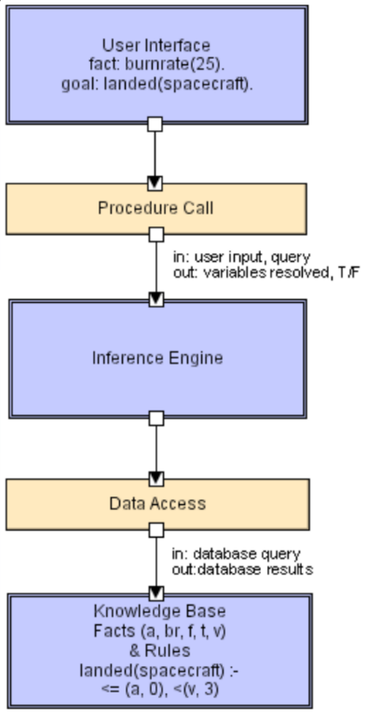
- Virtual Machine architecture pretends some functionality, which is not native to the hardware and/or software on which it is implemented.
- A virtual machine is built upon an existing system and provides a virtual abstraction, a set of attributes, and operations.
- In most cases, a virtual machine splits a programming language or application environment from an execution platform.
- The examples of virtual machines are rule-based systems, syntactic shells and command language processors.
- It introduces modifications at runtime and provides flexibility through the ability to interrupt.
- It provides portability and machine platform independence.

The Two examples of virtual machines are :

1) Rule-based system –

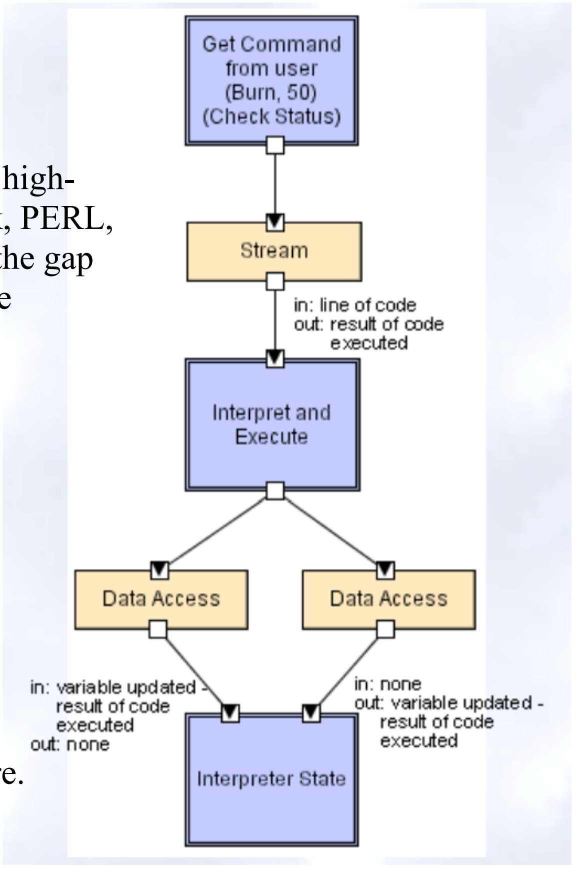
- Programs are expressed in the form of rules or predicates that hold true.
- The input data for applications is generally represented by a set of assertions or facts.
- The output can either be the product of the rule activation or a set of assertions that holds true for the given input data.
- The set of rules or predicates identifies the knowledge base that can be queried to infer properties about the system. This approach is quite peculiar, since it allows expressing a system or a domain in terms of its behavior rather than in terms of the components.
- Rule-based systems are very popular in the field of artificial intelligence.

- Practical applications can be found in the field of process control, where rule-based systems are used to monitor the status of physical devices by being fed from the sensory data collected and processed by PLCs. (A programmable logic controller (PLC)
- Another interesting use of rule-based systems can be found in the networking domain: network intrusion detection systems (NIDS) often rely on a set of rules to identify abnormal behaviors connected to possible intrusions in computing systems.
- Caution: When a large number of rules are involved, understanding the interactions between multiple rules affected by the same facts can become very difficult.



2) Interpreter –

- The core feature of the interpreter style is the presence of an engine
- used to interpret a pseudo-program expressed in a format acceptable
- for the interpreter.
- The interpretation of the pseudo-program constitutes the execution of the program itself.
- Systems modeled according to this style exhibit four main components:
  - The interpretation engine that Executes the core activity of this style.
  - An Internal Memory that contains the pseudo-code to be interpreted.
  - A Representation of the current state of the engine.
  - A Representation of the current state of the program being executed.
- This model is quite useful in designing virtual machines for high-level programming (Java, C#) and scripting languages (Awk, PERL, and so on). Within this scenario, the virtual machine closes the gap between the end-user abstractions and the software/hardware environment in which such abstractions are executed.



Advantages of Virtual Machine

- Portability and machine platform independency.
- Simplicity of software development.
- Provides flexibility through the ability to interrupt and query the program.
- Simulation for disaster working model.
- Introduce modifications at runtime.

Disadvantages of Virtual Machine

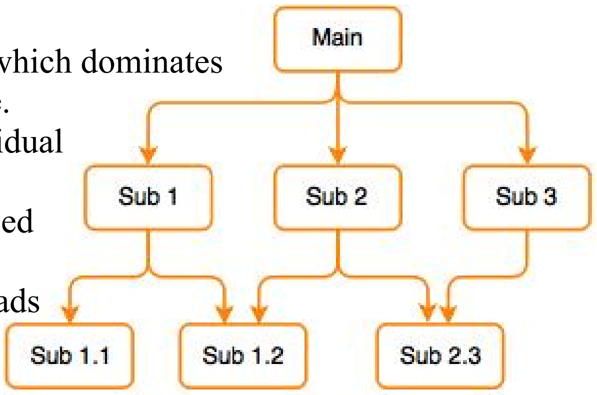
- Slow execution of the interpreter due to the interpreter nature.
- There is a performance cost because of the additional computation involved in execution.

Call and Return –

This category identifies all systems that are organized into components mostly connected together by method calls.

There are four major subcategories, which differentiate by the way the system is structured and how methods are invoked -

1) Main program and Subroutine Call –

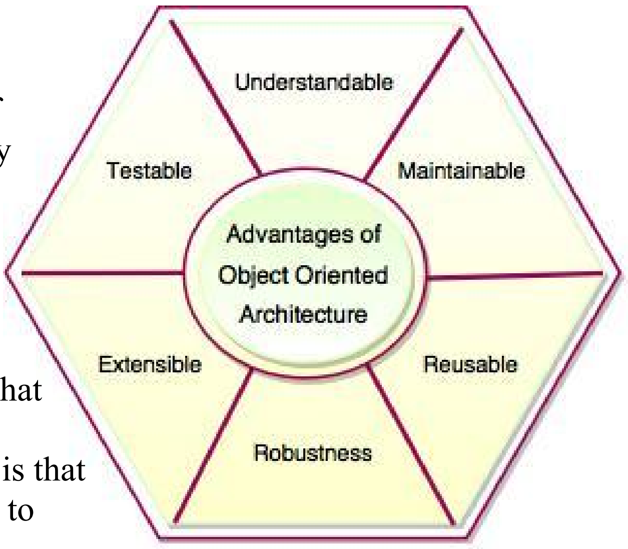


- Main-subroutine is a style of hierarchical architecture which dominates the software design methodologies for a very long time.
- Main-subroutine reuses the subroutines and have individual subroutines developed independently.
- Using main-subroutine, a software system is decomposed into subroutines hierarchically refined according to the desired functionality of the system and each module reads input files and write output files.
- Main-subroutine has a benefit, it is easy to decompose the system based on the definition of the tasks in a top-down refinement manner.
- Main-subroutine has a limitation because of tight coupling; it may cause more ripple effects of changes as compared to object-oriented design.

2) Top-Down Systems –

- This architectural style is quite representative of systems developed with imperative programming, which leads to a divide-and-conquer approach to problem resolution.
- Systems developed according to this style are composed of one large main program that accomplishes its tasks by invoking subprograms or procedures.
- Components in this style are procedures and subprograms, and connections are method calls or invocation. The calling program passes information with parameters and receives data from return values or parameters.
- Method calls can also extend beyond the boundary of a single process by leveraging techniques for remote method invocation, such as remote procedure call (RPC) and all its descendants.
- This architectural style is quite intuitive from a design point of view but hard to maintain and manage in large systems.

3) Object-Oriented Systems –



- This architectural style encompasses a wide range of systems that have been designed and implemented by leveraging the abstractions of object oriented programming (OOP).
- Systems are specified in terms of classes and implemented in terms of objects.
- Classes define the type of components by specifying the data that represent their state and the operations that can be done over these data.
- One of the main advantages over the top-down style is that there is a coupling between data and operations used to manipulate them.
- Object instances become responsible for hiding their internal state representation and for protecting its integrity while providing operations to other components. This leads to a better decomposition process and more manage- able systems. Disadvantages of this style are mainly two –
    - Each object needs to know the identity of an object if it wants to invoke operations on it
    - Shared objects need to be carefully designed in order to ensure the consistency of their state.

Advantages of Object-Oriented Architecture Style

- This architecture maps the application to real world objects to make it more understandable.
- It is easy to maintain and improves the quality of the system due to program reuse.
- This architecture provides reusability through polymorphism and abstraction.
- It has the ability to manage the errors during execution. (Robustness)
- It has the ability to extend new functionality and does not affect the system.
- It improves testability through encapsulation.
- Object-Oriented architecture reduces the development time and cost.

Disadvantages of Object-Oriented Architecture Style

- Object-Oriented architecture has difficulty determining all the necessary classes and objects required for a system.
- It is difficult to complete a solution within estimated time and budget because object-oriented architecture offers a new kind of project management.
- This methodology does not lead to successful reuse on a large scale without an explicit reuse procedure.

4) Layered Style –

- In Layered style, it decomposes the system into a number of higher and lower layers and each layer has its responsibility.
- Using layered architecture, applications involve distinct classes of services that can be organized hierarchically and have clear divisions between core services, critical services, user interface services etc.
- Layered architecture design is based on incremental levels of abstraction.
- It is implemented by using component-based technology which makes the system much easier to allow for plug-and-play of new components.
- Using layered architecture, It is easy to decompose the system based on the definition of the tasks in a top-down refinement manner.

Independent components –

This class of architectural style models systems in terms of independent components that have their own life cycles, which interact with each other to perform their activities.

- Component-based architecture focuses on the decomposition of the design into individual functional or logical components that represent well-defined communication interfaces containing methods, events, and properties.
- It provides a higher level of abstraction and divides the problem into sub-problems, each associated with component partitions.
- The primary objective of component-based architecture is to ensure component reusability.

There are two major categories in independent components -

1) Communicating process –

- In this architectural style, components are represented by independent processes that leverage IPC facilities for coordination management.
- This is an abstraction that is quite suitable to modeling distributed systems that, being distributed over a network of computing nodes, are necessarily composed of several concurrent processes.
- Each of the processes provides other processes with services and can leverage the services exposed by the other processes.

- The conceptual organization of these processes and the way in which the communication happens vary according to the specific model used, either peer-to-peer or client/server.
- Connectors are identified by IPC facilities used by these processes to communicate.

2) Event Systems –

- In this architectural style, the components of the system are loosely coupled and connected.
- In addition to exposing operations for data and state manipulation, each component also publishes (or announces) a collection of events with which other components can register.
- In general, other components provide a callback that will be executed when the event is activated
- During the activity of a component, a specific runtime condition can activate one of the exposed events, thus triggering the execution of the callbacks registered with it.
- Event activation may be accompanied by contextual information that can be used in the callback to handle the event.
- This information can be passed as an argument to the callback or by using some shared repository between components.
- Event-based systems have become quite popular, and support for their implementation is provided either at the API level or the programming language level.

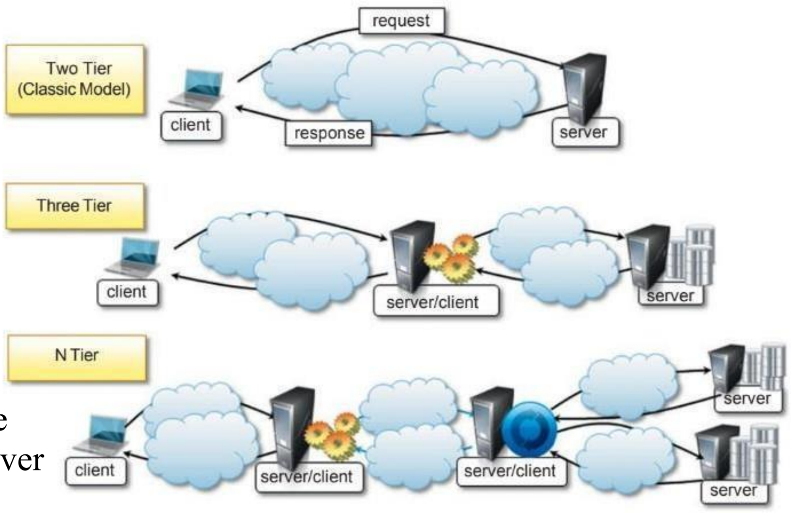
Q.9) What is the fundamental system architectural style?

>> System architectural styles cover the physical organization of components and processes over a distributed infrastructure.

In the system architectural style, two fundamental reference styles –

1) Client/Server –

- This architecture is very popular in distributed computing and is suitable for a wide variety of applications.
- As depicted in Figure, The Client/Server model features two major components -
    1) Server
    2) Client
- These two components interact with each other through a network connection using a given protocol.
- Communication is unidirectional.
  (The client issues a request to the server, and after processing the request the server returns a response, There could be multiple client components issuing requests to a server that is passively waiting for them.)



For the client design, we identify two major models -

Thin-client model:- In this model, the load of data processing and transformation is put on the server side, and the client has a light implementation that is mostly concerned with retrieving and returning the data it is being asked for, with no considerable further processing.

Fat-client model:- In this model, the client component is also responsible for processing and transforming the data before returning it to the user, whereas the server features a relatively light implementation that is mostly concerned with the management of access to the data.

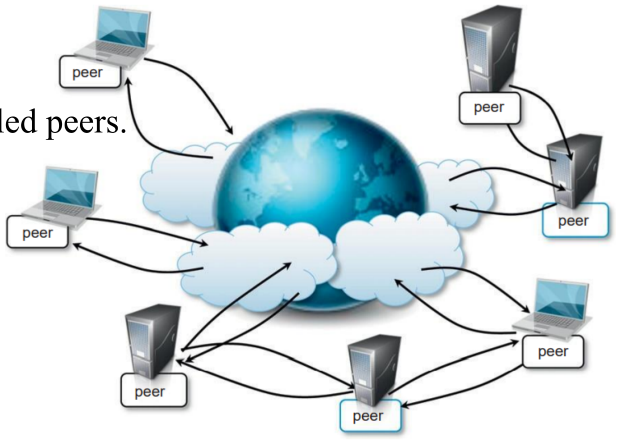
The three major components in the client-server model - Presentation, application logic, and data maintenance can be seen as conceptual layers, which are more appropriately called Tiers.

The mapping between the conceptual layers and their physical implementation in modules and components allows differentiating among several types of architectures, which go under the name of Multi-Tiered architectures.

Two-Tier Architecture – This architecture partitions the systems into two tiers, which are located one in the client component and the other on the server. The client is responsible for the presentation tier by providing a user interface.

Three-Tier architecture/N-Tier Architecture – The three-tier architecture separates the presentation of data, the application logic, and the data storage into three tiers. This architecture is generalized into an N-tier model in case it is necessary to further divide the stages composing the application logic and storage tiers. This model is generally more scalable than the two-tier one because it is possible to distribute the tiers into several computing nodes, thus isolating the performance bottlenecks.

2) Peer-To-Peer –



- The peer-to-peer model, depicted in Figure introduces a symmetric architecture in which all the components, called peers.
- It plays the same role and incorporates both client and server capabilities of the client/server model.
- Each peer acts as a server when it processes requests from other peers and as a client when it issues requests to other peers.
- This model is quite suitable for highly decentralized architecture, which can scale better along the dimension of the number of peers.
- The disadvantage of this approach is that the management of the implementation of algorithms is more complex than in the client/server model.
- Examples of peer-to-peer systems are constituted by file-sharing applications such as Gnutella, BitTorrent, and Kazaa. Another interesting example of peer-to-peer architecture is represented by the Skype network.

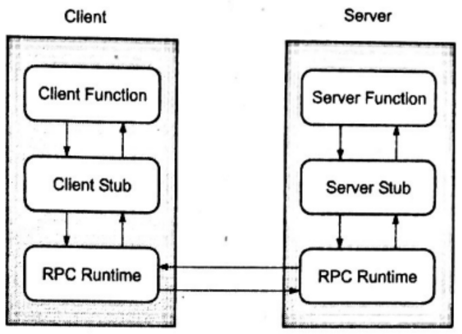
Q.10) What is the most relevant abstraction for interprocess communication in a distributed system? >>

- Distributed systems are made up of a collection of concurrent processes interacting with each other through a network connection.
- IPC (Inter-process communications) is a fundamental aspect of distributed systems design and implementation.
- IPC is either used for the exchange of data & information or to coordinate the activity of processes.
- IPC is that which binds together the various components of a distributed system, thus making them function as a single system.
- There are many different models in which processes can interact with each other.
- These maps are in different abstracts for IPC.
- The most relevant that we can mention is **shared memory**, **remote procedure call (RPC)**, and **message passing**.

1) **Shared Memory -** IPC through shared memory is a concept where two or more processes can access the common memory. And communication is done via this shared memory where changes made by one process can be viewed by another process. Shared memory provides a way by letting

two or more processes share a memory segment. With Shared Memory the data is only copied twice – from input file into shared memory and from shared memory to the output file.

2) **Remote Procedure Call -** RPC is one of the IPC mechanisms that are utilized for client-server applications. It is also called as a function call or a subroutine call. RPC translates a client request or call and send it to the server. After that the server receives the request and provide an appropriate response to the client. All this happens in a synchronous way, client is blocked till server is processing the request or call and it can resume its execution when the server is finished with its processing of the call.



3) **Message Passing –** In the distributed systems two applications can communicate with each other by means of exchanging messages. Messaging provides Asynchronous, high speed, program-to-program communication with reliable or trusted delivery. Two operations involved in message communication : send and receive, specified in form of source, destinations and messages.
   a) **Serialization or Marshalling :** is the mechanism of dividing a message into small pieces, so that it can be transferred easily over a network in the form in which it can be reconstructed perfectly at the receiver side.
   b) **Deserialization or Unmarshalling :** is the mechanism of combining gathered pieces of message in the form of bytes at receiver side to generate original message same as sender's message.

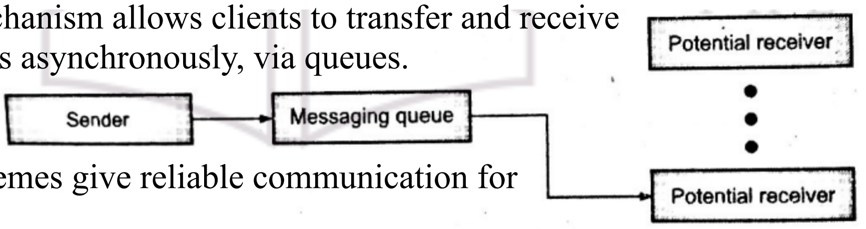
Q.11) Discuss the most important model for message-based communication. >>

- In the development of models and technologies, message abstraction is a necessary aspect that enables distributed computing.
- The term 'message' identifies any discrete amount of information that is passed from one entity to another. It encompasses any form of data representation that is limited in size and time, whereas this is an invocation to a remote procedure or a serialized object instance or a generic message.
- Therefore, the term message-based communication model can be used to refer to any model for IPC, which does not necessarily rely on the abstraction of data streaming.
- Several distributed programming paradigms eventually use message-based communication despite the abstractions that are presented to developers for programming the interaction of distributed components.

Here are some of the most popular and important –

1) Point-To-Point Message Model –

- The point-to-point messaging mechanism allows clients to transfer and receive messages synchronously as well as asynchronously, via queues.



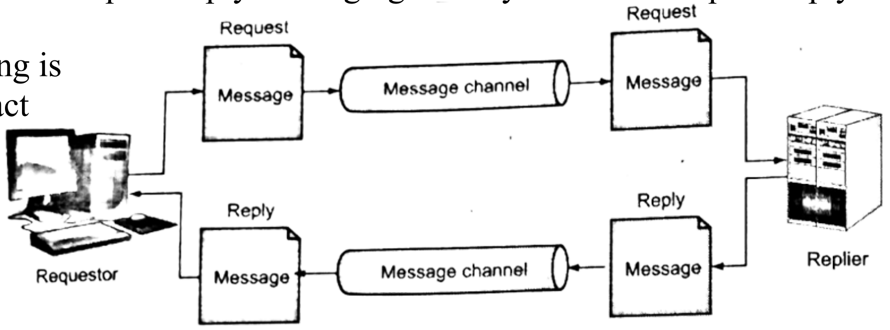
- The point-to-point messaging schemes give reliable communication for multi-staged applications.
- This technique was originally a pull-based or polling-based, where messages are stored in a queue and receive it from the queue, instead of automatically pushing the message directly to the receiver.
- The point-to-point messaging architecture is useful for implementing systems that are mostly based on one-to-one or many-to-one communication.

2) Publish-and-Subscribe Message Model –

- This mechanism is a highly scalable way of messaging.
- If a publisher application publishes messages on a particular topic, any number of subscribing applications can subscribe to this topic and receive the messages published by the publishing application.
    - Publishers publish messages on particular topics.
    - A message server maintains track of all the messages, and all its currently active and durable Subscribers. The message server gives security for the messaging system by providing authentication and authorization.
    - Whenever messages are published on a particular topic, they are transferred to all of its subscribers.
- A message will be provided to all subscribers who have registered for the related event. There are two major strategies for sending this event to customer :
    - Push strategy : In this strategy, It is the responsibility of the publisher to notify all subscribers. For example, with a method invocation.
    - Pull strategy : In this strategy, the publisher only provides messages for a specific event, and it is the responsibility of the subscriber to check whether there are messages on recorded events.
- The published-and-subscribe paradigm is very suited for implementing systems based on one-to-many communication models and make it easy, the implementation of indirect communication pat-turn.
- In fact, it is not necessary for the publisher to know the identity of the subscribers in order to create the communication.

3) Request-Reply Message Model –

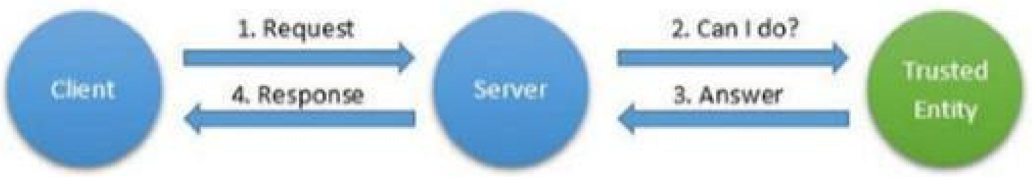
- On many events applications require that request/reply messaging transactions be performed.
- It can happen in two ways: synchronous request/reply messaging and asynchronous request/reply messaging.
- Synchronous request/reply messaging is mostly needed when trying to interact with a Web service client that blocks and wait for a synchronous response to get from the receiver.
- In the asynchronous mode of request/reply messaging, the sender expects the reply to arrive at a later time and continue its work with interruption.



- From figure, it is clear that in a simple request/reply asynchronous messaging scenario delivery channels are not bidirectional.
- To perform a request/reply transaction the sender and receiver should use two channels: one for the request and one for the reply.

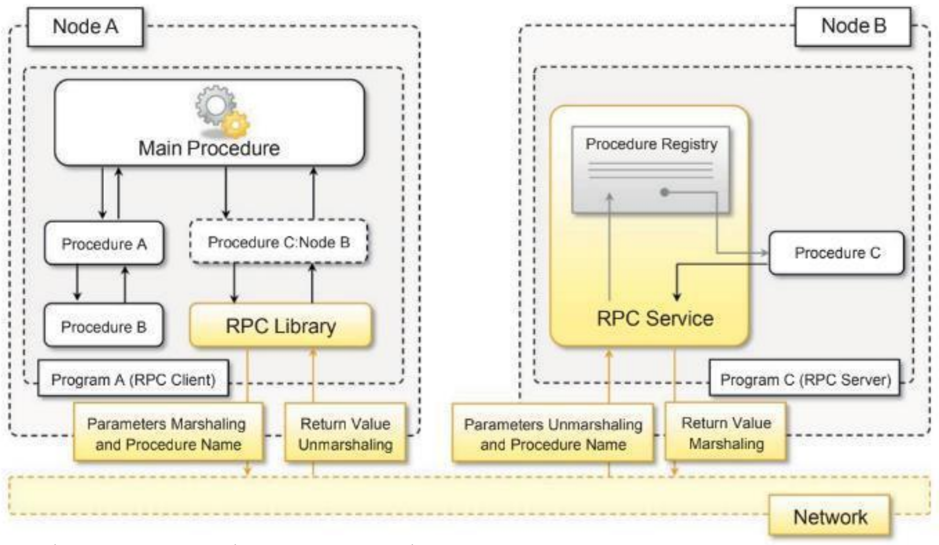
Q.12) Discuss RPC and how it enables interprocess communication? >>

- 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.
- RPC is the fundamental abstraction enabling the execution of procedures on client's request.



- It allows extending the concept of a procedure call beyond the boundaries of a process and a single memory address space. The called procedure and calling procedure may be on the same system or they may be on different systems in a network.

How RPC Enables IPC?



- RPC is a powerful, robust, efficient, and secure IPC mechanism that enables data exchange and invocation of functionality residing in a different process.
- RPCs are a form of inter-process communication (IPC), in that different processes have different address spaces.
- That different process can be on the same machine, on the local area network, or across the Internet.
- This section explains the RPC programming model and the model for distributed systems that can be implemented using RPC.
- RPC maintains the synchronous pattern that is natural in IPC and function calls. Therefore, the calling process thread remains blocked until the procedure on the server process has completed its execution and the result (if any) is returned to the client.
- The RPC runtime, on the other hand, is not only responsible for parameter packing and unpacking but also for handling the request-reply interaction that happens between the client and the server process in a completely transparent manner.
- Therefore, developing a system leveraging RPC for IPC consists of the following steps -
    - Design & implementation of the server procedures that'll be exposed for remote invocation.
    - Registration of remote procedures with the RPC server on the node where they'll be made available.
    - Design and implementation of the client code that invokes the remote procedure(s).

Advantages of Remote Procedure Call –

- Remote procedure calls support process oriented and thread-oriented models.
- The internal message passing mechanism of RPC is hidden from the user.
- The effort to re-write and re-develop the code is minimal in remote procedure calls.
- Remote procedure calls can be used in distributed environments as well as the local environment.
- Many of the protocol layers are omitted by RPC to improve performance.

Disadvantages of Remote Procedure Call –

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

Q.13) What is the difference between distributed objects and RPC? >>

Remote procedure call (RPC) –

- This paradigm extends the concept of procedure call beyond the boundaries of a single process, thus triggering the execution of code in remote processes. In this case, underlying client/server architecture is implied.
- A remote process hosts a server component, thus allowing client processes to request the invocation of methods, and returns the result of the execution.
- Messages, automatically created by the RPC implementation, convey the information about the procedure to execute along with the required parameters and the return values.
- The use of messages within this context is also referred to as marshaling of parameters and return values.
- RPC is the fundamental abstraction enabling the execution of procedures on client's request.
- RPC allows extending the concept of a procedure call beyond the boundaries of a process and a single memory address space. The called procedure and calling procedure may be on the same system or they may be on different systems in a network.
- The concept of RPC has been discussed since 1976 and completely formalized by Nelson and Birrell in the early 1980s. From there on, it has not changed in its major components.
- Even though it is a quite old technology, RPC is still used today as a fundamental component for IPC in more complex systems.
- An important aspect of RPC is marshaling, which identifies the process of converting parameter and return values into a form that is more suitable to be transported over a network through a sequence of bytes. The term unmarshaling refers to the opposite procedure.
- Each RPC implementation generally provides client and server application programming interfaces (APIs) that facilitate the use of this simple and powerful abstraction.
- For instance, RPyC is an RPC implementation for Python. There also exist platform independent solutions such as XML-RPC and JSON-RPC, which provide RPC facilities over XML and JSON, respectively.
- Thrift is the framework developed at Facebook for enabling a transparent cross-language RPC model.

Distributed Objects –

- This is an implementation of the RPC model for the object-oriented paradigm and contextualizes this feature for the remote invocation of methods exposed by objects.
- Each process registers a set of interfaces that are accessible remotely. Client processes can request a pointer to these interfaces and invoke the methods available through them.
- The underlying runtime infrastructure is in charge of transforming the local method invocation into a request to a remote process and collecting the result of the execution.
- The communication between the caller and the remote process is made through messages.
- With respect to the RPC model that is stateless by design, distributed object models introduce the complexity of object state management and lifetime.
- The methods that are remotely executed operate within the context of an instance, which may be created for the sole execution of the method, exist for a limited interval of time, or are independent from the existence of requests.
- Examples of distributed object infrastructures are Common Object Request Broker Architecture (CORBA), Component Object Model (COM, DCOM, and COM1), Java Remote Method Invocation (RMI), and .NET Remoting.
- Distributed agents and active objects. Programming paradigms based on agents and active objects involve by definition the presence of instances, whether they are agents of objects, despite the existence of requests. This means that objects have their own control thread, which allows them to carry out their activity.

- These models often make explicit use of messages to trigger the execution of methods, and a more complex semantics is attached to the messages.
- Distributed object frameworks extend object-oriented programming systems by allowing objects to be distributed across a heterogeneous network and provide facilities so that they can coherently act as though they were in the same address space.
- Distributed object frameworks leverage the basic mechanism introduced with RPC and extend it to enable the remote invocation of object methods and to keep track of references to objects made available through a network connection.
- Distributed object frameworks give the illusion of interaction with a local instance while invoking remote methods. This is done by a mechanism called a proxy skeleton.
- Proxy and skeleton always constitute a pair: the server process maintains the skeleton component, which is in charge of executing the methods that are remotely invoked, while the client maintains the proxy component, allowing its hosting environment to remotely invoke methods through the proxy interface.

Q.14) What are object activation and lifetime? How do they affect the consistency of state within a distributed system? >>

> What are Object Activation and Lifetime –

The management of distributed objects poses additional challenges with respect to the simple invocation of a procedure on a remote node. Methods live within the context of an object instance, and they can alter the internal state of the object as a side effect of their execution. In particular, the lifetime of an object instance is a crucial element in distributed object - oriented systems. Within a single memory address space scenario, objects are explicitly created by the programmer, and their references are made available by passing them from one object instance to another. The memory allocated for them can be explicitly reclaimed by the programmer or automatically by the runtime system when there are no more references to that instance. A distributed scenario introduces additional issues that require a different management of the lifetime of objects exposed through remote interfaces.

> How do they affect the consistency of state within a distributed system are as Follows –

- The first element to be considered is the object's activation, which is the creation of a remote object. Various strategies can be used to manage object activation, from which we can distinguish two major classes - 1) **server-based activation** and 2) **client-based activation**.
- In server-based activation, the active object is created in the server process and registered as an instance that can be exposed beyond process boundaries.
- In this case, the active object has a life of its own and occasionally executes methods as a consequence of a remote method invocation. In client-based activation the active object does not originally exist on the server side; it is created when a request for method invocation comes from a client.
- This scenario is generally more appropriate when the active object is meant to be stateless and should exist for the sole purpose of invoking methods from remote clients. For example, if the remote object is simply a gateway to access and modify other components hosted within the server process, client-based activation is a more efficient pattern.
- The second element to be considered is the lifetime of remote objects. In the case of server based activation, the lifetime of an object is generally user-controlled, since the activation of the remote object is explicit and controlled by the user.
- In the case of client-based activation, the creation of the remote object is implicit, and therefore its lifetime is controlled by some policy of the runtime infrastructure. Different policies can be considered;

- The simplest one implies the creation of a new instance for each method invocation. This solution is quite demanding in terms of object instances and is generally integrated with some lease management strategy that allows objects to be reused for subsequent method invocations if they occur within a specified time interval.
- Another policy might consider having only a single instance at a time, and the lifetime of the object is then controlled by the number and frequency of method calls.
- Different frameworks provide different levels of control of this aspect. Object activation and lifetime management are features that are now supported to some extent in almost all the frameworks for distributed object programming, since they are essential to understanding the behavior of a distributed system.
- In particular, these two aspects are becoming fundamental in designing components that are accessible from other processes and that maintain states.
- Understanding how many objects representing the same component are created and for how long they last is essential in tracking inconsistencies due to erroneous updates to the instance internal data.

Q.15) Discuss CORBA. >>

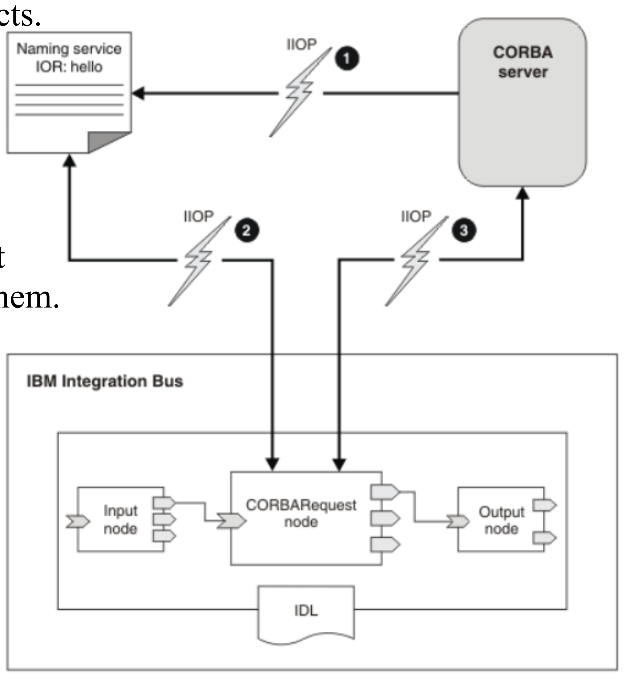
> Common Object Request Broker Architecture (CORBA) –

- The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together.
- CORBA is a standard for distributing objects across networks so that operations on those objects can be called remotely.
- CORBA is not associated with a particular programming language, and any language with a CORBA binding can be used to call and implement CORBA objects.
- Objects are described in a syntax called Interface Definition Language (IDL).

> CORBA includes 4 Components –

- **Object Request Broker (ORB) –** The ORB handles the communication, marshaling, and unmarshaling of parameters so that the parameter handling is transparent for a CORBA server and client applications.
- **CORBA Server –** It creates CORBA objects and initializes them with an ORB. The server places references to the CORBA objects inside a naming service so that clients can access them.
- **Naming Service –** It holds references to CORBA objects.
- **CORBA Request Node –** It acts as a CORBA client.

The following diagram shows the layers of communication between IBM Integration Bus and CORBA -



The diagram illustrates the following steps.

1) CORBA server applications create CORBA objects and put object references in a naming service so that clients can call them.

2) At deployment time, the node contacts a naming service to get an object reference.

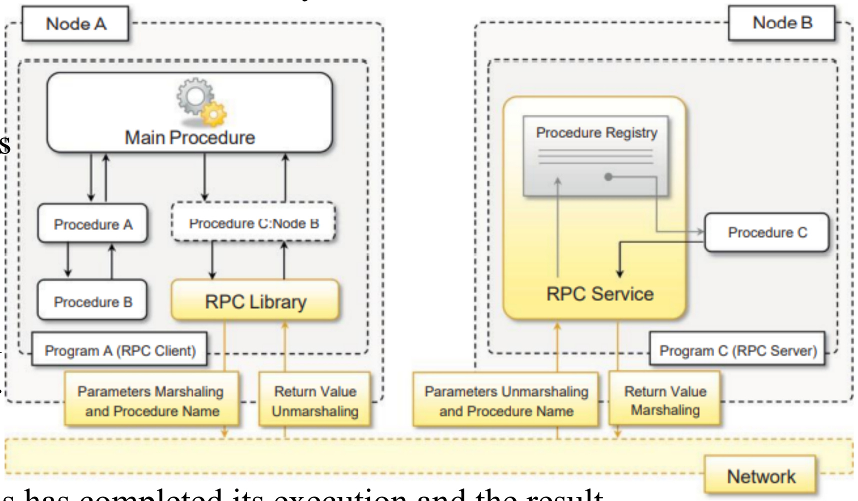
3) When a message arrives, the node uses the object reference to call an operation on an object in the CORBA server.

Q.16) What are the most relevant technologies for distributed object programming? >>

>> Distributed Computing Technologies that provide detailed implementations of interaction models,which mostly rely on message-based communication. These technologies are remote procedure call (RPC), distributed object frameworks, and service-oriented computing.

1) Remote Procedure Call (RPC) –

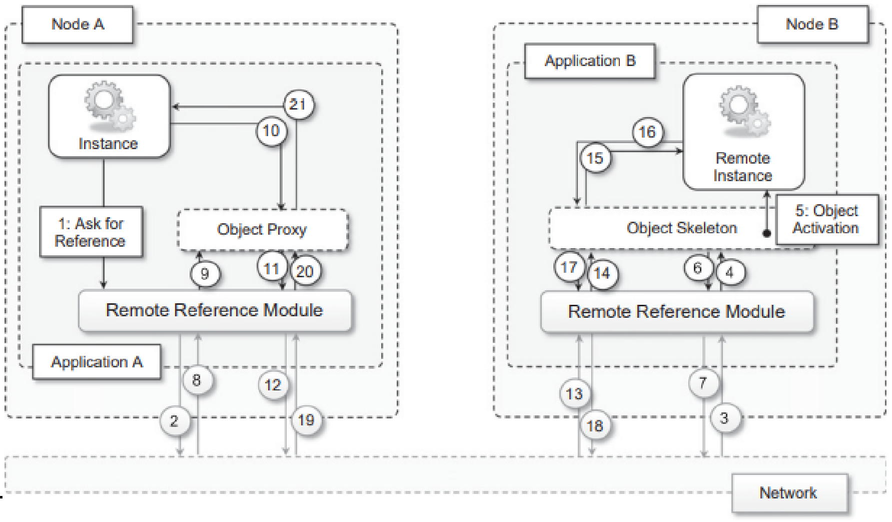
- RPC is the fundamental abstraction enabling the execution of procedures on client's request.
- RPC allows extending the concept of a procedure call beyond the boundaries of a process and a single memory address space. The called procedure and calling procedure may be on the same system or they may be on different systems in a network.
- The concept of RPC has been discussed since 1976 and completely formalized by Nelson and Birrell in the early 1980s.
- Figure illustrates the major components that enable an RPC system.
- The system is based on a client/server model. The server process maintains a registry of all the available procedures that can be remotely invoked and listens for requests from clients that specify which procedure to invoke, together with the values of the parameters required by the procedure.
- RPC maintains the synchronous pattern that is natural in IPC and function calls.
- Therefore, the calling process thread remains blocked.
  until the procedure on the server process has completed its execution and the result (if any) is returned to the client.



- An important aspect of RPC is marshaling, which identifies the process of converting parameter and return values into a form that is more suitable to be transported over a network through a sequence of bytes.
- The term unmarshaling refers to the opposite procedure. Marshaling and unmarshaling are performed by the RPC runtime infrastructure, and the client and server user code does not necessarily have to perform these tasks.
- Developing a system leveraging RPC for IPC consists of the following steps:
  - Design and implementation of the server procedures that'll be exposed for remote invocation.
  - Registration of remote procedures with the RPC server on the node where they'll be made available.
  - Design and implementation of the client code that invokes the remote procedure(s).
- RPyC is an RPC implementation for Python.
- Thrift is the framework developed at Facebook for enabling a transparent cross-language RPC model.
- The term RPC implementations encompass a variety of solutions including frameworks such as distributed object programming (CORBA, DCOM, Java RMI, and .NET Remoting) and Web services that evolved from the original RPC concept.

2) Distributed Object Frameworks –

- The Distributed Object Frameworks extend object-oriented programming systems by allowing objects to be distributed across a heterogeneous network and provide facilities so that they can coherently act as though they were in the same address space.
- Distributed object frameworks leverage the basic mechanism introduced with RPC and extend it to enable the remote invocation of object methods and to keep track of references to objects made available through a network connection.
- With respect to the RPC model, the infrastructure manages instances that are exposed through well-known interfaces instead of procedures. Therefore, the common interaction pattern is the following –
- The server process maintains a registry of active objects that are made available to other processes. According to the specific implementation, active objects can be published using interface definitions or class definitions
- The client process, by using a given addressing scheme, obtains a reference to the active remote object. This reference is represented by a pointer to an instance that is of a shared type of interface and class definition.
- The client process invokes the methods on the active object by calling them through the reference previously obtained. Parameters and return values are marshaled as happens in the case of RPC.
- Distributed object frameworks give the illusion of interaction with a local instance while invoking remote methods. This is done by a mechanism called a proxy skeleton.
- Figure gives an overview of how this infrastructure works.
- Proxy and skeleton always constitute a pair: the server process maintains the skeleton component, which is in charge of executing the methods that are remotely invoked, while the client maintains the proxy component, allowing its hosting environment to remotely invoke methods through the proxy interface.
- Distributed object frameworks introduce objects as first-class entities for IPC. They are the principal gateway for invoking remote methods but can also be passed as parameters and return values.



3) Service-Oriented Computing –

- SAME AS BELOW 👇

Q.17) What is service-oriented computing? >>

- Service orientation is the core reference model for cloud computing systems.
- This approach adopts the concept of services as the main building blocks of application and system development.
- Service-oriented computing (SOC) supports the development of rapid, low-cost, flexible, interoperable, and evolvable applications and systems.
- A service is an abstraction representing a self-describing and platform-agnostic component that can perform any function—anything from a simple function to a complex business process.
- Virtually any piece of code that performs a task can be turned into a service.

- A service is supposed to be loosely coupled, reusable, programming language independent, and location transparent.
- Loose coupling allows services to serve different scenarios more easily and makes them reusable. Independence from a specific platform increases services accessibility.
- Thus, a wider range of clients, which can look up services in global registries and consume them in a location-transparent manner, can be served
- Services are composed and aggregated into a service-oriented architecture (SOA), which is a logical way of organizing software systems to provide end users or other entities distributed over the network with services through published and discoverable interfaces.
- Service-oriented computing introduces and diffuses two important concepts, which are also fundamental to cloud computing: quality of service (QoS) and Software-as-a-Service (SaaS).

> Quality of service (QoS) –

- Quality of service identifies a set of functional and non-functional attributes that can be used to evaluate the behavior of a service from different perspectives.
- These could be performance metrics such as response time, or security attributes, transactional integrity, reliability, scalability, and availability.
- QoS requirements are established between the client and the provider via an SLA that identifies the minimum values (or an acceptable range) for the QoS attributes that need to be satisfied upon the service call.

> Software-as-a-Service (SaaS) –

- SaaS is also known as "On-Demand Software".
- It is a software distribution model in which services are hosted by a cloud service provider.
- These services are available to end-users over the internet so, the end-users do not need to install any software on their devices to access these services.

There are the following services provided by SaaS providers -

➔ **Business Services -** SaaS Provider provides various business services to start-up the business. The SaaS business services include **ERP** (Enterprise Resource Planning), **CRM** (Customer Relationship Management), **billing**, and **sales**.
➔ **Document Management -** SaaS document management is a software application offered by a third party (SaaS providers) to create, manage, and track electronic documents.
Example: Slack, Samepage, Box, and Zoho Forms.
➔ **Social Networks -** As we all know, social networking sites are used by the general public, so social networking service providers use SaaS for their convenience and handle the general public's information.
➔ **Mail Services -** To handle the unpredictable number of users and load on e-mail services, many email providers offer their services using SaaS.

Advantages of SaaS

- SaaS is easy to buy.
- One to Many.
- Less hardware required for SaaS.
- Low maintenance required for SaaS.
- No special software or hardware versions required.
- Multi Device support.
- No client-side installation.
- API Integration.

Disadvantages of SaaS

- Data is stored in the cloud, so security may be an issue for some users. However, cloud computing is not more secure than in-house deployment.
- There is a possibility that there may be greater latency when interacting with the application compared to local deployment.
- Without an internet connection, most SaaS applications are not usable. A Total Dependency on the Internet.
- Switching between SaaS vendors is difficult.

Q.18) What is market-oriented cloud computing? >>

- Computing is being transformed to a model consisting of services that are commoditised and delivered in a manner similar to utilities such as water, electricity, gas, and telephony.
- In such a model, users access services based on their requirements without regard to where the services are hosted.
- Several computing paradigms have promised to deliver this utility computing vision and they include Grid computing, P2P computing, and more recently Cloud computing.
- The latter term denotes the infrastructure as a “Cloud” in which businesses and users are able to access applications from anywhere in the world on demand.
- Hence, Cloud computing can be classed as a new paradigm for the dynamic creation of next-generation Data Centers by assembling services of networked Virtual Machines (VMs).
- Thus, the computing world is rapidly transforming towards developing software for millions to consume as a service rather than creating software for millions to run on their PCs.
- Market Oriented Cloud Computing goes one step further by allowing spread into multiple public and hybrid environments dynamically composed by trading service.
- Market Oriented Computing has the same characteristics as Cloud Computing; therefore it is a dynamically provisioned unified computing resource allowing you to manage software and data storage as on aggregate capacity resulting in “real-time” infrastructure across public and private infrastructures.
- Market-Oriented Cloud Computing (MOCC) From Cloud Computing, the presence of a virtual marketplace where IT services are traded and brokered dynamically.
- The presence of a demand based marketplace represents an opportunity for enterprises to shape their infrastructure for dynamically reacting to workload spikes and for cutting maintenance costs.
- It also allows the possibility to temporarily lease some in-house capacity during low usage periods, thus giving a better return on investment. These developments will lead to the complete realization of market oriented cloud computing.

Q.19) What is SOA? >>

- Service-oriented architecture (SOA) is an architectural style supporting service orientation.
- It organizes a software system into a collection of interacting services.
- SOA encompasses a set of design principles that structure system development and provide means for integrating components into a coherent and decentralized system.
- SOA-based computing packages functionalities into a set of interoperable services, which can be integrated into different software systems belonging to separate business domains.
- There are two major roles within SOA: 1) the service provider and 2) the service consumer.
- The service provider is the maintainer of the service and the organization that makes available one or more services for others to use.
- The provider can publish them in a registry, together with a service contract that specifies the nature of the service, how to use it, the requirements for the service, and the fees charged.

- The service consumer can locate the service metadata in the registry and develop the required client components to bind and use the service.
- Service providers and consumers can belong to different organization bodies or business domains.
- Service Orchestration, which more generally describes the automated arrangement, coordination, and management of complex computer systems, middleware, and services.
- Another important interaction pattern is Service Choreography, which is the coordinated interaction of services without a singlepoint of control.
- The Characteristics of SOA are as follows –
    - Standardized service contract
    - Loose coupling
    - Abstraction
    - Reusability
    - Autonomy
    - Lack of state
    - Discoverability
    - Interoperability
    - Composability
- SOA can be realized through several technologies. The first implementations of SOA have leveraged distributed object programming technologies such as CORBA and DCOM.

Advantages of SOA

- Easy to Integrate - In a service-oriented architecture, the integration is a service specification that provides implementation transparency.
- Manage Complexity - Due to service specification, the complexities get isolated, and integration becomes more manageable.
- Platform Independence - The services are platform-independent as they can communicate with other applications through a common language.
- Loose coupling - It facilitates the implementation of services without impacting other applications or services.
- Parallel Development - As SOA follows layer-based architecture, it provides parallel development.
- Available - The SOA services are easily available to any requester.
- Reliable - As services are small in size, it is easier to test and debug them.

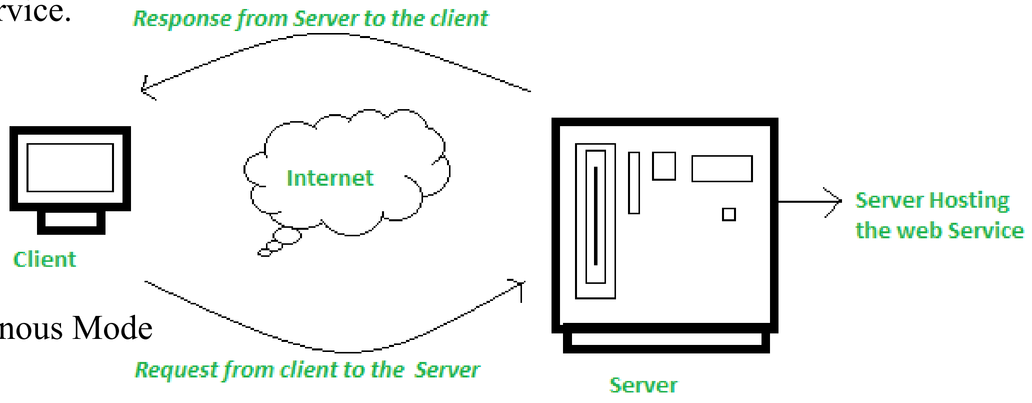
Q.20) Discuss the most relevant technologies supporting service computing. >>

- Service Computing has become a cross discipline that covers the science and technology of bridging the gap between business services and IT services. It is a new computing discipline with web service and service-oriented system architecture as basic supporting technology.
- The most relevant technology supporting the service computing are –

1) Web Service –

- The term web service is a software module which is available via the internet.
- It’s a self-describing, self-contained web application component used for client server communication.
- It's an outside application which gets coupled with other applications whenever they need that service to fulfill their requirements such as solve some problems, to get some information & extra resources.
- There are mainly two types of web services.
    - SOAP web services.
    - RESTful web services.

- So web service is any service that –
    - Is available over the network.
    - Is a piece of software for communication between two devices.
    - Is a protocol for exchanging information between devices.
    - Is a XML based service.

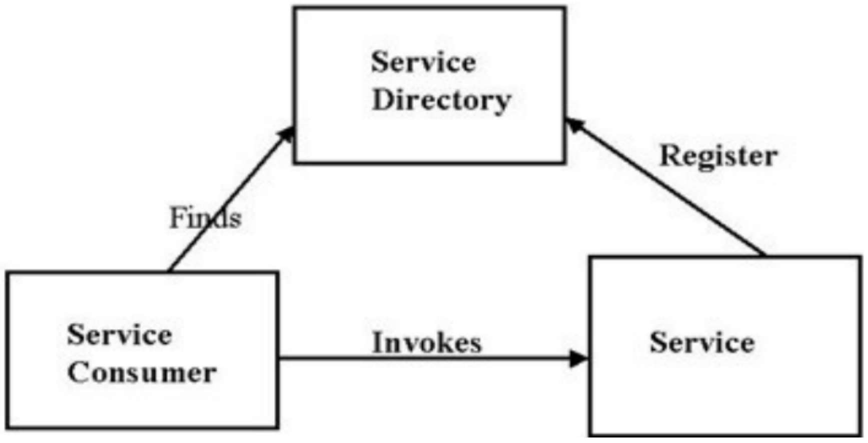

Response from Server to the client
Internet
Client
Server Hosting the web Service
Request from client to the Server
Server

Features of Web Service –

- Loosely Coupled
- XML based
- Interoperability
- Distributed in nature
- Supports RPC
- Synchronous & Asynchronous Mode
- Dynamic

2) Service Oriented Architecture (SOA) –

- Service orientation is the main reference model for cloud computing.
- This approach adopts the service concept as the main building block of application & system development.
- A SOA is a construction format in computer software design in which application parts to other parts by using standard protocols.
- Web services based on SOA architecture inclined to make web service extremely independent.
- It guarantees that web services on a network can communicate with each & other flawlessly.
- Refer Points from Q.19



Features of SOA –

- Loose coupling
- Service abstraction
- Service Reusability
- Service autonomy
- Service stateless
- Service composability
- Service Interoperability

Q.21) Write a short note on IaaS, PaaS, SaaS? >>

- IaaS, PaaS, and Saas are the three main categories of cloud computing.
- Cloud computing is using a network of different servers that host, store, manage, and process data online in the cloud.

Infrastructure As A Service (IaaS) –

- IAAS means delivering computing infrastructure as on-demand services. It is one of the three fundamental cloud service models.
- The user purchases servers, software data center space, or network equipment and rents those resources through a fully outsourced, on-demand service model.
- It allows dynamic scaling and the resources are distributed as a service. It generally includes multiple-users on a single piece of hardware.

- IaaS is the utilization of APIs to manage the lowest levels of network infrastructure, including networking, storage, servers, and virtualization.

Examples –

- **Amazon Web Services(AWS) :** AWS is overseen by amazon and is used for on-demand cloud computing and purchased on a recurring subscription basis. AWS helps companies store data and deliver content — in fact, it's helping you read this blog post right now.
- **Google Cloud :** Google Cloud is an IaaS platform that businesses can use to natively run Windows, Oracle, and SAP. Additionally, a business can manage its enterprise database and use AI solutions to increase operational efficiency within the firm.
- **IBM Cloud :** IBM Cloud is another IaaS product that allows businesses to "allocate your computer, network, storage and security resources on demand." In other words, businesses only use resources when needed, increasing efficiency.

Advantages of IaaS

- The resources can be deployed by the provider to a customer's environment at any given time.
- Its ability to offer the users to scale the business based on their requirements.
- The provider has various options when deploying resources including virtual machines, applications, storage, and networks.
- It has the potential to handle an immense number of users.
- It is easy to expand and saves a lot of money. Companies can afford the huge costs associated
- with the implementation of advanced technologies.

Platform As A Service (PaaS) –

- PAAS is a cloud delivery model for applications composed of services managed by a third party.
- It provides elastic scaling of your application which allows developers to build applications and services over the internet and the deployment models include public, private and hybrid.
- PaaS offers an even greater abstraction of cloud service, offering users the capability to build or deploy applications using tools (i.e. programming languages, libraries, services) without maintaining the underlying infrastructure.
- Users instead have control over the applications themselves.

Examples –

- **Google App Engine :** Google App Engine allows developers to build and host web applications in cloud-based data centers that Google manages.
- **Kinsta :** Kinsta provides Application, Database, and Managed Wordpress Hosting solution that make it quick and easy to deploy any web application in minutes, without worrying about the hosting infrastructure.
- **Red Hat Openshift :** Red Hat OpenShift is an on-premises containerization PaaS software.
- **Heroku :** Developers can use this PaaS tool to build, manage, and grow consumer-facing apps.
- **Apprenda :** Apprenda is a PaaS product that allows developers and businesses to host an entire application portfolio. Build and deploy applications of all types on this platform.

Advantages of PaaS

- Programmers need not worry about what specific database or language the application has been programmed in.
- It offers developers the applications to be built without the overhead of the underlying operating system or infrastructure.

- Provides the freedom to developers to focus on the application’s design while the platform takes care of the language and the database.

Software As A Service (SaaS) –

- SaaS allows users to run existing online applications.
- It is a model software that is deployed as a hosting service and is accessed over Output Rephrased/Re-written Text the internet or software delivery model during which software and its associated data are hosted centrally and accessed using their client, usually an online browser over the web.
- SaaS services are used for the development and deployment of modern applications.
- SaaS enables users to use and access the cloud provider’s applications that are running on the provider’s infrastructure from thin client or program interfaces.
- SaaS products are among the most popular cloud computing services used by companies to build and grow businesses.
- SaaS is highly scalable and easy to use and manage because it doesn’t always require download and installation on individual devices for entire company use. It gives access to the end user.

Examples –

- **Hubspot :** HubSpot is a CRM, marketing, sales, and service SaaS platform that businesses use to connect with and retain customers.
- **JIRA :** JIRA is a project management software that's delivered by Atlassian and can be purchased on a subscription basis by customers.
- **Dropbox :** Dropbox is a file-sharing SaaS tool that allows multiple users within a group or organization to upload and download different files.

Advantages of IaaS

- It is a cloud computing service category providing a wide range of hosted capabilities and services. These can be used to build and deploy web-based software applications.
- It provides a lower cost of ownership than on-premises software. The reason is it does not require the purchase or installation of hardware or licenses.
- It can be easily accessed through a browser along a thin client.

Basis Of	IaaS	PaaS	SaaS
Uses	IAAS is used by network architects.	PAAS is used by developers.	SAAS is used by the end user.
Access	IAAS gives access to resources like virtual machines and virtual storage.	PAAS gives access to the run time environment to deployment and development tools for application.	SAAS gives access to the end user.
Model	It is a service model that provides virtualized computing resources over the internet.	It is a cloud computing model that delivers tools that are used for the development of applications.	It is a service model in cloud computing that hosts software to make it available to clients.
Technical	It requires technical	Some knowledge is	There is no requirement

understanding	knowledge.	required for the basic setup.	about technicalities, the company handles everything.
Popularity	It is popular among developers and researchers.	It is popular among developers who focus on the development of apps and scripts.	It is popular among consumers and companies, such as file sharing, email, and networking.
Cloud Services	Amazon Web Services, sun, vCloud Express.	Facebook, and Google search engine.	MS Office web, Facebook and Google Apps.
Enterprise services	AWS virtual private cloud.	Microsoft Azure.	IBM cloud analysis.
Outsourced cloud services	Salesforce	Force.com, Gigaspaces.	AWS, Terremark
User Controls	Operating System, Runtime, Middleware, and Application data	Data of the application.	Nothing.

...