CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | | |

Internal Assessment Test 1 – Feb. 2025

Sub:	Cloud Computing							Sub Code:	22MCA332
Date:	07/2/2025	Duration:	90 min's	Max Marks:	50	Sem:	III	Branch:	MCA

Note : Answer FIVE FULL Questions, choosing ONE full question from each PART

		MARKS	OBE	
			CO	RBT
	PART I			
1	Briefly summarize the Cloud Computing Reference Model.	[10]	CO1	L2
	OR			
2	Discuss RPC and how it enables interprocess communication.	[10]	CO2	L3
	PART II			
3	What is cloud? List and explain characteristics and benefits of cloud computing.	[10]	CO1	L1
	OR			
4	Discuss examples of distributed framework	[10]	CO2	L2
	PART III			
5	What are the major distributed computing technologies that led to cloud computing?	[10]	CO1	L1
	OR			
6	Compare the characteristics of parallel and distributed system. Draw and explain the layered view of distributed system	[10]	CO2	L2
	PART IV			
7	Discuss hardware Architecture for Parallel Processing.	[10]	CO2	L2
	OR			
8	Discuss the most important model for message-based communication.	[10]	CO2	L2
	PARTV			
9	Discuss Service Oriented Architecture (SOA)	[10]	CO2	L2
	OR			
10	With a neat diagram discuss distributed object Programming Model (Distributed Object Framework).	[10]	CO2	L3

Q1) Briefly summarize the Cloud Computing Reference Model

A fundamental characteristic of cloud computing is the capability to deliver, on demand, a variety of IT services that are quite diverse from each other. This variety creates different perceptions of what cloud computing is among users. Despite this lack of uniformity, it is possible to classify cloud computing services offerings into three major categories: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS).

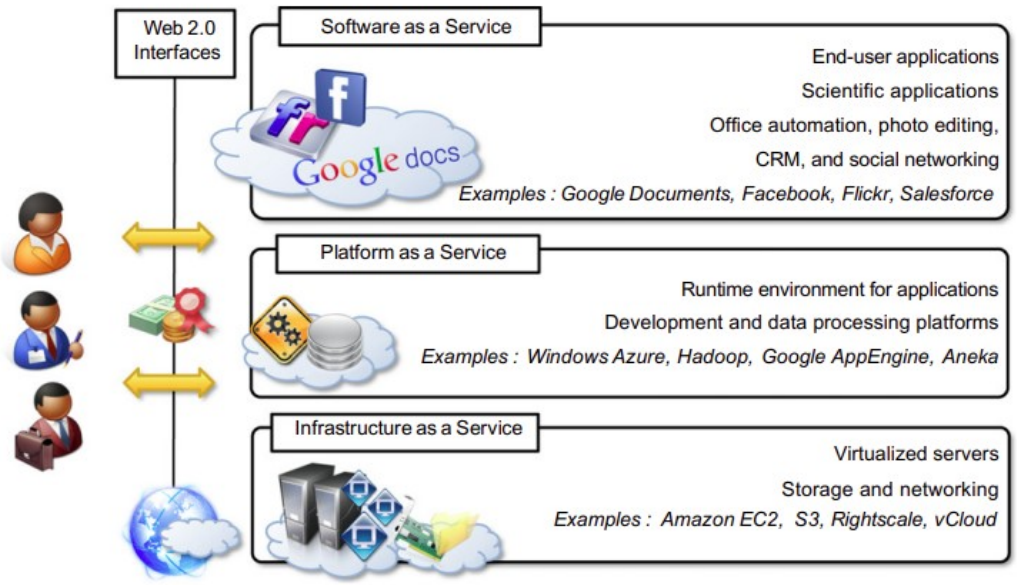


FIGURE 1.5

The Cloud Computing Reference Model.

At the base of the stack, **Infrastructure-as-a-Service** solutions deliver infrastructure on demand in the form of virtual hardware, storage, and networking. Virtual hardware is utilized to provide compute on demand in the form of virtual machine instances. These are created at users' request on the provider's infrastructure, and users are given tools and interfaces to configure the software stack installed in the virtual machine. The pricing model is usually defined in terms of dollars per hour, where the hourly cost is influenced by the characteristics of the virtual hardware. Virtual storage is delivered in the form of raw disk space or object store.. Virtual networking identifies the collection of services that manage the networking among virtual instances and their connectivity to the Internet or private networks.

Platform-as-a-Service solutions are the next step in the stack. They deliver scalable and elastic runtime environments on demand and host the execution of applications. These services are backed by a core middleware platform that is responsible for creating the abstract environment where applications are deployed and executed. It is the responsibility of the service provider to provide scalability and to manage fault tolerance, while users are requested to focus on the logic of the application developed by leveraging the provider's APIs and libraries. This approach increases the level of abstraction at which cloud computing is leveraged but also constrains the user in a more controlled environment.

At the top of the stack, **Software-as-a-Service** solutions provide applications and services on demand. Most of the common functionalities of desktop applications—such as office automation, document management, photo editing, and customer relationship management (CRM) software—are replicated on the provider's infrastructure and made more scalable and accessible through a browser on demand. These applications are shared across multiple users whose interaction is isolated from the other users. The SaaS layer is also the area of social networking Websites, which leverage cloud-based infrastructures to sustain the load generated by their popularity.

Each layer provides a different service to users. IaaS solutions are sought by users who want to leverage cloud computing from building dynamically scalable computing systems requiring a specific software stack. IaaS services are therefore used to develop scalable Websites or for background processing. PaaS solutions provide scalable programming platforms for developing applications and are more appropriate when new systems have to be developed. SaaS solutions target mostly end users who want to benefit from the elastic scalability of the cloud without doing any software development, installation, configuration, and maintenance. This solution is appropriate when there are existing SaaS services that fit users needs (such as email, document management, CRM, etc.) and a minimum level of customization is needed.

Q2) Discuss RPC and how it enables interprocess communication.

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. Figure 2.14 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.

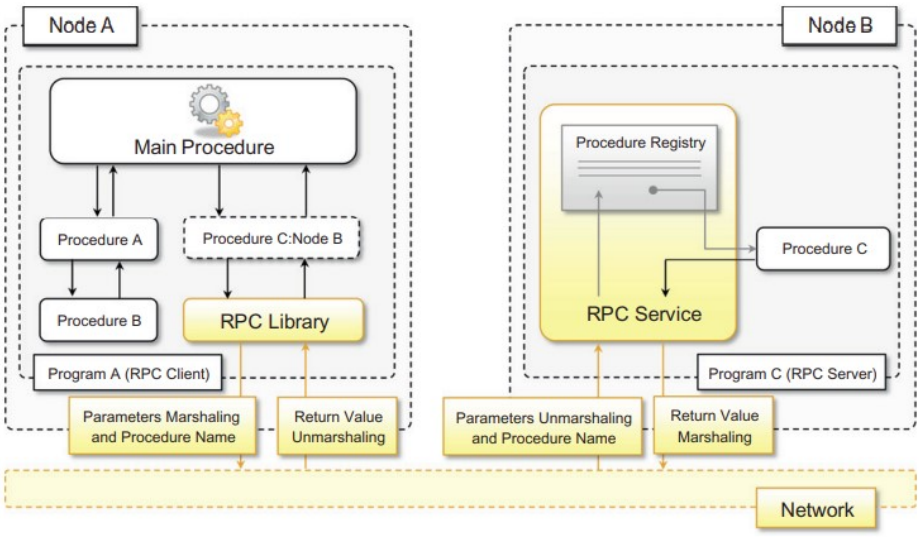


FIGURE 2.14
The RPC reference model.

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. 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 and implementation of the server procedures that will be exposed for remote invocation.
• Registration of remote procedures with the RPC server on the node where they will be made available. • Design and implementation of the client code that invokes the remote procedure(s).

Each RPC implementation generally provides client and server application programming interfaces (APIs) that facilitate the use of this simple and powerful abstraction. An important observation has to be made concerning the passing of parameters and return values. Since the server and the client processes are in two separate address spaces, the use of parameters passed by references or pointers is not suitable in this

scenario, because once unmarshaled these will refer to a memory location that is not accessible from within the server process. Second, in user-defined parameters and return value types, it is necessary to ensure that the RPC runtime is able to marshal them.

This is generally possible, especially when user-defined types are composed of simple types, for which marshaling is naturally provided. RPC has been a dominant technology for IPC for quite a long time, and several programming languages and environments support this interaction pattern in the form of libraries and additional packages. For instance, RPyC is an RPC implementation for Python. There also exist platformindependent solutions such as XML-RPC and JSON-RPC, which provide RPC facilities over XML and JSON, respectively. Thrift [113] is the framework developed at Facebook for enabling a transparent cross-language RPC model. Currently, the term RPC implementations encompass a variety of solutions including frameworks such distributed object programming (CORBA, DCOM, Java RMI, and .NET Remoting) and Web services that evolved from the original RPC concept.

Q3) What is cloud? List and explain characteristics and benefits of cloud computing.

The term cloud has historically been used in the telecommunications industry as an abstraction of the network in system diagrams. It then became the symbol of the most popular computer network: the Internet. This meaning also applies to cloud computing, which refers to an Internet-centric way of computing.
The Internet plays a fundamental role in cloud computing, since it represents either the medium or the platform through which many cloud computing services are delivered and made accessible. This aspect is also reflected in the definition given by Armbrust et al. [28]:
Cloud computing refers to both the applications delivered as services over the Internet and the hardware and system software in the datacenters that provide those services.
This definition describes cloud computing as a phenomenon touching on the entire stack: from the underlying hardware to the high-level software services and applications. It introduces the concept of everything as a service, mostly referred as XaaS, 2 where the different components of a system—IT infrastructure, development platforms, databases, and so on—can be delivered, measured, and consequently priced as a service
This notion of multiple parties using a shared cloud computing environment is highlighted in a definition proposed by the U.S. National Institute of Standards and Technology (NIST):
Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.
According to Reese [29], we can define three criteria to discriminate whether a service is delivered in the cloud computing style:

- ***The service is accessible via a Web browser (nonproprietary) or a Web services application programming interface (API).***
- ***Zero capital expenditure is necessary to get started.***
- ***You pay only for what you use as you use it.***

The utility-oriented nature of cloud computing is clearly expressed by Buyya et al. [30]:
A cloud is a type of parallel and distributed system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers.

Cloud computing has some interesting characteristics that bring benefits to both cloud service consumers (CSCs) and cloud service providers (CSPs). These characteristics are:

- No up-front commitments
- On-demand access
- Nice pricing
- Simplified application acceleration and scalability
- Efficient resource allocation

- Energy efficiency
- Seamless creation and use of third-party services

Q4) Discuss examples of distributed framework

Common object request broker architecture (CORBA)

CORBA is a specification introduced by the Object Management Group (OMG) for providing cross-platform and cross-language interoperability among distributed components. The specification was originally designed to provide an interoperation standard that could be effectively used at the industrial level. The current release of the CORBA specification is version 3.0 and currently the technology is not very popular, mostly because the development phase is a considerably complex task and the interoperability among components developed in different languages has never reached the proposed level of transparency. A fundamental component in the CORBA architecture is the *Object Request Broker (ORB)*, which acts as a central object bus. A CORBA object registers with the ORB the interface it is exposing, and clients can obtain a reference to that interface and invoke methods on it. The ORB is responsible for returning the reference to the client and managing all the low-level operations required to perform the remote method invocation. To simplify cross-platform interoperability, interfaces are defined in *Interface Definition Language (IDL)*, which provides a platform-independent specification of a component. An IDL specification is then translated into a *stub-skeleton* pair by specific CORBA compilers that generate the required client (stub) and server (skeleton) components in a specific programming language. These templates are completed with an appropriate implementation in the selected programming language. This allows CORBA components to be used across different runtime environment by simply using the stub and the skeleton that match the development language used. A specification meant to be used at the industry level, CORBA provides interoperability among different implementations of its runtime. In particular, at the lowest-level ORB implementations communicate with each other using the *Internet Inter-ORB Protocol (IIOP)*, which standardizes the interactions of different ORB implementations. Moreover, CORBA provides an additional level of abstraction and separates the ORB, which mostly deals with the networking among nodes, from the *Portable Object Adapter (POA)*, which is the runtime environment in which the skeletons are hosted and managed. Again, the interface of these two layers is clearly defined, thus giving more freedom and allowing different implementations to work together seamlessly.

Distributed component object model (DCOM/COM+)

DCOM, later integrated and evolved into COM+, is the solution provided by Microsoft for distributed object programming before the introduction of .NET technology. DCOM introduces a set of features allowing the use of COM components beyond the process boundaries. A COM object identifies a component that encapsulates a set of coherent and related operations; it was designed to be easily plugged into another application to leverage the features exposed through its interface. To support interoperability, COM standardizes a binary format, thus allowing the use of COM objects across different programming languages. DCOM enables such capabilities in a distributed environment by adding the required IPC support. The architecture of DCOM is quite similar to CORBA but simpler, since it does not aim to foster the same level of interoperability; its implementation is monopolized by Microsoft, which provides a single runtime environment. A DCOM server object can expose several interfaces, each representing a different behavior of the object. To invoke the methods exposed by the interface, clients obtain a pointer to that interface and use it as though it were a pointer to an object in the client's address space. The DCOM runtime is responsible for performing all the operations required to create this illusion. This technology provides a reasonable level of interoperability among Microsoft-based environments, and there are third-party implementations that allow the use of DCOM, even in Unix-based environments. Currently, even if still used

in industry, this technology is no longer popular and has been replaced by other approaches, such as .NET Remoting and Web Services.

Java remote method invocation (RMI)

Java RMI is a standard technology provided by Java for enabling RPC among distributed Java objects. RMI defines an infrastructure allowing the invocation of methods on objects that are located on different Java Virtual Machines (JVMs) residing either on the local node or on a remote one. As with CORBA, RMI is based on the *stub-skeleton* concept. Developers define an interface extending *java.rmi.Remote* that defines the contract for IPC. Java allows only publishing interfaces while it relies on actual types for the server and client part implementation. A class implementing the previous interface represents the *skeleton* component that will be made accessible beyond the JVM boundaries. The *stub* is generated from the skeleton class definition using the *rmic* command-line tool. Once the *stub-skeleton* pair is prepared, an instance of the skeleton is registered with the RMI registry that maps URIs, through which instances can be reached, to the corresponding objects. The RMI registry is a separate component that keeps track of all the instances that can be reached on a node. Clients contact the RMI registry and specify a URI, in the form *rmi://host:port/serviceName*, to obtain a reference to the corresponding object. The RMI runtime will automatically retrieve the class information for the stub component paired with the skeleton mapped with the given URI and return an instance of it properly configured to interact with the remote object. In the client code, all the services provided by the skeleton are accessed by invoking the methods defined in the remote interface. RMI provides a quite transparent interaction pattern. Once the development and deployment phases are completed and a reference to a remote object is obtained, the client code interacts with it as though it were a local instance, and RMI performs all the required operations to enable the IPC. Moreover, RMI also allows customizing the security that has to be applied for remote objects. This is done by leveraging the standard Java security infrastructure, which allows specifying policies defining the permissions attributed to the JVM hosting the remote object.

.NET remoting

Remoting is the technology allowing for IPC among .NET applications. It provides developers with a uniform platform for accessing remote objects from within any application developed in any of the languages supported by .NET. With respect to other distributed object technologies, Remoting is a fully customizable architecture that allows developers to control the transport protocols used to exchange information between the proxy and the remote object, the serialization format used to encode data, the lifetime of remote objects, and the server management of remote objects. Despite its modular and fully customizable architecture, Remoting allows a transparent interaction pattern with objects residing on different application domains. An application domain represents an isolated execution environment that can be accessible only through Remoting channels. A single process can host multiple application domains and must have at least one.

Q5) What are the major distributed computing technologies that led to cloud computing?

Three major milestones have led to cloud computing: mainframe computing cluster computing, and grid computing.

- Mainframes. These were the first examples of large computational facilities leveraging multiple processing units. Mainframes were powerful, highly reliable computers specialized for large data movement and massive input/output (I/O) operations. They were mostly used by large organizations for bulk data processing tasks such as online transactions, enterprise resource planning, and other operations involving the processing of significant amounts of data. One of the most attractive features of mainframes was the ability to be highly reliable computers that were "always on" and capable of tolerating failures transparently. No system shutdown was required to replace failed components, and the system could work without interruption. Now their popularity and deployments have reduced, but evolved versions of such systems are still in use for transaction

processing (such as online banking, airline ticket booking, supermarket and telcos, and government services).

- Clusters. Cluster computing started as a low-cost alternative to the use of mainframes and supercomputers. The technology advancement that created faster and more powerful mainframes and supercomputers eventually generated an increased availability of cheap commodity machines as a side effect. These machines could then be connected by a high-bandwidth network and controlled by specific software tools that manage them as a single system. Starting in the 1980s, clusters become the standard technology for parallel and high-performance computing. Built by commodity machines, they were cheaper than mainframes and made high-performance computing available to a large number of groups, including universities and small research labs. One of the attractive features of clusters was that the computational power of commodity machines could be leveraged to solve problems that were previously manageable only on expensive supercomputers. Moreover, clusters could be easily extended if more computational power was required.
- Grid computing appeared in the early 1990s as an evolution of cluster computing. In an analogy to the power grid, grid computing proposed a new approach to access large computational power, huge storage facilities, and a variety of services. Users can "consume" resources in the same way as they use other utilities such as power, gas, and water. Grids initially developed as aggregations of geographically dispersed clusters by means of Internet connections. These clusters belonged to different organizations, and arrangements were made among them to share the computational power. Different from a "large cluster," a computing grid was a dynamic aggregation of heterogeneous computing nodes, and its scale was nationwide or even worldwide. Several developments made possible the diffusion of computing grids: (a) clusters became quite common resources; (b) they were often underutilized; (c) new problems were requiring computational power that went beyond the capability of single clusters; and (d) the improvements in networking and the diffusion of the Internet made possible long-distance, high-bandwidth connectivity. All these elements led to the development of grids, which now serve a multitude of users across the world

Q6) Compare the characteristics of parallel and distributed system. Draw and explain the layered view of distributed system

Parallel Computing	Distributed Computing
Many operations are performed simultaneously.	System components are located at different locations.
Single computer is required.	Uses multiple computers.
Multiple processors perform multiple operations.	Multiple computers perform multiple operations.
It may have shared or distributed memory.	It have only distributed memory.
Processors communicate with each other through bus.	Computer communicate with each other through message passing.
Improves the system performance.	Improves system scalability, fault tolerance and resource sharing capabilities.

A distributed system is the result of the interaction of several components that traverse the entire computing stack from hardware to software. It emerges from the collaboration of several elements that—by working together—give users the illusion of a single coherent system

Below figure provides an overview of the different layers that are involved in providing the services of a distributed system.

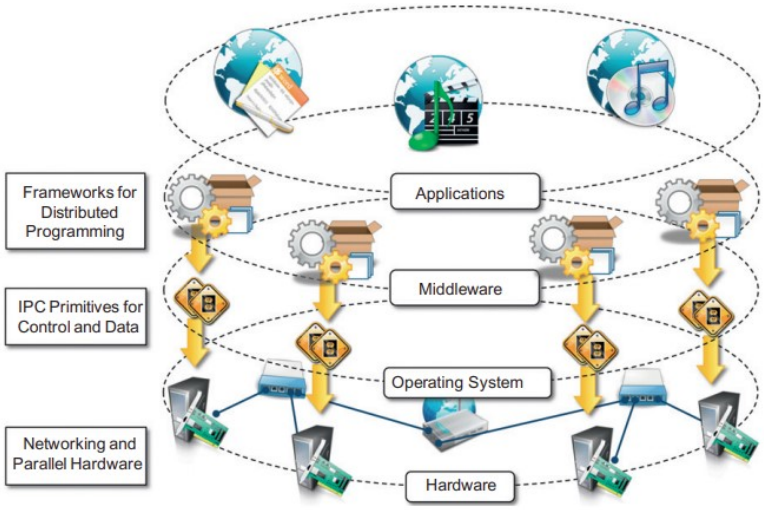


FIGURE 2.10

A layered view of a distributed system.

- At the very bottom layer, computer and network hardware constitute the physical infrastructure; these components are directly managed by the operating system, which provides the basic services for interprocess communication (IPC), process scheduling and management, and resource management in terms of file system and local devices. Taken together these two layers become the platform on top of which specialized software is deployed to turn a set of networked computers into a distributed system
- The middleware layer leverages such services to build a uniform environment for the development and deployment of distributed applications. By relying on the services offered by the operating system, the middleware develops its own protocols, data formats, and programming language or frameworks for the development of distributed applications. All of them constitute a uniform interface to distributed application developers that is completely independent from the underlying operating system and hides all the heterogeneities of the bottom layers.
- The top of the distributed system stack is represented by the applications and services designed and developed to use the middleware. These can serve several purposes and often expose their features in the form of graphical user interfaces (GUIs) accessible locally or through the Internet via a Web browser.

Figure 2.11 shows an example of how the general reference architecture of a distributed system is contextualized in the case of a cloud computing system.

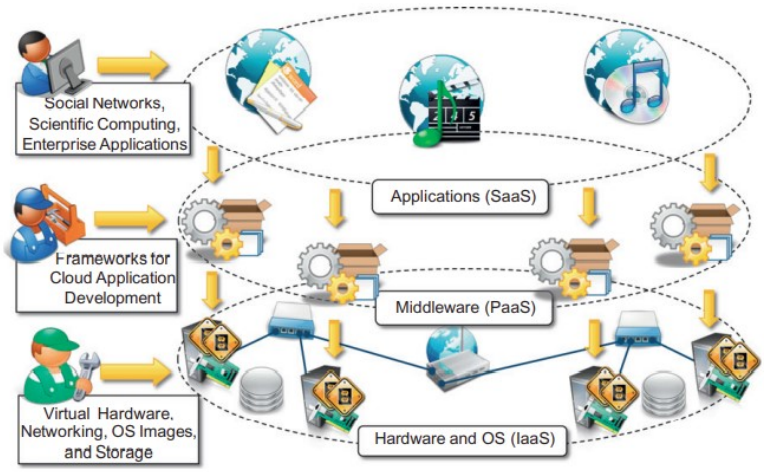


FIGURE 2.11

A cloud computing distributed system.

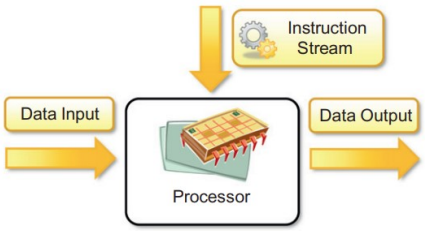
- Hardware and operating system layers make up the bare-bone infrastructure of one or more datacenters, where racks of servers are deployed and connected together through high-speed connectivity. This infrastructure is managed by the operating system, which provides the basic capability of machine and network management.
- The core logic is then implemented in the middleware that manages the virtualization layer, which is deployed on the physical infrastructure in order to maximize its utilization and provide a customizable runtime environment for applications.
- The middleware provides different facilities to application developers according to the type of services sold to customers. These facilities, offered through Web 2.0-compliant interfaces, range from virtual infrastructure building and deployment to application development and runtime environment

Q7) Discuss hardware Architecture for Parallel Processing.

The core elements of parallel processing are CPUs. Based on the number of instruction and data streams that can be processed simultaneously, computing systems are classified into the following four categories:
• Single-instruction, single-data (SISD) systems • Single-instruction, multiple-data (SIMD) systems
• Multiple-instruction, single-data (MISD) systems • Multiple-instruction, multiple-data (MIMD) systems

Single-instruction, single-data (SISD) systems

An SISD computing system is a uniprocessor machine capable of executing a single instruction, which operates on a single data stream



SINGLE-INSTRUCTION, MULTIPLE SYSTEMS

An SIMD computing system is a multiprocessor machine capable of executing the same instruction on all the CPUs but operating on different data streams

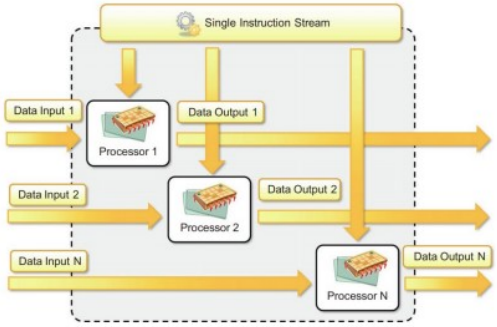


FIGURE 2.3
Single-instruction, multiple-data (SIMD) architecture.

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

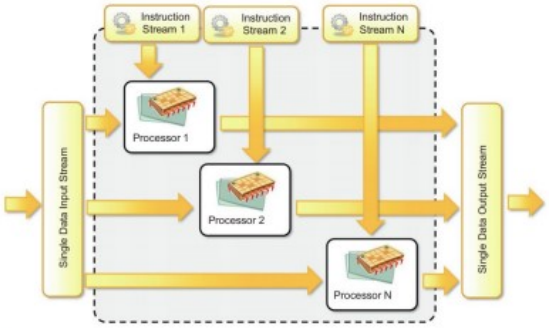
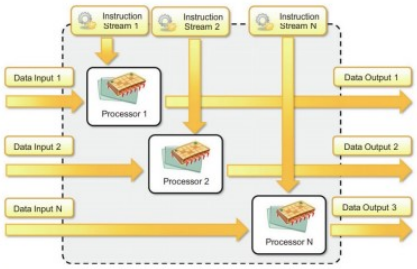


FIGURE 2.4
Multiple-instruction, single-data (MISD) architecture.

MULTIPLE-INSTRUCTION, MULTIPLE SYSTEMS-DATA
An MIMD computing system is a multiprocessor machine capable of executing multiple instructions on multiple data sets

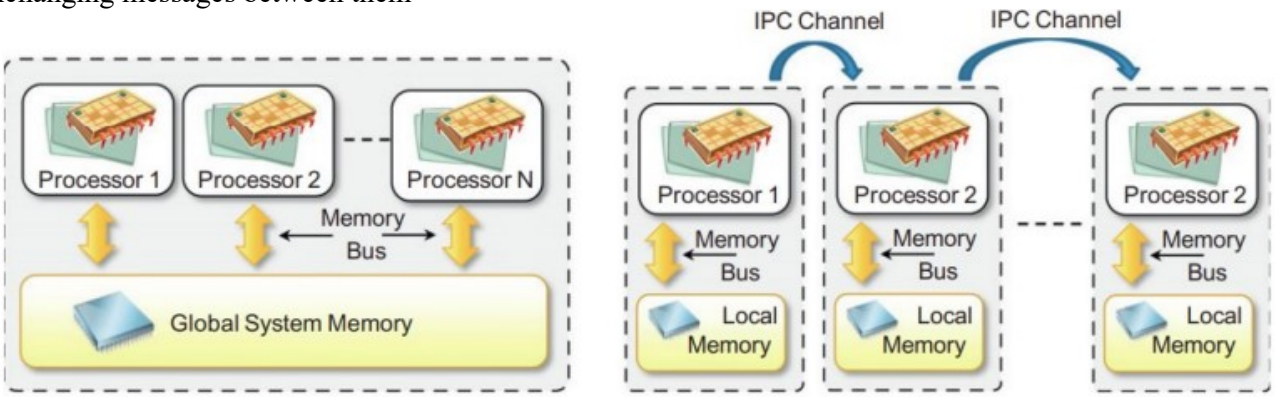


MULTIPLE-INSTRUCTION, MULTIPLE SYSTEMS MIMD
machines are broadly categorized into shared memory MIMD based on the way PEs are coupled to the main memory.

Shared memory
MIMD machines In the shared memory MIMD model, all the PEs are connected to a single global memory and they all have access to it Systems based on this model are also called tightly coupled multiprocessor systems. The communication between PEs in this model takes place through the shared memory; modification of the data stored in the global memory by one PE is visible to all other PEs. INSTRUCTION, MULTIPLE-DATA (MIMD) MIMD machines are broadly categorized into shared-memory MIMD and distributed

Distributed memory

MIMD machines In the distributed memory MIMD model, all PEs have a local memory. Systems based on this model are also called loosely coupled multiprocessor systems. The communication between PEs in this model takes place through the interconnection network (the interprocess communication channel, or IPC). The network connecting PEs can be configured to tree, mesh, cube, and so on. Each PE operates asynchronously, and if communication/synchronization among tasks is necessary, they can do so by exchanging messages between them



Q8) Discuss the most important model for message-based communication

Models for message-based communication

- **Point-to-point message model** This model organizes the communication among single components. Each message is sent from one component to another, and there is a direct addressing to identify the message receiver. In a point-to-point communication model it is necessary to know the location of or how to address another component in the system. There is no central infrastructure that dispatches the messages, and the communication is initiated by the message sender. It is possible to identify two major subcategories: direct communication and queue-based communication. In the former, the message is sent directly to the receiver and processed at the time of reception. In the latter, the receiver maintains a message queue in which the messages received are placed for later processing. The point-topoint message model is useful for implementing systems that are mostly based on one-to-one or many-to-one communication.
- **Publish-and-subscribe message model** This model introduces a different strategy, one that is based on notification among components. There are two major roles: the publisher and the subscriber. The former provides facilities for the latter to register its interest in a specific topic or event. Specific conditions holding true on the publisher side can trigger the creation of messages that are attached to a specific event. A message will be available to all the subscribers that registered for the corresponding event. There are two major strategies for dispatching the event to the subscribers:
  - Push strategy. In this case it is the responsibility of the publisher to notify all the subscribers— for example, with a method invocation.
  - Pull strategy. In this case the publisher simply makes available the message for a specific event, and it is responsibility of the subscribers to check whether there are messages on the events that are registered.

  The publish-and-subscribe model is very suitable for implementing systems based on the oneto-many communication model and simplifies the implementation of indirect communication patterns. It is, in fact, not necessary for the publisher to know the identity of the subscribers to make the communication happen.
- **Request-reply message model** The request-reply message model identifies all communication models in which, for each message sent by a process, there is a reply. This model is quite popular

and provides a different classification that does not focus on the number of the components involved in the communication but rather on how the dynamic of the interaction evolves. Point-to-point message models are more likely to be based on a request-reply interaction, especially in the case of direct communication. Publishand-subscribe models are less likely to be based on request-reply since they rely on notifications

Q9) Discuss Service Oriented Architecture (SOA)

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 andprovide 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:

- Service Provider
- Service Consumer

The following guiding principles, which characterize SOA platforms, are winning features within an enterprise context:

- **Standardized service contract**. Services adhere to a given communication agreement, which is specified through one or more service description documents.

- **Loose coupling**. Services are designed as self-contained components, maintain relationships that minimize dependencies on other services, and only require being aware of each other. Service contracts will enforce the required interaction among services. This simplifies the flexible aggregation of services and enables a more agile design strategy that supports the evolution of the enterprise business.

- **Abstraction**. A service is completely defined by service contracts and description documents. They hide their logic, which is encapsulated within their implementation.The use of service description documents and contracts removes the need to consider the technical implementation details and provides a more intuitive framework to definesoftware systems within a business context.

- **Reusability**. Designed as components, services can be reused more effectively, thus reducing development time and the associated costs. Reusability allows for a more agile design and cost- effective system implementation and deployment. Therefore, it is possible to leverage third- party services to deliver required functionality by paying an appropriate fee rather developing the same capability in-house.

- **Autonomy**. Services have control over the logic they encapsulate and, from a service consumer point of view, there is no need to know about their implementation.

- **Lack of state**. By providing a stateless interaction pattern (at least in principle), services increase the chance of being reused and aggregated, especially in a scenario in which a single service is used by multiple consumers that belong to different administrative and business domains.

- **Discoverability**. Services are defined by description documents that constitute supplemental metadata through which they can be effectively discovered. Service discovery provides an effective means for utilizing third-party resources.
- **Composability**. Using services as building blocks, sophisticated and complex operations can be implemented. Service orchestration and choreography provide a solid support for composing services and achieving business goals.

Q10) With a neat diagram discuss distributed object Programming Model (Distributed Object Framework).

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:

1. 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.

2. 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.

3. 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.

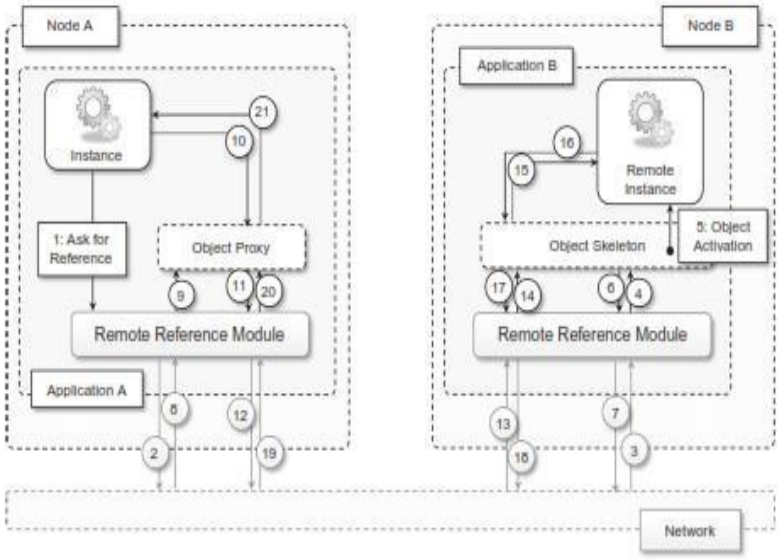


FIGURE 2.15

The distributed object programming model.

Examples of distributed Object frameworks

- Common Object Request Broker Architecture (CORBA): cross platform and crosslanguage interoperability among distributed components.
- Distributed Component Object Model (DCOM/COM+): Microsoft technology for distributed object programming before the introduction of .NET technology.
- Java Remote Method Invocation (RMI): technology provided by Java for enabling RPC among distributed Java objects.
- .NET Remoting: IPC among .NET applications, a uniform platform for accessing remote objects from within any application developed in any of the languages supported by .NET.