UNIT I

1

INTRODUCTION TO DISTRIBUTED COMPUTING CONCEPTS

Unit Structure

1.0 OBJECTIVE

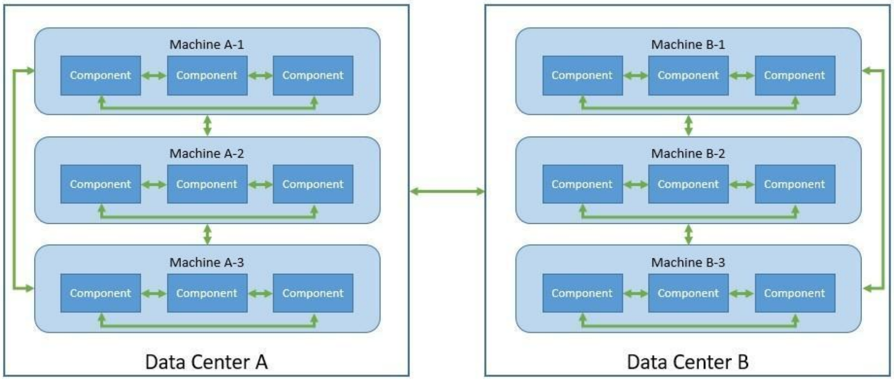
This chapter will able you to understand the following concept:

- Bezier curve and surface
- properties of Bezier curve
- Design techniques using Bezier curve
- Cubic Bezier curve
- Bezier surface

1.1 INTRODUCTION

1.1.1 What Is a Distributed System?

The definition of distributed system is consisting of many components together, it can also have multiple geographic boundaries and it can also communicate and coordinate with many components for message passing among the actor outside this system.



Now we will talk about the Decentralized system in which the distributed systems are not having specific components to take the decision but every component own their part of decision, none of them have complete information. Hence, the resulted decision is depending upon some sort of consensus between all components.

Distributed system is also called as parallel system as it is very close to parallel computing. In both the terms it is been refer to scaling-up the computational capability, but they achieve this in different way. **In parallel computing, we use multiple processors on a single machine to perform multiple tasks simultaneously**, possibly with shared memory. Whereas in distributed computing, multiple autonomous machines with no shared memory and communicating with message passing is used for message passing.

1.1.2 Distributed system concepts:

1. Availability:

This term is defining the percentage of time the service is operational is called as high availability. This is one of the most important feature of right software.

100% availability is the feature given by best developer, achieving this feature is the dream of every developer, it can be very challenging and expensive.

Distributed software systems can be made up of machines with a lower level of availability. To develop an application with 99.99% availability you can use machines/nodes that have the four nines availability.

2. Consistency:

In this type of feature the same information can be share all nodes simultaneously and, all nodes see and return the same information. Hence they should work in synchronization to get all nodes to exchange messages and work.

There some minor problem can be there like some difficulties while passing message through the network between the nodes. Other example of problem is in passing some message the delivery of message may fail during communication or may it get lost or some nodes may be unavailable at some point.

We can conclude that, if the function having weaker level of consistency, the system cannot run faster – but at the same time the higher chances that it won't return the latest dataset.

3. Idempotency:

Idempotency means in specific request is executed when the actual event execution will occur only one time regardless the number of times. As long as the level of idempotency, develop by manager is try to avoid bad consequences then it can have dropped connections, request errors, and more.

For example, after shopping if the customer tries to make a payment but nothing happens, he/she tries for many times, when the system is idempotent, the payment will be charged only one time, while not using idempotent systems one cannot give guarantee the lack of double charges and users returning their money back.

4. Data durability:

The term durability means that once data is added to the data storage it is one of the key concerns of distributed systems, it works even if some system's nodes are offline or have their data corrupted.

Level of durability is depending on different distributed databases used. Some of them are support data durability at the machine/node level, where as in some cases it maintains the cluster level, and in few cases it doesn't offer this functionality out of the box.

While developing high-scalable applications data durability takes an important role in which it is able to process millions of events per day.

In organization world sometime the companies or the owners do not allow the data loss as it is very crucial data. In some special cases when it deals with critical operations and transactions. Hence the developers aim should be providing a high level of data durability and strong connection data.

Nowadays, most distributed data storage services, e.g. Cassandra, MongoDB, and Dynamodb, offer durability support at different levels and can be all configured to ensure data durability at the cluster level.

5. Message Persistence:

In message passing the many of the time it happens that while processing a message the nodes through which a message passed goes offline or sometime it may occur failure, then there is a risk of message loss or some part of message loss. Message persistence assures that the message is saved and will be processed after the issue is solved.

When we talk about quality application message persistence is one of the most important characteristics.

When we need to protect the system from losses, we can take an example of a messaging app of Uber where billions of users are there with millions of payments per day, it seems very difficult and requires proven technologies and developers' expertise.

The solution to this challenge can be a creation of a messaging system that delivers a message at least one time and it should implement at least once.

In speaking of distributed systems, messaging is generally ensured by some distributed messaging service like RabbitMQ or Kafka, supporting various levels of reliability in delivering messages and allowing to build successful app architectures.

1.2 TYPES OF DISTRIBUTED SYSTEMS

The arrangement of nodes in the distributed system is like client/server systems or peer to peer systems.

1.2.1 Client/Server Distributed Systems:

In this type of relation, the client requests a resource to server and the server provides that resource back to the client. Multiple clients can be handled by the server at the same time while a client is in contact with only one server. The computer network is must be there to communicate with client and server in distributed system.

1.2.2 Peer to Peer Distributed Systems:

In this type of system nodes are having equal participants in data sharing through the computer network. Every task is equally divided between all the nodes. To complete the task nodes are interacting with each other as required as share resources. With using the network this can be done.

1.3 DISTRIBUTED SYSTEMS OVERVIEW

1.3.1 Advantages of a Distributed System:

While distributed systems are definitely more complex to design and build, it pays off the benefits they bring along.

- Scalability is one of the most important constraint of hardware limitation.
- In distributed system the multiple components are used to message passing with multiple machines hence the data can be replicate on multiple nodes hence the Reliability is more important factor of the distributed system as it should pass the message with reduced capacity.

- The large problem is divided into smaller part so that the task can be perform at multiple machines at the same time is the typical applications of distributed computing. Hence, this will result into increase in **performance of many complex workloads**, like matrix multiplications.

- Transfer of data from one node to another node can be easily done as all the nodes are connected to each other in distributed system.

- Addition of nodes at any point of time can be easily done in distributed system i.e. it can be scaled as required.

- In the entire distributed system if one node fails the other remaining nodes can still communicate with each other. The entire system will not break down.

- Printers and scanner can be shared with multiple nodes rather than being restricted to just one.

1.3.2 Disadvantages of Distributed Systems:

- The nodes and connection used in distributed system need to secured hence it's not provide adequate security.

- While transferring data from one node to another node messages may have lost due to network failure.

- Single user system is much easier to handle as compare to distributed system as the database connection is quite complicated and difficult.

- If all nodes want to send data at same time the load is on the distributed system hence the Overloading may occur in the network.

1.3.3 Challenges in a Distributed System:

While using benefits of distributed system there are some challenges which are discussed here.

- **Consistency vs. Availability:** Because of partition tolerance a distributed system can provide either consistence or availability. It provides by definition partition tolerance, as constrained by the CAP theorem.

- **Data Distribution:** data distribution is the main challenge in network as it uses multiple nodes to data transfer hence it need the complex algorithm to solve the issue of distribution with effective partition.

- **Coordination:** the data replication can be there in distributed system as it goes on multiple nodes hence the workload in a distributed system is the main problem in which it is very tricky to find the solution of the issue. To resolve the issue, it need a complex protocol for participated nodes which can take a decision.

Often in enterprise applications, under a transaction we require multiple operations to happen at same time. For example, in a single unit of data we may need to make several updates. While this has become quite complex when we distribute data over a cluster of nodes. Many systems do provide transactions like semantics in a distributed environment using complex protocols like Paxos and Raft.

1.4 DESIGNING ISSUES OF DISTRIBUTED SYSTEM

1. **Heterogeneity:** In the computer network, computer hardware, operating system and implementation of different developers is Heterogeneity used. Client-server environment is middleware is a key component of the heterogeneous distributed system. To communicate with end to end customer Middleware is a set of services to interact with the application.

2. **Openness:** new resources can be share and made available to the users at any time can be known as openness of the distributed system. Fact that their key interfaces are the main characteristics of the open system. A uniform communication mechanism can be helpful for access to shared resources in distributed commuting. The construction can be done through heterogeneous hardware and software.

3. **Scalability:** The term Scalability can be achieved through a significant increase in the number of users and resources connected and its efficient use.

4. **Security:** The main challenge of distributed system is the security of information. Hence the problem can be resolve with three components Confidentially, integrity and availability and with Encryption we can protects shared resources.

5. **Failure Handling:** while using system there may be some case where some faults occur in hardware and the software program, it may produce false results or sometime it may hold the activity before they have finished the intended computation. bust handling failure is more complex in distributed systems because the failure is the failure can be in some components fail while others continue to function.

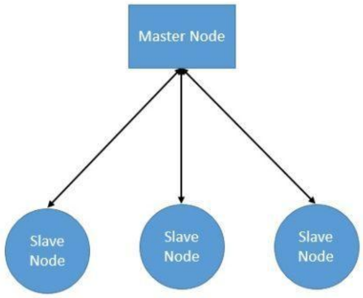
6. **Concurrency:** the resources can be shared at the same time by client can lead the issue of concurrency as multiple users require the same resources at the same time and they have the access of read, write, and update. Every resource should be safe in a concurrent environment.

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

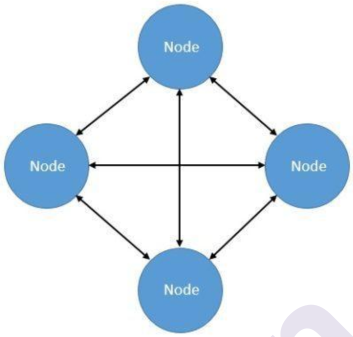
1.5 DISTRIBUTED SYSTEM ARCHITECTURE

The architecture of a distributed system depends on the use-case which can be used anywhere in the management .it shows the flow of the data. However, with some general patterns we can explore more cases.

The following diagram can represent the fact, about the core distribution models which the architecture used



Master-Slave Model



Peer-to-Peer Model

- **Master-slave**: In this model, there is one node who is master node of the distributed system and it plays the role of king. Hence he is having all the necessary information about the system and its control and based on these it can also take the decision whenever is needed. The other node can act as salves and they can be responsible for the task given by the master and they should report to the master only. The changes in any architecture can be done by master node.

- **Peer-to-peer:** in peer to peer distributed system there is no master slave relationship. The master node is not present in this system. All nodes are master nodes in peer to peer distributed system in this model. All the **nodes equally share the responsibility of the master**.

Most of the distributed system can be combination of both the architecture i.e. Peer to peer and master slaves. One can also choose one of the architecture for models.

Data distribution is advantage of peer-to-peer model, whereas data replication in the same architecture can be advantage of master-slave model.

1.6 CATEGORIES OF DISTRIBUTED SYSTEM

To design a distributed system there can be several rationales. For example, we need to perform computations like matrix multiplications at a massive scale in machine learning models. These are impossible to accommodate on a single machine.

Similarly, on a single machine it is impossible to store all large files and handling huge files and processing at least highly inefficient.

So, categorization of distributed system can be **depending upon the use-case,** the following categories are.

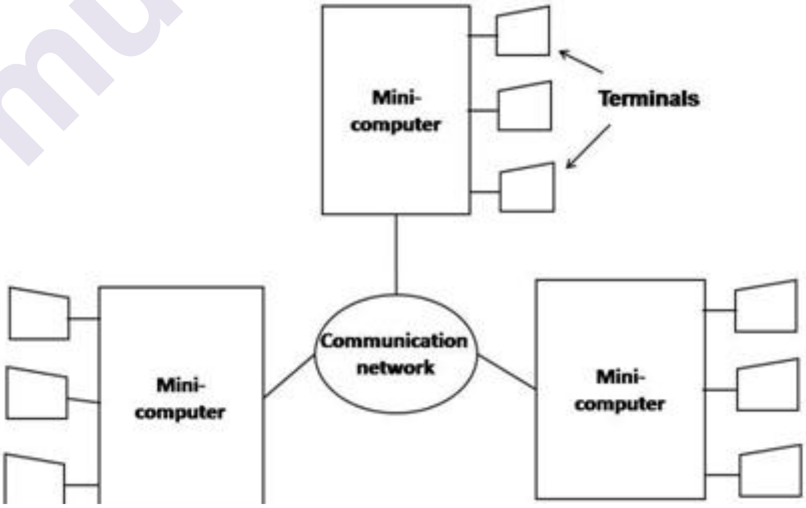
- Data stores
- Messaging
- Computing
- Ledgers
- File-systems
- Applications

Traditionally, we can store the data in relational databases were the default choice of data store for quite simple some time. However, the data can be **growth in terms of volume, velocity of data and variety,** in this database is not sufficient for the data process hence it started to fall short of the expectation. The solution for this problem we introduced NoSQL databases with their distributed architecture which is more useful.

Similarly, the distributed system provides the features like durability, performance, scalability, usability over the traditional messaging systems which could not remain insulated to the challenges of the modern scale of data. in this area today there are several options that can provide multiple semantics like publish-subscribe and point-to-point.

The 5 categories of various models that are used for building distributed computing systems

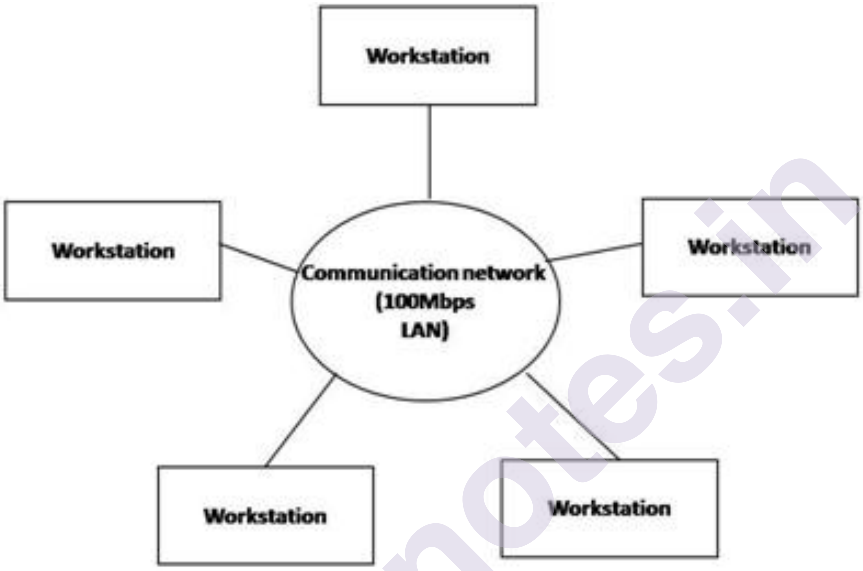
1. Minicomputer Model:



- The extension of the centralized time-sharing system is called as minicomputer model.
- Few minicomputers interconnected by a communication network in the distributed computing system based. multiple users can simultaneously have logged on to each minicomputer.

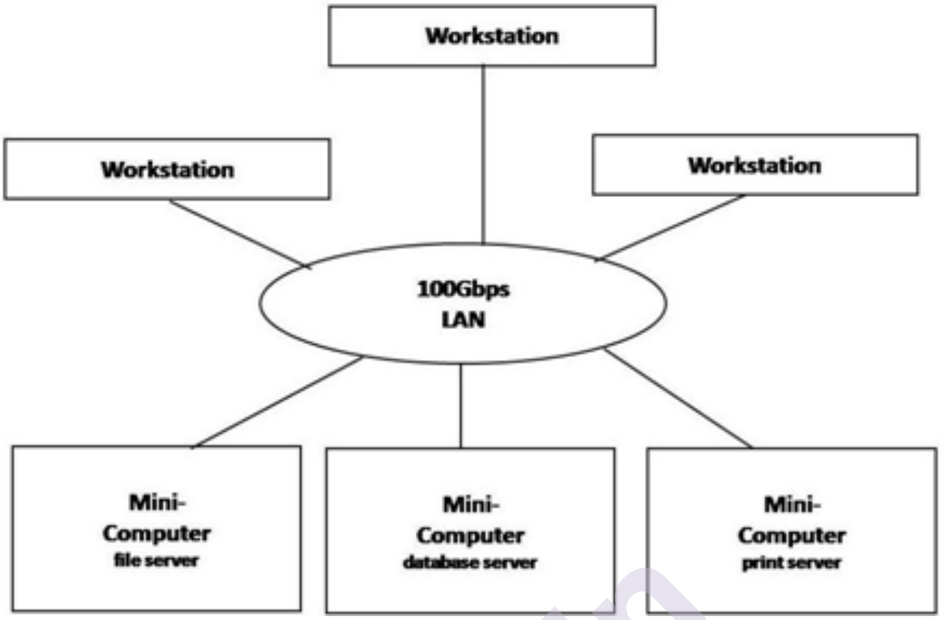
- Each minicomputer is connected to Several interactive terminals. With every user login they get remote access to other minicomputers while using minicomputer.
- Every user will get an access of remote resources on network which are available on some machine other than the one on to which the user is currently logged. maybe when one need to share resource sharing with remote users is desired The minicomputer model is used.
- One of the best example of distributed computing system based on the minicomputer model is the early ARPA net.

2. Workstation Model:



- A several workstations interconnected by a communication network in distributed computing system based on the workstation model.
- Throughout an infrastructure several workstations were located in an organization. every workstation has its well equipped with its own disk & serves as a single-user computer.
- In such type of environment, at any one time out of total workstations some of the workstations are idle which results in the waste of large amounts of CPU time.
- Therefore, the idea of the workstation model is to interconnect all these workstations by a high-speed LAN so that idle workstations may be used to process jobs of users who are logged onto other workstations & do not have sufficient processing power at their own workstations to get their jobs processed efficiently.
- Example: Sprite system & Xerox PARC.

3. Workstation–Server Model:



- A network of personal workstations having its own hard disk and local files can be together called as server workstation model.

- Diskful workstation can be referred as a workstation with its own local disk & a workstation without a local disk can be referred as a diskless workstation. in network environments these workstations have become more popular than diskful workstations.

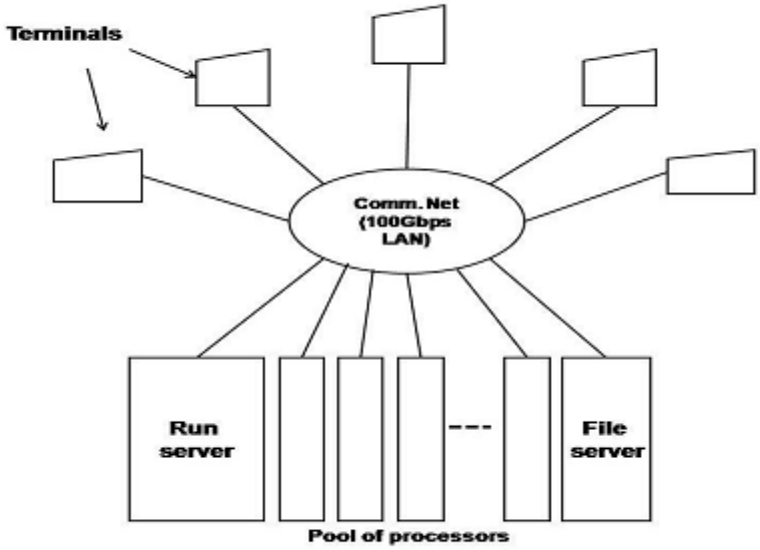
- A few minicomputers & several workstations interconnected by a communication in network are together come under distributed computing system based on the workstation-server model.

- In this model, a user logs onto a workstation with the help of his or her home workstation. Normal algorithmic or numeric activities required by the user's processes are performed at the user's home workstation, but server process the requests coming from special servers. Out of which some are sent to a server which provided by user's requested activity & returns the result of request processing to the user's workstation.

- Therefore, in this model, the user's processes need not migrated to the server machines for getting the work done by those machines.

- Example: The V-System.

4. Processor–Pool Model:



- Some time there are some cases in which the user does not need any computing power but once in a while the user may need a very large amount of computing power for a short time this computation can consider under the processor-pool model. It works on the observation of the utilization of resources.

- As we know that the workstation-server model uses a processor is allocated to each user for the task, but in processor-pool model the processors are used for the task to pooled together and the resources are shared by the users as and when needed.

- a large number of microcomputers & minicomputers attached to the network in the pool of processors.

- In this model every workstation in the pool has its own memory to load the data & run a system program or an application program of the distributed computing system.

- No home machine is present & the user does not log onto any machine in this model.

- Better utilization of processing power & greater flexibility is the highlighting advantage of the model.

- Example: Amoeba & the Cambridge Distributed Computing System.

5. Hybrid Model:

- The workstation-server model has a large number of computer users only performing simple interactive tasks &-executing small programs.

- The processor-pool model is more attractive & suitable for the users or the group of users who need to do massive computation for job performing.

- The feature of Workstation-server & processor-pool models can combine together is called as hybrid model which can be used to build a distributed system.
- The allocation of processor can be done dynamically for computations that are too large or require several computers for execution.
- This model assures that the interactive jobs can be processed in local workstation of the user account in the hybrid mode.

A distributed system is a system in which components are situated in distinct places, these distinct places refer to networked computers which can easily communicate and coordinate their tasks by just exchanging messages to each other. These components can communicate with each other to conquer one common goal as a task.

There are many algorithms are used to achieve Distributed Computing and these are broadly divided into 2 categories: Token-Based Algorithms and Non-Token Based Algorithms.

1.7 DISTINGUISH BETWEEN TOKEN BASED AND NON-TOKEN BASED ALGORITHMS IN DISTRIBUTED SYSTEM

Sr. No.	Token Based Algorithms	Non-Token Based Algorithms
1.	In the distributed computing system, a unique token is shared among all the sites in Token-based algorithm.	Where as in this algorithm the token is not present and even it is not sharing any token for accessing the data.
2.	When the token is processes the site is allowed to enter the into the Computer System.	Here, two or more successive rounds of messages are exchanged between sites to determine which site is to enter the Computer System next.
3.	The request sequence order can be used to allocate the token in token based algorithm for request for the Computer Systems and to resolve the conflict for the simultaneous requests for the System.	The use of timestamp in processing the request for the Computer Systems and to resolve the conflict for the simultaneous requests for the System in non-token based algorithm.
4.	The token-based algorithm produces less message traffic as compared to Non-Token based Algorithm.	As compared to the Token-based Algorithm Non-Token based Algorithm produces more message traffic.
5.	They are free from deadlock (i.e. here there are no two or more processes are in the queue in order to wait for messages that will actually	They are not free from the deadlock problem as they are based on timestamp only.

	can't come) because of the existence of unique token in the distributed system.	
6.	Here, it is follow the sequence of request executed as the order they are made in.	Here there is no such type of execution order hence.
7.	Token-based algorithms are more scalable as they can free your server from having to store session state and also they contain all the necessary information which they need for authentication.	Non-Token based algorithms are less scalable than the Token-based algorithms because here server is not free from its tasks.
8.	Here the access control is quite Fine-grained because here inside the token roles, permissions and resources can be easily specifying for the user.	Here the access control is not so fine as there is no token which can specify roles, permission, and resources for the user.
9.	Token-based algorithms make authentication quite easy.	Non-Token based algorithms can't make authentication easy.
10.	Examples of Token-Based Algorithms are: (i) Singhal's Heuristic Algorithm (ii) Raymonds Tree Based Algorithm (iii) Suzuki - Kasami Algorithm	Examples of Non-Token Based Algorithms are: (i) Lamport's Algorithm (ii) Ricart-Agarwala Algorithm (iii) Maekawa's Algorithm

1.8 SUMMARY

The definition of distributed system is consisting of many components together, it can also have multiple geographic boundaries and it can also communicate and coordinate with many components for message passing among the actor outside this system. Distributed software systems can be made up of machines with a lower level of availability. To develop an application with 99.99% availability you can use machines/nodes that have the four nines availability. The architecture of a distributed system depends on the use-case which can be used anywhere in the management .it shows the flow of the data. However, with some general patterns we can explore more cases. A distributed system is a system in which components are situated in distinct places, these distinct places refer to networked computers which can easily communicate and coordinate their tasks by just exchanging messages to each other. These components can communicate with each other to conquer one common goal as a task.

1.9 UNIT AND EXCERCISE

1. What is distributed system?
2. List and explain types of distributed system.
3. Explain advantages of distributed system
4. Explain challenges of distributed system
5. Explain issues while designing of distributed system
6. Explain categories of distributed system
7. Write distinguishes between token base and non-token base algorithm.
8. Write a note on

   1. Hybrid model

   2. Processor pool model

   3. Workstation server model

*****

2

INTRODUCTION TO DISTRIBUTED COMPUTING CONCEPTS

Unit Structure

2.0 OBJECTIVE

This chapter will able you to understand the following concept:

- What is IPC (Inter-Process Communication) and its need
- Different modes of IPC
- Different approaches of IPC
- Group communication overview with its types
- RPC and its working
- Remote Method Invocation

2.1 INTRODUCTION

What Is IPC?

To exchange the data, the cooperating processes, need to communicate with each this transaction of process can be called as Inter-process communication. It is the mechanism of communicating among processes.

To share data or resources the process can be used this process refers **inter - process communication** or **interprocess communication** (**IPC**) and it is specially used in operating system in computer science. This mechanism of an operating system provides to allow the processes to manage shared data. Typically, applications of an IPC can be client server relationship in which client ask for resources from server and server will reply on the client request.

In designing of microkernels and nano-kernels IPC is very important factor, which reduce the number of functionalities provided by the kernel. Those functionalities are then obtained by communicating with servers via IPC, leading to a large increase in communication when compared to a regular monolithic kernel. Variable analytic framework structures are used in IPC. The IPC model relies on multi vector protocols which can ensure the compatibility between them. An IPC mechanism is either synchronous or asynchronous.

Need for IPC:

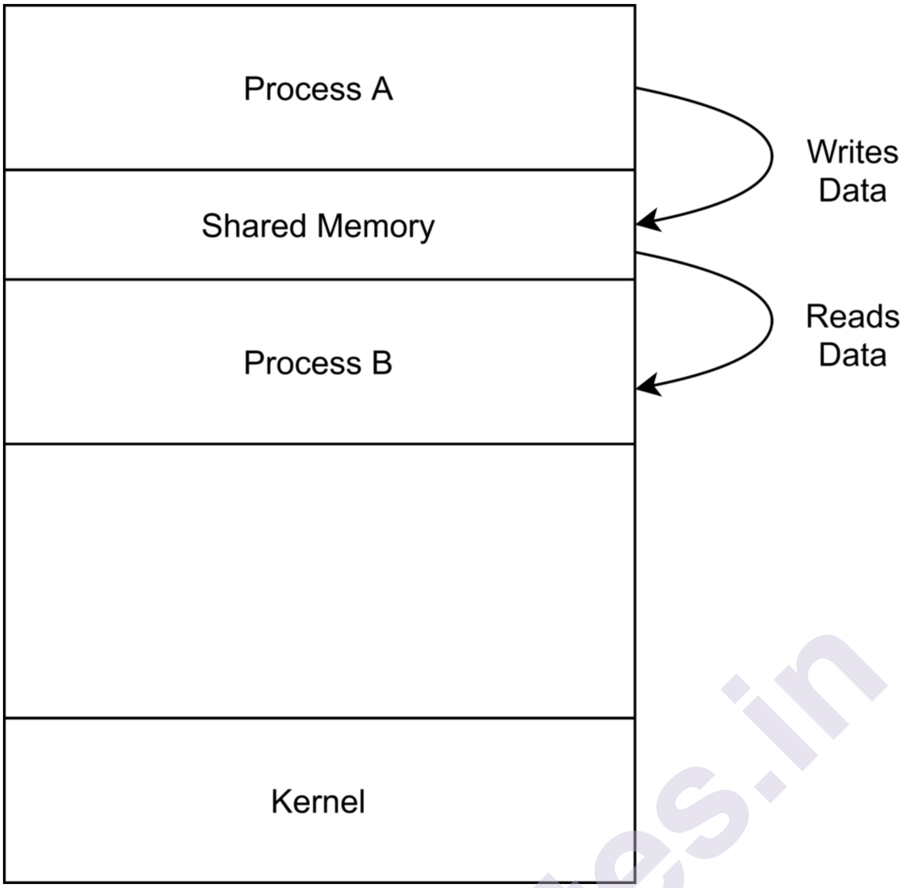
- **Information Sharing:** the resources sharing is the main component as many users could demand for same information at the same time. Thus, if there would be easiest path in resource sharing it will available all the time.
- **Computation Speedup:** many of the task could perform at same time hence the big task needs to split into several sub task for its fast execution. This also requires related processes to exchange information related to the task
- **Modularity:** Most of the time applications are built in a modular fashion and divided into separate processes. For instance, the Google Chrome web browser spawns a separate process for each new tab.

2.2 MODES OF INTER-PROCESS COMMUNICATION

Shared memory and message passing are the two modes through which processes can communicate with each other. The process is divided into several sections.

2.2.1. Shared Memory:

To established a shared memory region, require communicating process which can run through the shared memory model. Every process has its own memory region as well as they have its own address space to communicate with each other, the other process which want to communicate with them, they need to attach their address space to this shared memory segment



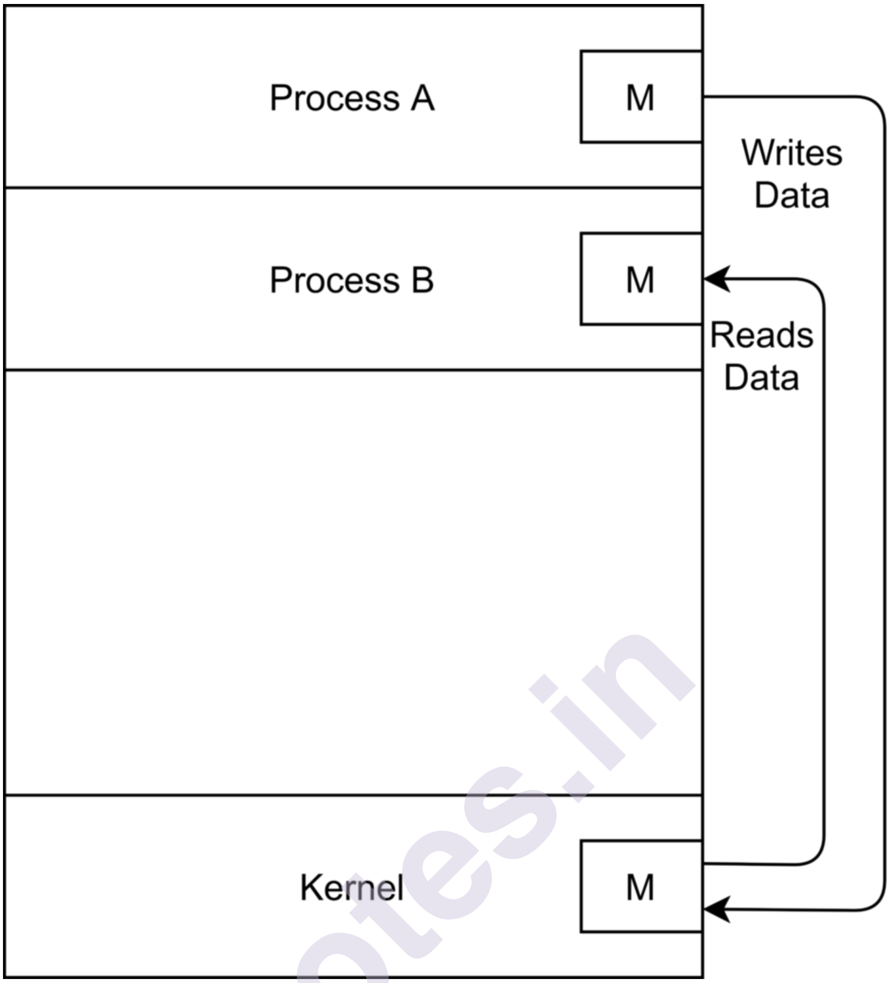
The above diagram explains the shared memory model of IPC. Message sharing or information exchanging done between Process A and process B with the help of a shared memory segment through the shared memory region.

By default, the operating system prevents processes from accessing other process memory. The shared memory model requires processes to agree to remove this restriction. Besides, as shared memory is established based on the agreement between processes, the processes are also responsible to ensure synchronization so that both processes are not writing to the same location at the same time.

2.2.2 Message Passing:

The shared memory model is very useful model for transferring the data, but it is not giving the suitable outcome for some processes. some process we can’t achieve through this for example if the data is at various system and the process needs to exchange the data through different computer systems in distributed computing they don’t have straightforward way to communicate with each other in a shared memory region.

The message passing mechanism provides an alternative means processes for communication. In this mode, processes interact with each other through messages with assistance from the underlying operating system:



In the above diagram two processes A, and B are communicating with each other through message passing. Process A sends a message M to the operating system (kernel). This message is then read by process B.

In order to successfully exchange messages, there needs to be a communication link between the processes. There are several techniques through which these communication links are established. Following points discussed the mechanisms:

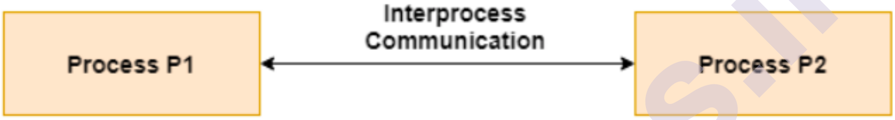
- **Direct Communication:** In this type the direct sender and receiver is including in the process, every process has explicit the recipient or the sender. For example, if process A needs to send a message to process B, it can use the primitive
- **Indirect Communication:** In this type the messages can be transferred indirectly with the help of, mailbox. A container that holds the messages is called mailbox. For example, if X is a mailbox, then process A can send a message to mailbox using the primitive
- **Synchronization:** in this type the additional option of synchronization is there which can have extension to direct and indirect communication. There is block to send and receive message

which can be based on the need of a process can choose to block while sending or receiving messages Besides, it can communicate without any blocking is called asynchronous.

- **Buffering:** The temporary queue for exchanged messages is called as buffering. These queues can be of zero, bounded, and unbounded capacity

We can explain the things with help of **producer-consumer problem of IPC**. where a process is producer or we can see application is also producers who can store data. Whereas the consumer can consume data which ultimately an application / process by which the data is produced. To communicate with each other the producer and consumer processes mechanism is used.

An operating system provides mechanism of Interprocess communication that allows processes to communicate with each other. In this communication one process can let know to the another process that some event is occurred or the transferring of data from one process to another.



2.2.3 Synchronization in Interprocess Communication:

Synchronization is a most important part of interprocess communication. It can be assured by either the interprocess control mechanism or handled by the communicating processes. Following are some of the methods to provide synchronization −

- **Semaphore:**

A multiple process demands common resources at same time which is sometime called as a semaphore with a variable that controls the access of the resources. The Min categories of semaphores are binary semaphores and counting semaphores.

- **Mutual Exclusion:**

There are some cases where the only one thread enter into critical section that situation is called as Mutual. This technique is useful for preventing race condition and it also can help in synchronization.

- **Barrier:**

As name implies barrier will not allow any individual processes to proceed until all the processes reach it. Many parallel languages and collective routines impose barriers.

- **Spinlock:**

This is one type of lock. The checking process of lock availability is called as spinlock where the processes trying to acquire this lock wait in a loop. This is known as busy waiting because the process is not doing any useful operation even though it is active.

2.3 APPROACHES TO INTER PROCESS COMMUNICATION

- **Pipe:**

A unidirectional channel of data transfer is called as pipe. In this one pipe is used for one-way data channel if both the way data transfer then we have to use two pipes for two processes. standard input and output methods can be used in this. Pipes are used in all POSIX systems as well as Windows operating systems.

- **Socket:**

The end point for sending or receiving data in a network is called as socket. This is true for data sent between processes on the same computer or data sent between different computers on the same network. Most of the operating systems use sockets for interprocess communication.

- **File:**

A set of data record is called as file it is stored on a disk or acquired on demand by a file server. As per the requirement multiple processes can access a file. Files can be used for data storage in all operating systems.

- **Signal:**

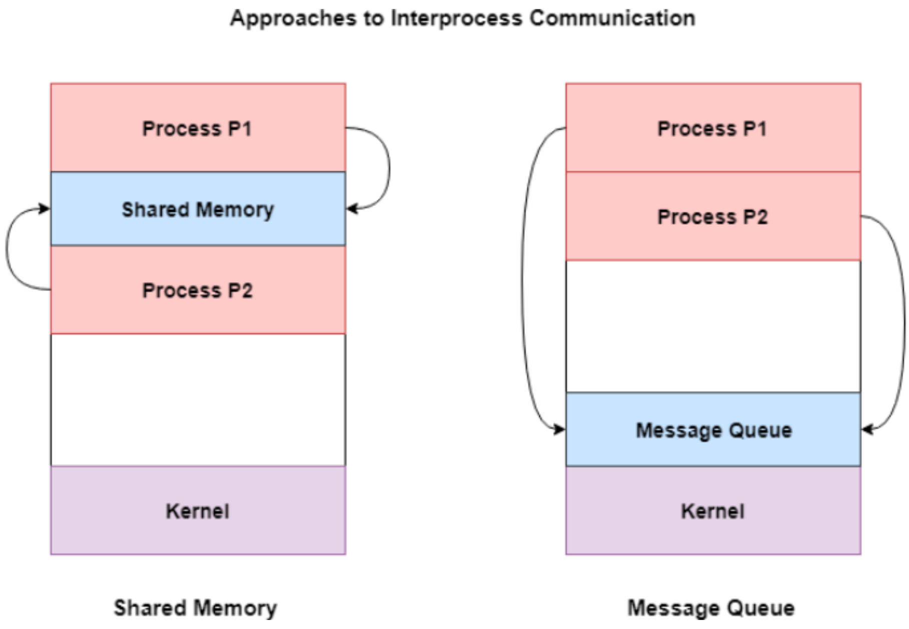
In a limited way of interprocess communication the signals are used. The messages are sent from one process to another. Normally, signals are not used to transfer data but are used for remote commands between processes.

- **Shared Memory:**

Multiple process at the same time shares the resources with the help of Shared memory. Due to this all process can communicate with one another in easiest way All POSIX systems, as well as Windows operating systems use shared memory.

- **Message Queue:**

Without being connected to each other multiple processes can read and write data to the message queue. Until their recipient retrieves, messages are stored in the queue. In interprocess communication Message queues are quite useful and are used by most operating systems.



2.4 GROUP COMMUNICATION OVERVIEW

Group Communication:

In group communication all the servers are connected together with each other hence the message is sent to a group and then this message is delivered to all server of the group.

There are so many members of group who can join or leave the group at any time. This can be done through multicast communication. There are three communication modes:

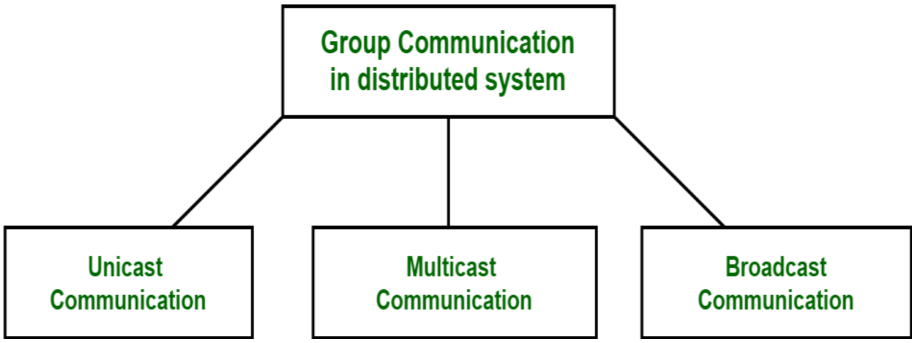
1. **Unicast:** Process to process communication
2. **Broadcast:** all the process can communicate with each other
3. **Multicast:** only a member of group or object of a class can communicate with each other

Object Group: Same set of invocations concurrently applied on a set of object which is together called as an Object group.

Client Group: operations on a single, local object, which acts as a proxy for the group is called as Client objects.

The proxy uses a group communication system to send the invocations to the members of the object group.

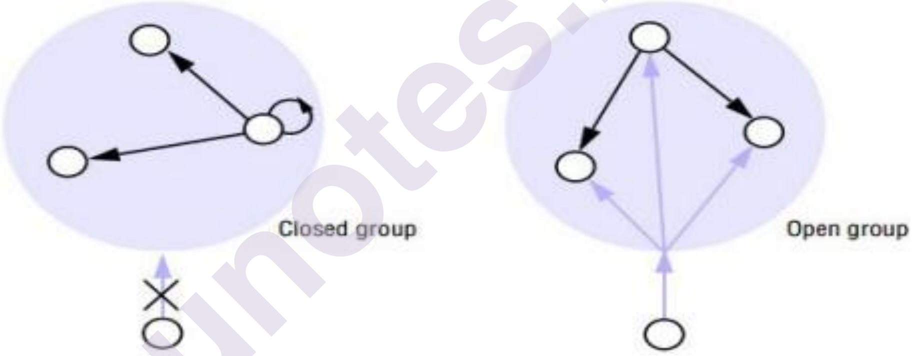
Various type of data (files, code, source file) can be exchanged over a network and this is between two processes in a distributed system. Sometime group communication concept is existing as there could be a situation where one source process tries to communicate with multiple processes at once. Abstraction of process or a group is a collection of interconnected processes. In abstraction it is hiding the message passing wo look the procedure in normal procedure. Different host works together while communicating in Group communication and they also perform operations in a synchronized manner, so that the improvement of performance of the system can take place.



Types of group communication:

Closed and open groups:

only a group member can send the message in group and multicast the message in group. In the closed group a process can deliver message to itself.



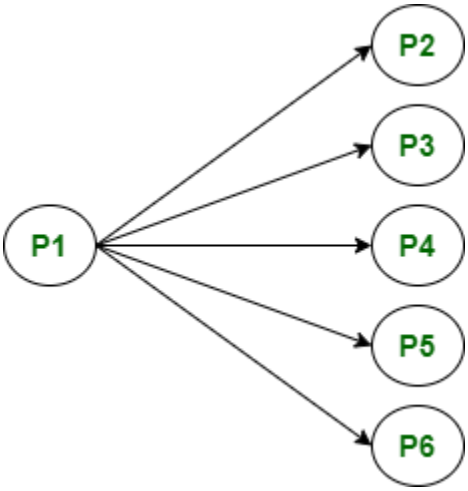
Overlapping and non-overlapping groups:

In overlapping groups, entities may be members of multiple groups, and non-overlapping groups imply that membership does not overlap to at most one.

Types of Group Communication in a Distributed System:

Broadcast Communication:

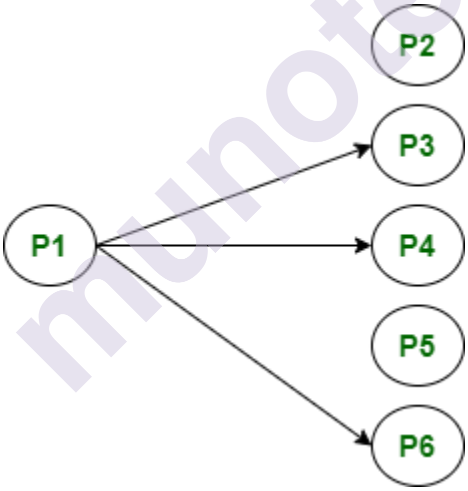
In distributed computing every process is connects with the host process and if the host process wants to communicate with the other process at same time it is called as broadcast communication. When one common information is sent over a trusted network Broadcast communication is used as the information is delivered to each and every process in most efficient manner possible. It does not require any extra processing thing the communication is very fast in comparison to other modes of communication. It is not suitable for large number of processes and cannot treat a specific process individually.



A broadcast Communication: P1 process communicating with every process in the system

Multicast Communication:

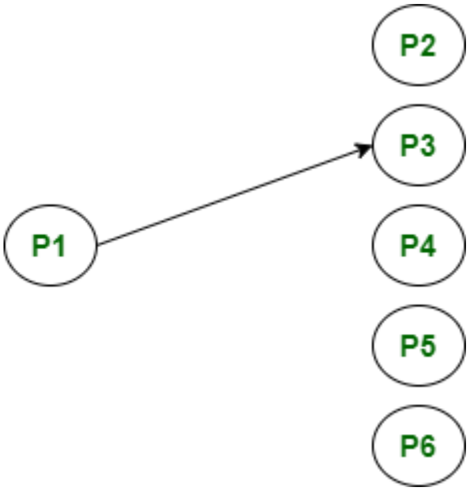
A group of process needs to controlled by a specific host process in a distributed system at the same time is called as multicast communication. The implementation is done in finding a way to address problem of a high workload on host system and redundant information from process in system. It mostly decreases the time taken for message handling.



A multicast Communication: P1 process communicating with only a group of the process in the system

Unicast Communication:

When a single process wants to communicate with the host process in a distributed system at the same time is called as Unicast communication. As the name imply it deals with single process. The same information can be pass to multiple processes. The specific process will be treating in best way hence this is the best for two processes for communication. It will lead to overheads as it has to find exact process and then exchange information/data.



Synchronous and asynchronous systems:

Group communication can be possible in both the environments. It is reliable and ordered in multicast message processing.

In group communication, it guarantees the copy of same message should deliver to the all members of a group.

Reliable multicast operation is based on two things:

Integrity: delivering the message correctly only once

Validity: assuring that a message sent will be delivered.

The following are the types of message ordering:

FIFO ordering: first process sends the message to first server and so on it will be delivered in this order for all the processes in the group.

Causal ordering: If a message happens before another message in the distributed system this so-called causal relationship will be preserved in the delivery of the associated messages at all processes.

Total ordering: during the entire process, if a message is delivered before another message at one process, then the same order will be preserved for all processes is called as total ordering process.

Following are the four main task of Group membership management

Providing an interface for group membership changes: the changes related to membership is done at this stage. The operations like addition of process, or destroy a process from a group or crating a process can have done through the membership service.

Failure detection: In case of crash and unreachability the service monitors the group members. Suspected or Unsuspected processes can be detecting by detector. If the process is unreachable or crashed the process is excluded from membership.

Notifying members of group membership changes: if the process is added to the group or excluded from the group then the service will indicate that to the other group members.

Performing group address expansion: When a process multicast a message, it supplies the group identifier rather than a list of processes in the group.

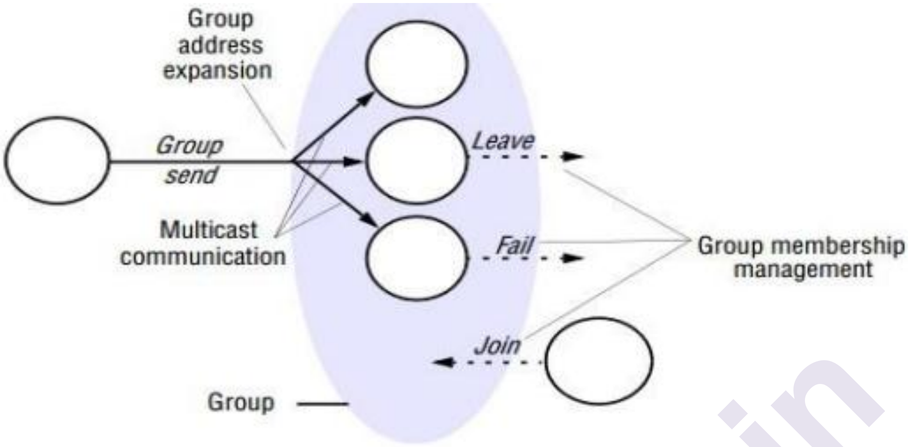


Fig 2.15: Group membership management

2.5 WHAT IS RPC?

Remote Procedure Call (RPC) is an interprocess communication technique. The extended form of RPC is Remote Procedure Call. The use of RPC is in used for client-server applications. RPC mechanisms are used when a computer program causes a procedure or subroutine to execute in a different address space, which is coded as a normal procedure call without the programmer specifically coding the details for the remote interaction.

RMI (Remote Method Invocation) is a way that a programmer, using the Java programming language and development environment, can write object-oriented programming in which objects on different computers can interact in a distributed network. RMI is the Java version of what is generally known as a remote procedure call (RPC), but with the ability to pass one or more objects along with the request. The object can include information that will change the service that is performed in the remote computer. Sun Microsystems, the inventors of Java, calls this "moving behaviour.

This procedure call also manages low-level transport protocol, such as User Datagram Protocol, Transmission Control Protocol/Internet Protocol etc. It is used for carrying the message data between programs.

2.5.1 Characteristics of RPC:

Here are the essential characteristics of RPC:

- The called procedure is in another process, which is likely to reside in another machine.

- The processes do not share address space.
- Parameters are passed only by values.
- RPC executes within the environment of the server process.
- It doesn't offer access to the calling procedure's environment.

2.5.2 Features of RPC:

Here are the important features of RPC:

- Simple call syntax
- Offers known semantics
- Provide a well-defined interface
- It can communicate between processes on the same or different machines

2.6 TYPES OF RPC

There are three types of RPC:

1. Callback RPC
2. Broadcast RPC
3. Batch-mode RPC

1. Callback RPC:

P2P paradigm between participating processes can enables RPC. With this client and server both can be work as service.

Functions of Callback RPC:

- Interactive application problems can be remotely processed
- It offers server with clients handle system
- Callback will allow the client process wait
- Will help in managing callback deadlocks
- It allows a peer-to-Peer paradigm among participating processes.

2. Broadcast RPC:

The client's request is called as Broadcast RPC; the processing request method can be processed by all servers on broadcast network.

Functions of Broadcast RPC:

- It allows you to specify that the client's request message has to be broadcasted.

- You can declare broadcast ports.
- Physical network can be reducing with the broadcast RPC

3. Batch-mode RPC:

Batch-mode RPC helps to queue, on the client-side, separate RPC requests, in a transmission buffer and then send them on a network in one batch to the server.

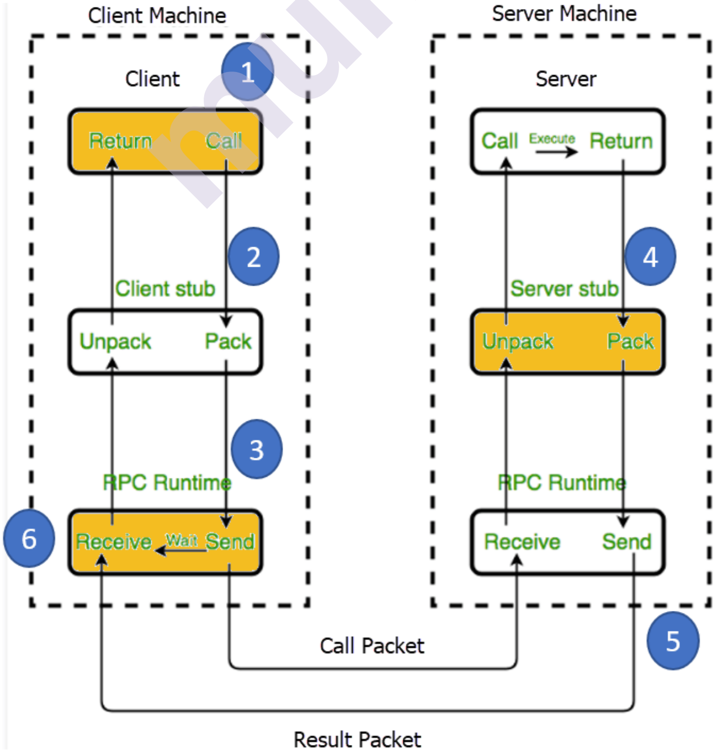
Functions of Batch-mode RPC:

- When request is sent over the network will minimizes overhead involved them over the network in one batch to the server.
- Only lower call rates applications are efficient for the type of RPC protocol.
- It needs a reliable transmission protocol.

2.7 RPC ARCHITECTURE

RPC architecture has mainly five components of the program:

1. Client
2. Client Stub
3. RPC Runtime
4. Server Stub
5. Server



RPC Architecture

How RPC Works?:

Step 1) **O**n the run time client execute, client stub, and one instance of RPC client machine.

Step 2) A client starts a client stub process by passing parameters. Client's own address space stores within the client stub. It acknowledges the server stub by local RPC Runtime.

Step 3) RPC called by the user by accessing regular Local Procedural Call. RPC Runtime manages broadcast the messages between the network across client and server. It also does acknowledgment, routing, performs the job of retransmission, and encryption.

Step 4) After getting over the server procedure, it gets back to the server stub, in which the message values are returned. In the transport layer the layer gets back the message by server stub.

Step 5) During the step, the client sends message result to the transport layer, which returns back a message to the client stub.

Step 6) In this stage, the return parameters called by the client stub, in the resulting packet, and the execution process returns to the caller.

2.8 ADVANTAGES & DISADVANTAGES OF RPC

Advantages of RPC:

- In high-level languages RPC method helps in transmission of message between clients and servers using the conventional procedure calls.
- on the local procedure call RPC method is modeled, but the called procedure is most likely to be executed in a different process and usually a different computer.
- Process and thread-oriented models can be used in RPC.
- RPC provides the feature of abstraction. The internal message passing mechanism hidden from the user in RPC.
- The effort needs to re-write and re-develop the code is minimum.
- To improve performance, it commits many of the protocol layers.
- In a distributed environment RPC allows the usage of the applications.

Disadvantages of RPC:

- In Remote Procedure the parameter is call by values and pointer which is not allowed in RPC.
- The time required for Remote procedure call is significantly lower than that for a local procedure.
- The probability of failure occurs is high as it involves a communication system, another machine, and another process.
- There are many different ways of RPC implementation, hence it is not standard.
- Flexibility is not offered in RPC for hardware architecture as It is mostly interaction-based.
- In remote procedure call, the cost of the process is increased.

2.9 RMI (REMOTE METHOD INVOCATION)

Remote Method Invocation:

The extended form of RMI is **Remote Method Invocation**. In this it allows an object residing in one system (JVM) to gain access /invoke an object running on another JVM.

In distributed application RMI is used. It is responsible for remote communication between Java programs. It is provided in the package **java.rmi**.

2.9.1 Goals of RMI:

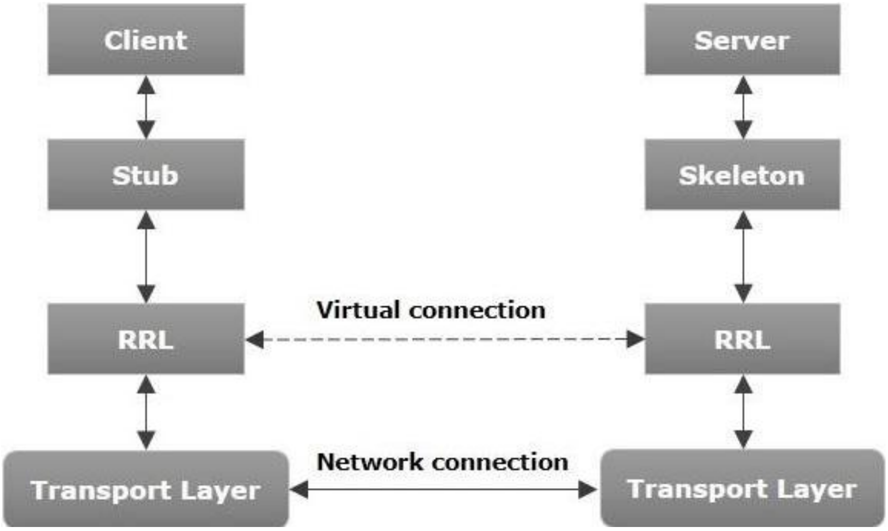
- To minimize the complexity of the application.
- To preserve type safety.
- Distributed garbage collection.
- Minimize the difference between working with local and remote objects.

2.9.2 Architecture of an RMI Application:

Server program and client program is the main feature of an RMI application, one has to write two individual programs for each.

- In the server program it is remote object which is created and reference of that object which can give access to the client.
- Whereas on the sever, the client program requests the remote objects and tries to invoke its methods.

The following diagram shows the architecture of an RMI application.



- **Transport Layer**: This layer is responsible for the connection between the client and the server. It also responsible to manage the existing connection and also sets up new connections.

- **Stub**: A representation (proxy) of the remote object at client is called as Stub. We will find it in the client system; it acts as a gateway for the client program.

- **Skeleton**: on the server side this object is residing. **stub** communicates with this skeleton to pass request to the remote object.

- **RRL (Remote Reference Layer)**: in this layer it manages the references made by the client to the remote object.

Working of an RMI Application:

- When the remote object gets call from the client object, it is first received by the stub and after that it eventually passes this request to the RRL.

- After getting request from the remote object to the client-side RRL, it invokes a method called **invoke ()** of the object **remoteRef**. RRL gets request form the server side.

- After passing the request on the server side it passes the request to the Skeleton (proxy on the server) which finally invokes the required object on the server.

- The result is passed all the way back to the client.

2.9.3 Marshalling and Unmarshalling:

Whenever a client invokes a method that accepts parameters on a remote object, the parameters are bundled into a message before being sent over the network. These parameters may be of primitive type or objects. In case of primitive type, the parameters are put together and a header is attached to it. In case the parameters are objects, then they are serialized. This process is known as **marshalling**.

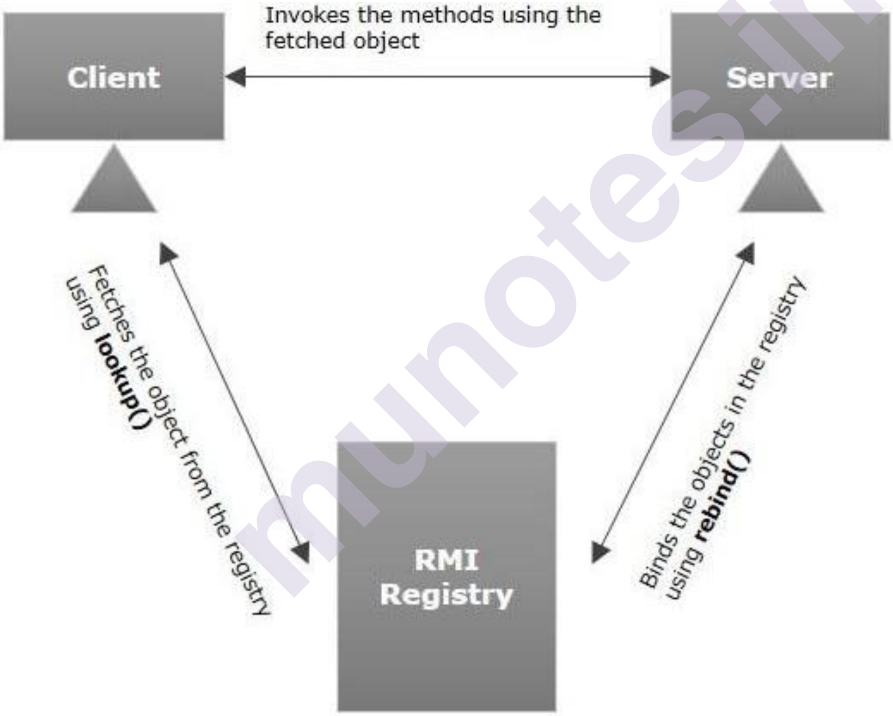
At the server side, the packed parameters are unbundled and then the required method is invoked. This process is known as **unmarshalling**.

RMI Registry:

RMI registry is a namespace on which all server objects are placed. Each time the server creates an object, it registers this object with the RMIregistry (using **bind()** or **reBind()** methods). These are registered using a unique name known as **bind name**.

To invoke a remote object, the client needs a reference of that object. At that time, the client fetches the object from the registry using its bind name (using **lookup()** method).

The following illustration explains the entire process:


Invokes the methods using the fetched object
Client
Server
Fetches the object from the registry using lookup()
Binds the objects in the registry using rebind()
RMI Registry

2.10 SUMMARY

To exchange the data, the cooperating processes, need to communicate with each this transaction of process can be called as Inter-process communication. it is the mechanism of communicating among processes. To established a shared memory region, require communicating process which can run through the shared memory model. The message passing mechanism provides an alternative means processes for communication. In this mode, processes interact with each other through messages with assistance from the underlying operating system. In group communication all the servers are connected together with each other hence the message is sent to a group and then this message is delivered to all server of the group. Group communication can be possible in both the environments. It is reliable and ordered in multicast message processing. The extended form of RMI is **Remote Method Invocation**. In this it allows an object residing in one system (JVM) to gain access /invoke an object running on another JVM. In case of primitive type, the parameters are put together and a header is attached to it. In case the parameters are objects, then they are serialized. This process is known as **marshalling**. At the server side, the packed parameters are unbundled and then the required method is invoked. This process is known as **unmarshalling**.

2.11 UNIT AND EXCERCISE

1. What is IPC? Explain Working of IPC.
2. Explain different modes of IPC
3. Explain different approaches of IPC
4. Explain RPC with its characteristics and features.
5. Explain RPC architecture with diagram.
6. List down advantages & disadvantages of RPC
7. What is RMI explain its working.
8. Explain concept of marshalling and unmarshalling
9. Write a note on RMI registry
10. Explain the concept of Group communication.

*****

UNIT II

3

CLOCK SYNCHRONIZATION

Unit Structure

3.1 INTRODUCTION

A distributed system is a collection of distinct processes that are spatially separated and run concurrently. In systems with multiple concurrent processes, it is economical to share the hardware or software resources among the concurrently executing processes. In such situation, sharing may be cooperative or competitive. Since the number of available resources in a computing system is restricted, one process must necessarily influence the action of other concurrently executing processes as it competes for resources. For example, a resource such as a tape drive that cannot be used simultaneously by multiple processes, a process willing to use it must wait if it is in use by another process. This chapter presents synchronization mechanisms that are suitable for distributed systems.

3.2 CLOCK SYNCHRONIZATION

Every computer needs a timer mechanism called a computer clock to keep track of current time, calculation of the time spent by a process in CPU utilization, disk I/O etc. so that the corresponding user can be charged properly. In a distributed system, an application may have processes that concurrently run on multiple nodes of the system. For correct results, such distributed applications require that the clocks of the nodes are synchronized with each other. For example, for a distributed **on-line reservation** system to be fair, seat booked (remaining) almost simultaneously from two different nodes should be offered to the client who booked first, even if the time difference between the two bookings is very small. It may not be possible to guarantee this if the clocks of the nodes of the system are not synchronized.

In a distributed system, synchronized clocks also enable one to measure the duration of distributed activities that start on one node and terminate on another node, for instance, calculating the time taken to transmit a message from one node to another at any arbitrary time. It is difficult to get the correct result in case if the clocks of the sender and receiver nodes are not synchronized.

The discussion above shows that it is the job of a distributed operating system designer to devise and use suitable algorithms for properly synchronizing the clocks of a distributed system. In this section we will describe such algorithms. However, for a better understanding of these algorithms, we will first discuss how computer clocks are implemented and what are the main issues in synchronizing the clocks of a distributed system.

3.2.1 How Computer Clocks Are Implemented:

A computer clock usually consists of three components- **a quartz crystal** that oscillates at a well-defined frequency, **a counter register**, and a **constant register**. The constant register is used to store a constant value that is decided based on the frequency of oscillation of the quartz crystal. The counter register is used to keep track of the oscillations of the quartz crystal. That is, the value in the counter register is decremented by 1 for each oscillation of the quartz crystal. When the value of the counter register becomes zero, an interrupt is generated, and its value is reinitialized to the value in the constant register. Each interrupt is called a **clock tick.**

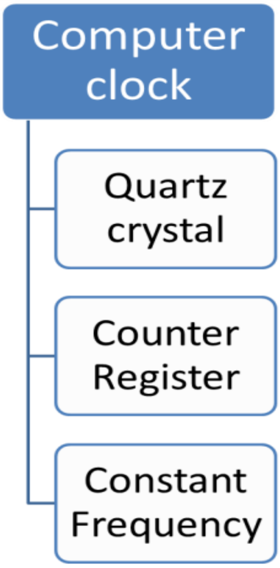


Figure 3.1

To make the computer clock function as an ordinary clock used by us in our day-to day life, the following things are done:

1. The value in the constant register is chosen so that 60 clock ticks occur in a second.

2. The computer clock is synchronized with real time (external clock). For this, two more values are stored in the system-a fixed starting date and time and the number of ticks. For example, in UNIX, time begins at 0000 on January 1, 1970. At the time of initial booting, the system asks the operator to enter the current date and time. The system converts the entered value to the number of ticks after the fixed starting date and time. At every clock tick, the interrupt service routine increments the value of the number of ticks to keep the clock running.

3.2.2 Drifting or Clocks:

A clock always runs at a constant rate because its quartz crystal oscillates at a well-defined frequency. However, the rates at which two clocks run are normally different from each other due to differences in the crystals. The difference in the oscillation period between two clocks might be extremely small, but the difference accumulated over many oscillations leads to an observable difference in the times of the two clocks, no matter how accurately they were initialized to the same value.

Therefore, as time passes, a computer clock drifts from the real-time clock that was used for its initial setting. For clocks based on a quartz crystal, the drift rate is approximately 10^{-6}, giving a difference of 1 second every 1,000,000 seconds, or 11.6 days [Coulouris et al. 1994]. Hence there is a need to resynchronize computer clock periodically with the real-time clock to keep it nonfaulty. Even nonfaulty clocks do not always maintain perfect time. A clock is considered nonfaulty if there is a bound on the amount of drift from real time for any given finite time interval.

Suppose that when the real time is t, the time value of a clock p is $C_p(t)$. If all clocks in the world were perfectly synchronized, we would have $\mathbf{C_p(t) = t}$ for all p and all t. That is, if C denotes the time value of a clock, in the ideal case **dC/dt** should be 1. Therefore, if the maximum drift rate allowable is p, a clock is said to be nonfaulty if the following condition holds for it:

$$1 - \rho \leq \frac{dC}{dt} \leq 1 + \rho$$

As shown in Figure 3.2, after synchronization with a perfect clock, slow and fast clocks drift in opposite directions from the perfect clock. This is because for slow clocks **dC/dt < 1** and for fast clocks **dC/dt > 1.**

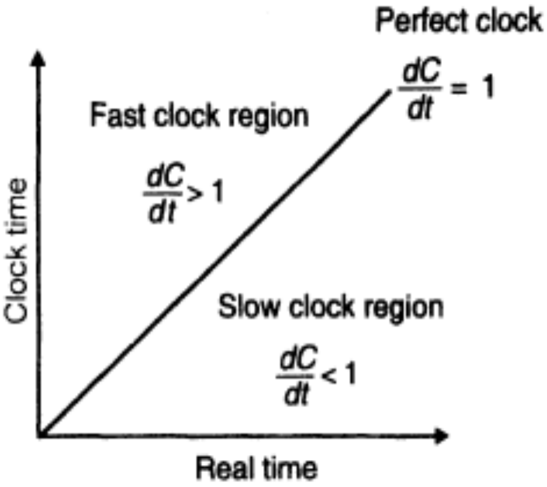


Figure 3.2 Slow, perfect, and fast clocks

A distributed system consists of several nodes, each with its own clock, running at its own speed. Because of the nonzero drift rates of all clocks, the set of clocks of a distributed system do not remain well synchronized without some periodic resynchronization. This means that the nodes of a distributed system must periodically resynchronize their local clocks to maintain a global time base across the entire system. Recall from Figure 3.2 that slow and fast clocks drift in opposite directions from the perfect clock.

Therefore, out of two clocks, if one is slow and one is fast, at a time Δt after they were synchronized, the maximum deviation between the time value of the two clocks will be $2\rho\Delta t$. Hence, to guarantee that no two clocks in a set of clocks ever differ by more than δ, the clocks in the set must be resynchronized periodically, with the time interval between two synchronizations being less than or equal to $\delta/2\rho$. Therefore, unlike a centralized system in which only the computer clock must be synchronized with the real-time clock, a distributed system requires the following types of clock synchronization:

1. Synchronization of the computer clocks with real-time (or external) clocks:

This type of synchronization is mainly required for real-time applications. That is, external clock synchronization allows the system to exchange information about the timing of events with other systems and users.

An external time source UTC (Coordinated Universal Time) is often used as a reference for synchronizing computer clocks with real time. The UTC is an international standard and many standard bodies disseminate UTC signals by radio, telephone, and satellite. For instance, the WWV radio station in the United States and the Geostationary Operational Environmental Satellites (GEOS) are two such standard bodies. Commercial devices (called time providers) are available to receive and interpret these signals. Computers equipped with time provider devices can synchronize their clocks with these timing signals.

2. Mutual (or internal) synchronization of the clocks of different nodes of the system:

This synchronization is mainly required for those applications that require a consistent view of time across all nodes of a distributed system as well as for the measurement of the duration of distributed activities that terminate on a node different from the one on which they start.

Note that externally synchronized clocks are also internally synchronized. However, the converse is not true because with the passage of time internally synchronized clocks may drift arbitrarily far from external time.

3.2.3 Clock Synchronization Issues:

We have seen that no two clocks can be perfectly synchronized. Therefore, in practice, two clocks are said to be synchronized at a instance of time if the difference in time values of the two clocks is less than some specified constant δ. The difference in time values of two clocks is called **clock skew**.

Clock synchronization requires each node to read the other nodes clock values. The actual mechanism used by a node to read other clocks differs from one algorithm to another. However, regardless of the actual reading mechanism, a node can obtain only an approximate view of its clock skew with respect to other nodes' clocks in the system. Errors occur mainly because of unpredictable communication delays during message passing used to deliver a clock signal or a clock message from one node to another. A minimum value of the unpredictable communication delays between two nodes can be computed by counting the time needed to prepare, transmit, and receive an empty message in the absence of transmission errors and any other system load. However, in general, it is rather impossible to calculate the upper bound of this value because it depends on the amount of communication and computation going on in parallel in the system, on the possibility that transmission errors will cause messages to be transmitted several times, and on other random events, such as page faults, process switches, or the establishment of new communication routes.

An important issue in clock synchronization is that time must never run backward because this could cause serious problems, such as the repetition of certain operations that may be hazardous in certain cases. Notice that during synchronization a fast clock has to be slowed down. However, if the time of a fast clock is readjusted to the actual time all at once, it may lead to running the time backward for that clock.

Therefore, clock synchronization algorithms are normally designed to gradually introduce such a change in the fast running clock instead of readjusting it to the correct time all at once. One way to do this is to make the interrupt routine more intelligent. When an intelligent interrupt routine is instructed by the clock synchronization algorithm to slow down its clock, it readjusts the amount of time to be added to the clock time for each interrupt. For example, suppose that if 8 msec is added to the clock time on each interrupt in the normal situation, when slowing down, the interrupt routine only adds 7 msec on each interrupt until the correction has been made. Although not necessary, for smooth readjustment, the intelligent interrupt routine may also advance its clock forward, if it is found to be slow, by adding 9 msec on each interrupt, instead of readjusting it to the correct time all at once.

3.2.4 Clock Synchronization Algorithms:

Clock synchronization algorithms may be broadly classified as centralized and distributed.

Centralized Algorithms:

In centralized clock synchronization algorithms one node has a real-time receiver usually called the **time server node**, and the clock time of this node is regarded as correct and used as the **reference time**. The goal of the algorithm is to keep the clocks of all other nodes synchronized with the clock time of the time server node. Examples of centralized algorithms are- Berkeley Algorithm, Passive Time Server, Active Time Server etc.

Distributed Algorithms:

Distributed is the one in which there is no centralized time server present. Instead the nodes adjust their time by using their local time and then, taking the average of the differences of time with other nodes.

Distributed algorithms overcome the issue of centralized algorithms like the scalability and single point failure. Examples of Distributed algorithms are – Global Averaging Algorithm, Localized Averaging Algorithm, NTP (Network time protocol) etc.

3.3 MUTUAL EXCLUSION

There are many resources in a system that need not be used simultaneously by multiple processes if program operation is to be correct. For example, a file cannot be simultaneously updated by multiple processes. Similarly, use of tape drives or printers must be restricted to a single process at a time. Therefore, exclusive access to such shared resource by a process must be ensured. This exclusiveness of access is called mutual exclusion between processes. The sections of a program that need exclusive access to shared resources are called as critical sections.

An algorithm for implementing mutual exclusion must satisfy the following requirements:

1. Mutual exclusion:

At any time only one process should access the shared resource. That is, a process that has been granted the resource must release it before it can be granted to another process.

2. No starvation:

If every process that is granted the resource eventually releases it, every request must be eventually granted.

Three basic approaches used by different algorithms for implementing mutual exclusion in distributed systems are described below. To simplify our description, we assume that each process resides at a different node.

3.3.1 Centralized Approach:

In this approach, one of the processes in the system is elected as the coordinator and coordinates the entry to the critical sections. Each process that wishes to enter a critical section must first seek permission from the coordinator. If no other process is currently in that critical section, the coordinator can immediately grant permission to the requesting process. However, if two or more processes concurrently ask for permission to enter same critical section, the coordinator grants permission to only one process at a time in accordance with some scheduling algorithm. After executing a critical section, when a process exits the critical section, it must notify the coordinator process so that the coordinator can grant permission to another process (if any) that has also asked for permission to enter the same critical section.

An algorithm for mutual exclusion using centralized approach is described here with the help of an example. As shown in Figure 3.3, assume that there is a coordinator process P_c and three other processes P_1, P_2 and P_3 in the system. Also assume that the requests are granted in the first-come, first-served manner for which the coordinator maintains a request queue. Suppose P_1 wants to enter a critical section for which it sends a **request** message to P_c. On receiving the request message, P_c checks to see whether some other process is currently in that critical section. Since no other process is in the critical section, P_c immediately sends back a **reply** message granting permission to P_l. When reply arrives, P_1 enters critical section.

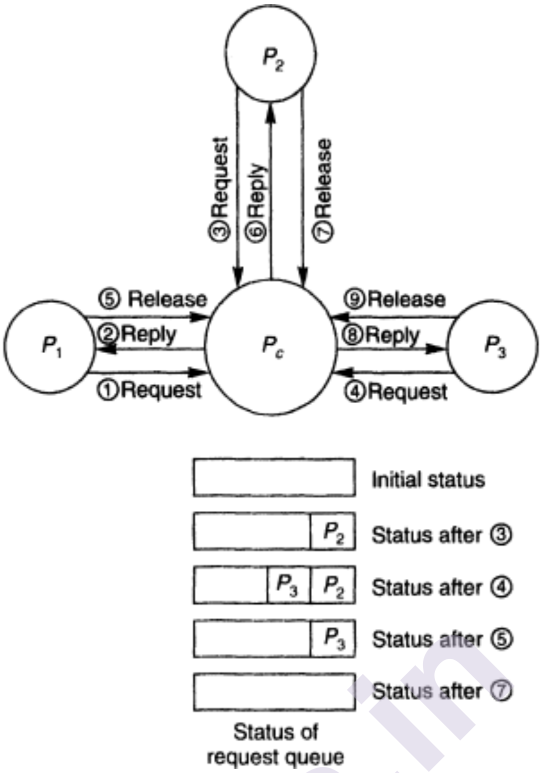


Figure 3.3 Example illustration centralize approach for mutual exclusion

Now suppose that while P_1 is in the critical section and at same time P_2 asks for permission to enter the same critical section by sending a request message to P_c. Since P_1 is already in the critical section, permission cannot be granted to P_2. The exact method used to deny permission varies from one algorithm to another. For our algorithm, let us assume that the coordinator does not return any **reply** and the process that made the request remains blocked until it receives the **reply** from the coordinator. Therefore, P_c does not send a reply to P_2 immediately and enters its request in the request queue.

Again, assume that while P_1 is still in the critical section and at same time P_3 also sends a request message to P_c asking for permission to enter same critical section. Obviously, P_3 cannot be granted permission, so no reply is sent immediately to P_3 by $P_{c;}$ and its request is queued in the request queue.

Now suppose P_1 exits the critical section and sends a **release** message to P_c releasing its exclusive access to the critical section. On receiving the release message, P_c takes the first request from the queue of **deferred requests** and sends a reply message to the corresponding process, granting it permission to enter the critical section. Therefore, in this case, P_c sends a reply message to P_2.

After receiving the reply message, P_2 enters the critical section, and when it exits the critical section, it sends a release message to P_c. Again P_c takes the first request from the request queue (in this case request of P_3) and sends a reply message to the corresponding process (P_3) On receiving the reply message, P_3 enters the critical section, it sends a release message to P_c, when it exits the critical section. Now since there are no more requests, P_c keeps waiting for the next request message.

This algorithm ensures mutual exclusion because, at a time, the coordinator allows only one process to enter a critical section. Use of first-come, first-served scheduling policy ensures that no starvation will occur. This algorithm is simple to implement and requires only three messages per critical section entry: **a request, a reply, and a release.**

However, it suffers from the usual drawbacks of centralized schemes. That is, a single coordinator is subject to a single point of failure and can become a performance bottleneck in a large system. Furthermore, for handling failure, mechanism must be provided to detect a failure of the coordinator, to elect a unique new coordinator, and to reconstruct its request queue before the computation can be resumed.

3.3.2 Distributed Approach:

In the distributed approach, the decision making for mutual exclusion is distributed across the entire system. All processes that want to enter the same critical section cooperate with each other before reaching to a decision on which process will enter the critical section next. The first such algorithm was presented by Lamport [1978] based on his event-ordering scheme. Later, Ricart and Agrawala [1981] proposed a more efficient algorithm which also requires that there should be a total ordering of all events in the system. As an example of a distributed algorithm for mutual exclusion, Ricart and Agrawala's algorithm is described below. In the following description we assume that Lamport's event ordering scheme is used to generate a unique timestamp for each event in the system.

When a process wants to enter a critical section, it sends a request message to all other processes. The message contains the following information:

1. The process identifier of the process.

2. The name of the critical section that the process wants to enter.

3. A unique timestamp generated by the process for the request message.

On receiving a request message, a process either immediately sends back a reply message to the sender or defers sending a reply based on the following rules:

1. If the receiver process is currently executing in the critical section, it simply queues the request message and defers sending a reply.

2. If the receiver process is currently not executing in the critical section but is waiting for its turn to enter the critical section, it compares the timestamp in the **received request message** with the timestamp in its **own request message** that it has sent to other processes. If the timestamp of the received request message is lower, it means that the sender process made a request before the receiver process to enter the

critical section. Therefore, the receiver process immediately sends back a reply message to the sender. On the other hand, if the receiver process's own request message has a lower timestamp, the receiver queues the received request message and defers sending a reply message.

3. If the receiver process neither is in the critical section nor is waiting for its turn to enter the critical section, it immediately sends back a reply message.

A process that sends out a request message keeps waiting for reply messages from other processes. It enters the critical section as soon as it has received reply messages from all processes. After finishing execution in the critical section, it sends reply messages to all processes in its queue and deletes them from its queue.

Let us consider the example of Figure 3.4, to illustrate how the algorithm works. There are four processes P_1, $P_{2,}$ P_3 and $P_{4.}$ While process P_4 is in a critical section, processes P_1 and P_2 want to enter the same critical section. To get permission from other processes, processes P_1 and P_2 send request messages with timestamps 6 and 4 respectively to other processes (Figure 3. 4(a)).

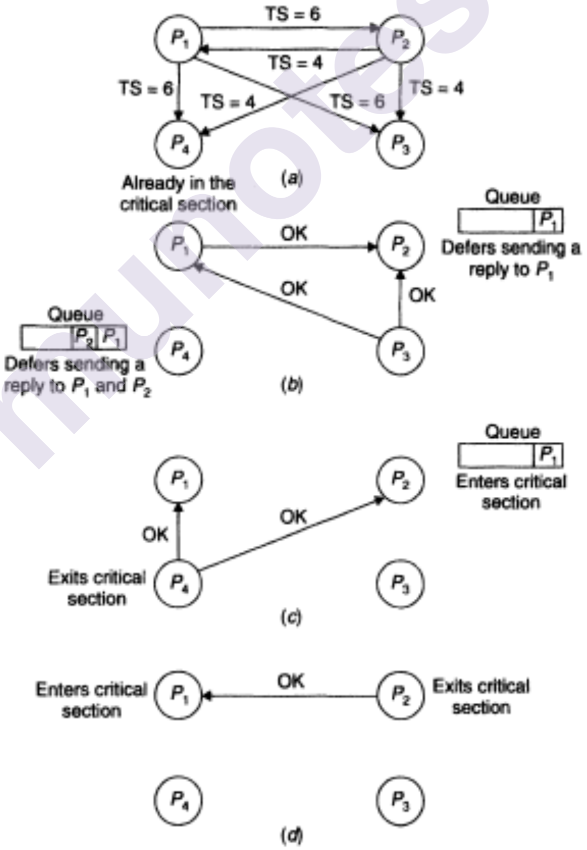


Figure 3.4 Example illustrating the distributed algorithm for mutual exclusion

(a) status when processes P_1 and P_2 send request messages to other processes while process P_4 is already in the critical section;

(b) status while process P_4 is still in critical section;

(c) status after process P_4 exits critical section;

(d) status after process P_2 exits critical section.

Now let us consider the situation in Figure 3.4(b). Since process P_4 is already in the critical section, it defers sending a reply message to P_1 and P_2 and enters them in its queue. Process P_3 is currently not interested in the critical section, so it sends a reply message to both P_1 and P_2.

Process P_2 defers sending a reply message to P_1 and enters P_1 in its queue, because the timestamp (4) in its own request message is less than the timestamp (6) in P_1'S request message. On the other hand, P_1 immediately replies to P_2 because the timestamp (6) in its request message is found to be greater than the timestamp (4) of P_2'S request message.

Next consider the situation in Figure 3.4(c). When process P_4 exits the critical section, it sends a reply message to all processes in its queue (in this case to processes P_1 and P_2) and deletes them from its queue. Now since process P_2 has received a reply message from all other processes ($P_{1,}$ P_3 and P_4), it enters the critical section. However, process P_1 continues to wait since it has not yet received a reply message from process P_2.

Finally, when process P_2 exits the critical section, it sends a reply message to P_1 (Fig. 3.4(d). Since process P_1 has received a reply message from all other processes, it enters the critical section.

This algorithm guarantees mutual exclusion because a process can enter its critical section only after getting permission from all other processes, and in the case of a conflict, only one of the conflicting processes can get permission from all other processes. Algorithm also ensures freedom from starvation since entry to the critical section is scheduled according to the timestamp ordering.

It has also been proved by Ricart and Agrawala [1981] that the algorithm is free from deadlock. Furthermore, if there are n processes, the algorithm requires n-1 request messages and n-1 reply messages, giving a total of 2(n-1) messages per critical section entry.

This algorithm suffers from the following drawbacks because of the requirement that all processes must participate in a critical section entry request by any process:

1. In a system having n processes, the algorithm is liable to n points of failure because if process fails, the entire scheme collapses. This is because the failed process will not reply to request messages that will be falsely interpreted as denial of permission by the requesting processes, causing all the requesting processes to wait indefinitely.

   To solve this problem **Tanenbaum [1995]** proposed a small modification to the algorithm. In the modified algorithm, instead of remaining silent by deferring the sending of the reply message in cases when permission cannot be granted immediately, the receiver sends a **"permission denied"** reply message to the requesting process and then later sends an **OK** message when the permission can be granted. Therefore, a reply message (either "permission denied" or OK) is immediately sent to the requesting process in any case. If the requesting process does not receive a reply from a process within a fixed time period, it either keeps trying until the process replies or concludes that the process has crashed. When the requesting process receives a **"permission denied"** reply message from one or more of the processes, it blocks until **OK** message is received from all of them.

2. The algorithm requires that each process knows the identity of all the processes participating in the mutual-exclusion algorithm. This requirement makes implementation of the algorithm complex because each process of a group needs to dynamically keep track of the processes entering or leaving the group. That is, when a process joins a group, it must receive the names of all the other processes in the group, and the name of the new process must be distributed to all the other processes in the group. Similarly, when a process leaves the group or crashes, all members of that group must be informed so that they can delete it from their membership list. Updating of the membership list is little bit difficult when request and reply messages are already being exchanged among the processes of the group. Therefore, the algorithm is suitable only for groups whose member processes are fixed and do not change dynamically.

3. In this algorithm, a process can enter a critical section only after communicating with all other processes and getting permission from them. Therefore, assuming that the network can handle only one message at a time, the waiting time from the moment the process makes a request to enter a critical section until it actually enters the critical section is the time for exchanging 2(n-1) messages in a system having n processes. This waiting time may be large if there are too many processes in the system. Therefore, the algorithm is suitable only for a small group of cooperating processes.

3.3.3 Token-Passing Approach:

In this approach, mutual exclusion is achieved by using a **single token** that is circulated among the processes in the system. A token is a special type of message that entitles its holder to enter a critical section. For fairness, the processes in the system are logically organized in a ring structure, and the token is circulated from one process to another around the ring always in the same direction (clockwise or anticlockwise).

When a process receives the token, it checks if it wants to enter a critical section and acts as follows:

- If it wants to enter a critical section, it keeps the token with it, enters the critical section, and exits from the critical section after finishing its work. It then passes the token along the ring to its neighbor process. Note that the process can enter only one critical section when it receives the token. If it wants to enter another critical section, it must wait until it gets the token again.

- If it does not want to enter a critical section, it just passes the token along the ring to its neighbor processes. Therefore, if no process is interested in entering a critical section, the token simply keeps on circulating around the ring.

Mutual exclusion is guaranteed by this algorithm because at any instance of time only one process can be in a critical section, since there is only a single token. Furthermore, since the ring is unidirectional and a process is permitted to enter only one critical section each time it gets the token, starvation cannot occur.

In this algorithm the number of messages per critical section entry may vary from 1 (when every process always wants to enter a critical section) to an unbounded value (when no process wants to enter a critical section). Moreover, for a total of n processes in the system, the waiting time from the moment a process wants to enter a critical section until its actual entry may vary from the time needed to exchange 0 to n-1 token-passing messages. Zero token-passing messages are needed when the process receives the token just when it wants to enter the critical section, whereas n-1 messages are needed when the process wants to enter a critical section just after it has passed the token to its neighbor process.

The algorithm requires the handling of the following types of failures:

1. Process failure:

A process failure in the system causes the logical ring to break. In such situation, a new logical ring must be established to ensure the continued circulation of the token among other processes. This requires detection of a failed process and dynamic reconfiguration of the logical ring when a failed process is detected or when a failed process recovers after failure.

Failed process can be easily detected by making it a rule that a process receiving the token from its neighbor always sends an acknowledgment message to its neighbor. With this rule, a process detects that its neighbor has failed when it sends the token to it but does not receive the acknowledgment message within a fixed time period.

On the other hand, dynamic reconfiguration of the logical ring can be done by maintaining the current ring configuration with each process. When a process detects that its neighbor has failed, it removes the failed process from the group by skipping it and passing the token to the next alive process in the sequence. When a process becomes alive after recovery, it simply informs its previous neighbor in the ring so that it gets the token during the next round of circulation.

2. Lost token:

If the token is lost, a new token must be generated. The algorithm must have mechanisms to detect and regenerate a lost token. One method to solve this problem is to designate one of the processes on the ring as a **"monitor"** process. The monitor process periodically circulates a **"who has the token?"** message on the ring. This message rotates around the ring from one process to another.

All processes simply pass this message to their neighbor process, except the process that has the token when it receives this message. This process writes its identifier in a special field of the message before passing it to its neighbor. After one complete round, when the message returns to the monitor process it checks the special field of the message. If there is no entry in this field, it concludes that the token has been lost, generates a new token, and circulates it around the ring.

There are two problems associated with this method- **the monitor process may itself fail** and **the "who has the token?" message may itself get lost**. Both problems may be solved by using more than one monitor processes. Each monitor process independently checks the availability of the token on the ring. However, when a monitor process detects that the token is lost, it holds an election with other monitor processes to decide which monitor process will generate and circulate a new token. An election is required to prevent the generation of multiple tokens that may happen when each monitor process independently detects that the token is lost, and each one generates a new token.

3.4 REFERENCE

- Pradeep K. Sinha, Distributed Operating System: Concepts and Design, PHI Learning.

*****

4

ELECTION ALGORITHMS

Unit Structure

4.1 DEADLOCK

In the previous section we saw that there are several resources in a system for which the resource allocation policy must ensure exclusive access by a process. A system consists of a finite number of units of each resource type for example, three printers, six tape drives, four disk drives, two CPUs, etc. When multiple concurrent processes have to compete to use resource, the sequence of events required to use a resource by a process is as follows:

1. Request:

The process first makes a request for the resource. If the requested resource is not available, possibly because it is being used by another process, the requesting process must wait until the requested resource is allocated to it by the system.

Note that if the system has multiple units of the requested resource type, the allocation of any unit of the type will satisfy the request. Process may request as many units of a resource as it requires with the restriction that the number of units requested may not exceed the total number of available units of the resource.

2. Allocate:

The system allocates the resource to the requesting process as soon as possible. It maintains a table in which it keeps records of each resource whether it is free or allocated and, if it is allocated, to which process its allocated. If the requested resource is currently allocated to another process, the **requesting process** is added to a queue of processes waiting for this resource. Once the system allocates the resource to the requesting process, that process can exclusively use the resource by operating on it.

3. Release:

Once the process has finished using the allocated resource, it releases the resource to the system. The system table records are updated at the time of allocation and release to reflect the current status of availability of resources.

The request and release of resources are called system calls, such as **request** and **release** for devices, open and close for files, and allocate and free for memory space. Notice that of the three operations, allocate is the only operation that the system can control. The other two operations are initiated by a process. If the total request made by multiple concurrent processes for resources of a certain type exceeds the available amount, some way is required to order the assignment of resources in time. Extra care must be taken that the strategy applied cannot cause a deadlock, that is, a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

It may happen that some processes that have entered the waiting state (because the requested resources were not available at the time of request) will never again change state, because the resources they have requested are held by other waiting processes. This situation is called **deadlock**, and the processes involved are said to be deadlocked. Hence, deadlock is the state of permanent blocking of a set of processes each of which is waiting for an event that only another process in the set can cause.

A deadlock situation can be explained with the help of an example given below. Suppose that a system has two processes P1 and P2 & two tape drives T1 and T2; resource allocation strategy is such that a requested resource is immediately allocated to the requester if the resource is free. Suppose that two concurrent processes P1 and P2 make requests for the tape drives in the following order:

1. P1 requests for one tape drive and the system allocates tape drive T1 to it.

2. P2 requests for one tape drive and the system allocates tape drive T2 to it.

3. Now P1 requests for one more tape drive and enters a waiting state because no tape drive is currently available.

4. Now P2 requests for one more tape drive and it also enters a waiting state because no tape drive is currently available.

Now onwards, P1 and P2 will wait for each other indefinitely, since P1 will not release T1 until it gets T2 to carry out its designated task, that is, not until P2 has released T2, whereas P2 will not release T2 until it gets T1. Therefore, the two processes are in a state of deadlock. Note that the requests made by the two processes are totally legal because each is requesting for only two tape drives, which is the total number of tapes drives available in the system. However, the deadlock problem occurs because the total requests of both processes exceed the total number of units for the tape drive and the resource allocation policy is such that it immediately allocates a resource on request if the resource is free.

In the context of deadlocks, the term "resource" applies not only to physical objects such as tape and disk drives, printers, CPU cycles, and memory space but also to logical objects such as a locked record in a database, files, tables, semaphores, and monitors. However, these resources should permit only exclusive use by a single process at a time and should be nonpreemptable. A nonpreemptable resource is one that cannot be taken away from a process to which it was allocated until the process voluntarily releases it. If taken away, it has ill effects on the computation already performed by the process. For example, a printer is a nonpreemptable resource because taking the printer away from a process that has started printing but has not yet completed its printing job and giving it to another process may produce printed output that contains a mixture of the output of the two processes. This is certainly unacceptable.

4.1.1 Necessary Conditions for Deadlock:

Coffman et al. [1971] stated that the following conditions are necessary for a deadlock situation to occur in a system:

1. Mutual-exclusion:

If a resource is held by a process, any other process requesting for that resource must wait until the resource has been released.

2. Hold-and-wait:

Processes can request for new resources without releasing the resources that they are currently holding.

3. No-preemption:

A resource that has been allocated to a process becomes available for allocation to another process only after it has been voluntarily released by the process holding it.

4. Circular-wait:

Two or more processes must form a circular chain in which each process is waiting for a resource that is held by another process.

A set of processes are waiting for each other in a circular fashion. For example, let’s say there are a set of processes {P0, P1, P2, P3} such that P0 depends on P1, P1 depends on P2, P2 depends on P3 and P3 depends on P0. This creates a circular relation between all these processes, and they have to wait forever to be executed.

All four conditions must hold simultaneously in a system for a deadlock to occur. If anyone of them is absent, no deadlock can occur. Notice that the four conditions are not completely independent because the circular-wait condition implies the hold-and-wait condition. Although these four conditions are somewhat interrelated, it is quite useful to consider them separately to devise methods for deadlock prevention.

Example:

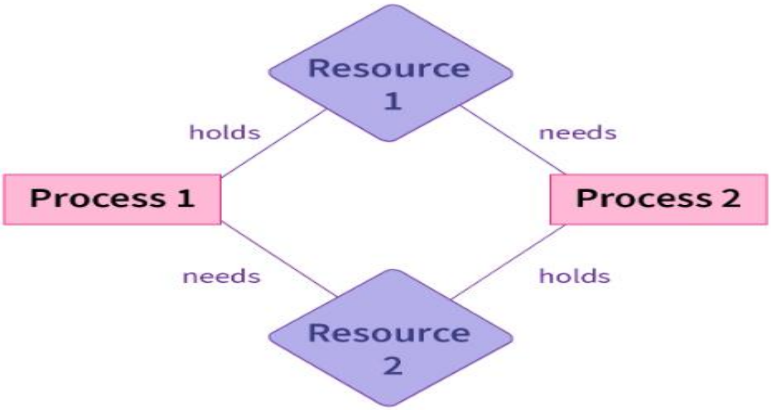


Figure 4.1 Deadlock example

In the above figure, there are two processes and two resources. Process 1 holds "Resource 1" and needs "Resource 2" while Process 2 holds "Resource 2" and requires "Resource 1". This creates a situation of deadlock because none of the two processes can be executed. Since the resources are non-shareable, they can only be used by one process at a time (Mutual Exclusion). Each process is holding a resource and waiting for the other process to release the resource it requires. None of the two processes releases their resources before their execution and this creates a circular wait. Therefore, all four conditions are satisfied.

4.1.2 Deadlock Modeling:

Deadlocks can be modeled using directed graphs. Before presenting a graphical model for deadlocks, some terminology from graph theory is needed:

1. Directed graph:

A directed graph is a pair (N, E), where N is a nonempty set of nodes and E is a set of directed edges. A directed edge is an ordered pair (a, b), where a and b are nodes in N.

2. Path:

A path is a sequence of nodes (a, b, c, ...i, j) of a directed graph such that (a, b), (b, c), ... , (i, j) are directed edges. Obviously, a path contains at least two nodes.

3. Cycle:

A cycle is a path whose first and last nodes are the same.

4. Reachable set:

The reachable set of a node a is the set of all nodes b such that a path exists from a to b.

5. Knot:

A knot. is a nonempty set K of nodes such that the reachable set of each node in K is exactly the set K. A knot always contains one or more cycles.

An example of a directed graph is shown in Figure 4.2.

The graph has a set of nodes {a, b, c, d, e, f} and a set of directed edges {(a, b), (b, c), (c, d), (d, e), (e, f) (f, a), (e, b)}. It has two cycles (a, b, c, d, e, f, a) and (b, c, d, e, b). It also has a knot {a, b, c, d, e, f} that contains the two cycles of the graph.

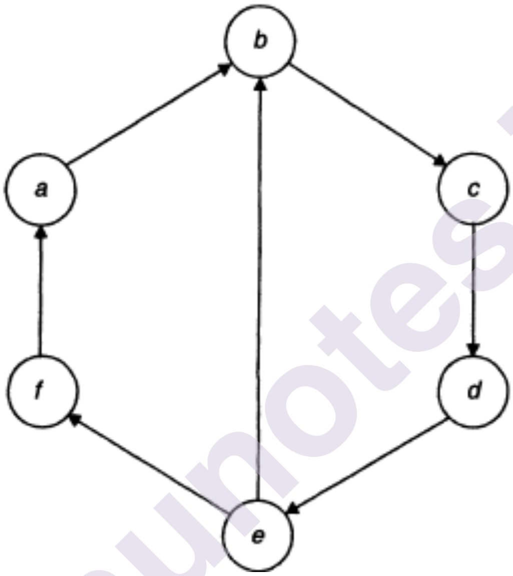


Figure 4.2 A directed graph

For deadlock modeling, a directed graph, called a resource allocation graph, is used in which both the set of nodes and the set of edges are partitioned into two types, resulting in the following graph elements:

1. Process nodes:

A process node represents a process of the system. In a resource allocation graph, it is normally shown as a circle, with the name of the process written inside the circle (nodes P1, P2, and P3 of Fig. 4.3)

2. Resource nodes:

A resource node represents a resource of the system. In a resource allocation graph, it is normally shown as a rectangle with the name of the resource written inside the rectangle. Since a resource type R, may have more than one unit in the system, each such unit is represented as a bullet within the rectangle. For instance, in the resource allocation graph of

Figure 4.3, there are two units of resource R1, one unit of R2, and three units of R3.

3. Assignment edges:

An assignment edge is a directed edge from a resource node to a process node. It signifies that the resource is currently held by the process. In multiple units of a resource type, the tail of an assignment edge touches one of the bullets in the rectangle to indicate that only one unit of the resource is held by that process. Edges (R1, P1), (R1, P3), and (R2, P2) are the three assignment edges in the resource allocation graph of Figure 4.3.

4. Request edges:

A request edge is a directed edge from a process node to a resource node. It signifies that the process made a request for a unit of the resource type and is currently waiting for that resource. Edges (P1, R2) and (P2, R1) are the two request edges in the resource allocation graph of Figure 4.3.

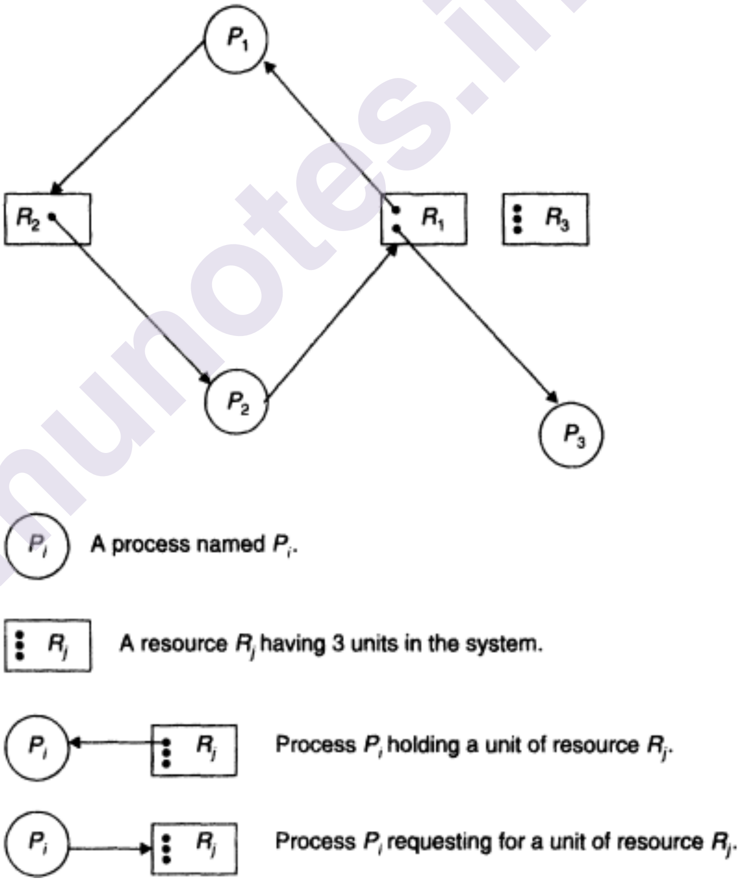

P_1
R_2
R_1
R_3
P_2
P_3
P_i
A process named P_i.
R_j
A resource R_j having 3 units in the system.
Process P_i holding a unit of resource R_j.
Process P_i requesting for a unit of resource R_j.

Figure 4.3 Resource allocation graph

A resource allocation graph provides an overall view of the processes holding or waiting for the various resources in the system. Therefore, the graph changes dynamically as the processes in the system request for or release resources or the system allocates a resource to a process. That is, when a process Pi requests for a unit of resource type Rj , a request edge

(Pi, Rj) is inserted in the resource allocation graph. When this request can be fulfilled, a unit of resource Rj is allocated to Pi and the request edge (Pi, Rj) is instantaneously transformed to an assignment edge (Rj , Pi). Later, when Pi releases Rj , the assignment edge (Rj , Pi) is deleted from the graph.

4.1.3 Deadlock Prevention:

We can prevent Deadlock by eliminating any of the below four conditions.

1. Mutual-exclusion
2. Hold-and-wait
3. No-preemption
4. Circular-wait

1. Eliminate Mutual Exclusion:

It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tape drive and printer, are inherently non-shareable.

Let's take a practical example to understand this issue. Jack and Jones share a bowl of soup. Both wants to drink the soup from the same bowl and use a single spoon simultaneously, which is not feasible.

2. Eliminate Hold and wait:

1. Allocate all required resources to the process at the start of its execution, this way hold and wait condition is eliminated but it will lead to low device utilization. For example, if a process requires printer later and we have allocated printer before the start of its execution printer will remain blocked till it has completed its execution.
2. The process will make a new request for resources after releasing the current set of resources. This solution may lead to starvation.

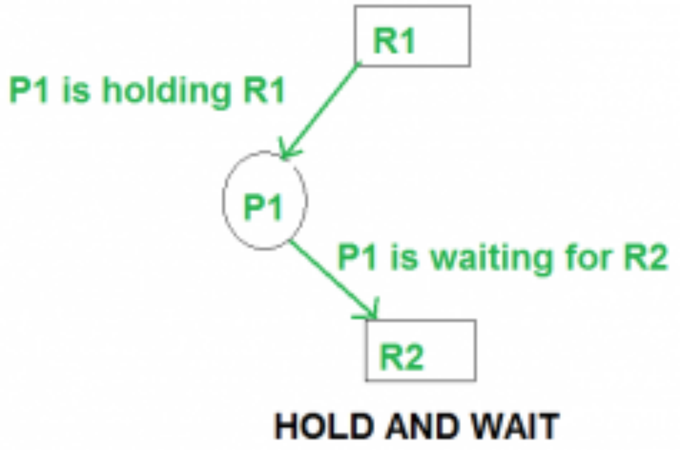


Figure 4.4

3. Eliminate No Preemption:

Preempt resources from the process when resources are required by other high priority processes.

4. Eliminate Circular Wait:

Each resource will be assigned a numerical value, and a process has to access the resource in increasing or decreasing order. For Example, if process P1 is allocated resource R5, now next time if P1 ask for R4 or R3 which is lesser than R5; this request will not be granted, only request for resources greater than R5 will be granted.

Distributed system:

Distributed system is a collection of independent computers that do not share their memory. Each processor has its own memory and they communicate via communication networks. Communication in networks is implemented in a process on one machine communicating with a process on another machine.

Distributed Algorithm is an algorithm that runs on a distributed system. Several distributed algorithms require that there be a coordinator process in the entire system that performs some type of coordination activity needed for the smooth running of other processes in the system. Since all other processes in the system must interact with the coordinator, they all must unanimously agree on who the coordinator is. Furthermore, if the coordinator process fails due to the failure of the site on which it is located, a **new coordinator** process must be elected to take up the job of the **failed coordinator**.

4.2 ELECTION ALGORITHMS

Election algorithms are meant for electing a coordinator process from the currently running processes in such a manner that at any instance of time there is a single coordinator for all processes in the system.

Election algorithms are based on the following assumptions:

1. Each process in the system has a unique priority number.

2. Whenever an election is held, the process having the highest priority number among the currently active processes is elected as the coordinator.

3. On recovery, a failed process can take appropriate actions to rejoin the set of active processes.

Therefore, whenever initiated, an election algorithm finds out which of the currently active processes has the highest priority number and then it informs this to all other active processes. Different election algorithms differ in the way they do this. Two such election algorithms are described below. In the description of both algorithms we will assume that there is only one process on each node of the distributed system.

4.2.1 The Bully Algorithm:

This algorithm was proposed by Garcia-Molina in 1982. In this algorithm it is assumed that every process knows the priority number of every other process in the system. The algorithm works as follows.

When any process (say Pi) sends a **request** message to the coordinator and does not receive a reply within a fixed time period, it assumes that the coordinator has failed. It then initiates an election by sending an **election** message to every process with a higher priority number than itself. If Pi does not receive any response to its election message within a fixed time period, it assumes that it has the highest priority number among the currently active processes.

Therefore, it takes up the job of the coordinator and sends a message (let us call it a coordinator message) to all processes having lower priority numbers than itself, informing that it is elected as their new coordinator. On the other hand, if Pi receives a response for its election message, this means that some other process having higher priority number is alive. Therefore, Pi does not take any further action and just waits to receive the result (a coordinator message from the new coordinator) of the election it initiated.

When a process (say P_j) receives an election message (obviously from a process having a lower priority number than itself), it sends -a response message (let us call it **alive** message) to the sender informing that it is alive and will take over the election activity. Now P_j holds an election if it is not already holding one. In this way, the election activity gradually moves on to the process that has the highest priority number among the currently active processes and eventually wins the election and becomes the new coordinator.

As part of the recovery action, this method requires that a failed process (say P_k) must initiate an election on recovery. If the current coordinator's priority number is higher than that of P_k, then the current coordinator will win the election initiated by P_k and will continue to be the coordinator. On the other hand, if P_k's priority number is higher than that of the current coordinator, it will not receive any response for its election message. So, it wins the election and takes over the coordinator's job from the currently active coordinator. Therefore, **the active process having the highest priority number always wins the election.**

Hence the algorithm is called the "bully" algorithm. If the process having the highest priority number recovers after a failure, it does not initiate an election because it knows from its list of priority numbers that all other processes in the system have lower priority numbers than that of its own. Therefore, on recovery, it simply sends a coordinator message to all other processes and bullies the current coordinator into submission.

Now we will see working of this algorithm with the help of an example. Suppose the system consists of five processes P_1, P_2, P_3, P_4 and P_5 and their priority numbers are 1, 2, 3, 4, and 5 respectively. Also assume that at an instance of time the system is in a state in which P_2 is crashed, and P_1, P_3, P_4 and P_5 are active. Starting from this state, the functioning of the bully algorithm with the changing system states is illustrated below.

1. P_5 is the coordinator in the starting state. Suppose P_5 crashes.

2. Process P_3 sends a request message to P_5 and does not receive a reply within the fixed time period.

3. Process P_3 assumes that P_5 has crashed and it initiates an election by sending an election message to P_4 and P_5. An election message is sent only to processes with higher priority numbers.

4. When P_4 receives P_3's election message, it sends an alive message to $P_{3,}$ informing that it is alive and will take over the election activity. Process P_5 cannot respond to P_3's election message because it is down.

5. Now P_4 holds an election by sending an election message to P_5.

6. Process P_5 does not respond to P_4's election message because it is down, and therefore, P_4 wins the election and sends a coordinator message to P_1, P_2 and P_3 informing them that from now onwards it is new coordinator. Obviously, this message is not received by P_2 because it is currently down.

7. Now suppose P_2 recovers from failure and initiates an election by sending an election message to P_3, P_4 and P_5. Since P_2's priority number is lower than that of P_4 (current coordinator), P_4 will win the election initiated by P_2 and will continue to be the coordinator.

8. Finally, suppose P_5 recovers from failure. Since P_5 is the process with the highest priority number, it simply sends a coordinator message to P_1, P_2, P_3 and P_4 and becomes the new coordinator.

4.2.2 A Ring Algorithm:

The following algorithm is based on the ring-based election algorithms presented in [Tanenbaum 1995, Silberschatz and Galvin 1994]. In this algorithm it is assumed that all the processes in the system are organized in a logical ring. The ring is unidirectional means all messages related to the election algorithm are always passed only in one direction (clockwise/anticlockwise).

Every process in the system knows the .structure of the ring, so while circulating a message over the ring, if the successor of the sender process is down, the sender can skip over the successor, or the one after that, until an active member is found. The algorithm works as follows.

When a process (say P_i) sends a request message to the current coordinator and does not receive a reply within a fixed time period, it assumes that the coordinator has crashed. Therefore, it initiates an election by sending an election message to its successor (to the first successor that is currently active). This message contains the priority number of process P_i. On receiving the election message, the successor appends its own priority number to the message and passes it on to the next active member in the ring. This member appends its own priority number to the message and forwards it to its own successor.

In this manner, the election message circulates over the ring from one active process to another and eventually returns to process Pi. Process P_i recognizes the message as its own election message by seeing that in the list of priority numbers held within the message. The first priority number is its own priority number.

Note that when process P_i receives its own election message, the message contains the list of priority numbers of all processes that are currently active. Therefore, of the available processes in this list, it elects the process having the highest priority number as the new coordinator. It then circulates a **coordinator message** over the ring to inform all the other active processes **who the new coordinator is**. When the coordinator message comes back to process P_i after completing its one round along the ring, it is removed by process P_i. At this point all the active processes know who the current coordinator is.

When a process (say P_j) recovers after failure, it creates an **inquiry message** and sends it to its successor. The message contains the identity of process P_j. If the successor is not the current coordinator, it simply forwards the enquiry message to its own successor. Hence the **inquiry message** moves forward along the ring until it reaches the current coordinator. On receiving an inquiry message, the current coordinator sends a reply to process P_j informing that it is the current coordinator.

In this algorithm two or more processes may almost simultaneously discover that the coordinator has crashed and then each one may circulate an election message over the ring. Although this results in a little waste of network bandwidth, it does not cause any problem because every process that initiated an election will receive the same list of active processes, and all of them will choose the same process as the new coordinator.

4.2.3 Discussion of the Election Algorithms:

In the **bully algorithm**, in a system having total n processes, when the process having the lowest priority number detects the coordinator's failure and initiates an election, altogether n-2 elections are performed one after another for the initiated one. That is, all the processes, except the active process with the highest priority number and the coordinator process that has just failed, perform elections by sending messages to all processes with higher priority numbers. Hence, in the worst case, the bully algorithm requires $O(n^2)$ messages. However, when the process having the priority number just below the failed coordinator detects that the coordinator has failed, it immediately elects itself as the coordinator and sends n-2 coordinator messages. Hence, in the best case, the bully algorithm requires only n-2 messages.

On the other hand, in the **ring algorithm**, irrespective of which process detects the failure of the coordinator and initiates an election, an election always requires 2(n-l) messages (assuming that only the coordinator process has failed); n-l messages are needed for one round rotation of the election message, and another n-l messages are needed for one round rotation of the coordinator message.

Next let us consider the complexity involved in the recovery of a process. In the **bully algorithm**, a failed process must initiate an election on recovery. Therefore, once again depending on the priority number of the process that initiates the recovery action, the bully algorithm requires $O(n^2)$ messages in the worst case, and n-l messages in the best case.

On the other hand, in the **ring algorithm**, a failed process does not initiate an election on recovery but simply searches for the current coordinator. Hence, the ring algorithm requires only n/2 messages on an average for recovery action.

In conclusion, as compared to the bully algorithm, the ring algorithm is more efficient and easier to implement.

4.3 SUMMARY

Several distributed algorithms require that there should be a coordinator process in the entire system. Election algorithms are meant for electing a coordinator process from among the currently running processes. We discussed two election algorithms in this chapter i.e the bully algorithm and the ring algorithm.

4.4 REFERENCE

- Pradeep K. Sinha, Distributed Operating System: Concepts and Design, PHI Learning.

*****

UNIT III

5

DISTRIBUTED SHARED MEMORY

Unit Structure

5.1 INTRODUCTION

DSM stands Distributed Shared Memory. It is form of memory architecture where shared means the address space of memory shared. DSM is a mechanism of allowing user processes to access shared data without using inter-process communications. DSM refers to shared memory paradigm applied to loosely coupled distributed memory systems. Shared memory exists virtual only which is similar virtual memory so sometimes it's also called DSVM-Distributed Shared Virtual Memory. It provides a virtual address space shared among processes on loosely coupled processors.

It is basically an abstraction that integrates the local memory of different machine into a single logical entity shared by cooperating processes.

In DSM, every implement shared node has its own memory and provides memory read and write services and it provide consistency protocols. The distributed shared memory (DSM) implements shared memory model in distributed system but doesn't have physical shared memory. All the nodes share the virtual address space provided by the shared memory model.

Distributed Shared memory:

1) Message Passing Paradigm:

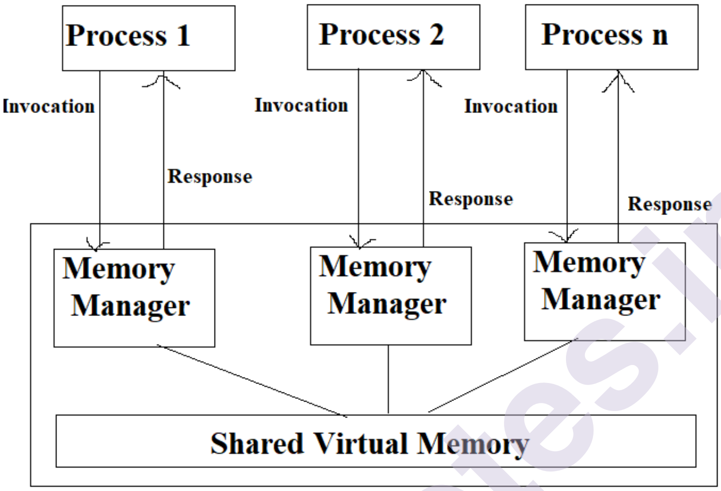
Send (recipient, data)

Receive(data)

2) Shared Memory Paradigm-

Data=Read(address)

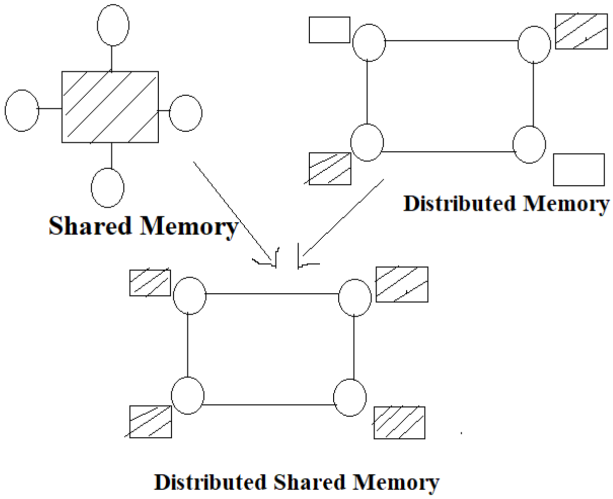
Write (address, data)



DSM can be achieved via software as well as hardware. DSM Provide service memory reads and write from any node's consistency protocols.

5.2 FUNDAMENTAL CONCEPTS OF DSM

Definition: Distributed Shared Memory is mechanism allowing end user processes to access shared data without using inter process communications. DSM system is to make inter process communication transparent to end user. By using DSM variables can shared directly and cost of communication is invisible.



Disturbed shared memory comparison with Message passing

Message passing	Distributed shared memory
Variables must be marshalled.	Variables are shared directly
Cost of communication is visible	Cost of communication is invisible
Processes are protected by having private address space	Process could cause error by altering data.
Process executes at same time.	Process executing may happen with non-overlapping lifetimes.

5.2.1 Architecture of DSM:

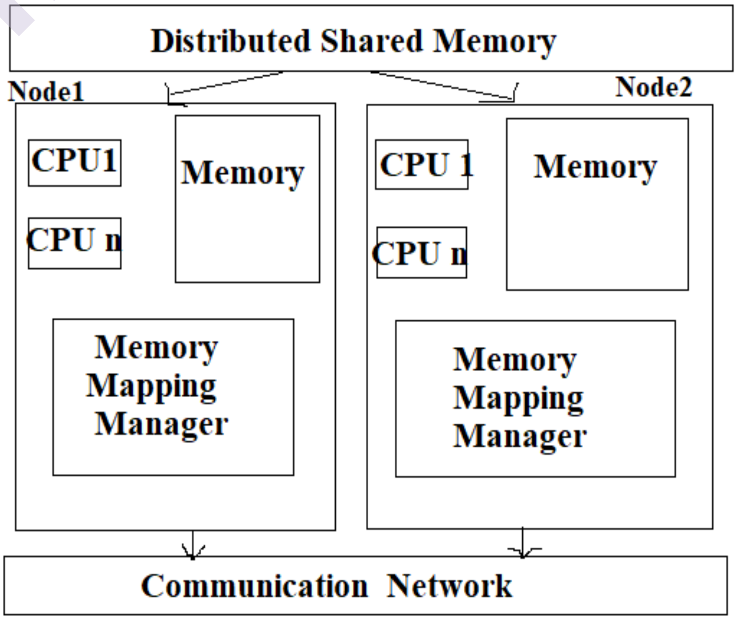
Each node of the system consists of one or more CPU's and memory unit. Nodes are connected by high-speed communication network. Simple message passing system for nodes to exchange information. Main memory of individual nodes is used to cache pieces of shared memory space. Main memory of individual nodes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

Memory Mapping Manger Unit:

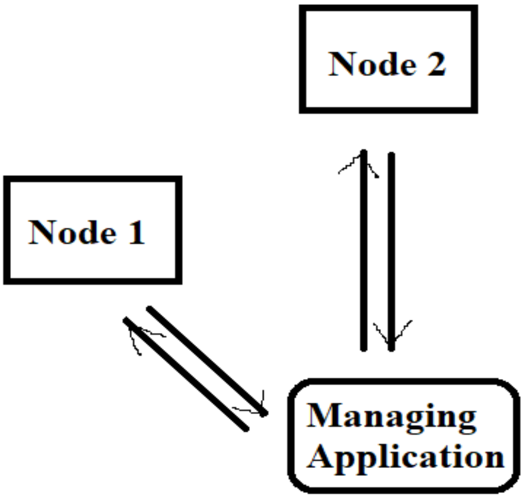
Memory mapping manager routine maps local memory to shared virtual memory. For mapping operation, the shared memory house is divided into blocks. DMA uses information caching to scale back network latency. Memory mapping manager of every node reads its native memory as enormous cache of the shared memory house for its native process.

Communication Network Unit:

Once method access information within shared address house mapping maps the shared memory address to the physical memory. Physical memory on every node holds pages of shared virtual address house native pages area in some node's memory.



In simple language we can say nodes can communicate through managing application. Refers the following diagram of distributed shared memory communication among nodes.



5.2.2 Advantages of DSM:

1) DSM system is scalable. Scales are good with many nodes.

2) It hides the message passing.

3) DSM can handle large and complex data bases without replication or sending them processes.

4) It provides large virtual memory space.

5) It provides portable as they use common DSM programming interface.

6) DSM can improve performance by speeding up data access.

7) DSM provides on demand data movement means data will eliminate the data exchange phase.

8) DSM is less expensive when compared to using a multiprocessor system.

9) No memory access bottleneck as no single bus.

5.2.3 Disadvantages of DSM:

1) It could cause a performance penalty.

2) It should provide for protection against simultaneous access to shared data such as lock.

3) Its performance of irregular problems could be difficult.

5.2.4 Types of Distributed Shared Memory (DSM)

1) On- Chip Memory

2) Bus-Based Multiprocessors

3) Ring-Based Multiprocessors

4) Switched Multiprocessors

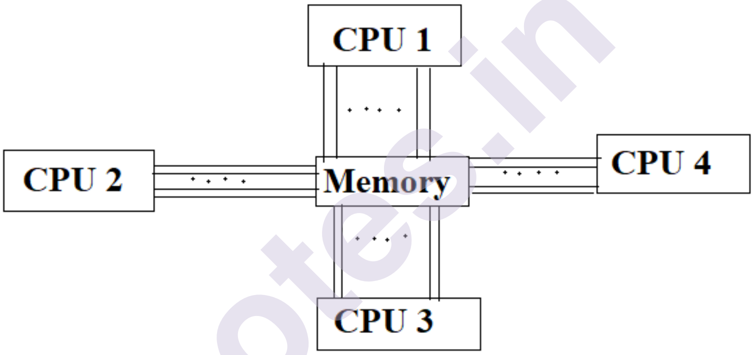
1) On- Chip Memory:

The data is present in the CPU portion of the chips.

Address lines are connected to memory.

On-chip memory DSM is complex and expensive.

On-chip Memory chips are widely used in appliances cars and even toys.



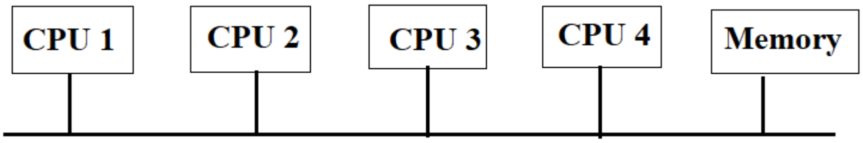
2) Bus-Based Multiprocessors:

A set of parallel wires called bus acts as a connection between CPU and memory.

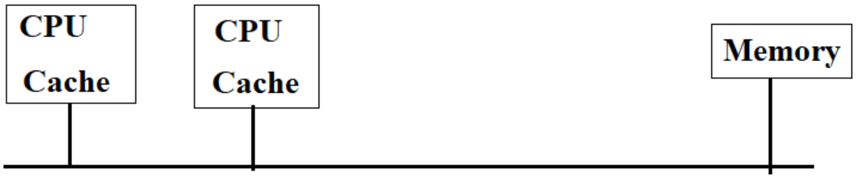
Network traffic is reduced by using caches with each CPU.

Some Algorithms are used to prevent two CPU trying to access same memory simultaneously.

As uses single bus makes it overloaded.



Bus based multiprocessor



Bus Based Multiprocessor with caching

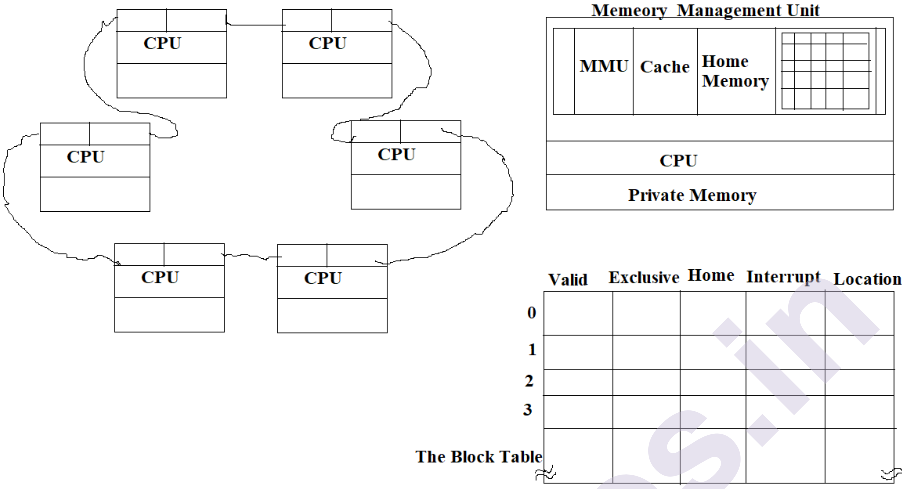
3) Ring-Based Multiprocessors:

A single address line is partitioned into a private area and shared area.

All nodes are connected via a token passing ring.

Shared area is divided into 32 bytes.

There is no global centralized memory present in ring-based multiprocessors.



Ring Baased Multiprocessor (Memnet)

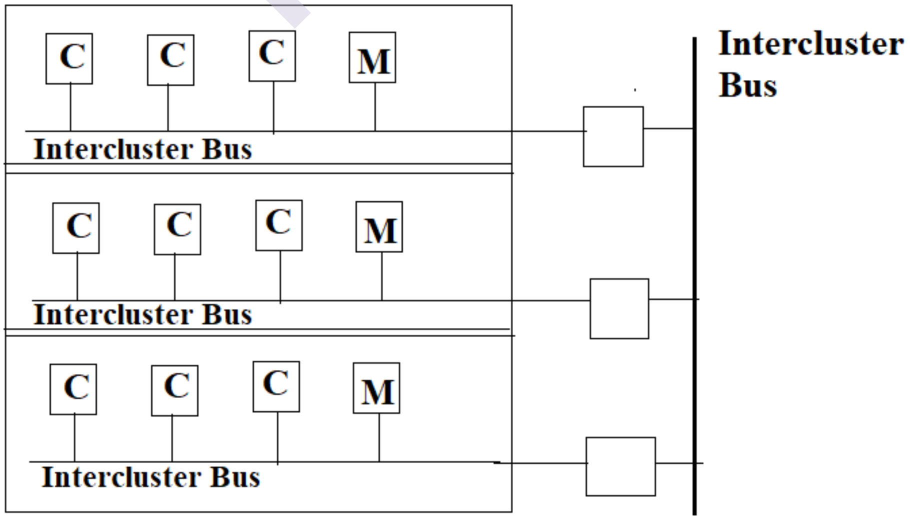
4) Switched Multiprocessors:

Two approaches can be taken to attack the problem of not enough bandwidth.

Reduce the amount of communication Ex. Caching

Increase the communication capacity Ex. Changing topology.

To build the system as a hierarchy build the system as multiple clusters.



5.2.5 Various hardware DSM systems:

Distributed shared memory (DSM) can be implemented in software or hardware.

Software implementation of distributed shared memory is easy, and which is used by software. Message passing on same cluster, so it is easy. When we apply software distributed shared memory there is no need to change hardware.

Software layer can be:

1) Page Based

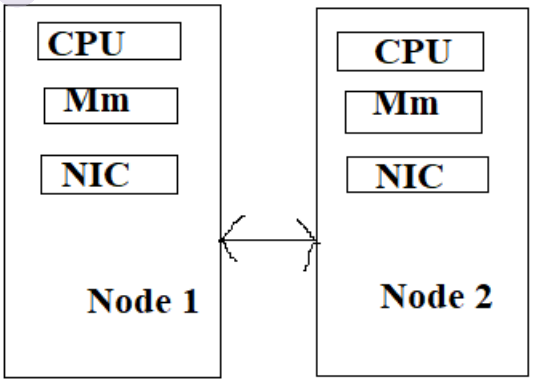
2) Shared Variable Based

3) Object Based

Whenever we talk about hardware it relies on interconnects physically.

- For hardware distributed shared memory network interfaces and cache coherence circuits are used.
- Special interfaces are used that supports shared memory operations.
- It gives higher performance.
- Hardware distributed shared memory is expensive.

Operating System Manges physical memory. Operating system running on node that is physical one which access physical memory which allows us to make connection between virtual to physical address of other nodes. Network interface card (NIC) used to translate remote message access to messages. NIC manages memory management, consistency, and atomics.



In this diagram,

CPU-Central Processing Unit

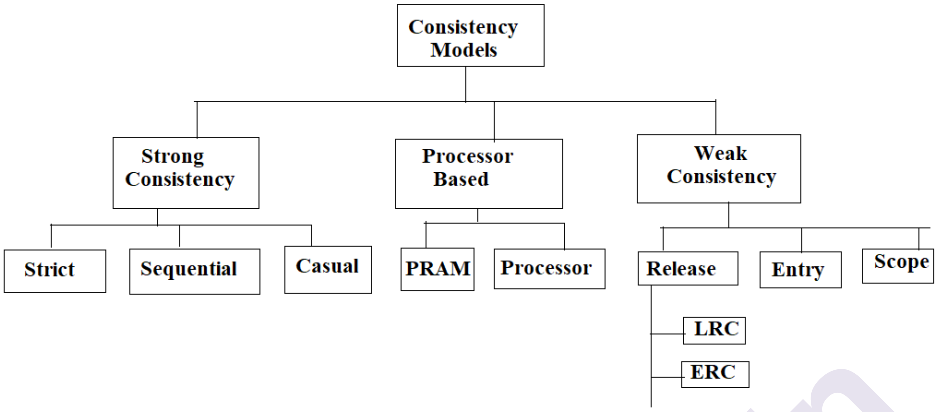
Mm-Memory Management

NIC-Network Interface Card

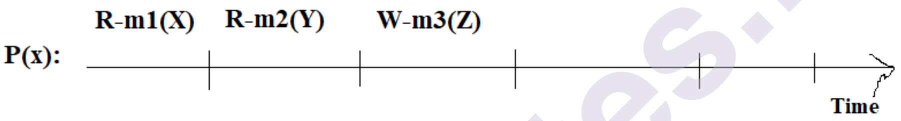
5.2.6 Consistency Models:

Consistency Model nothing but the degree of consistency that must be maintained for shared memory data. Memory (states) accordingly by access ordering and propagation or visibility of updated.

Classification of consistency model



Notations:



R-m1(X)= X was read from memory location m1

W-m1(Y)=Y was written to memory location m1

Consistency Model types:

1) Strict Consistency Model

2) Sequential Consistency Model

3) Causal Consistency Model

4) Weak Consistency Model

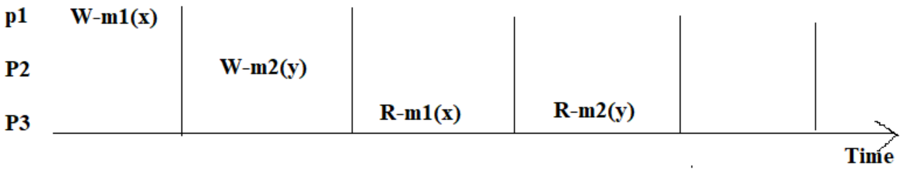
5) Pipelined Random Access Consistency Model (PRAM)

6) Processor Consistency Model

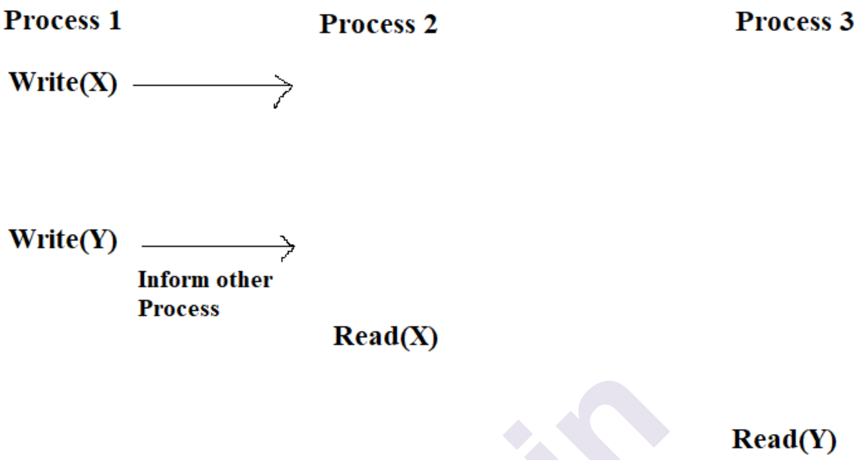
7) Release Consistency Model

1) Strict Consistency Model:

This type of model has strong type of memory integration with a strong need for consistency. The amount returned for the performance learned at the memory address remains the same as the value written for the most transaction at that address. For writes its visible to all the process.



Let’s us consider the transaction example for strict consistency model



2) Sequential Consistency Mode:

The shared memory system is said to support sequential compatibility model when all processes detect the same order. If three functions are read (r1), write (w1), read (r2) are done in the memory field for that program any orders (r1,w1,r2),(r1,r2,w1),(w1,r1,r2),(w1,r2,r1),(r2,r1,w1),(r2,w1,r1) these all processes are in same orders.

3) Causal Consistency Model:

All processes detect only those memory reference functions in their sequence that may be related memory reference functions may be identified by various process in different sequences. The function of the memory reference is said to be related to other memory reference function if one may be influenced by the other.

4) Weak Consistency Model:

If any process caused an error or any process in critical phase, there is no restriction to show the change in the memory caused by an activity in other process. The DSM system that supports a weak static model uses a special alternative called synchronization variable.

5) Pipelined Random Access Consistency Model (PRAM):

It provides a simplistic semantics that is weaker than the symmetry model described ensure that all writing process performed in one process are recognized by all other process in sequence. All single process writing tasks are on track.

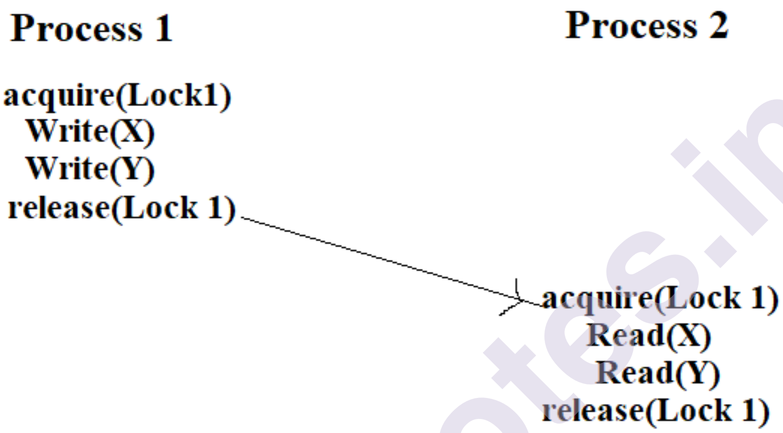
6) Processor Consistency Model:

This is like Pipelined Random Access Consistency Model (PRAM). Memory consistency means that in memory location all process agrees on the same sequence of all writing activities performed in same memory space. If two processes w1, w2 writing in same memory location with sequence w1, w2 or w2, w1.

7) Release Consistency Model:

To enhance weak consistency, model this model is used. acquire used to tell the system it is entering release used to tell the system just excited Results are release by process of another node.

Let's consider the transaction for release consistency model



5.2.7 Issues in designing and implementing DSM systems:

1) Granularity

2) Structure of shared memory

3) Memory Coherence and access synchronization

4) Data Location and access

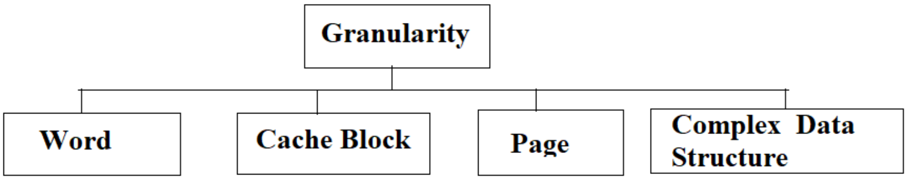
5) Replacement Strategy

6) Thrashing

7) Heterogeneity

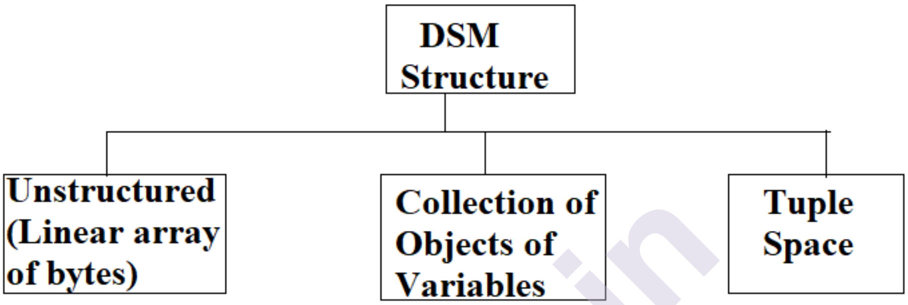
1) Granularity:

Granularity means block size of shared memory. When data sharing and transfer across the network that time block faults occurs may be word or page faults. A chunk of memory contains the word and issue is size of the chunk.



2) Structure of shared memory:

It is layout of shared data in the memory. This structure of shared memory depends on different application which is supported by Distributed Shared Memory (DSM).


DSM Structure
Unstructured (Linear array of bytes)
Collection of Objects of Variables
Tuple Space

3) Memory Coherence and access synchronization:

Distributed Shared Memory (DSM) system allows duplicate of shared data items, copied of shared data items available in main memories of number of nodes. To deal with this issue the consistency of a piece of shared data lying in the main memories of two or more nodes.

4) Data Location and access:

As shared memory required shared data, to shared data in a Distributed Shared Memory (DSM) to locate and retrieve the data accessed by a user process.

5) Replacement Strategy:

In this strategy the data block must be replaced by new data block. When the local memory of the node is full, a cache misses at that node from fetched of access of data block from remote node and replacement also.

6) Thrashing:

Data block of memory migrate between nodes on demands. Different technique used to less thrashing.

Application Controlled Lock

Algorithm to shared date used pattern a

b

page access by using

two variables with same page

Suppose we must access pages by using different variables we can use same pages. consider two variables respectively and b.

6) Heterogeneity:

It is applied on operating system, computer hardware, networks on system with implementation of developers. Heterogeneity provides environment like client-server .it treats as middleware which is set of services which interacts to end users. Basically, Heterogeneity works on

1) **Networks**: Internet protocols is used in network for communication purpose.
2) **Computer Hardware:** Internal representation of different processor.
3) **Programming Language:** Data structured are represented for data using different programming languages.
4) **Operating System:**To communicate different operating system used to send message.
5) **Implementation of different Developers:** Follows different standards for communication.

5.3 SUMMARY

DSM means Distributed Shared Memory which is not physical memory it is virtual memory. It is a form of memory architecture where shared means the address space of memory shared. DSM is a mechanism of allowing user processes to access shared data without using inter-process communications. DSM refers to the shared memory paradigm applied to loosely coupled distributed memory systems. We learned what is Distributed shared memory with its architecture that what is the role of memory mapping manager and communication network unit. What is the difference between message passing and distributed shared memory? There are different types of distributed shared memory likes On- Chip Memory, Bus-Based Multiprocessors, Ring-Based Multiprocessors and Switched Multiprocessors.

As Distributed shared memory can be implemented in software as well as in hardware. In software there are three layers Page Based, Shared Variable Base, Object Based. In Hardware CPU (Central Processing Unit), MU (Memory Unit) and NIC (Network Interface Card) required. We learned different Consistency Models of distributed shared memory.

We discussed different issues of distributed shared memory.

5.4 BIBLIOGRAPHY

1) https://www.cc.gatech.edu/classes/AY2010/cs4210_fall/papers/DSM_protic.pdf

2) https://www.geeksforgeeks.org/what-is-distributed-shared-memory-and-its-advantages/

3) https://vedveethi.co.in/eNote/DistSys/CS-702%20U-2.htm

4) https://en.wikipedia.org/wiki/Distributed_shared_memory

5) https://www.slideserve.com/dianne/distributed-computing

6) https://slideplayer.com/slide/6897641/

7) https://nptel.ac.in/courses/106102114/

8) Pradeep K. Sinha, Distributed Operating System: Concepts and Design, PHI Learning, ISBN No. 978-81-203-1380-4

9) Andrew S. Tanenbaum, Distributed Operating Systems, Pearson Education, ISBN No. 978-81-317-0147-8

5.6 REFERENCE FURTHER READING TOPICS

MemNet Architecture We learned in types of distributed shared memory in this one of the types is ring based multiprocessor that is MemNet. You can learn in details studies of Memnet with its architecture.

5.7 QUESTIONS

1) Explain Architecture of Distributed Shared Memory.

2) What is Distributed shared memory? Explain its architecture.

3) Explain various consistency model in details.

4) Explain Different types of Distributed Shared Memory.

5) What is the issue of design and implementation of distributed shared memory?

6) Explain advantages and disadvantages of distributed shared memory.

7) Explain Distributed shared memory with its architecture?

*****

UNIT IV

6

DISTRIBUTED SYSTEM MANAGEMENT

Unit Structure

6.1 INTRODUCTION TO DISTRIBUTED SYSTEM

A distributed system contains multiple nodes that are physically separate but linked together using the network. All the nodes in this system communicate with each other and handle processes in tandem. Each of these nodes contains a small part of the distributed operating system software.

6.1.1 Types of Distributed Systems:

The nodes in the distributed systems can be arranged in the form of client/server systems or peer to peer systems. Details about these are as follows −

Client/Server Systems: In client server systems, the client requests a resource and the server provides that resource. A server may serve multiple clients at the same time while a client is in contact with only one server. Both the client and server usually communicate via a computer network and so they are a part of distributed systems.

Peer to Peer Systems: The peer to peer systems contains nodes that are equal participants in data sharing. All the tasks are equally divided between all the nodes. The nodes interact with each other as required as share resources. This is done with the help of a network.

6.1.2 Advantages of Distributed Systems:

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

6.1.3 Disadvantages of Distributed Systems:

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

6.1.4 How does a distributed system work?:

Distributed systems have evolved over time, but today's most common implementations are largely designed to operate via the internet and, more specifically, the cloud. A distributed system begins with a task, such as rendering a video to create a finished product ready for release. The web application, or distributed applications, managing this task — like a video editor on a client computer — splits the job into pieces. In this simple example, the algorithm that gives one frame of the video to each of a dozen different computers (or nodes) to complete the rendering. Once the frame is complete, the managing application gives the node a new frame to work on. This process continues until the video is finished and all the pieces are put back together. A system like this doesn't have to stop at just 12 nodes — the job may be distributed among hundreds or even thousands of nodes, turning a task that might have taken days for a single computer to complete into one that is finished in a matter of minutes.

There are many models and architectures of distributed systems in use today. Client-server systems, the most traditional and simple type of distributed system, involve a multitude of networked computers that interact with a central server for data storage, processing or other common goal. Cell phone networks are an advanced type of distributed system that share workloads among handsets, switching systems and internet-based devices. Peer-to-peer networks, in which workloads are distributed among hundreds or thousands of computers all running the same software, are another example of a distributed system architecture. The most common forms of distributed systems in the enterprise today are those that operate over the web, handing off workloads to dozens of cloud-based virtual server instances that are created as needed, then terminated when the task is complete.

6.2 INTRODUCTION TO RESOURCE MANAGEMENT

Distributed systems contain a set of resources interconnected by a network. Processes are migrated to fulfill their resource requirements. The Resource manager is responsible to control the assignment of resources to processes. The resources can be logical (shared file) or physical (CPU) resource. Scheduling is the way in which processors are assigned to run on the available resources. In a distributed computing system, the scheduling of various modules on particular processing nodes may be preceded by appropriate allocation of modules of the different tasks to various processing nodes and then only the appropriate execution characteristic can be obtained. The task allocation becomes the important most and major activity in the task scheduling within the operating system of a DCS.

6.2.1 Architecture of Resource Management In Distributed Environment:

In this architecture all the processes involved in a task like resource management play similar roles interacting co-operatively as peers without any distinction between client and server processes or the computers they run . The aim of peer to peer architecture is to exploit the resources in a large number of participating computers for the fulfilment of a given task. In this architecture applications are composed of large numbers of peer processes running on separate computers and the pattern of communication between them depends entirely on application requirements. A large number of data objects like files are shared. Each object or file is replicated in several computers to further distribute the load and to provide compensation in the event of disconnection or network failure.

The user interface for distributed resource environment needs several kinds of transparency for resources which are distributed on many systems connected to the network of the distributed system. Access transparency and location transparency are general concept for distributed resources. The advantage of the distributed environment is the replication of the resources in any site. Besides the two transparencies, there is semantics transparency in which a user can access resource by semantic name rather than by physical name for the resource.

Besides the distributed architecture of Resource Management, Scheduling is done to make resources available to an application whenever they are needed. Processes need to have resources assigned to them according to priority. A resource Scheduler determines the priority of processes based on certain criteria. Scheduling methods need to be applied to all resources that affect the performance of an application.

There are two kinds of scheduling. Fair Scheduling and Real-Time Scheduling. The scheduling also depends upon the types of resources. A resource is any object that a process can request and wait for. A resource can consist of any number of identical units and a process can request any number of units of a resource. The types of resources can be given as:

- **Reusable Resources:** A reusable resource does not vanish as a result of its use but can be used over again and again. In a system, a reusable resource has a fixed number of units and these units can neither be created nor destroyed.

- **Consumable Resources**: A consumable resource vanishes as a result of its use. When a unit of consumable resource is allocated to a process it as consumed and ceases to exist. There is no fixed number of units of a consumable resource in a system since the resource units can be created and consumed.

In a distributed resource management system, resources can be stored at any machine and the computation can be performed at any machine. When a machine needs to access a file stored on remote machine, the remote machine performs the necessary file access operations and returns the data if a read operation is performed. The two most important services present in a distributed resource management system are the name server and cache manager. A name server is a process that maps names specified by clients to stored objects such as files and directories. A cache manager is a process that implements file caching. In file caching, a copy of data stored at file server is brought to the client's machine when referenced by the client. Subsequent accesses to data are performed locally at the client, there by reducing the access delays due to network latency. Cache managers can be present at both clients and file servers. Cache managers at the server caches files in the main memory to reduce the delays due to disk latency.

Typcally, accessing remote resources is more expensive than accessing local resources because of the communication delay that occur in the network and the CPU overhead incurred to process communication protocols. The impetus behind the development of Distributed Computing was the availability of powerful processors at low cost and also advances in communication technology. The availability of powerful yet cheap processors led to the development of powerful workstations that satisfy a single user's needs. These powerful standalone workstations satisfy user need by providing such things as bit-mapped displays and visual interfaces, which traditional time-sharing systems do not support. When a group of people work together, there is generally a need to communicate with each other, to share data, and to share expensive resources such as printers, disk drives etc. This requires interconnecting computers and resources.

Technically, a completely pure peer-topeer application must implement only peering protocols that do not recognize the concepts of server" and "client". Such pure peer applications are rare. Most networks and application described as peer-to-peer actually contain or rely on some non-peer.

6.3 SCHEDULING ALGORITHMS

Scheduling is a decision-making process that is used on a regular basis in many manufacturing and services industries. It deals with the allocation of resources to tasks over given time periods and its goal is to optimize one or more objectives

6.3.1 Features of Scheduling algorithms:

General purpose:

- A scheduling approach should make few assumptions about and have few restrictions to the types of applications that can be executed.
- Interactive jobs, distributed and parallel applications, as wellas non-interactive batch jobs, should all be supported with good performance.
- This property is a straightforward one, but to some extent difficult to achieve.
- Because different kinds of jobs have different attributes, their requirements to the scheduler may contradict.
- To achieve the general purpose, a tradeoff may have to be made.

Efficiency:

- It has two meanings: one is that it should improve the performance of scheduled jobs as much as possible; the other is that the scheduling should incur reasonably low overhead so that it would not counter attack the benefits.

Fairness:

- Sharing resources among users raises new challenges in guaranteeing that each user obtains his/her fair share when demand is heavy is fairness.
- In a distributed system, this problem could be exacerbated such that one user consumes the entire system.
- There are many mature strategies to achieve fairness on a single node.

Dynamic:

- The algorithms employed to decide where to process a task should respond to load changes, and exploit the full extent of the resources available.

Transparency:

- The behavior and result of a task's execution should not be affected by the host(s) on which it executes.
- In particular, there should be no difference between local and remote execution.
- No user effort should be required in deciding where to execute a task or in initiating remote execution; a user should not even be aware of remote processing.
- Further, the applications should not be changed greatly.
- It is undesirable to have to modify the application programs in order to execute them in the system.

Scalability:

- A scheduling algorithm should scale well as the number of nodes increases.
- An algorithm that makes scheduling decisions by first inquiring the workload from all the nodes and then selecting the most lightly loaded node has poor scalability.
- This will work fine only when there are few nodes in the system.
- This is because the inquirer receives a flood of replies almost simultaneously, and the time required to process the reply messages

for making a node selection is too long as the number of nodes (N) increase.

- Also the network traffic quickly consumes network bandwidth.
- A simple approach is to probe only m of N nodes for selecting a node.

Fault tolerance:

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

Quick decision making capability:

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

Balanced system performance and scheduling overhead:

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

Stability:

- Fruitless migration of processes, known as processor thrashing, must be prevented.

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

6.3.2 Local Scheduling

In a distributed system, local scheduling means how an individual workstation should schedule those processes assigned to it in order to maximize the overall performance. It seems that local scheduling is the same as the scheduling approach on a stand-alone workstation. However, they are different in many aspects. In a distributed system, the local scheduler may need global information from other workstations to achieve the optimal overall performance of the entire system. For example, in the extended stride scheduling of clusters, the local schedulers need global ticket information in order to achieve fairness across all the processes in the system.

In recent years, there have been many scheduling techniques developed in different models. Here, we introduce two of them: one is a proportional-sharing scheduling approach, in which the resource consumption rights of each active process are proportional to the relative shares that it is allocated. The other is predictive scheduling, which is adaptive to the CPU load and resource distribution of the distributed system.

The traditional priority-based schedulers are difficult to understand and give more processing time to users with many jobs, which leads to unfairness among users. Numerous researches have been trying to find a scheduler that is easy to implement and can solve the problem of allocating resources to users fairly over time. In this environment, proportional-share scheduling was brought out to effectively solve this problem. With proportional-share scheduling, the resource consumption rights of each active process are proportional to the relative shares that it is allocated.

6.3.3 Stride Scheduling:

As a kind of proportional-share scheduling strategies, stride scheduling allocates resources to competing users in proportion to the number of tickets they hold. Each user has a time interval, or stride, inversely proportional to his/her ticket allocation, which determines how frequently it is used. A pass is associated with each user. The user with a minimum pass is scheduled at each interval; a pass is then incremented by the job's stride.

6.3.3.1 Extension to Stride Scheduling:

The original stride scheduling only deals with CPU-bound jobs. If the proportional-share schedulers are to handle the interactive and I/O intensive job workloads, they must be extended to improve the responsive time and I/O throughput, while not penalizing competing users. Here we discuss two extensions to stride scheduling that give credits to jobs not competing for resources. In this way, jobs are given incentive to relinquish the processor when not in use and will receive their share of resources over a longer time-interval. Thus, because interactive jobs are scheduled more frequently when they awaken, they can receive better response time. The first approach is loan & borrow, and the second approach is system credit. Both approaches are built upon exhaustible tickets, which are simple tickets with expiration time.

Loan & Borrow: In this approach, exhausted tickets are traded among competing clients. When a user temporarily exits the system, other users can borrow these otherwise inactive tickets. The borrowed tickets expire when the user rejoins the system. When the sleeping user wakes up, it stops loaning tickets and is paid back in exhaustible tickets by the borrowing users. In general, the lifetime of the exhaustible tickets is equal to the length the original tickets were borrowed. This policy can keep the total number of tickets in the system constant over time; thus, users can accurately determine the amount of resources they receive. However, it also introduces an excessive amount of computation into the scheduler on every sleep and wake-up event, which we don’t expect.

System Credit: This second approach is an approximation of the first one. With system credits, clients are given exhaustible tickets from the system when they awaken. The idea behind this policy is that after a client sleeps and awakens, the scheduler calculates the number of exhaustible tickets for the clients to receive its proportional share over some longer interval. The system credit policy is easy to implement and does not add significant overhead to the scheduler on sleep and wakeup events.

Proportional-share of resources can be allocated to clients running sequential jobs in a cluster. In the cluster, users are guaranteed a proportional-share of resources if each local stride-scheduler is aware of the number of tickets issued in its currency across the cluster and if the total number of base tickets allocated on each workstation is balanced. The solution for the first assumption is simple: each local scheduler is informed of the number of tickets issued in each currency, and then correctly calculates the base funding of each local job. The solution for distributing tickets to the stride-schedulers is to run a user-level tickets-sever on each of the nodes in the cluster. Each stride-scheduler periodically contacts the local ticket server to update and determine the value of currencies.

Further, for parallel jobs in a distributed cluster, proportional-share resources can be provided through a combination of stride-scheduling and implicit coscheduling. Preliminary simulations of implicit coscheduling for a range of a communication patterns and computation granularity indicate that the stride-scheduler with system credit performs similarly to the Solaris time-sharing scheduler which is used in the Berkeley NOW environment

6.3.4 Predictive Scheduling:

Predictive scheduling differs from other scheduling approaches in that it provides intelligence, adaptivity and proactivity so that the system implementing predictive scheduling can adapt to new architectures and/or algorithms and/or environmental changes automatically. Predictive scheduling can learn new architectures, algorithms and methods that are embedded into the system. They provide some guarantees of service.

Furthermore, they are able to anticipate significant changes to its environment and avoid those changes to become the system performance bottleneck.

Predictive scheduling can be roughly decomposed into three components: H-cell, S-cell and allocator. The H-cell receives information of hardware resource changes such as disk traffic, CPU usage, memory availability, etc., and provides near-real-time control. Meanwhile, S-cell provides long-term control of computational demands--such as what the deadline of a task is and what its real-time requirement is--by interrogating the parallel program code. H-cell and S-cell respectively collect information about computational supply and computational demand, and provide to the allocator the raw data or some intelligent recommendations. The allocator reconciles the recommendations sent by the H-cells and S-cells and schedules jobs according to their deadline, while guaranteeing constraints and enforcing the deadline.

In the allocator, the previous inputs, in the form of a vector of performance information (such as memory, CPU, disk usage etc.), are aggregated into sets. Each set corresponds to a scheduling decision. The allocator re-organizes the sets dynamically to keep a limited memory demand by splitting or merging sets. If a new input matches one of the pattern categories, a decision will be made due to the corresponding decision of that pattern set, otherwise a new pattern category is built to associate this new input pattern with corresponding scheduling decision.

Most of the scheduling policies are used either when a process blocks or at the end of a time slice, which may reduce the performance because there can be a considerable lapse of time before scheduling is done. Predictive scheduling solves this problem by predicting when a scheduling decision is necessary, or predicting the parameters needed by the scheduling decision when not known in advance. Based on the collected static information (machine type, CPU power, etc.) and dynamic information (memory free space, CPU load, etc.), predictive scheduling tries to make an educated guess about the future behavior, such as CPU idle time slot, which can be used to make scheduling decisions in advance. Predicting the future performance based on past information is a common strategy, and it can achieve a satisfactory performance in practical work.

Predictive scheduling is very effective in performance and reliability enhancement, even with the simplest methods, but at the cost of design complexity and management overhead. Furthermore, it is observed that the more complicated method is used, the more design complexity and management overhead, and the less performance and reliability enhancement.

6.3.5 Coscheduling:

In 1982, Outsterhout introduced the idea of coscheduling , which schedules the interacting activities (i. e., processes) in a job so that all the activities execute simultaneously on distinct workstations. It can produce benefits in both system and individual job efficiency. Without coordinated scheduling, the processor thrashing may lead to high communication latencies and consequently degraded overall performance. With systems connected by highperformance networks that already achieve latencies within tens microseconds, the success of coscheduling becomes a more important factor in deciding the performance.

6.3.6 Gang Scheduling:

Gang scheduling is a typical coscheduling approach, which has already been introduced for a long time but still plays a fundamental role. Moreover, there are still many research projects in progress to improve gang scheduling. The approach identifies a job as a gang and its components as gang members. Further, each job is assigned to a class that has the minimum number of workstations that meet the requirement of its gang members based on a oneprocess- one-workstation policy. The class has a local scheduler, which can have its own scheduling policy. When a job is scheduled, each of its gang members is allocated to a distinct workstation, and thus, the job executes in parallel. When a time-slice finishes, all running gang members are preempted simultaneously, and all processes from a second job are scheduled for the next time-slice. When a job is rescheduled, effort is also made to run the same processes on the same processors.

The strategy bypasses the busy-waiting problem by scheduling all processes at the same time. According to the experience, it works well for parallel jobs that have a lot of inter-process communications. However, it also has several disadvantages. First, it is a centralized scheduling strategy, with a single scheduler making decisions for all jobs and all workstations. This centralized nature can easily become the bottleneck when the load is heavy. Second, although this scheduler can achieve high system efficiency on regular parallel applications, it has difficulty in selecting alternate jobs run when processes block, requiring simultaneous multi-context switches across the nodes. Third, to achieve good performance requires long scheduling quanta, which can interfere with interactive response, making them a less attractive choice for use in a distributed system. These limitations motivate the integrated approaches.

The requirement of centralized control and the poor timesharing response of previous scheduling approaches have motivated new, integrated coscheduling approaches. Such approaches extend local timesharing schedulers, preserving their interactive response and autonomy. Further, such approaches do not need explicitly identified sets of processes to be coscheduled, but rather integrate the detection of a coscheduling requirement with actions to produce effective coscheduling.

6.3.7 Implicit Coscheduling:

Implicit coscheduling is a distributed algorithm for time-sharing communicating processes in a cluster of workstations. By observing and reacting to implicit information, local schedulers in the system make independent decisions that dynamically coordinate the scheduling of communicating processes. The principal mechanism involved is two-phase spin-blocking: a process waiting for a message response spins for some amount of time, and then relinquishes the processor if the response does not arrive.

The spin time before a process relinquishes the processor at each communication event consists of three components. First, a process should spin for the baseline time for the communication operation to complete; this component keeps coordinated jobs in synchrony. Second, the process should increase the spin time according to a local cost-benefit analysis of spinning versus blocking. Third, the pairwise cost-benefit, i.e., the process, should spin longer when receiving messages from other processes, thus considering the impact of this process on others in the parallel job.

- The baseline time comprises the round-trip time of the network, the overhead of sending and receiving messages, and the time to awake the destination process when the request arrives.
- The local cost-benefit is the point at which the expected benefit of relinquishing the processor exceeds the cost of being scheduled again. For example, if the destination process will be scheduled later, it may be beneficial to spin longer and avoid the cost of losing coordination and being rescheduled later. On the other hand, when a large load imbalance exists across processes in the parallel job, it may be wasteful to spin for the entire load-imbalance even when all the processes are coscheduled.
- The pairwise spin-time only occurs when other processes are sending to the currently spinning process, and is therefore conditional. Consider a pair of processes: the receiver who is performing a two-phase spin-block while waiting for a communication operation to complete, and a sender who is sending a request to the receiver. When waiting for a remote operation, the process spins for the base and local amount, while recording the number of incoming messages. If the average interval between requests is sufficiently small, the process assumes that it will remain beneficial in the future to be scheduled and continues to spins for an additional spin time. The process continues conditionally spinning for intervals of spin time until no messages are received in an interval.

6.3.8 Dynamic Coscheduling:

Dynamic coscheduling makes scheduling decisions driven directly by the message arrivals. When an arriving message is directed to a process that isn't running, a schedule decision is made. The idea derives from the observation that only those communicating processes need to be coscheduled. Therefore, it doesn't require explicit identification to specify the processes need coscheduling.

The implementation consists three parts:

Monitoring Communication/Thread Activity:

A firmware, which is on the network interface card, monitors the thread activities by periodically reading the host's kernel memory. If the incoming message is sent to the process currently running, the scheduler should do nothing.

Causing Scheduling Decisions:

If a message received is not sent to the process currently running, an interrupt will be produced and invoke the interrupt routine. When the routine finds that it would be fair to preempt the process currently running, the process receiving the message has its priority raised to the maximum allowable priority for user mode timesharing processes, and is placed at the front of the dispatcher queue. Flags are set to cause a scheduling decision based on the new priorities. This will cause the process receiving the message to be scheduled unless the process currently running has a higher priority than the maximum allowable priority for user mode.

Making a Decision Whether to Preempt:

In dynamic coscheduling, the process receiving the message is scheduled only if doing so would not cause unfair CPU allocation. The fairness is implemented by limiting the frequency of priority boosts that therefore limits the frequency of preemption. In jobs with fine-grained communication, the sender and receiver are scheduled together and run until one of them blocks or is preempted. Larger collections of communicating processes are coscheduled by transitivity. The experiments taken in HPVM project indicate that dynamic coscheduling can provide good performance for a parallel process running on a cluster of workstations in competition with serial processes. Performance was able to close to ideal: CPU times were nearly the same as for batch processing, and reduced job response times by up to 20% over implicit scheduling while maintaining near-perfect fairness. Further, it claims that dynamic-coscheduling-like approaches can be used to implement coordinated resource management in a much broader range of cases, although most of which are still to be explored.

6.4 TASK ASSIGNMENT APPROACH

Each process is viewed as a collection of tasks. These tasks are scheduled to suitable processor to improve performance. This is not a widely used approach because:

- It requires characteristics of all the processes to be known in advance.
- This approach does not take into consideration the dynamically changing state of the system.

In this approach, a process is considered to be composed of multiple tasks and the goal is to find an optimal assignment policy for the tasks of an individual process. The following are typical assumptions for the task assignment approach:

- Minimize IPC cost (this problem can be modeled using network flow model)
- Efficient resource utilization
- Quick turnaround time
- A high degree of parallelism

A Distributed System is a Network of Machines that can exchange information with each other through Message-passing. It can be very useful as it helps in resource sharing. In this article, we will see the concept of the Task Assignment Approach in Distributed systems.

6.4.1 Resource Management:

One of the functions of system management in distributed systems is Resource Management. When a user requests the execution of the process, the resource manager performs the allocation of resources to the process submitted by the user for execution. In addition, the resource manager routes process to appropriate nodes (processors) based on assignments.

Multiple resources are available in the distributed system so there is a need for system transparency for the user. There can be a logical or a physical resource in the system. For example, data files in sharing mode, Central Processing Unit (CPU), etc.

As the name implies, the task assignment approach is based on the division of the process into multiple tasks. These tasks are assigned to appropriate processors to improve performance and efficiency. This approach has a major setback in that it needs prior knowledge about the features of all the participating processes. Furthermore, it does not take into account the dynamically changing state of the system. This approach's major objective is to allocate tasks of a single process in the best possible manner as it is based on the division of tasks in a system. For that, there is a need to identify the optimal policy for its implementation.

6.4.2 Working of Task Assignment Approach:

In the working of the Task Assignment Approach, the following are the assumptions:

- The division of an individual process into tasks.
- Each task's computing requirements and the performance in terms of the speed of each processor are known.
- The cost incurred in the processing of each task performed on every node of the system is known.

- The IPC (Inter-Process Communication) cost is known for every pair of tasks performed between nodes.
- Other limitations are also familiar, such as job resource requirements and available resources at each node, task priority connections, and so on.

6.4.3 Goals of Task Assignment Algorithms:

- Reducing Inter-Process Communication (IPC) Cost
- Quick Turnaround Time or Response Time for the whole process
- A high degree of Parallelism
- Utilization of System Resources in an effective manner

The above-mentioned goals time and again conflict. To exemplify, let us consider the goal-1 using which all the tasks of a process need to be allocated to a single node for reducing the Inter-Process Communication (IPC) Cost. If we consider goal-4 which is based on the efficient utilization of system resources that implies all the tasks of a process to be divided and processed by appropriate nodes in a system.

Note: The possible number of assignments of tasks to nodes:

For m tasks and n nodes= m x n

But in practice, the possible number of assignments of tasks to nodes < m x n because of the constraint for allocation of tasks to the appropriate nodes in a system due to their particular requirements like memory space, etc.

6.4.4 Need for Task Assignment in a Distributed System:

The need for task management in distributed systems was raised for achieving the set performance goals. For that optimal assignments should be carried out concerning cost and time functions such as task assignment to minimize the total execution and communication costs, completion task time, total cost of 3 (execution, communication, and interference), total execution and communication costs with the limit imposed on the number of tasks assigned to each processor, and a weighted product of cost functions of total execution and communication costs and completion task time. All these factors are countable in task allocation and turn, resulting in the best outcome of the system.

6.4.5 Example of Task Assignment Approach:

Let us suppose, there are two nodes namely n1 and n2, and six tasks namely t1, t2, t3, t4, t5, and t6. The two task assignment parameters are:

- **execution cost: $\mathbf{x_{ab}}$** refers to the cost of executing a task an on node b.

- **inter-task communication cost: c_{ij}** refers to inter-task communication cost between tasks i and j.

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

Execution Cost

	Nodes	
Tasks	**n1**	**n2**
t1	5	10
t2	2	infinity
t3	4	4
t4	6	3
t5	5	2
t6	infinity	4

Note: The execution of the task (t2) on the node (n2) and the execution of the task (t6) on the node (n1) is not possible as it can be seen from the above table of Execution costs that resources are not available.

Case1: Serial Assignment

Task	**Node**
t1	n1
t2	n1
t3	n1
t4	n2
t5	n2
t6	n2

Cost of Execution in Serial Assignment:

$t_{11} + t_{21} + t_{31} + t_{42} + t_{52} + t_{62} = 5 + 2+ 4 + 3 + 2 + 4$

$= 20$ (Refer Execution Cost table)

Cost of Communication in Serial Assignment:

$= c_{14} + c_{15} + c_{16} + c_{24} + c_{25} + c_{26} + c_{34} + c_{35} + c_{36}$

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

$= 38$ (Refer Inter-task Communication Cost table)

Hence, Total Cost in Serial Assignment

$= 20 + 38$

$= 58$

Case2: Optimal Assignment

Task	**Node**
t1	n1
t2	n1
t3	n1
t4	n1
t5	n1
t6	n2

Cost of Execution in Optimal Assignment:

= $t_{11} + t_{21} + t_{31} + t_{41} + t_{51} + t_{62}$

= 5 + 2+ 4 + 6 + 5 + 4

= 26 (Refer Execution Cost table)

Cost of Communication in Optimal Assignment:

= c16 + c26 + c36 + c46 + c56

= 12 + 0+ 0 + 0 + 0

= 12 (Refer Inter-task Communication Cost table)

Hence, Total Cost in Optimal Assignment

= 26 + 12

= 38

6.5 Load balancing approach

6.5.1 Classification of Load Balancing Algorithms

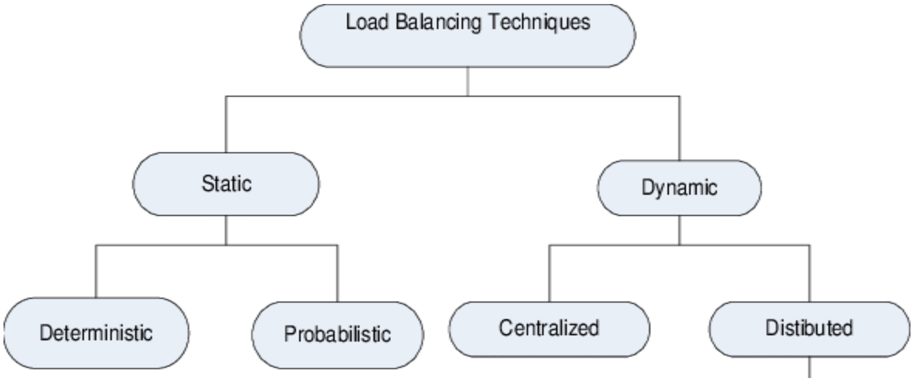


Fig 5.11: Classification of Load Balancing Algorithms

Static versus Dynamic:

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

Deterministic versus Probabilistic:

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

Centralized versus Distributed:

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

Cooperative versus Non-cooperative:

Cooperative Algorithms	Non-cooperative Algorithms
In Co-operative algorithms distributed entities cooperate with each other.	In Non-cooperative algorithms entities act as autonomous ones and make scheduling decisions independently from other entities.
Cooperative algorithms are more complex and involve larger overhead	They are simpler.
Stability of Cooperative algorithms are better.	Stability is comparatively poor.

6.5.2 Issues in Load Balancing Algorithms:

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

6.6 LOAD-SHARING APPROACH

The following problems in load balancing approach led to load sharing approach:

- Load balancing technique with attempting equalizing the workload on all the nodes is not an appropriate object since big overhead is generated by gathering exact state information.
- Load balancing is not achievable since number of processes in a node is always fluctuating and temporal unbalance among the nodes exists every moment

6.6.1 Basic idea:

It is necessary and sufficient to prevent nodes from being idle while some other nodes have more than two processes.

- Load-sharing is much simpler than load-balancing since it only attempts to ensure that no node is idle when heavily node exists.
- Priority assignment policy and migration limiting policy are the same as that for the load-balancing algorithms.

6.6.2 Load Estimation Policies:

Since load-sharing algorithms simply attempt to avoid idle nodes, it is sufficient to know whether a node is busy or idle.

- Thus these algorithms normally employ the simplest load estimation policy of counting the total number of processes.
- In modern systems where permanent existence of several processes on an idle node is possible, algorithms measure CPU utilization to estimate the load of a node

6.6.3 Process Transfer Policies:

- The load sharing algorithms normally use all-or-nothing strategy.
- This strategy uses the threshold value of all the nodes fixed to 1.
- Nodes become receiver node when it has no process, and become sender node when it has more than 1 process.

To avoid processing power on nodes having zero process load-sharing algorithms uses a threshold value of 2 instead of 1. When CPU utilization is used as the load estimation policy, the double-threshold policy should be used as the process transfer policy

Policies:

The location policy decides whether the sender node or the receiver node of the process takes the initiative to search for suitable node in the system, and this policy can one of the following:

Sender-initiated location policy: Sender node decides where to send the process. Heavily loaded nodes search for lightly loaded nodes

Receiver-initiated location policy: Receiver node decides from where to get the process. Lightly loaded nodes search for heavily loaded nodes

initiated location policy: Node becomes overloaded, it either broadcasts or randomly probes the other nodes one by one to find a node that is able to receive remote processes. When broadcasting, suitable node is known as soon as reply arrives

initiated location policy: Nodes becomes underloaded, it either broadcast or randomly probes the other nodes one by one to indicate its willingness to receive remote processes. Receiver-initiated policy require preemptive process migration facility since scheduling decisions are usually made at process departure epochs

- Both policies gives substantial performance advantages over the situation in which no load-sharing is attempted.
- Sender-initiated policy is preferable at light to moderate system loads.
- Receiver-initiated policy is preferable at high system loads.
- Sender-initiated policy provide better performance for the case when process transfer cost significantly more at receiver-initiated than at sender-initiated policy due to the pre-emptive transfer of processes.

information exchange policies:

In load-sharing algorithms it is not necessary for the nodes to periodically exchange state information, but needs to know the state of other nodes when it is either underloaded or overloaded. The following are the two approaches followed when there is state change:

Broadcast when state changes:

- In sender-initiated/receiver-initiated location policy a node broadcasts State Information Request when it becomes overloaded/ underloaded.
- It is called broadcast-when-idle policy when receiver-initiated policy is used with fixed threshold value of 1

Poll when state changes:

- In large networks polling mechanism is used.
- Polling mechanism randomly asks different nodes for state information until find an appropriate one or probe limit is reached.
- It is called poll-when-idle policy when receiver-initiated policy is used with fixed threshold value value of 1.

6.6.4 Differences between Load Balancing and Load Sharing:

	Load Balancing	Load Sharing
1.	Load balancing equally distributes network traffic or load across different channels and can be achieved using both static and dynamic load balancing techniques.	Load sharing delivers a portion of the traffic or load to one connection in the network while the remainder is routed through other channels.
2.	Focuses on the notion of traffic dispersion across connections.	Works with the notion of traffic splitting across connections.
3.	The creation of Ratios, Least connections, Fastest, Round robin, and observed approaches are used in load balancing.	Load Sharing is based on the notion of sharing traffic or network load among connections based on destination IP or MAC address selections.
4.	It is Uni-Directional.	It is Uni-Directional

5.	No instance is load sharing.	All instances are load sharing.
6.	Accurate Load Balancing is not an easy task.	Load sharing is easy compared with load balancing.

6.7 SUMMARY

A **distributed file system (DFS)** is a network file system wherein the file system is distributed across multiple servers. DFS enables location transparency and file directory replication as well as tolerance to faults. Some implementations may also cache recently accessed disk blocks for improved performance. Though distribution of file content increases performance considerably, efficient management of metadata is crucial for overall file system performance. It has been shown that 75% of all file system calls access file metadata [15] and distributing metadata load is important for scalability. Scaling metadata performance is more complex than scaling raw I/O performance since even a small inconsistency in metadata can lead to data corruption.

Resource management is the process by which businesses manage their various resources effectively. Those resources can be intangible – people and time – and tangible – equipment, materials, and finances.

It involves planning so that the right resources are assigned to the right tasks. Managing resources involves schedules and budgets for people, projects, equipment, and supplies. While it is often used in reference to project management, it applies to many other areas of business management. A small business, in particular, will pay attention to resource management in a number of areas

6.8 REFERENCE FOR FURTHER READING

1. https://www.shopify.in/encyclopedia/resource-management

2. https://www.wrike.com/blog/what-is-resource-management/#:~:text=Resource%20management%20is%20the%20process,or%20the%20adoption%20of%20software.

3. https://www.scality.com/topics/what-is-a-distributed-file-system

4. https://www.geeksforgeeks.org/what-is-dfsdistributed-file-system/

5. https://www.weka.io/learn/distributed-file-system/

6. https://www.techopedia.com/definition/1825/distributed-file-system-dfs

6.9 MODEL QUESTIONS

1. Discuss on Types of Distributed Systems

2 Explain Advantages and disadvantages of Distributed Systems

3. How does a distributed system work?
4. Explain the Architecture of resource Management in distributed Environment
5. Write a short note on Features of scheduling algorithms
6. Explain any two scheduling algorithms in detail
7. Explain Resource Management
8. Discuss on Working of Task Assignment Approach
9. Explain the Goals of Task Assignment Algorithms
10. Discuss the Need for Task Assignment in a Distributed System
11. Explain with example the Task Assignment Approach
12. Explain the concept of Classification of Load Balancing Algorithms
13. What are the Issues in Load Balancing Algorithms
14. Explain the Basic idea of load sharing
15. Differentiate between Load Balancing and Load Sharing

*****

7

DISTRIBUTED SYSTEM MANAGEMENT

Unit Structure

7.1 PROCESS MANAGEMENT

7.1.1 Introduction:

All business organizations involve processes. It can be as simple as buying ingredients, baking bread, selling bread, and receiving payment for a bakery. It can also be more complex, like a multistep purchasing process for vendor management. In either case, without an efficient system, unorganized processes can lead to problems that can adversely affect a business. Thus, it is important to implement process management regardless of the size of the company. To understand its importance, here's a quick run-through of how **process management benefits** business organizations.

7.1.2 What is Process Management?:

Process management is a systematic approach to ensure that effective and efficient business processes are in place. It is a methodology used to align business processes with strategic goals. In contrast to project management, which is focused on a single project, process management addresses repetitive processes carried out on a regular basis. It looks at every business process, individually and as a whole, to create a more efficient organization. It analyzes current systems, spots bottlenecks, and identifies areas of improvement. Process management is a long-term strategy that constantly monitors business processes so they maintain optimal efficiency. Implemented properly, it significantly helps boost business growth.

7.1.3 The Importance of Process Management:

When managing any organization, it is imperative to understand why process management is important. More than creating seamless workflows, it enables all aspects of business operations to run at an optimal pace. With business processes systematically implemented, you reduce time wasted on repetitive tasks and minimize errors due to human inefficiency. It also prevents the loss of data and missed steps within a process. Moreover, it ensures that resources are used properly so your business becomes more cost-efficient. Aside from improving business operations, process management also aligns your processes with the needs of your customers. This increases customer satisfaction and leads to higher revenues.

7.1.4 Realtime Process Management Examples:

Process management encompasses all aspects of business. Some business organizations use process management software to automate their systems, while others still use traditional methods of flowcharts and manuals. If you have been running your own company, you are most likely using some form of business process management (BPM).

Here are some real-world examples of how BPM is used in various industries.

Onboarding New Employees | Human Resource Department:

Without a proper system, the onboarding process can be chaotic and time-consuming. With BPM, forms and documents can be filled up and submitted electronically. Software is used to automatically filter data, find the best matches for a position, send messages, schedule interviews, and facilitate employee onboarding.

Managing Logistics | Shipping Company:

Logistics for a shipping company entails a long chain of complex processes while dealing with potentially thousands of people in various locations. BPM standardizes and optimizes routines involved to streamline the entire process and deliver quality service. It integrates production, finance, quality control, HR, customer service, and other departments. It centralizes data to facilitate easy retrieval of information in every phase of the business operations.

Loan Processing | Banking Firms:

With BPM, processing loans can be done in a much shorter time. It creates an efficient flow from document submission to credit and risk checks to loan approval. It also enables tracking of applications through the entire loan processing system.

Compliance Management| Insurance Companies:

BPM improves the overall regulatory compliance of insurance companies. It reduces human error and prevents data loss through proper documentation management. It also ensures that the company is able to comply with the latest state and federal regulations.

Customer Service | Retail Business:

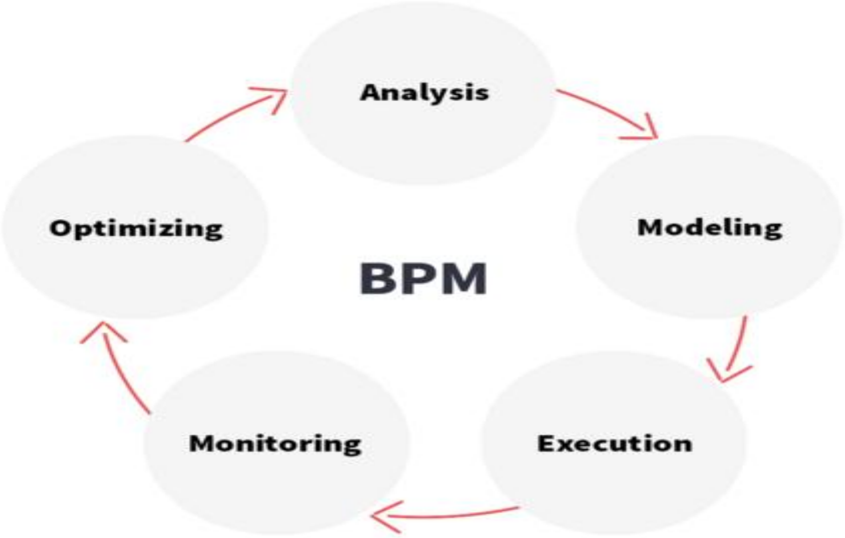
BPM enables customer-centric operations. It unifies all systems and departments for a smooth workflow that ensures all customer needs are met. It also identifies bottlenecks in the buyer's journey so the entire purchasing process can be improved.

7.1.5 How does BPM differ from workflow management?:

Workflow management is another term that is often mentioned in connection with process management. While workflow management is about coordinating and organizing processes, process management is about integrating these individual business processes into a larger whole. Workflow management is therefore a subfield, but by no means a synonym of process management. However, the aim of efficient processes is the same in both approaches.

7.1.6 Distinction between Digital Process Automation (DPA) and Process Management:

Digital Process Automation (DPA) is the next evolution in process management and supports companies in their efforts towards end-to-end digital transformation. In contrast to DPA, process management initially has nothing to do with digital transformation. However, in order to achieve the desired improvements, digital transformation usually involves companies digitizing and ultimately automating processes. In the case of DPA, company-wide business processes with external participants come more into focus. DPA is a technology that ensures that processes and systems are not only digitized, but that tasks and sequences that would normally require manual intervention are automated.



Life-cycle phases of Business Process Management (BPM)

Examples of Business Process Management (BPM) phases

Analysis:

In the analysis phase, processes are first identified and then analyzed (people, activities, individual steps, points in time, etc.).

Modeling:

The modeling phase essentially involves selecting and adapting processes that are to be implemented.

Execution:

Once processes are defined, the execution phase begins, including efforts to automate business processes.

Monitoring:

The monitoring phase is the prerequisite for subsequent optimization and is used for targeted process control.

Optimization:

The optimization phase begins following monitoring. The knowledge gained during monitoring makes it possible to improve the processes. It is possible, for example, that there are subtasks that have not been automated, that unnecessary steps are still being carried out, or that the data structure in general may require readjustment.

7.1.8 Digital process management: Application examples:

Throughout a company or in individual departments, there are various workflows that contain individual steps, as well as instructions for actions and responsibilities. These processes can be both internal and external. Examples of internal processes are the hiring of new employees or the ordering of office supplies. Invoicing processes (processing accounts payable invoices or creating invoices), on the other hand, are external processes because outsiders (customers, partners or suppliers) are involved.

Use cases that affect the entire company are, for example:

- Accounting and finance
- Purchasing decisions
- Administrative activities
- Customer services
- Facility management
- Personnel management
- Order processing
- Performance measurement
- Warehousing, logistics
- Standard operating procedures
- Employee performance and training
- Supplier and customer portal

In addition, there are processes whose origin can be assigned to a specific department. All these workflows can be mapped digitally. Even if the actual process management is detached from digital solutions, it can be implemented much more easily. Digital and automated business processes improve performance in all departments - this reduces overhead and enables flexibility in the company.

Human Resources:

With the advancement of organization-wide digital strategies, the HR department is also changing - and gaining in importance. In addition to administrative tasks such as approving vacation requests or checking applications, the HR department also has a strategic role to play in the "war for talent". In the future, fewer administrative tasks will be on the agenda than creative, intelligent activities in the corporate environment. This, in turn, requires companies to digitize and automate HR processes that used to be time-consuming and to connect them with the relevant data and departments. More flexible work structures, further developing corporate culture and supporting employees in digital transformation and change management are today's new priorities. To free up time for strategic and value-creating activities, HR employees must be relieved of their administrative tasks. This is done through digital business processes. Here are some workflows that can be digitized and automated in HR:

- Vacation request
- Digital personnel file
- Travel expense report
- Application management
- Employee onboarding
- Expense report
- Business trip application

And here are the reasons why digital automaton is particularly worthwhile in the HR sector:

- Promotion of new and agile ways of working and mobile working
- Increased employee satisfaction through fast processing
- Transparency in employee responsibilities
- Simplified change management
- Information exchange without interruption or integration problems

Administration:

Who hasn't had to handle a contract or obtain a permit? These tasks are often time-consuming when performed in analog (manual) form. Digitizing administrative processes is therefore an essential part of any digitization strategy. In addition, workflows can only be accelerated effectively when all systems, applications and company departments are connected locally and across decentralized locations. To achieve this, companies have to rethink their analog processes, sometimes redesigning and streamlining them. As a result, the heterogeneous IT landscape becomes interconnected, the company remains agile, and decision-makers can focus on management tasks. This frees up employees to use their time more effectively - and allows the company to maintain control over all physical and electronic records at all times. Additionally, the newly created visibility helps identify operational bottlenecks and continuously improve processes. Here are a few workflows that can be digitized and automated in administration:

- Contract management
- Training management
- Approval processes
- Digital construction file / digital project file
- Maintenance order / Production order
- Fleet management

And these are the reasons why digital automation in administration is worthwhile:

- Enormous time savings across all departments
- Faster and more up-to-date accounting
- Faster, more secure access to all documents and data
- Efficient workflows thanks to end-to-end processes
- Optimized service thanks to prompt, proactive contact
- Real-time daily accounting and payroll reports
- Reduced overall process costs
- Well-founded decisions thanks to increased transparency
- Strict security standards for documents and data exchange

Finance:

The finance department is undergoing a transformation. Administrative tasks still fill the business day, but according to McKinsey Research, as much as 75-79 percent of general accounting operations, cash dispurement, and revenue management tasks in finance can be fully automated in the future. Experts are convinced that with digitization, employees in finance departments are developing into business partners, answering trend-setting questions, interpreting data and contributing increasing value to their organization. The problem arises when the number of cross-departmental, control-relevant data continues to increase - but beneficial processes for management are not implemented. As real-time and ad-hoc analysis grows in importance along the entire value chain, it will become a priority to set up an interconnected value-creation system. Here are some workflows that can be digitized and automated in finance:

- Treasury Management
- Risk Assessment
- Purchase-to-Pay
  - Invoice Receipt Verification
  - Outgoing invoice processing
  - Payment processing
- Expenditure planning
- bank report
- data controlling
- Accounts Payable / Accounts Receivable

And these are the reasons why digital automation in finance is worthwhile:

- More performance through integrated data exchange
- Creation of electronic invoices with information linking
- Establishment of a paperless filing system as a central reference for all documents
- Ability to exchange, match and archive documents without material costs
- Optimized cost control

Purchasing:

It is no secret that digital procurement processes can now be fully automated. Companies also rely on operational and in-house digital processes such as requirements gathering and pricing, with data from various sources being integrated. However, it is the comprehensive exchange of information that brings the full advantage of digital procurement to light. Sustainable processes require more than strategic data management. The degree of interconnection between employees, departments and systems determines how digital and efficient purchasing can be. In day-to-day business, this can be seen by optimizing the supply chain and maximizing response time. With digital processes, the modern buyer maintains full control and transparency over processes, tasks and figures at all times, and can make decisions in real time, despite the large number of purchasing processes involved.

Here are some workflows that can be digitized and automated in purchasing:

- Investment request
- Goods receipt process
- Order processing
- Inventory process
- Delivery release

And these are the reasons why digital automation in purchasing is worthwhile:

- Contract, supplier and procurement management with seamless system and data integration
- Optimized supply chains
- Maximized reaction speed
- Automated routine processes (article dispositions, creation of order proposals or price inquiries)

- Transparent bookings and stock levels
- New savings potential
- Reduced processing time

Sales:

Today, more than 50 percent of new employees already belong to the "digital native" generation. They have has grown up with digital tools and different ways of working and are transferring this experience to their everyday working lives. They don't think much of mountains of documents and Excel lists with manually prepared data or paperwork. Customers have also opened up new information channels. By the time the first contact with sales is made, the decision has often already been made, so advance work must be done - on all channels. After all, the customer should have the choice of how to get in touch and the employee should be able to switch seamlessly between channels to qualify a lead efficiently. Information overload and attention deficits among prospects demand that companies streamline all sales processes and optimize them digitally. Information must be quick and easy to find, and data must be efficient to use. Here are some workflows that can be digitized and automated in sales:

- Order processing
- Information download
- Quotation approval
- Compilation of product sheets
- B2B sales process

And these are the reasons why digital automation in sales is worthwhile:

- Simplification of processes, communication and advice
- Increased reach and sales
- Optimized sales productivity
- Build trust and prevent mistrust
- Reduced costs for administration and organization
- Increased effectiveness and reduced susceptibility to errors
- Simplified contact, data maintenance & collaboration
- Sustainable competitive advantage

HR, finance and accounting are high on the digital agenda:

Recent surveys show that companies reported they are gaining momentum digitally, especially in HR, finance and accounting, but are still far from exploiting all the possibilities. Companies that want to introduce process management should pursue a holistic strategy while taking small steps forward in practice. Time will tell if other areas should wait until later in the journey and therefore lag behind digitally.

In particular, HR, finance, and accounts payable topics are digitized: In which of the following areas does your company already use or plan to use digital solutions for planning and controlling company resources?

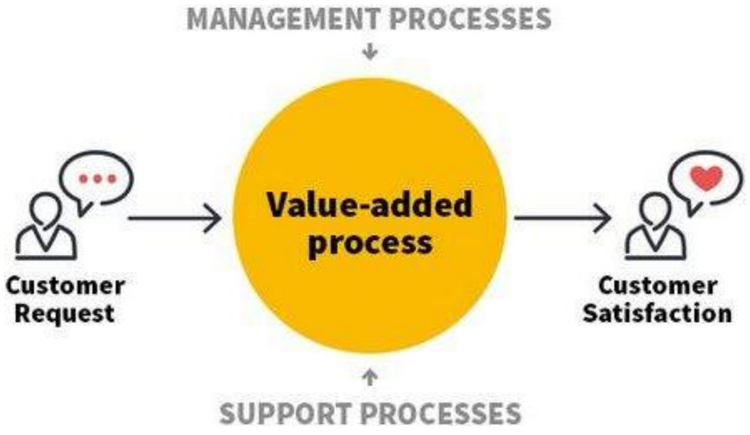
7.1.9 How can process management be implemented?:

How process management works can be explained in theory with the BPM lifecycle. But theory and practice are often only comparable to a limited extent. In practice, you should be aware that process management is not a rigid process of working through to-do lists, but has a lot to do with communication and teamwork. Pilot projects for digital automation in companies often start in isolation from one another. This silo approach can result in projects that become stalled, terminate unsuccessfully, or create islands of information that impede information flow to other teams across the organization. Surveys by Accenture and McKinsey indicate that when companies approach digital process management holistically from the outset with the necessary communication, failure can be prevented. It is also important to have the backing of all stakeholders, including management, in order to introduce and successfully drive digital process management forward to its full potential.

Learn about three essential steps to implement process management here.

1. Organization-wide business process structure:

Once you have gained support in the company for this topic, the first step is to get an overview of the external and internal processes in your company. Talk to colleagues from other departments and outline all workflows. Ideally, you should already record all the processes that take place in the company. Alternatively, you can take a department-specific approach and, for example, first analyze business processes in human resources or administration. It is useful to know that there are different types of processes that fulfill different tasks and purposes and ultimately help you to structure business processes. As a rule, value creation, support and management processes are distinguished from one another.



Value creation processes:

Value-adding processes are essential for the creation of a product or the provision of a service. They describe all corporate activities that are geared to customer needs. The value-adding processes a company has depend to a large extent on its industry focus or its core competencies. Typically, sales and marketing processes are almost always part of the value creation processes. It is also characteristic that different departments of a company are integrated into the value creation processes.

Support processes:

Support processes, also known as supporting processes, are not customer-oriented at first, but are necessary in order to carry out, control and optimize value creation and management processes. These include, for example, personnel selection and qualification as well as purchasing or the payment of invoices. In contrast to value-added processes, support processes can often be assigned to a single department.

Management processes:

Management processes relate to the company as a whole, contribute to the planning and control of core and support processes, and serve to strategically manage a company. Similar to support processes, this type of process is not directly related to the value creation of a company. Examples of management processes would include, but are not limited to: Aligning the company strategically, defining the corporate mission statement or formulating corporate goals.

2. Create process map:

The second step emerges from the discussions with internal and external stakeholders. The process map graphically depicts all processes in the company. Division into core, support and management processes is supplemented by the interactions/dependencies of business processes with each other. Figuratively speaking, the process map is your compass to keep focus on the path to process management. In addition, the process map can be consulted to explain the process management projects to employees. This clear representation helps to ensure understanding of the (change) project. In addition, a graphical representation makes it particularly easy to identify optimization potential. You can create a process map by enriching your core, support and management processes with additional information (e.g., responsible department/employee) and further subdividing them into sub-processes (if appropriate) and adding required third-party systems, documents and key figures.

3. Implement and optimize processes:

Once an overview has been created, you can work through the business processes from your map step-by-step, digitizing and even automating them. With subsequent process optimization, the BPM lifecycle then begins again.

Before you start with the implementation, however, you should have completed important tasks:

- Involve employees early on and offer training.
- Record process steps, responsibilities and other important information in documentation.
- Identify and contact all stakeholders in a timely manner. Also consider those who have no direct contact with the process but must approve it - for example, the works council or data protection officers.

It is also advisable to set regular coordination and review dates even before the introduction. In this way, you can ensure from the outset that processes are constantly scrutinized and, if necessary, adapted or even eliminated. We have also compiled five additional tips to help you successfully implement your project management.

Five tips for successful process management

1. Understand the current process:

Let's take an ordering process as an example. Do you know who orders what in your company and what approvals need to be obtained? No? Then you are in the same situation as many others. That is why you should get rid of the idea that you, or even company management, can have detailed knowledge of all processes. From a practical point of view, the best people to talk to are still those who work with the process on a regular basis. Don't get too hung up on theory. Talking to colleagues will tell you about individual process steps and everything else you need to know to get started.

2. Start small:

"It's hard to get started" - a saying that couldn't be more apt for process management. So don't make things unnecessarily difficult for yourself and start small. Choose simple processes that address the current situation and avoid being overwhelmed. Especially if process management is new. You should aim for small stages that can be implemented quickly and easily. Success will inspire you to do more - so you can gradually tackle other, as yet untried processes in your company.

3. Create a schedule:

You will probably realize pretty quickly that many things will take much longer than initially anticipated. Nevertheless, it makes sense to create a schedule that includes generous milestones and that you can regularly review and adjust. Process management is not something you can just do on the side. It is important to give all involved employees sufficient time to deal with the topic in detail and successfully implement their tasks.

4. Enable the exchange of information, ideas:

As with any new big project, sooner or later challenges crop up. Restructuring, new service providers and much more can mean that you have to throw your well-thought-out plan out the window and start over again. Regular meetings are a good way to learn about changes early on and to support each other. Bring yourself up to date, discuss problems and inspire each other. The intervals at which you schedule meetings depend on how much you need to discuss. It is advisable to choose shorter intervals, especially during critical phases, such as at the beginning. Once everything has settled in, weekly or monthly meetings are also advisable.

5. Use tools:

With your process map, you already have a powerful tool at hand. It is the basis for analysis, meetings and further developments and should always be included in the regular coordination meetings. But the choice of a suitable means of communication is also crucial. E-mails often lead to misunderstandings and ultimately cause more confusion than they help the process. Therefore, only write e-mails if your request can be explained succinctly and be sure to file any important documents in an agreed-upon location. This is where project management tools have proven to be the tool of choice, as they bundle all communication including documents, schedules, and to-do lists, etc. There are a number of software products available, some of which can even be used free of charge - for example, Asana or Trello.

7.1.10 Selection criteria for good BPM software:

With the help of BPM software, companies can design, implement, optimize and, above all, automate business processes. All tasks are performed in one platform. Accordingly, a BPM tool helps to approach process management holistically and to view and improve processes from start to finish. As a result, processes can not only be made more efficient, but resources can be adjusted according to need, errors can be reduced, time can be saved, and ultimately the entire value creation of a company can be increased.

In order to take full advantage of this, it is important to choose a suitable software. We have collected some criteria that experience has shown to play an important role in the search for and selection of good BPM software.

Monitoring:

Make sure that you can use the BPM software to monitor key business indicators in real time, if possible. Ideally, you should be able to visualize the data in a dashboard.

Scalability:

Every company has its own special features and unique requirements. Therefore, make sure that the selected process management software can handle them. This includes integration to third-party systems as well as preferred data types or archiving and search functions for documents. A scalable solution is also necessary for companies with growth and expansion plan. This enables them to handle not just current requirements, but also future changes as well.

Security:

Recent surveys across many industries shows that data privacy and security have become top priorities driving digital transformation initiatives. Security is therefore one of the most important criteria when it comes to selecting a suitable BPM tool. Particularly in the case of cloud solutions where the question of data residency, where data is stored, is at the forefront. In many countries and states, government regulations can be particularly strict. It is therefore advisable to choose a software provider that offers cloud options that meet data residency requirements. In addition, process management software should meet the requirements of data protection regulations such as GDPR.

Analysis and simulation:

You should make sure that the software provides an analysis and simulation function. With a heat map, for example, it is possible to analyze weak points within a process. The simulation function allows you to test the process from the end user's point of view. This allows you to check the functionality of subsequent steps, system activities, control conditions or scripts and to detect errors in advance.

Usability:

For process designers, a flexible and user-friendly interface is essential. This enables them to model and adapt even complicated process forms quickly and easily. Of course, end users also benefit from user-friendly BPM software that makes it particularly easy to start and control processes.

7.1.11 Benefits of Process Management:

Streamlined Processes: BPM restructures tangled operations into smooth workflows, simplifying operations and improving business agility.

Increased Productivity: BPM makes sure that resources and capital are utilized properly. It also improves business processes and working conditions to increase overall productivity.

Minimized Risks: By clearly defining responsibilities, BPM demands higher accountability. This minimizes risks due to human error and reduces inefficiencies.

Reduced Costs: BPM helps spot inefficiencies so they can be corrected. It also tracks the usage of resources. With fewer inefficiencies and proper utilization of resources, BPM can reduce costs and expenditures.

Business Process Management (BPM) is a method for analyzing, designing, controlling and ultimately improving business processes. Ideally, all of a company's business processes are included in the analysis. These can include, for example, processes with other companies, systems, customers, suppliers or partners. The goal is to improve business processes in such a way that they contribute optimally to achieving the company's goals. Process management or business process management are the terms for BPM and are used synonymously. Generally speaking, process management includes both analog and digital processes.

In summary, process management ensures

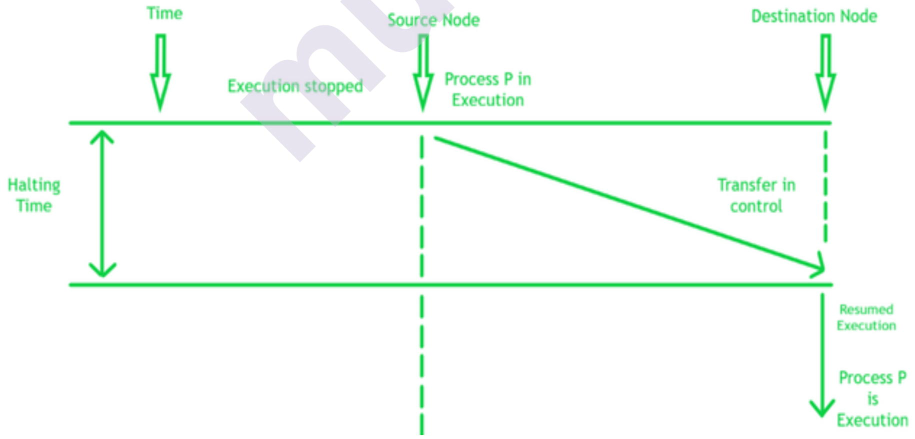
- transparent costs and responsibilities,
- efficient cross-departmental processes,

Information and knowledge exchange between different departments. Depending on the circumstances, the individual phases vary, but they usually include a modeling, execution and monitoring phase

7.2 PROCESS MIGRATION IN DISTRIBUTED SYSTEM

A process is essentially a program in execution. The execution of a process should advance in a sequential design. A process is characterized as an entity that addresses the essential unit of work to be executed in the system. A process is characterized as an entity that addresses the essential unit of work to be executed in the system.

Process migration is a particular type of process management by which processes are moved starting with one computing environment then onto the next.



There are two types of Process Migration:

- **Non-preemptive process:** If a process is moved before it begins execution on its source node which is known as a non-preemptive process.
- **Preemptive process:** If a process is moved at the time of its execution that is known as preemptive process migration. Preemptive

process migration is all the more expensive in comparison to the non-preemptive on the grounds that the process environment should go with the process to its new node.

7.2.1 Why use Process Migration?:

The reason to use process migration are:

- **Dynamic Load Balancing:** It permits processes to exploit less stacked nodes by relocating from overloaded ones.
- **Accessibility:** Processes that inhibit defective nodes can be moved to other perfect nodes.
- **System Administration:** Processes that inhabit a node if it is going through system maintenance can be moved to different nodes.
- **The locality of data:** Processes can exploit the region of information or other extraordinary abilities of a specific node.
- **Mobility:** Processes can be relocated from a hand-operated device or computer to an automatic server-based computer before the device gets detached from the network.
- **Recovery of faults:** The component to stop, transport and resume a process is actually valuable to support in recovering the fault in applications that are based on transactions.

7.2.2 What are the steps involved in Process Migration?

The steps which are involved in migrating the process are:

- The process is chosen for migration.
- Choose the destination node for the process to be relocated.
- Move the chosen process to the destination node.

The subcategories to migrate a process are:

- The process is halted on its source node and is restarted on its destination node.
- The address space of the process is transferred from its source node to its destination node.
- Message forwarding is implied for the transferred process.
- Managing the communication between collaborating processes that have been isolated because of process migration.

7.2.3 Methods of Process Migration:

The methods of Process Migration are:

1. Homogeneous Process Migration: Homogeneous process migration implies relocating a process in a homogeneous environment where all systems have a similar operating system as well as architecture. There are two unique strategies for performing process migration. These are i) User-level process migration ii) Kernel level process migration.

- **User-level process migration:** In this procedure, process migration is managed without converting the operating system kernel. User-level migration executions are more simple to create and handle but have usually two issues: i) Kernel state is not accessible by them. ii) They should cross the kernel limit utilizing kernel demands which are slow and expensive.

- **Kernel level process migration:** In this procedure, process migration is finished by adjusting the operating system kernel. Accordingly, process migration will become more simple and more proficient. This facility permits the migration process to be done faster and relocate more types of processes.

Homogeneous Process Migration Algorithms:

There are five fundamental calculations for homogeneous process migration:

- Total Copy Algorithm
- Pre-Copy Algorithm
- Demand Page Algorithm
- File Server Algorithm
- Freeze Free Algorithm

2. Heterogeneous Process Migration:

Heterogeneous process migration is the relocation of the process across machine architectures and operating systems. Clearly, it is more complex than the homogeneous case since it should review the machine and operating designs and attributes, as well as send similar data as homogeneous process migration including process state, address space, file, and correspondence data. Heterogeneous process migration is particularly appropriate in the portable environment where is almost certain that the portable unit and the base help station will be different machine types. It would be attractive to relocate a process from the versatile unit to the base station as well as the other way around during calculation. In most cases, this couldn't be accomplished by homogeneous migration. There are four essential types of heterogeneous migration:

- **Passive object:** The information is moved and should be translated

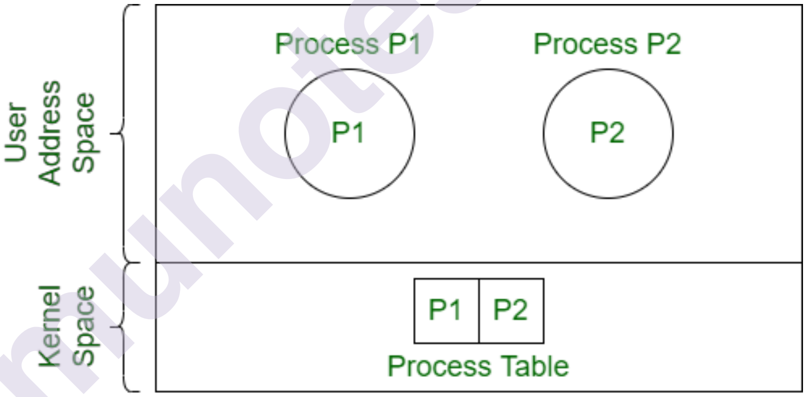
- **Active object, move when inactive:** The process is relocated at the point when it isn't executing. The code exists in the two areas, and just the information is moved and translated.
- **Active object, interpreted code:** The process is executing through an interpreter so just information and interpreter state need be moved.
- **Active object, native code:** Both code and information should be translated as they are accumulated for a particular architecture.

7.3 THREADS

A thread is a light weight process which is similar to a process where every process can have one or more threads. Each thread contains a Stack and a Thread Control Block. There are four basic thread models :

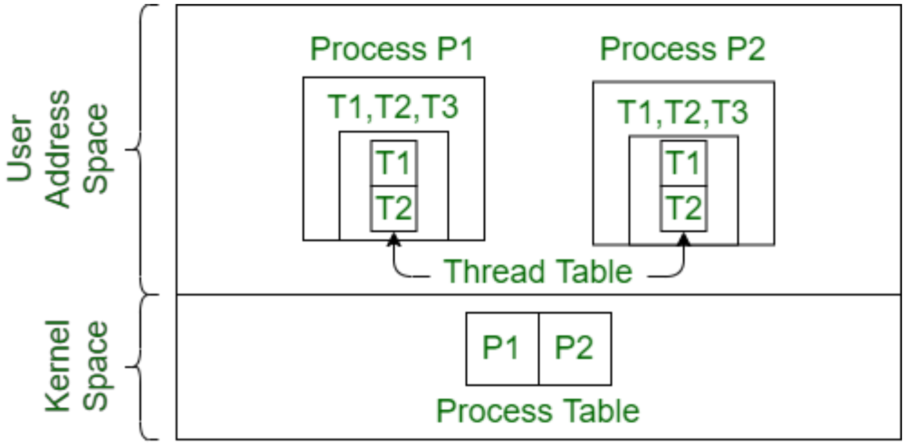
1. User Level Single Thread Model:

- Each process contains a single thread.
- Single process is itself a single thread.
- process table contains an entry for every process by maintaining its PCB.



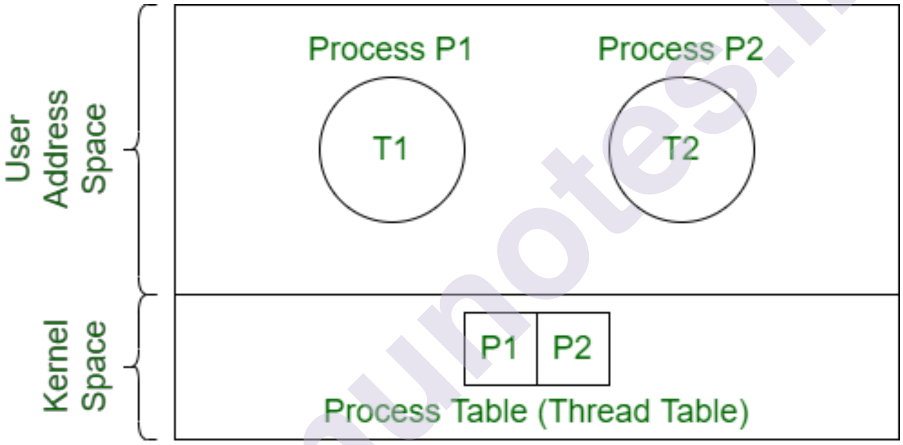
2. User Level Multi Thread Model:

- Each process contains multiple threads.
- All threads of the process are scheduled by a thread library at user level.
- Thread switching can be done faster than process switching.
- Thread switching is independent of operating system which can be done within a process.
- Blocking one thread makes blocking of entire process.
- Thread table maintains Thread Control Block of each thread of a process.
- Thread scheduling happens within a process and not known to Kernel.



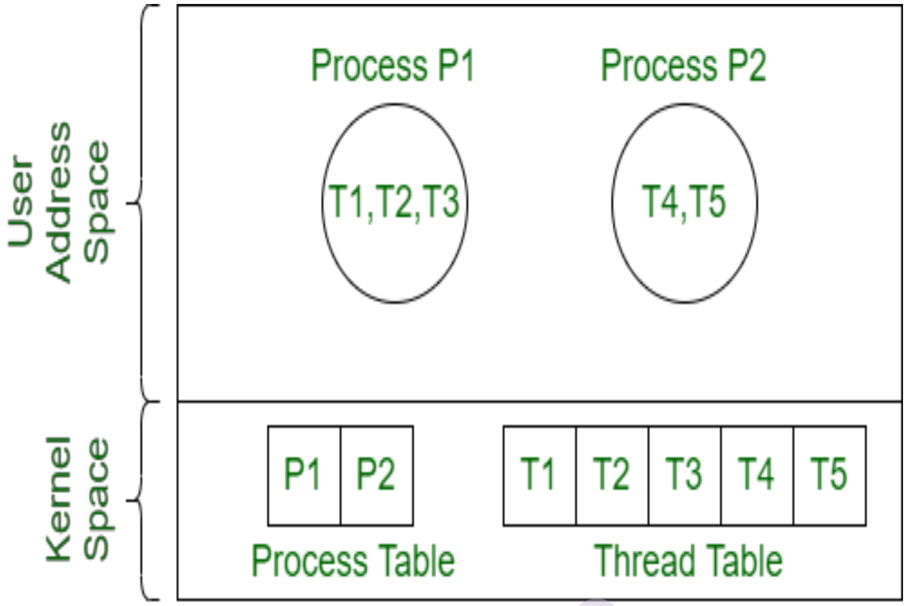
3. Kernel Level Single Thread Model:

- Each process contains a single thread.
- Thread used here is kernel level thread.
- Process table works as thread table.



4. Kernel Level Multi Thread Model:

- Thread scheduling is done at kernel level.
- Fine grain scheduling is done on a thread basis.
- If a thread blocks, another thread can be scheduled without blocking the whole process.
- Thread scheduling at Kernel process is slower compared to user level thread scheduling.
- Thread switching involves switch.



7.4 SUMMARY

A thread is also known as lightweight process. The idea is to achieve parallelism by dividing a process into multiple threads. For example, in a browser, multiple tabs can be different threads. MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc. The primary difference is that threads within the same process run in a shared memory space, while processes run in separate memory spaces. Threads are not independent of one another like processes are, and as a result threads share with other threads their code section, data section, and OS resources (like open files and signals). But, like process, a thread has its own program counter (PC), register set, and stack space

7.5 REFERENCE FOR FURTHER READING

1. https://www.geeksforgeeks.org/thread-in-operating-system/
2. https://www.geeksforgeeks.org/threads-and-its-types-in-operating-system/
3. https://www.javatpoint.com/threads-in-operating-system
4. https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/4_Threads.html
5. https://en.wikipedia.org/wiki/Thread_(computing)
6. https://padakuu.com/threads-summary-184-article
7. https://www.tutorialspoint.com/operating_system/os_multi_threading.htm
8. https://www.studytonight.com/operating-system/multithreading

7.6 MODEL QUESTIONS

1. What is Process Management?

2. Explain the Importance of Process Management?

3. Explain Realtime Process Management with Examples

4. How does BPM differ from workflow management?

5. Give the distinction between Digital Process Automation (DPA) and Process Management ?

7. Explain Business Process Management (BPM) phases with examples?

8. Explain application of Digital process management with examples?

9. How can process management be implemented?

10. What are Selection criteria for good BPM software?

11. Discuss the Benefits of Process Management ?

12. Why use Process Migration?

13. What are the steps involved in Process Migration?

14. Discuss the methods of Process Migration?

15. Explain Threads?

*****

8

DISTRIBUTED SYSTEM MANAGEMENT

Unit Structure

8.1 DATA FILE SYSTEM

A Distributed File System (DFS) is a file system that is distributed on multiple file servers or multiple locations. It makes the programs to access or to store isolated files with the local ones, allowing programmers to access files from any network or computer. It manages files and folders on different computers. It is mainly designed to provide file storage and access controlled to files over LAN and WAN.

A DFS is also called a client-server architecture based application, which allows the user or clients to access the data from the server as it is stored in their own computer. It provides location transparency and redundancy help to improve the data availability. And also use data replication strategy on multiple servers to prevent data access failure.

8.1.1 What is DFS (Distributed File System)?:

A **Distributed File System (DFS)** as the name suggests, is a file system that is distributed on multiple file servers or multiple locations. It allows programs to access or store isolated files as they do with the local ones, allowing programmers to access files from any network or computer.

The main purpose of the Distributed File System (DFS) is to allows users of physically distributed systems to share their data and resources by using a Common File System. A collection of workstations and mainframes connected by a Local Area Network (LAN) is a configuration on Distributed File System. A DFS is executed as a part of the operating system. In DFS, a namespace is created and this process is transparent for the clients.

DFS has two components:

Location Transparency:

Location Transparency achieves through the namespace component.

Redundancy:

Redundancy is done through a file replication component.

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

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

8.1.2 Features of DFS:

Transparency:

Structure transparency:

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

Access transparency:

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

Naming transparency:

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

Replication transparency:

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

User mobility:

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

Performance:

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

Simplicity and ease of use:

The user interface of a file system should be simple and the number of commands in the file should be small.

High availability:

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

Scalability:

Since growing the network by adding new machines or joining two networks together is routine, the distributed system will inevitably grow over time. As a result, a good distributed file system should be built to scale quickly as the number of nodes and users in the system grows. Service should not be substantially disrupted as the number of nodes and users grows.

High reliability:

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

Data integrity:

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

Security:

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

Heterogeneity:

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

8.1.3 History:

The server component of the Distributed File System was initially introduced as an add-on feature. It was added to Windows NT 4.0 Server and was known as “DFS 4.1”. Then later on it was included as a standard component for all editions of Windows 2000 Server. Client-side support has been included in Windows NT 4.0 and also in later on version of Windows. Linux kernels 2.6.14 and versions after it come with an SMB client VFS known as “cifs” which supports DFS. Mac OS X 10.7 (lion) and onwards supports Mac OS X DFS.

8.1.4 Applications:

NFS:

NFS stands for Network File System. It is a client-server architecture that allows a computer user to view, store, and update files remotely. The protocol of NFS is one of the several distributed file system standards for Network-Attached Storage (NAS).

CIFS:

CIFS stands for Common Internet File System. CIFS is an accent of SMB. That is, CIFS is an application of SIMB protocol, designed by Microsoft.

SMB:

SMB stands for Server Message Block. It is a protocol for sharing a file and was invented by IMB. The SMB protocol was created to allow computers to perform read and write operations on files to a remote host over a Local Area Network (LAN). The directories present in the remote host can be accessed via SMB and are called as “shares”.

Hadoop:

Hadoop is a group of open-source software services. It gives a software framework for distributed storage and operating of big data using the MapReduce programming model. The core of Hadoop contains a storage part, known as Hadoop Distributed File System (HDFS), and an operating part which is a MapReduce programming model.

NetWare:

NetWare is an abandon computer network operating system developed by Novell, Inc. It primarily used combined multitasking to run different services on a personal computer, using the IPX network protocol.

8.1.5 Working of DFS:

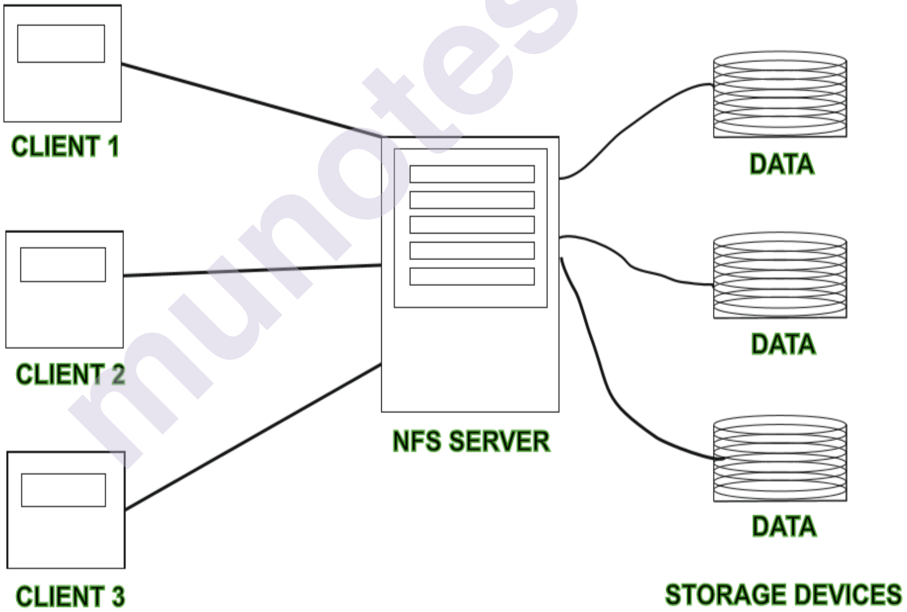
There are two ways in which DFS can be implemented:

Standalone DFS namespace:

It allows only for those DFS roots that exist on the local computer and are not using Active Directory. A Standalone DFS can only be acquired on those computers on which it is created. It does not provide any fault liberation and cannot be linked to any other DFS. Standalone DFS roots are rarely come across because of their limited advantage.

Domain-based DFS namespace:

It stores the configuration of DFS in Active Directory, creating the DFS namespace root accessible at \\**<domainname>\<dfsroot>** or \\**<FQDN>\<dfsroot>**



A file system is a set of data structures, interfaces, abstractions, and APIs that work together to manage any type of file on any type of storage device, in a consistent manner. Each operating system uses a particular file system to manage the files.

In the early days, Microsoft used **FAT** (FAT12, FAT16, and FAT32) in the **MS-DOS** and **Windows 9x** family. Starting from Windows **NT 3.1**, Microsoft developed **New Technology File System (NTFS)**, which had many advantages over FAT32, such as supporting bigger files, allowing longer filenames, data encryption, access management, journaling, and a lot more.

NTFS has been the default file system of the Window NT family (2000, XP, Vista, 7, 10, etc.) ever since. NTFS isn't suitable for non-Windows environments.

For instance, you can **only read** the content of an NTFS-formatted storage device (like flash memory) on a Mac OS, but you won't be able to write anything to it - unless you install an NTFS driver with write support. Or you can just use the **exFat** file system. **Extended File Allocation Table (exFAT)** is a lighter version of NTFS created by Microsoft in 2006. exFAT was designed for high-capacity removable devices, such as external hard disks, USB drives, and memory cards. exFAT is the default file system used by **SDXC cards**.

Unlike NTFS, exFAT has **read and write** support on Non-Windows environments as well, including Mac OS — making it the best cross-platform file system for high-capacity removable storage devices. So basically, if you have a removable disk you want to use on Windows, Mac, and Linux, you need to format it to exFAT. Apple has also developed and used various file systems over the years, including **Hierarchical File System (HFS)**, **HFS+**, and recently **Apple File System (APFS)**. Just like NTFS, APFS is a journaling file system and has been in use since the launch of **OS X High Sierra** in 2017. But how about file systems in Linux distributions?

The Extended File System (ext) family of file systems was created for the Linux kernel - the core of the Linux operating system. The first version of **ext** was released in 1991, but soon after, it was replaced by the **second extended file system** (**ext2**) in 1993. In the 2000s, the **third extended filesystem** (**ext3**) and **fourth extended filesystem (ext4)** were developed for Linux with journaling capability. **ext4** is now the default file system in many distributions of Linux, including Debian and Ubuntu. You can use the findmnt command on Linux to list your ext4-formatted partitions:

```
findmnt -lo source,target,fstype,used -t ext4
```

The output would be something like:

```
SOURCE    TARGET FSTYPE  USED

/dev/vda1 /      ext4    3.6G
```

Architecture of file systems:

A file system installed on an operating system consists of three layers:

- **Physical file system**
- **Virtual file system**
- **Logical file system**

These layers can be implemented as independent or tightly coupled abstractions. When people talk about file systems, they refer to one of these layers or all three as one unit. Although these layers are different across operating systems, the concept is the same. The physical layer is the concrete implementation of a file system; It's responsible for data storage and retrieval and space management on the storage device (or precisely: partitions).

The physical file system interacts with the storage hardware via device drivers. The next layer is the virtual file system or **VFS**. The virtual file system provides a **consistent view** of various file systems mounted on the same operating system. So does this mean an operating system can use multiple file systems at the same time? The answer is yes!

It's common for a removable storage medium to have a different file system than that of a computer.

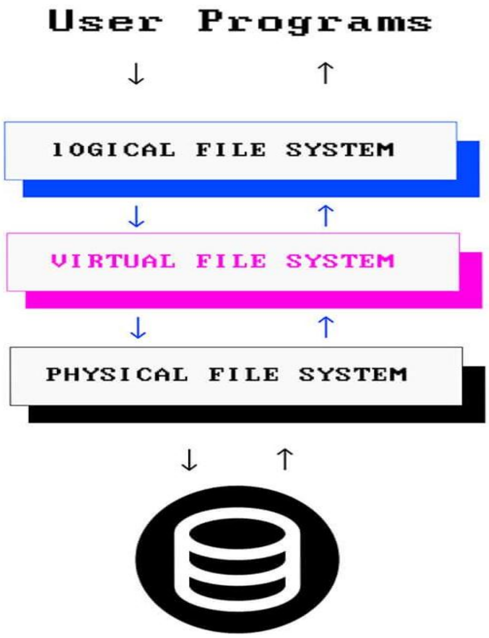
For instance, on Windows (which uses NTFS as the primary file system), a flash memory might have been formatted to exFAT or FAT32. That said, the operating system should provide a **unified interface** between computer programs (file explorers and other apps that work with files) and the different mounted file systems (such as NTFS, APFS, ext4, FAT32, exFAT, and UDF).

For instance, when you open up your file explorer program, you can copy an image from an ext4 file system and paste it over to your exFAT-formatted flash memory - without having to know that files are managed differently under the hood. This convenient layer between the user (you) and the underlying file systems is provided by the VFS. A VFS defines a contract that all physical file systems must implement to be supported by that operating system.

However, this compliance isn't built into the file system core, meaning the source code of a file system doesn't include support for every operating system's VFS. Instead, it uses a **file system driver** to adhere to the VFS rules of every file system. A driver is a program that enables software to communicate with another software or hardware.

Although VFS is responsible for providing a standard interface between programs and various file systems, computer programs don't interact with VFS directly. Instead, they use a unified API between programs and the VFS. Can you guess what it is? Yes, we're talking about the **logical file system**.

The logical file system is the user-facing part of a file system, which provides an API to enable user programs to perform various file operations, such as OPEN, READ, and WRITE, without having to deal with any storage hardware. On the other hand, VFS provides a bridge between the logical layer (which programs interact with) and a set of the physical layer of various file systems.



A high-level architecture of the file system layers

What does it mean to mount a file system?:

On Unix-like systems, the VFS assigns a **device ID** (for instance, dev/disk1s1) to each partition or removable storage device. Then, it creates a **virtual directory tree** and puts the content of each device under that directory tree as separate directories. The act of assigning a directory to a storage device (under the root directory tree) is called **mounting**, and the assigned directory is called a **mount point**. That said, on a Unix-like operating system, all partitions and removable storage devices appear as if they are directories under the root directory.

For instance, on Linux, the mounting points for a removable device (such as a memory card), are usually under the /media directory. That said, once a flash memory is attached to the system, and consequently, auto mounted at the default mounting point (/media in this case), its content would be available under the /media directory. However, there are times you need to mount a file system manually.

On Linux, it’s done like so:

```
mount /dev/disk1s1 /media/usb
```

In the above command, the first parameter is the device ID (/dev/disk1s1), and the second parameter (/media/usb) is the mount point. Please note that the mount point should already exist as a directory. If it doesn’t, it has to be created first:

```
mkdir -p /media/usb
```

```
mount /dev/disk1s1 /media/usb
```

If the mount-point directory already contains files, those files will be hidden for as long as the device is mounted.

Files metadata:

File metadata is a data structure that contains **data about a file**, such as:

- File size
- Timestamps, like creation date, last accessed date, and modification date
- The file's owner
- The file's mode (who can do what with the file)
- What blocks on the partition are allocated to the file
- and a lot more

Metadata isn't stored with the file content, though. Instead, it's stored in a different place on the disk - but associated with the file.

In Unix-like systems, the metadata is in the form of data structures, called **inode**. Inodes are identified by a unique number called the inode number. Inodes are associated with files in a table called inode tables. Each file on the storage device has an inode, which contains information about it such as the time it was created, modified, etc.The inode also includes the address of the blocks allocated to the file; On the other hand, where exactly it's located on the storage device In an ext4 inode, the address of the allocated blocks is stored as a set of data structures called **extents** (within the inode). Each extent contains the address of the first data block allocated to the file and the number of the continuous blocks that the file has occupied.

Whenever you open a file on Linux, its name is first resolved to an inode number.

Having the inode number, the file system fetches the respective inode from the inode table. Once the inode is fetched, the file system starts to compose the file from the data blocks registered in the inode. You can use the df command with the -i parameter on Linux to see the inodes (total, used, and free) in your partitions:

```
df -i
```

The output would look like this:

```
udev          4116100    378 4115722    1% /dev

tmpfs          4118422    528 4117894    1% /run

/dev/vda1      6451200 175101 6276099    3% /
```

As you can see, the partition /dev/vda1 has a total number of 6,451,200 inodes, of which 3% have been used (175,101 inodes). To see the inodes associated with files in a directory, you can use the ls command with -il parameters.

```
ls -li
```

And the output would be:

```
1303834 -rw-r--r--  1 root www-data  2502 Jul  8  2019 wp-links-opml.php

1303835 -rw-r--r--  1 root www-data  3306 Jul  8  2019 wp-load.php

1303836 -rw-r--r--  1 root www-data 39551 Jul  8  2019 wp-login.php

1303837 -rw-r--r--  1 root www-data  8403 Jul  8  2019 wp-mail.php

1303838 -rw-r--r--  1 root www-data 18962 Jul  8  2019 wp-settings.php
```

The first column is the inode number associated with each file.

The number of inodes on a partition is decided when you format a partition. That said, as long as you have free space and unused inodes, you can store files on your storage device.

It's unlikely that a personal Linux OS would run out of inodes. However, enterprise services that deal with a large number of files (like mail servers) have to manage their inode quota smartly.

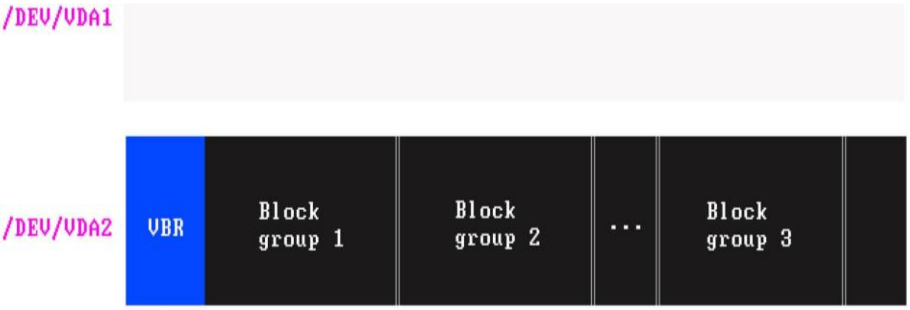
On NTFS, the metadata is stored differently, though.

NTFS keeps file information in a data structure called the **Master File Table (MFT)**. Every file has at least one entry in MFT, which contains everything about it, including its location on the storage device - similar to the inodes table. On most operating systems, you can grab metadata via the graphical user interface. For instance, when you right-click on a file on Mac OS, and select **Get Info** (Properties in Windows), a window appears with information about the file. This information is fetched from the respective file's metadata.

Space Management:

Storage devices are divided into fixed-sized blocks called **sectors**. A sector is the **minimum storage unit** on a storage device and is between 512 bytes and 4096 bytes (Advanced Format). However, file systems use a high-level concept as the storage unit, called **blocks.** Blocks are an abstraction over physical sectors; Each block usually consists of multiple sectors. Depending on the file size, the file system allocates one or more blocks to each file. Speaking of space management, the file system is aware of every used and unused block on the partitions, so it'll be able to allocate space to new files or fetch the existing ones when requested.

The most basic storage unit in ext4-formatted partitions is the block. However, the contiguous blocks are grouped into **block groups** for easier management.



The layout of a block group within an ext4 partition

Each block group has its own data structures and data blocks.

Here are the data structures a block group can contain:

- **Super Block:** a metadata repository, which contains metadata about the entire file system, such as the total number of blocks in the file system, total blocks in block groups, inodes, and more. Not all block groups contain the superblock, though. A certain number of block groups store a copy of the super as a backup.

- **Group Descriptors:** Group descriptors also contain bookkeeping information for each block group

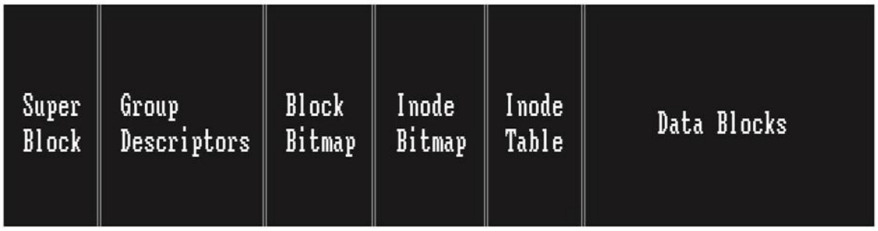
- **Inode Bitmap:** Each block group has its own inode quota for storing files. A block bitmap is a data structure used to identify used and unused inodes within the block group. 1 denotes used and 0 denotes unused inode objects.

- **Block Bitmap:** a data structure used to identify used & unused data blocks within the block group. 1 denotes used and 0 denotes unused data blocks

- **Inode Table:** a data structure that defines the relation of files and their inodes. The number of inodes stored in this area is related to the block size used by the file system.

- **Data Blocks:** This is the zone within the block group where file contents are stored. Ext4 file system even takes one step further (comparing to ext3), and organizes block groups into a bigger group called flex block groups.

The data structures of each block group, including the block bitmap, inode bitmap, and inode table, are concatenated and stored in the first block group within each flex block group. Having all the data structures concatenated in one block group (the first one) frees up more contiguous data blocks on other block groups within each flex block group. These concepts might be confusing, but you don't have to master every bit of them. It's just to depict the depth of file systems. The layout of the first block group looks like this:



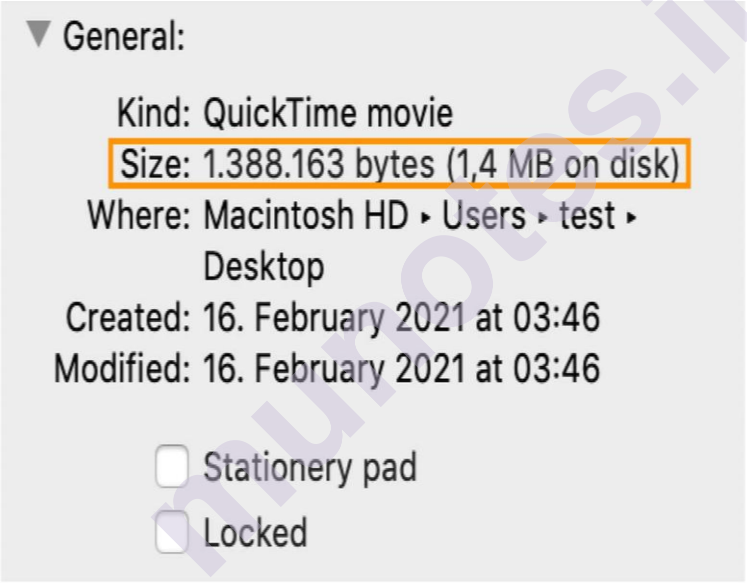
The layout of the first block in an ext4 flex block group

When a file is being written to a disk, it is written to one or more blocks within a block group.

Managing files at the block group level improves the performance of the file system significantly, as opposed to organizing files as one unit.

Size vs size on disk

Have you ever noticed that your file explorer displays two different sizes for each file: **size,** and **size on disk**.



▼ General:

Kind: QuickTime movie

Size: 1.388.163 bytes (1,4 MB on disk)

Where: Macintosh HD ▸ Users ▸ test ▸ Desktop

Created: 16. February 2021 at 03:46

Modified: 16. February 2021 at 03:46

□ Stationery pad

□ Locked

Size and Size on disk:

Why are size and size on disk slightly different?

We already know depending on the file size, one or more blocks are allocated to a file. One block is the minimum space that can be allocated to a file. This means the remaining space of a partially-filled block cannot be used by another file. This is the rule!

Since the size of the file isn't an integer multiple of blocks, the last block might be partially used, and the remaining space would remain unused - or would be filled with zeros.

So "size" is basically the actual file size, while "size on disk" is the space it has occupied, even though it's not using it all.

You can use the du command on Linux to see it yourself.

```
du -b "some-file.txt"
```

The output would be something like this:

```
623 icon-link.svg
```

And to check the size on disk:

```
du -B 1 "icon-link.svg"
```

Which will result in:

```
4096    icon-link.svg
```

Based on the output, the allocated block is about 4kb, while the actual file size is 623 bytes. This means each block size on this operating system is 4kb.

8.1.6 Advantages:

- DFS allows multiple user to access or store the data.
- It allows the data to be share remotely.
- It improved the availability of file, access time, and network efficiency.
- Improved the capacity to change the size of the data and also improves the ability to exchange the data.
- Distributed File System provides transparency of data even if server or disk fails.

8.1.7 Disadvantages:

- In Distributed File System nodes and connections needs to be secured therefore we can say that security is at stake.
- There is a possibility of lose of messages and data in the network while movement from one node to another.
- Database connection in case of Distributed File System is complicated.
- Also handling of the database is not easy in Distributed File System as compared to a single user system.
- There are chances that overloading will take place if all nodes tries to send data at once.

8.1.8 Benefits of DFS Models:

The distributed file system brings with it some common benefits.

A DFS makes it possible to restrict access to the file system, depending on access lists or capabilities on both the servers and the clients, depending on how the protocol is designed. Also, since the server also provides a single central point of access for data requests, it is thought to be fault-tolerant (as mentioned above) in that it will still function well if some of the nodes are taken offline. This dovetails with some of the reasons that DFS was developed in the first place – the system can still have that integrity if a few workstations get moved around.

8.1.9 DFS and Backup:

Ironically enough, even though a DFS server is prized for being a single central point of access, another server may also be in play. However, that doesn't mean that there won't be that single central access point. The second server will be for backup.

Because businesses invest in having one central DFS server, they will worry that the server could be compromised somehow. Backing all of the data up at a separate location ensures the right kind of redundancy to make the system fully fault-tolerant, even if the king itself (the primary server) is toppled by something like a DDoS attack or something else. DFS systems, like other systems, continue to innovate. With new kinds of net

8.1.10 The challenges associated with DFS:

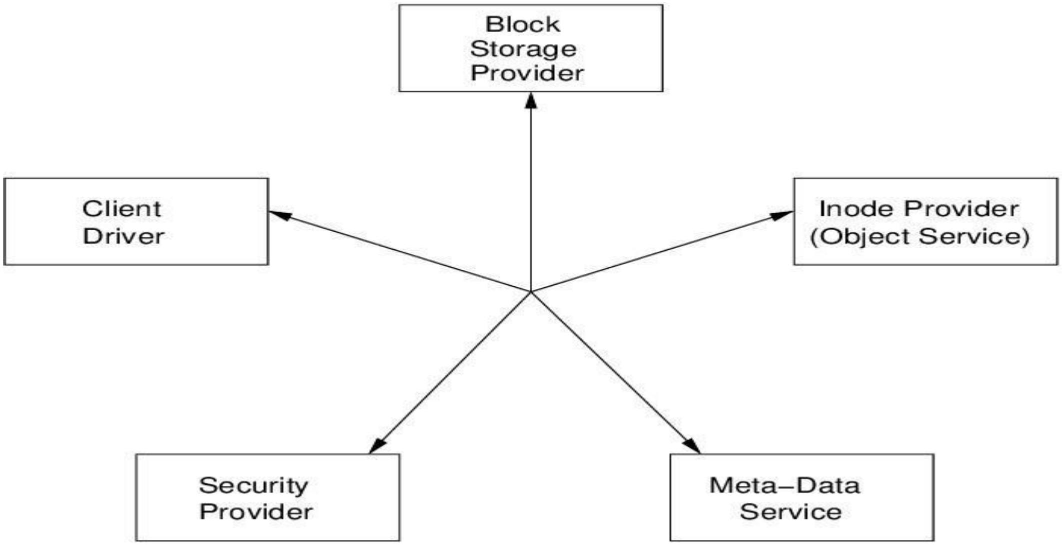
- Data redundancy and inconsistency.
- Difficulty in accessing data.
- Data isolation
- Integrity problems
- Unauthorized access is not restricted.
- It coordinates only physical access.

8.1.11 Components:

The components of DFS are as follows:

- Block Storage provider
- Client Driver
- Security provider
- Meta- Data Service
- Object service.

These components are pictorially represented below:



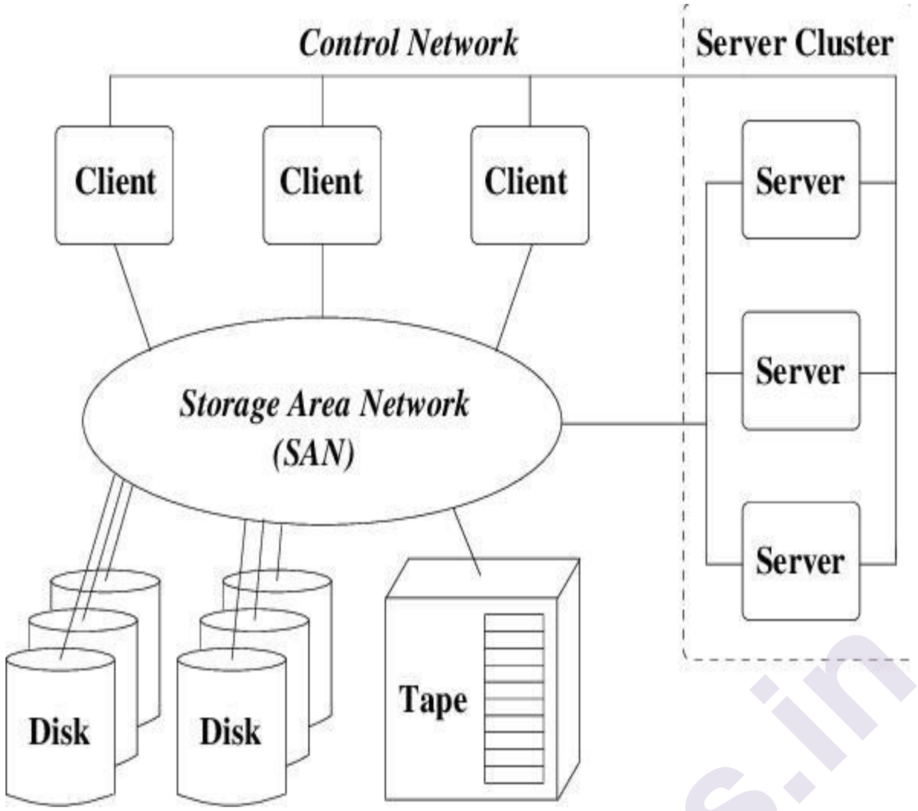
Features:

The features of DFS are as follows:

- User mobility
- Easy to use
- High availability
- Performance
- Coherent access
- Location independence
- File locking
- Multi-networking access
- Local gateways
- Multi-protocol access

Example:

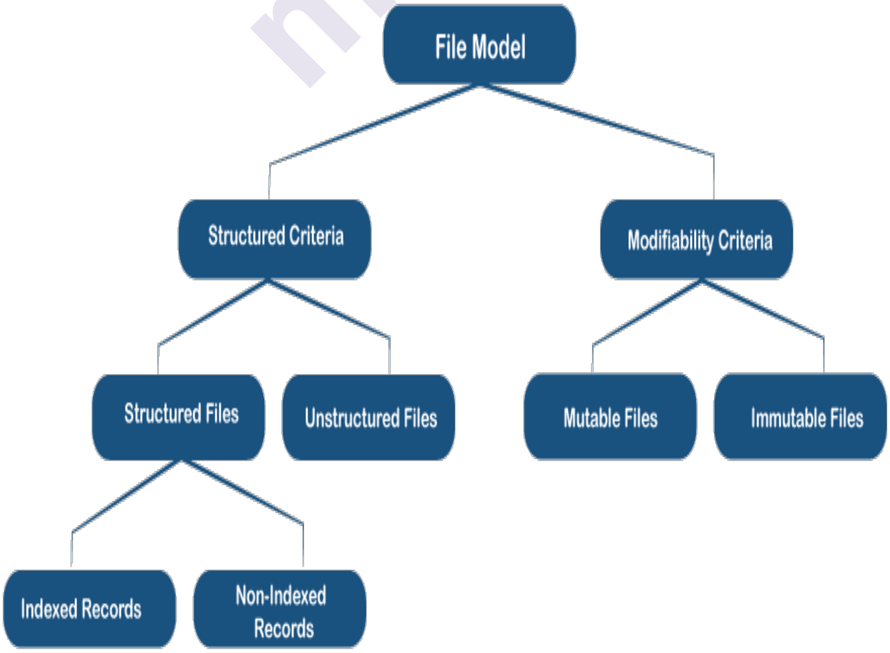
Given below is an example of DFS Structure:



8.2 FILE MODELS IN DISTRIBUTED SYSTEM

Several machines are utilized in **Distributed File Systems (DFS)** to supply the file system's facility. Various file systems frequently use various conceptual models. Models based on structure and mobility is frequently used for file modeling. In this article, you will learn about the file models in the distributed operating system.

Types of Files models in the distributed operating systems



There are mainly two types of file models in the distributed operating system.

1. **Structure Criteria**

2. **Modifiability Criteria**

Structure Criteria

There are two types of file models in structure criteria. These are as follows:

1. **Structured Files**

2. **Unstructured Files**

Structured Files

The **Structured file model** is presently a rarely used file model. In the structured file model, a file is seen as a collection of records by the file system. Files come in various shapes and sizes and with a variety of features. It is also possible that records from various files in the same file system have varying sizes. Despite belonging to the same file system, files have various attributes. A record is the smallest unit of data from which data may be accessed. The read/write actions are executed on a set of records. Different "File Attributes" are provided in a hierarchical file system to characterize the file. Each attribute consists of two parts: a name and a value. The file system used determines the file attributes. It provides information on files, file sizes, file owners, the date of last modification, the date of file creation, access permission, and the date of last access. Because of the varied access rights, the Directory Service function is utilized to manage file attributes.

The structured files are also divided into two types:

1. Files with Non-Indexed records

2. Files with Indexed records

1. Files with Non-Indexed records:

Records in non-indexed files are retrieved based on their placement inside the file. For instance, the second record from the starting and the second from the end of the record.

2. Files with Indexed records:

Each record contains a single or many key fields in a file containing indexed records, each of which may be accessed by specifying its value. A file is stored as a B-tree or similar data structure or hash table to find records quickly.

Unstructured Files:

It is the most important and widely used file model. A file is a group of unstructured data sequences in the unstructured model. Any substructure does not support it. The data and structure of each file available in the file system is an uninterrupted sequence of bytes such as UNIX or DOS. Most latest OS prefer the unstructured file model instead of the structured file model due to sharing of files by multiple apps. It has no structure; therefore, it can be interpreted in various ways by different applications.

Modifiability Criteria:

There are two files model in the Modifiability Criteria. These are as follows:

1. Mutable Files
2. Immutable Files

1. Mutable Files:

The existing operating system employs the mutable file model. A file is described as a single series of records because the same file is updated repeatedly once new material is added. After a file is updated, the existing contents are changed by the new contents.

2. Immutable Files:

The **Immutable file model** is used by **Cedar File System (CFS)**. The file may not be modified once created in the immutable file model. Only after the file has been created can it be deleted. Several versions of the same file are created to implement file updates. When a file is changed, a new file version is created. There is consistent sharing because only immutable files are shared in this file paradigm. Distributed systems allow caching and replication strategies, overcoming the limitation of many copies and maintaining consistency. The disadvantages of employing the immutable file model include increased space use and disc allocation activity. CFS uses the **"Keep"** parameter to keep track of the file's current version number. When the parameter value is 1, it results in the production of a new file version. The previous version is erased, and the disk space is reused for a new one. When the parameter value is greater than 1, it indicates the existence of several versions of a file. If the version number is not specified, CFS utilizes the lowest version number for actions such as **"delete"** and the highest version number for other activities such as **"open"**.

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

On the basis of the unit of data access, following file access models might be used in order to access the specific file.

1. File-level transfer model
2. Block-level transfer model
3. Byte-level transfer model
4. Record-level transfer model

1. **File-level transfer model:** In file-level transfer model, the complete file is moved while a particular operation necessitates the file data to be transmitted all the way through the distributed computing network amongst client and server. This model has better scalability and is efficient.

2. **Block-level transfer model:** In block-level transfer model, file data transfers through the network amongst client and a server is accomplished in units of file blocks. In short, the unit of data transfer in block-level transfer model is file blocks. The block-level transfer model might be used in distributed computing environment comprising several diskless workstations.

3. **Byte-level transfer model:** In byte-level transfer model, file data transfers the network amongst client and a server is accomplished in units of bytes. In short, the unit of data transfer in byte-level transfer model is bytes. The byte-level transfer model offers more flexibility in comparison to the other file transfer models since, it allows retrieval and storage of an arbitrary sequential subrange of a file. The major disadvantage of byte-level transfer model is the trouble in cache management because of the variable-length data for different access requests.

4. **Record-level transfer model:** The record-level file transfer model might be used in the file models where the file contents are structured in the form of records. In record-level transfer model, file data transfers through the network amongst client and a server is accomplished in units of records. The unit of data transfer in record-level transfer model is record.

8.3 SUMMARY

DFS **allows multiple user to access or store the data**. It allows the data to be share remotely. It improved the availability of file, access time, and network efficiency. Improved the capacity to change the size of the data and also improves the ability to exchange the data.

The main purpose of the Distributed File System (DFS) is to allows users of physically distributed systems to share their data and resources by using a Common File System.

A collection of workstations and mainframes connected by a Local Area Network (LAN) is a configuration on Distributed File System. A DFS is executed as a part of the operating system. In DFS, a namespace is created and this process is transparent for the clients.

8.4 REFERENCE FOR FURTHER READING

1. https://www.unf.edu/~sahuja/cis6302/filesystems.html

2. https://www.geeksforgeeks.org/what-is-dfsdistributed-file-system/

3. https://www.ques10.com/p/2247/what-are-the-good-features-of-a-distributed-file-1/

4. https://www.javatpoint.com/distributed-file-system

5. https://en.wikipedia.org/wiki/Distributed_File_System_(Microsoft)

6. https://erandipraboda.medium.com/characteristics-of-distributed-file-systems-bf5988f85d3

8.5 MODEL QUESTIONS

1. What is DFS (Distributed File System)?
2. What are the Features of DFS
3. Discuss on the History of DFS?
4. What are the Applications of DFS?
5. Explain the Working of DFS?
6. What are the Advantages of DFS?
7. What are the Disadvantages of DFS?
8. Explain the Benefits of DFS Models?
9. Discuss on DFS and Backup?
10. Discuss The challenges associated with DFS?
11. What are the Components of DFS?
12. Explain the concept of File Models in Distributed System?

*****

UNIT V

9

INTRODUCTION TO CLOUD COMPUTING

Unit Structure

9.0 OBJECTIVE

After studying this module, you will be able to understand

- Cloud Computing and its characteristics
- Benefits of Cloud Computing
- Cloud Computing Architecture
- Cloud based Services

9.1 INTRODUCTION

- Cloud Computing is a model for enabling convenient, on-demand network access to a shared pool of resources that can be rapidly provided and released with minimum management efforts.
- Cloud Computing intends to realize the concept of Computing as a utility like water, gas, electricity etc.

- Cloud Computing referred as the accessing and storing of data and provide services related to computing over the internet.
- It simply referred as it remote services on the internet manage and access data online rather than any local drives.
- The data can be anything like images, videos, audios, documents, files etc.
- The potential of cloud computing has been recognized by industry.
- Cloud computing is in huge demand so, big organization providing the service like Amazon AWS, Microsoft Azure, Google Cloud, Alibaba cloud etc. are some Cloud Computing service Provide

9.1.1 History and evolution:

- In early days before Cloud Computing was come into existence, client Server Architecture was used.
- In this all the data and control of client resides in Server side, so if user wants to access data it has to take the permission. It is having many disadvantages.
- So, After Client Server computing, Distributed Computing was come into existence.
- In this all the computers are in network so user can share resources as and when required. This also having certain limitations.
- So to remove limitations faced in distributed system, cloud computing was emerged.
- During 1961, John MacCharty delivered his speech at MIT that "Computing Can be sold as a Utility.
- This concept of computing was not appreciated so after few years this is implemented by Salesforce.com in 1999.
- This company started delivering an enterprise application over the internet and in this way Cloud Computing was started.
- In 2002, Amazon started Amazon Web Services (AWS), Amazon will provide storage, computation over the internet.
- In 2006 Amazon will launch Elastic Compute Cloud Commercial Service which is open for Everybody to use.
- In 2009, Google Play also started providing Cloud Computing Enterprise Application.
- In 2009, Microsoft launch Microsoft Azure and after that other companies like Alibaba, IBM, Oracle, HP also introduces their Cloud Services.

9.2 CHARACTERISTICS OF CLOUD COMPUTING

1.	On demand self-service	Here the user is able to use web services and resources on demand. User can logon to the website any time and use them.
2.	Ubiquitous Access-	As Cloud Computing is completely web based user can have accessed it from anywhere and anytime.
3.	Resource Pooling-	Resource pooling allows cloud providers to pool large-scale IT resources to serve multiple cloud consumers. Different physical and virtual IT resources are dynamically assigned and reassigned according to cloud consumer demand, typically followed by execution through statistical multiple xing.
4.	Rapid Elasticity	Elasticity is the automated ability of acloud to transparent scale IT resources, as required in response to runtime conditions or as pre-determined by the cloud consumer or cloud provider.
5.	Measured Usage	The measured usage characteristic represents the ability of a Cloud platform to keep track of the usage of its IT resources, primarily by cloud consumers. Based on what is measured, the cloud provider can charge a cloud consumer only for the IT resources used and/or for the time frame during which access to the IT resources was granted.
6.	Resiliency	Resilient computing is a form of failover that distributes redundant Implementations of IT resources across physical locations.

9.3 CLOUD COMPUTING EXAMPLES

Here we are discussing about two popular cloud computing facilities:

1. Amazon Elastic Computing Cloud (EC2):

- It is a part of set of standalone services which includes object storage service, for hosting and simple database.

- With EC2 user may rent virtual machine instances to run their own software and also can monitor number of VMs as demand changes.

- If user wants to use Amazon EC2 then he has to create Amazon Machine Image(AMI), then upload AMI to Amazon S3, select OS and start instances, then monitor and control via web interface or API.

2. **Google App Engine:**

- It is an end –to-end service.

- It allows developer to run their web application on Googles infrastructure for that

- Download App engine SDK

- Develop an application as a set of python programs

- Register for an application ID

- Submit the application to Google

9.4 BENEFITS OF CLOUD COMPUTING

Various advantages are provided by Cloud Computing

1. On-demand access to pay-as-you-go computing resources on a short-term basis (such as processors by the hour), and the ability to release these computing resources when they are no longer needed.

2. The perception of having unlimited computing resources that are available on demand, Thereby reducing the need to prepare for provisioning.

3. The ability to add or remove IT resources at a fine-grained level, such as modifying Available storage disk space by single gigabyte increments.

4. Abstraction of the infrastructure so applications are not locked into devices or location And can be easily moved if needed

5. An IT resource with increased availability is accessible for longer periods of time (for example, 22 hours out of a 24-hour day). Cloud providers generally offer “resilient” IT resources for which they are able to guarantee high levels of availability.

6. An IT resource with increased reliability is able to better avoid & recover from exception conditions. The modular architecture of cloud environments provides extensive fail over support that increases reliability.

9.5 RISKS OF CLOUD COMPUTING

Some downsides of Cloud Computing are discussed below:

1. **Security and privacy**:

It is always a risk to handover the sensitive information since data management and infrastructure management in cloud computing by third party.

2. **Lock in:**

To switch from one Cloud Service provider(CSP) to another is very difficult. It results in dependency on particular CSP.

3. **Isolation Failure:**

It involves the failure of isolation mechanism which will separate storage, memory, routing between different tenants.

4. **Management Interface compromise:**

In case of public cloud provider, customer management interface is accessible through the internet.

5. **Insecure data deletion:**

Sometime data requested for deletion may not get deleted. This can happen because extra copies of data are stored but not available on disk or disk may destroyed.

9.6 CLOUD COMPUTING ARCHITECTURE

In current time Cloud Computing is giving a new shape to every organization by providing on demand virtualized services/resources. Starting from small to medium and medium to large, every organization use cloud computing services in storing information and accessing that from anywhere and anytime only with the help of internet. Transparency, scalability, security and intelligent monitoring are some of the most important constraints which every cloud infrastructure should experience.

9.6.1 Cloud Computing Architecture:

Cloud Computing Architecture:

The cloud architecture is divided into 2 parts i.e.

1. Frontend

2. Backend

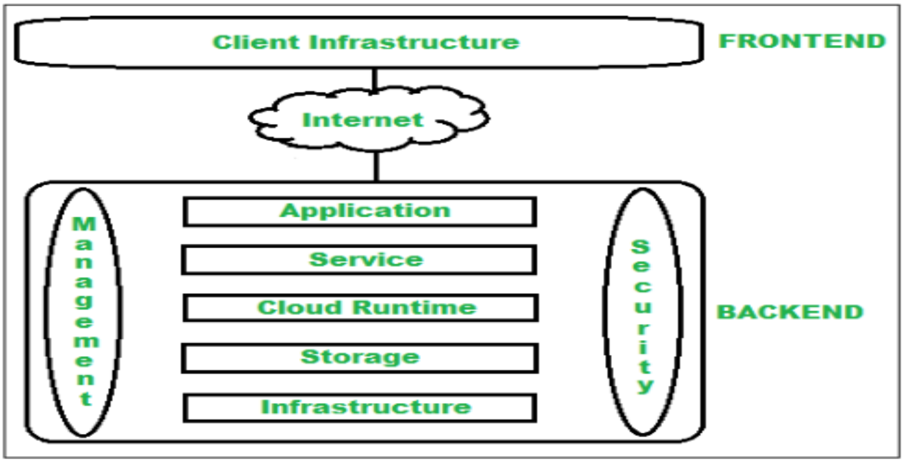


Figure - Internal Architectural view of Cloud Computing

1. Frontend:

Frontend of the cloud architecture refers to the client side of cloud computing system. Means it contains all the user interfaces and applications which are used by the client to access the cloud computing services/resources.

For example, use of a web browser to access the cloud plat form.

Client Infrastructure:

Client Infrastructure refers to the frontend components. It contains the applications and user interfaces which are required to access the cloud platform.

2. Backend:

Backend refers to the cloud itself which is used by the service provider. It contains the resources as well as manages the resources and provides security mechanisms. Along with this it includes huge storage, virtual applications, virtual machines, traffic control mechanisms, deployment models etc.

1. Application:

Application in backend refers to a software or platform to which client accesses. Means it provides the service in backend as per the client requirement.

2. Service:

Service in backend refers to the major three types of cloud based services like **SaaS, PaaS and IaaS**. Also manages which type of service the user accesses.

3. Cloud Runtime:

Runtime cloud in backend refers to provide of execution and runtime platform/environment to the virtual machine.

4. Storage:

Storage in backend refers to provide flexible and scalable storage service and management of stored data.

5. Infrastructure:

Cloud Infrastructure in backend refers to hardware and software components of cloud like it includes servers, storage, network devices, virtualization software etc.

6. Management:

Management in backend refers to management of backend components like application, service, runtime cloud, storage, infrastructure, and other security mechanisms etc.

7. Security:

Security in backend refers to implementation of different security mechanisms in the backend for secure cloud resources, systems, files, and infrastructure to end-users.

8. Internet:

Internet connection acts as the medium or a bridge between frontend and backend and establishes the interaction and communication between frontend and backend.

Benefits of Cloud Computing Architecture:

- Makes overall cloud computing system simpler.
- Improves data processing requirements.
- Helps in providing high security.
- Makes it more modularized.
- Results better disaster recovery.
- Gives good user accessibility.
- Reduces IT operating costs.

9.6.2 Types of Cloud:

A cloud deployment model represents a specific type of cloud environment, primarily distinguished by ownership, size, and access.

There are four common cloud deployment models:

- Public cloud
- Community cloud
- Private cloud
- Hybrid cloud

Public Cloud:

- A public cloud is a publicly accessible cloud environment owned by a third-party cloud provider
- The IT resources on public clouds are usually provisioned via cloud delivery models and are generally offered to cloud consumers at a cost or are commercialized via other avenues (such as advertisement).
- The cloud provider is responsible for the creation and on-going maintenance of the public cloud and its IT resources.
- Example-Google, Oracle, Microsoft

Community Clouds:

- A community cloud is similar to a public cloud except that its access is limited to a specific community of cloud consumers.
- The community cloud may be jointly owned by the community members or by third-party cloud provider that provisions a public cloud with limited access.
- The member cloud consumers of the community typically share the responsibility for defining and evolving the community Cloud.
- Example- Government agency

Private Clouds:

- A private cloud is owned by a single organization.
- Private clouds enable an organization to use cloud computing technology as a means of centralizing access to IT resources by different parts, locations,or departments the organization.
- The use of a private cloud can change how organizational and trust boundaries definedand applied
- The actual administration of a private cloud environment may be carried out by internal or outsourced staff.
- Example-HP data centre, Ubuntu

Hybrid Clouds:

- A hybrid cloud is a cloud environment comprised of two or more different cloud deployment models.
- The service of a hybrid cloud can be distributed in multiple cloud types.
- Example-Amazon Web service

9.6.3 Cloud based Services:

A cloud delivery model represent a specific, pre-packaged combination of IT resources offered by a cloud provider. Three common cloud delivery models have become widely established and formalized:

- Software-as-a-Service (SaaS)
- Platform-as-a-Service (PaaS)
- Infrastructure-as-a-Service (IaaS)

9.6.3.1 Software as a service (SaaS):

- A software program positioned as a shared cloud service and made available as a "product" or generic utility represents the typical profile of a SaaS offering.
- The SaaS delivery model is typicallyused to make a reusable cloud service widely available (often commercially) to a range of cloud consumers.
- SaaS application includes billing and invoice system, CRM, Help desk application, HR solutions.
- SaaS provides Application programming Interface(API), which allows the developer to develop a customized application.

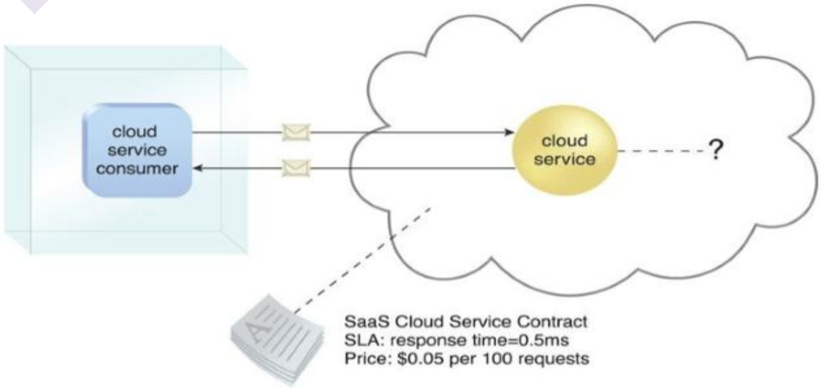


Figure 1: The cloud service consumer is given access the cloud service contract, but not to any underlying IT resources or implementation details

Benefits:

1. It is beneficial in terms of scalability, efficiency, performance.

2. Modest software tools

3. Efficient use of software licence

4. Centralized management and data

5. Platform responsibility managed by provider

6. Multitenant solution

9.6.3.2 Platform as a service (PaaS):

- The PaaS delivery model represents a pre-defined "ready-to-use" environment typically comprised of already deployed and configured IT resources.

- PaaS relies on (and is primarily defined by) the usage of a readymade environment that establishes a set of pre-packaged products and tools used to support the entire delivery lifecycle of custom applications.

- Common reasons a cloud consumer would use and invest in a PaaS environment include:

- The cloud consumer wants to extend on premise environments in to the cloud for scalability and economic purposes.

- The cloud consumer uses the ready-made environment to entirely substitute an on premise environment.

- The cloud consumer wants to become a cloud provider and deploys its own cloud services to be made available to other external cloud consumers.

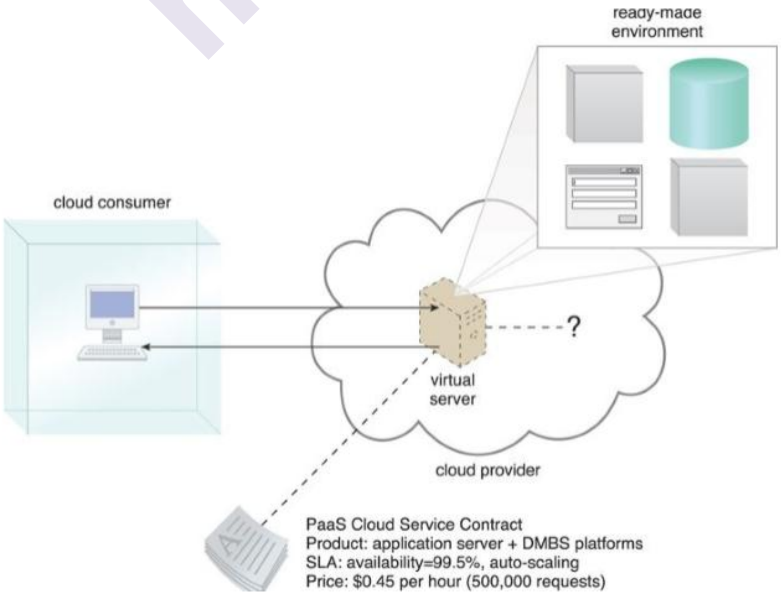

ready-made environment
cloud consumer
?
virtual server
cloud provider
PaaS Cloud Service Contract
Product: application server + DMBS platforms
SLA: availability=99.5%, auto-scaling
Price: $0.45 per hour (500,000 requests)

Figure 2: A cloud consumer is accessing a ready-made PaaS environment. The question mark indicates that the cloud consumer is intentionally shielded from the implementation details of the platform.

Benefits:

1. Lower administrative overhead:

Consumer need not to bother much about the administrative because its responsibility of cloud provider.

2. Lower total cost of ownership:

Consumer need not purchase expensive hardware, servers, power and data storage.

3. Scalable Solution:

It is easy to scale up and down automatically based upon application resources

Issues:

1. Lack of portability between PaaS clouds
2. Event based processor scheduling
3. Security engineering of PaaS application

9.6.3.3 Infrastructure as a service (IaaS)

- The IaaS delivery model represents a self-contained IT environment comprised of infrastructure-centric IT resources that can be accessed and managed via cloud service-based interfaces and tools.
- This environment can include hardware, network, connectivity, operating systems And other “raw” IT resources.
- The general purpose of an IaaS environment is to provide cloud consumer with high level of control and responsibility over its configuration and utilization.
- The IT resources provided by IaaS are generally not pre-configured, placing administrative responsibility directly upon the cloud consumer
- This model is therefore used by cloud consumers that require a high level of control over the cloud-based environment they intend to create.

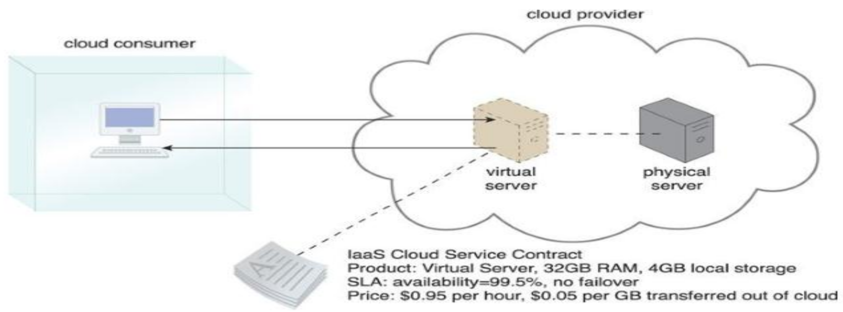

cloud provider
cloud consumer
virtual server
physical server
IaaS Cloud Service Contract
Product: Virtual Server, 32GB RAM, 4GB local storage
SLA: availability=99.5%, no failover
Price: $0.95 per hour, $0.05 per GB transferred out of cloud

Figure 3: A cloud consumer is using a virtual server within an IaaS environment. Cloud consumers are provided with a range of contractual guarantees by The cloud provider, pertaining to characteristics such as capacity, performance, and availability.

Benefits:

Some of the key benefits of IaaS are listed below:

1. **Full control of computing resources through administrative access to VMs:**

- It allows the consumer to access computing resources through administrative access to virtual machine in the following manner
- Consumer issues administrative command to cloud provide to run the virtual machine or to save data on cloud server.
- Consumer issues administrative command to virtual machines they owned to start web or installing new application.

2. **Flexible and efficient renting of computer hardware**

- IaaS resources such as virtual machine, storage, bandwidth, IP address, monitoring services, firewall all are made available to the consumer on rent.

3. **Portability, interoperability with legacy application**

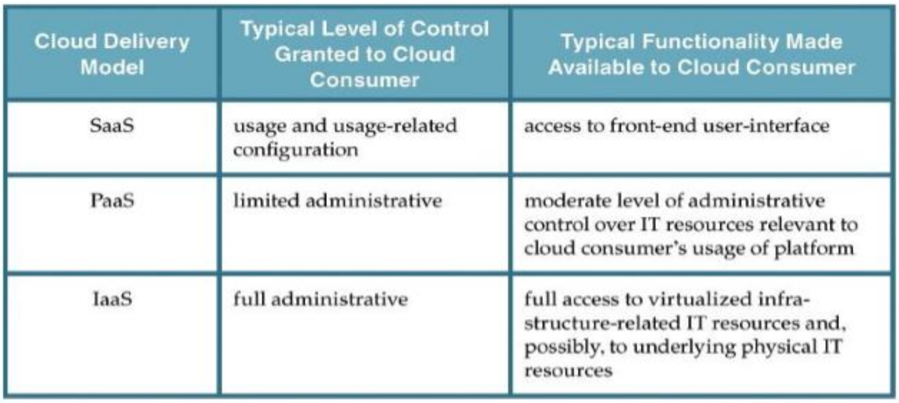
- It is possible to maintain legacy between application and workloads between IaaS cloud.

Issues:

1. Compatibility with legacy security vulnerability
2. Virtual machine Sprawl
3. Robustness of VM level isolation
4. Data erase practices

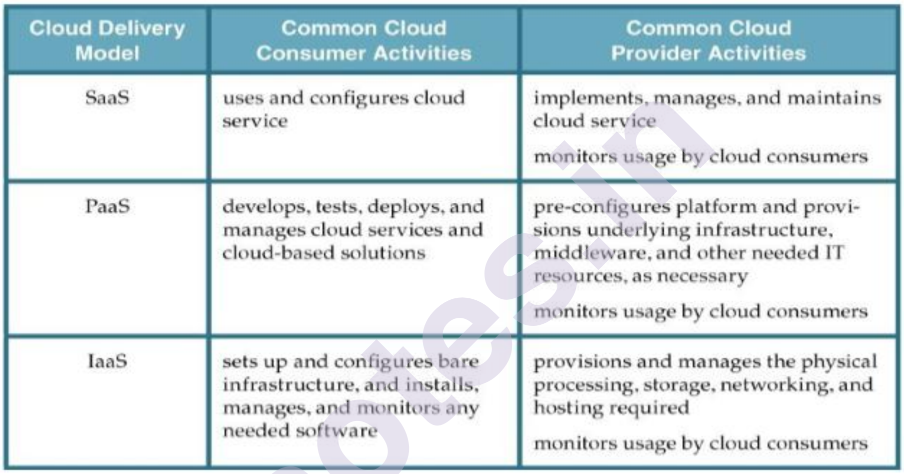
Comparing Cloud Delivery Models:

A comparison of typical cloud delivery model control levels



Cloud Delivery Model	Typical Level of Control Granted to Cloud Consumer	Typical Functionality Made Available to Cloud Consumer
SaaS	usage and usage-related configuration	access to front-end user-interface
PaaS	limited administrative	moderate level of administrative control over IT resources relevant to cloud consumer's usage of platform
IaaS	full administrative	full access to virtualized infrastructure-related IT resources and, possibly, to underlying physical IT resources

Typical activities carried out by cloud consumers and cloud providers in relation to the cloud delivery models.



Cloud Delivery Model	Common Cloud Consumer Activities	Common Cloud Provider Activities
SaaS	uses and configures cloud service	implements, manages, and maintains cloud service monitors usage by cloud consumers
PaaS	develops, tests, deploys, and manages cloud services and cloud-based solutions	pre-configures platform and provisions underlying infrastructure, middleware, and other needed IT resources, as necessary monitors usage by cloud consumers
IaaS	sets up and configures bare infrastructure, and installs, manages, and monitors any needed software	provisions and manages the physical processing, storage, networking, and hosting required monitors usage by cloud consumers

9.7 SUMMARY

- Cloud environments are comprised of highly extensive infrastructure that Offers pools of IT resources that can be leased using a pay-for-use model whereby Only the actual usage of the IT resources is billable.

- Cloud environments can introduce distinct security challenges, some of which pertain to overlapping trust boundaries imposed by a cloud provider sharing IT resources with multiple cloud consumers.

- A cloud consumer's operational governance can be limited within cloud environments due to the control exercised by a cloud provider over its platforms.

- The portability of cloud-based IT resources can be inhibited by dependencies upon proprietary characteristics imposed by a cloud.

- The geographical location of data and IT resources can be out of a cloud consumer's control when hosted by a third-party cloud provider. This can introduce various legal and regulatory compliance concerns.

- The IaaS cloud delivery model offers cloud consumers a high level of administrative control over "raw" infrastructure-based IT resources.
- The PaaS cloud delivery model enables a cloud provider to offer a preconfigured environment that cloud consumers can use to build and deploy cloud services and solutions, albeit with decreased administrative control.
- SaaS is a cloud delivery model for shared cloud services that can be positioned as commercialized products hosted by clouds.
- Different combinations of IaaS, PaaS, and SaaS are possible, depending on how cloud consumers and cloud providers choose to leverage the natural hierarchy established by these base cloud delivery models.
- A public cloud is owned by a third party and generally offers commercialized cloud services and IT resources to cloud consumer organizations.
- A private cloud is owned by an individual organization and resides within the organization'spremises.
- A community cloud is normally limited for access by a group of cloud consumers that may also share responsibility in its ownership.
- A hybrid cloud is a combination of two or more other cloud deployment models.

Self-Learning Topics: Cluster computing, Grid computing, Fog computing

9.8 REFERENCES-

- Cloud Computing Concepts, Technology & Architecture by Thomas Erl, Zaigham Mahmood, and Ricardo Puttini
- James Broberg and Andrzej M. Goscinski, Cloud Computing: Principles and Paradigms Wiley, First edition, ISBN No. 978-04-708-8799-8
- Rajkumar Buyya, Christian Vecchiola, S. ThamaraiSelvi, Mastering Cloud Computing, Tata Mcgraw Hill, ISBN No. 978-12-590-2995-0

*****

UNIT VI

10

CLOUD COMPUTING

Unit Structure

10.0 OBJECTIVE

The main motivation behind cloud computing is to enable businesses to get access to data centres and manage tasks from a remote location. Cloud computing works on the pay-as-you-go pricing model, which helps businesses lower their operating cost and run infrastructure more efficiently.

What is cloud computing?:

Cloud computing is a general term for anything that involves delivering hosted services over the internet. These services are divided into three main categories or types of cloud computing: infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS).

A cloud can be private or public. A public cloud sells services to anyone on the internet. A private cloud is a proprietary network or a data center that supplies hosted services to a limited number of people, with certain access and permissions settings. Private or public, the goal of cloud computing is to provide easy, scalable access to computing resources and IT services.

Cloud infrastructure involves the hardware and software components required for proper implementation of a cloud computing model. Cloud computing can also be thought of as utility computing or on-demand computing.

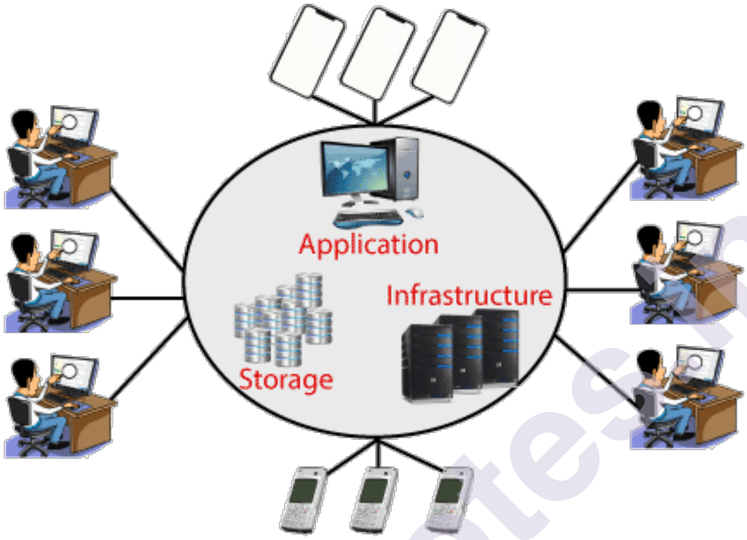
The name cloud computing was inspired by the cloud symbol that's often used to represent the internet in flowcharts and diagrams.

How does cloud computing work?:

Cloud computing works by enabling client devices to access data and cloud applications over the internet from remote physical servers, databases and computers.

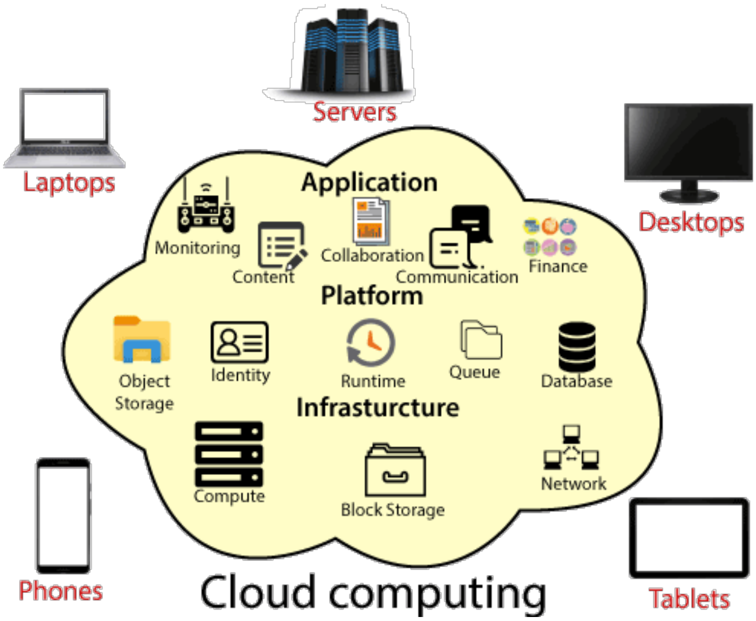
10.1 INTRODUCTION

Cloud Computing is the delivery of computing services such as servers, storage, databases, networking, software, analytics, intelligence, and more, over the Cloud (Internet).



Cloud Computing provides an alternative to the on-premises datacentre. With an on-premises datacentre, we have to manage everything, such as purchasing and installing hardware, virtualization, installing the operating system, and any other required applications, setting up the network, configuring the firewall, and setting up storage for data. After doing all the set-up, we become responsible for maintaining it through its entire lifecycle.

But if we choose Cloud Computing, a cloud vendor is responsible for the hardware purchase and maintenance. They also provide a wide variety of software and platform as a service. We can take any required services on rent. The cloud computing services will be charged based on usage.



The cloud environment provides an easily accessible online portal that makes handy for the user to manage the compute, storage, network, and application resources. Some cloud service providers are in the following figure.

rackspace
Joyent
DigitalOcean
Microsoft Azure
terremark
vmware
Google Cloud Platform
amazon web services
IBM Cloud

Advantages of cloud computing:

- **Cost:** It reduces the huge capital costs of buying hardware and software.
- **Speed:** Resources can be accessed in minutes, typically within a few clicks.

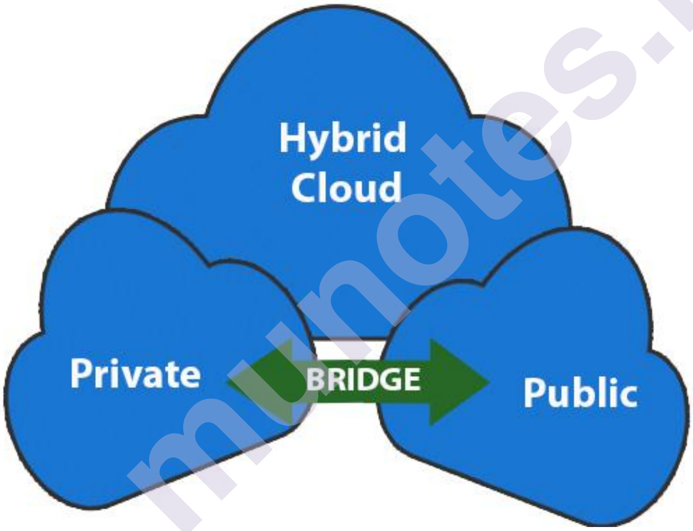
- **Scalability:** We can increase or decrease the requirement of resources according to the business requirements.

- **Productivity:** While using cloud computing, we put less operational effort. We do not need to apply patching, as well as no need to maintain hardware and software. So, in this way, the IT team can be more productive and focus on achieving business goals.

- **Reliability:** Backup and recovery of data are less expensive and very fast for business continuity.

- **Security:** Many cloud vendors offer a broad set of policies, technologies, and controls that strengthen our data security.

Types of Cloud Computing:

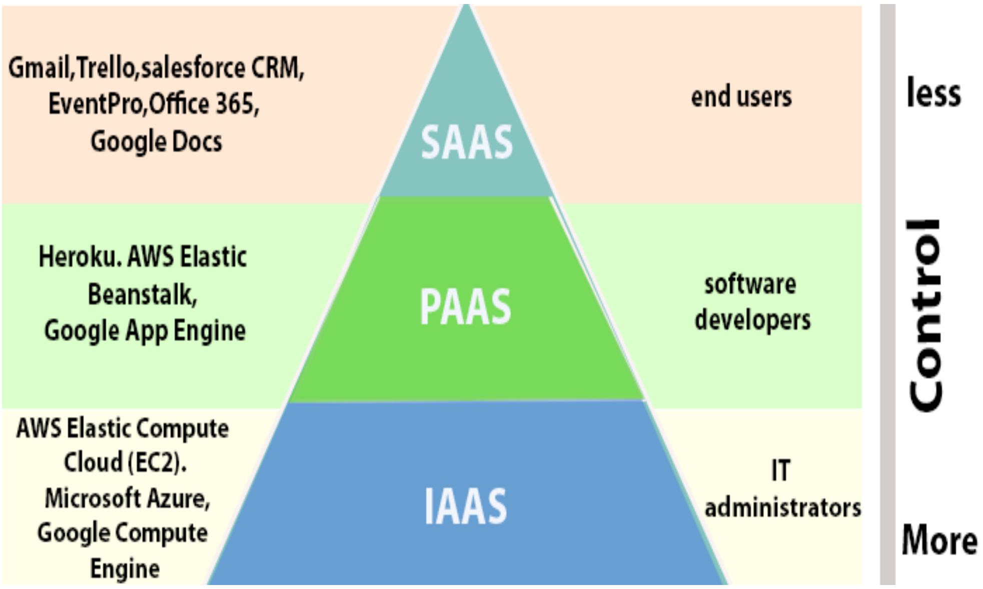


- **Public Cloud:** The cloud resources that are owned and operated by a third-party cloud service provider are termed as public clouds. It delivers computing resources such as servers, software, and storage over the internet

- **Private Cloud:** The cloud computing resources that are exclusively used inside a single business or organization are termed as a private cloud. A private cloud may physically be located on the company's on-site datacentre or hosted by a third-party service provider.

- **Hybrid Cloud:** It is the combination of public and private clouds, which is bounded together by technology that allows data applications to be shared between them. Hybrid cloud provides flexibility and more deployment options to the business.

Types of Cloud Services:


Gmail,Trello,salesforce CRM, EventPro,Office 365, Google Docs
SAAS
end users
less
Heroku. AWS Elastic Beanstalk, Google App Engine
PAAS
software developers
Control
AWS Elastic Compute Cloud (EC2). Microsoft Azure, Google Compute Engine
IAAS
IT administrators
More

1. Infrastructure as a Service (IaaS):

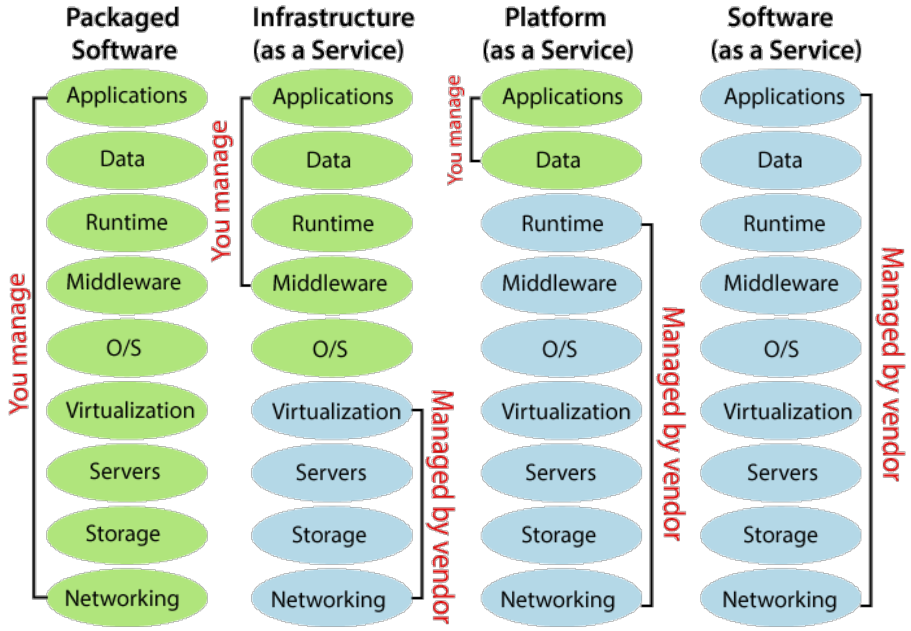
In IaaS, we can rent IT infrastructures like servers and virtual machines (VMs), storage, networks, operating systems from a cloud service vendor. We can create VM running Windows or Linux and install anything we want on it. Using IaaS, we don't need to care about the hardware or virtualization software, but other than that, we do have to manage everything else. Using IaaS, we get maximum flexibility, but still, we need to put more effort into maintenance.

2. Platform as a Service (PaaS):

This service provides an on-demand environment for developing, testing, delivering, and managing software applications. The developer is responsible for the application, and the PaaS vendor provides the ability to deploy and run it. Using PaaS, the flexibility gets reduce, but the management of the environment is taken care of by the cloud vendors.

3. Software as a Service (SaaS):

It provides a centrally hosted and managed software services to the end-users. It delivers software over the internet, on-demand, and typically on a subscription basis. E.g., Microsoft One Drive, Dropbox, WordPress, Office 365, and Amazon Kindle. SaaS is used to minimize the operational cost to the maximum extent.



10.2 AMAZON WEB SERVICES

AWS stands for Amazon Web Services which uses distributed IT infrastructure to provide different IT resources on demand.

Our AWS tutorial includes all the topics such as introduction, history of aws, global infrastructure, features of AWS, IAM, Storage services, Database services, etc.

What are AWS?:

- AWS stands for Amazon Web Services.
- The AWS service is provided by the Amazon that uses distributed IT infrastructure to provide different IT resources available on demand. It provides different services such as infrastructure as a service (IaaS), platform as a service (PaaS) and packaged software as a service (SaaS).
- Amazon launched AWS, a cloud computing platform to allow the different organizations to take advantage of reliable IT infrastructure.

Uses of AWS:

- A small manufacturing organization uses their expertise to expand their business by leaving their IT management to the AWS.
- A large enterprise spread across the globe can utilize the AWS to deliver the training to the distributed workforce.
- An architecture consulting company can use AWS to get the high-compute rendering of construction prototype.

- A media company can use the AWS to provide different types of content such as ebox or audio files to the worldwide files.

Pay-As-You-Go:

Based on the concept of Pay-As-You-Go, AWS provides the services to the customers.

AWS provides services to customers when required without any prior commitment or upfront investment. Pay-As-You-Go enables the customers to procure services from AWS.

- Computing
- Programming models
- Database storage
- Networking

Advantages of AWS:

1) Flexibility:

- We can get more time for core business tasks due to the instant availability of new features and services in AWS.
- It provides effortless hosting of legacy applications. AWS does not require learning new technologies and migration of applications to the AWS provides the advanced computing and efficient storage.
- AWS also offers a choice that whether we want to run the applications and services together or not. We can also choose to run a part of the IT infrastructure in AWS and the remaining part in data centres.

2) Cost-effectiveness:

AWS requires no upfront investment, long-term commitment, and minimum expense when compared to traditional IT infrastructure that requires a huge investment.

3) Scalability/Elasticity:

Through AWS, autoscaling and elastic load balancing techniques are automatically scaled up or down, when demand increases or decreases respectively. AWS techniques are ideal for handling unpredictable or very high loads. Due to this reason, organizations enjoy the benefits of reduced cost and increased user satisfaction.

4) Security:

- AWS provides end-to-end security and privacy to customers.
- AWS has a virtual infrastructure that offers optimum availability while managing full privacy and isolation of their operations.

- Customers can expect high-level of physical security because of Amazon's several years of experience in designing, developing and maintaining large-scale IT operation centers.
- AWS ensures the three aspects of security, i.e., Confidentiality, integrity, and availability of user's data.

10.3 MICROSOFT AZURE AND GOOGLE CLOUD

What is Azure:

Microsoft Azure is a growing set of cloud computing services created by Microsoft that hosts your existing applications, streamline the development of a new application, and also enhances our on-premises applications. It helps the organizations in building, testing, deploying, and managing applications and services through Microsoft-managed data centers.

Azure Services:

- **Compute services:** It includes the Microsoft Azure Cloud Services, Azure Virtual Machines, Azure Website, and Azure Mobile Services, which processes the data on the cloud with the help of powerful processors.
- **Data services:** This service is used to store data over the cloud that can be scaled according to the requirements. It includes Microsoft Azure Storage (Blob, Queue Table, and Azure File services), Azure SQL Database, and the Redis Cache.
- **Application services:** It includes services, which help us to build and operate our application, like the Azure Active Directory, Service Bus for connecting distributed systems, HDInsight for processing big data, the Azure Scheduler, and the Azure Media Services.
- **Network services:** It helps you to connect with the cloud and on-premises infrastructure, which includes Virtual Networks, Azure Content Delivery Network, and the Azure Traffic Manager.

Why Should You Choose Microsoft Azure Services?:

While every business can have their individual reasons to choose Microsoft Azure services, there are several unbeatable advantages that all can leverage upon, like:

- Microsoft Azure is capable of providing an enticing combination of Infrastructure as a platform (IaaS) and Platform as a service (PaaS) that helps enterprises create and deploy their own web apps without hustling over the groundwork.
- Microsoft Azure has designed Security Development Lifecycle, an industry-standard security process that considers all security features

including getting licenses and ensuring the best safety in all operations.

- There is a vast user base already on Microsoft Azure, but the infrastructure is constantly scaling up, by using more processes for applications and selling storage through the cloud. It can run without any additional coding.

- A hybrid cloud computing ecosystem is still a unique feature of Microsoft Azure. It can improve the performance by utilizing Virtual Private Networks (VPNs), ExpressRoute connections, caches, CDNs, etc.

- As most enterprises rely on MS office tools, it is wise to invest in a cloud platform that integrates well with all Microsoft products. Additionally, knowing C++, C, and Visual Basic can help you steer your career in Microsoft Azure. If you require further validation, then you can try out the Azure certification courses for Windows certificates.

- Microsoft Azure has intelligence and analytics capacities to improve the business process with the help of machine learning bots, cognitive APIs, and Blockchain as a Service (BaaS).

- Microsoft Azure also has SQL and noSQL data processing facilities to get deeper and actionable insights from the available data.

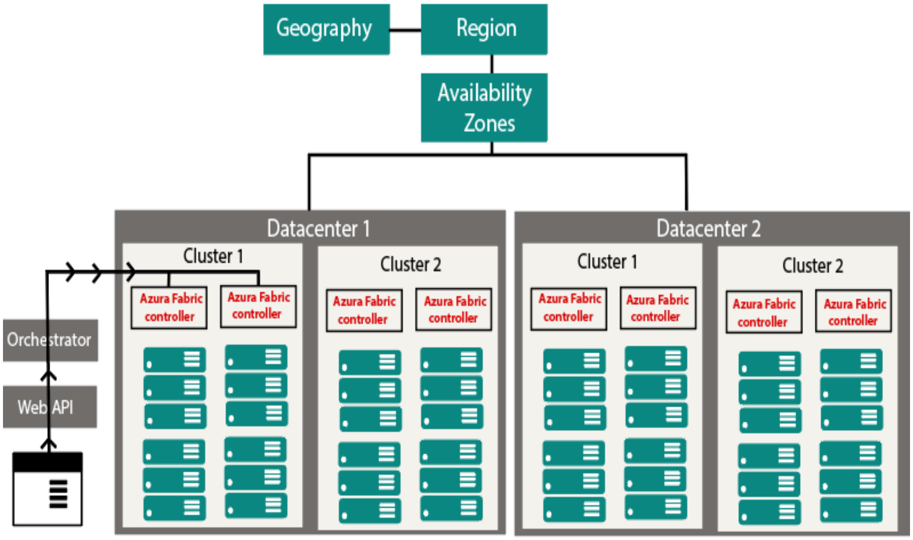
- One of the major reasons to choose Azure services is the affordability, as the virtual infrastructure maintenance is extremely cost-efficient.

Curated List of Top Azure Services:

While there is no long list of competitors in cloud servicing, the top runners like Google and AWS continue to give a tough fight to Microsoft Azure in the race of being the most used cloud service. Despite intense competition, Microsoft Azure continues growing and evolving over the years, especially through the phase of remote working due to a pandemic in 2020 and 2021. Offering top Azure services, the platform has maintained its integrity and popularity. Now let's delve deeper into understanding more about the top 10 most popular Azure services.

How Azure works:

It is essential to understand the internal workings of Azure so that we can design our applications on Azure effectively with high availability, data residency, resilience, etc.



Microsoft Azure is completely based on the concept of virtualization. So, similar to other virtualized data center, it also contains racks. Each rack has a separate power unit and network switch, and also each rack is integrated with a software called Fabric-Controller. This Fabric-controller is a distributed application, which is responsible for managing and monitoring servers within the rack. In case of any server failure, the Fabric-controller recognizes it and recovers it. And Each of these Fabric-Controller is, in turn, connected to a piece of software called Orchestrator. This Orchestrator includes web-services, Rest API to create, update, and delete resources.

When a request is made by the user either using PowerShell or Azure portal. First, it will go to the Orchestrator, where it will fundamentally do three things:

1. Authenticate the User
2. It will Authorize the user, i.e., it will check whether the user is allowed to do the requested task.
3. It will look into the database for the availability of space based on the resources and pass the request to an appropriate Azure Fabric controller to execute the request.

Combinations of racks form a cluster. We have multiple clusters within a data center, and we can have multiple Data Centers within an Availability zone, multiple Availability zones within a Region, and multiple Regions within a Geography.

- **Geographies:** It is a discrete market, typically contains two or more regions, that preserves data residency and compliance boundaries.
- **Azure regions:** A region is a collection of data centers deployed within a defined perimeter and interconnected through a dedicated regional low-latency network.

Azure covers more global regions than any other cloud provider, which offers the scalability needed to bring applications and users closer around the world. It is globally available in 50 regions around the world. Due to its availability over many regions, it helps in preserving data residency and offers comprehensive compliance and flexible options to the customers.

10.3.1 Compute services:

Azure compute services are the hosting services responsible for hosting and running the application workloads. These include Azure Virtual Machines (VMs), Azure Container Service, Azure App Services, Azure Batch, and Azure ServiceFabric.

Azure Compute Services: Azure Virtual Machines (VMs) and Azure Container Service:

Azure Virtual Machines (VMs):

A Microsoft Azure Virtual Machine (VM) is an on-demand, scalable computing resource. You don't need to buy any physical hardware and bear its maintenance cost; you have the flexibility of virtualization. Your cloud administrators only need to select the operating system, configure the required resources, and create the web server – all this gets done within a few minutes.

Azure Container Service:

Azure helps you leverage the modern container-based development practices and microservices architecture. You can migrate your .NET applications to microservices using Windows Server containers with Azure Service Fabric. Further, you can use Azure Kubernetes Service to scale and orchestrate Linux Containers.

You can choose between Docker Hub and Azure Container Registry to store your images and deploy to any preferred target. Moreover, it simplifies the configuration process and optimizes it for the cloud. The major advantage is that it consumes less space as compared to VMs and starts instantly; hence speeding up the processes.

10.3.2 Storage services:

The Azure Storage platform is Microsoft's cloud storage solution for modern data storage scenarios. Azure Storage offers highly available, massively scalable, durable, and secure storage for a variety of data objects in the cloud. Azure Storage data objects are accessible from anywhere in the world over HTTP or HTTPS via a REST API. Azure Storage also offers client libraries for developers building applications or services with .NET, Java, Python, JavaScript, C++, and Go. Developers and IT professionals can use Azure PowerShell and Azure CLI to write scripts for data management or configuration tasks. The Azure portal and Azure Storage Explorer provide user-interface tools for interacting with Azure Storage.

Benefits of Azure Storage:

Azure Storage services offer the following benefits for application developers and IT professionals:

- **Durable and highly available:** Redundancy ensures that your data is safe in the event of transient hardware failures. You can also opt to replicate data across data centers or geographical regions for additional protection from local catastrophe or natural disaster. Data replicated in this way remains highly available in the event of an unexpected outage.
- **Secure:** All data written to an Azure storage account is encrypted by the service. Azure Storage provides you with fine-grained control over who has access to your data.
- **Scalable:** Azure Storage is designed to be massively scalable to meet the data storage and performance needs of today's applications.
- **Managed:** Azure handles hardware maintenance, updates, and critical issues for you.
- **Accessible**: Data in Azure Storage is accessible from anywhere in the world over HTTP or HTTPS. Microsoft provides client libraries for Azure Storage in a variety of languages, including .NET, Java, Node.js, Python, PHP, Ruby, Go, and others, as well as a mature REST API. Azure Storage supports scripting in Azure PowerShell or Azure CLI. And the Azure portal and Azure Storage Explorer offer easy visual solutions for working with your data.

Azure Storage data services:

The Azure Storage platform includes the following data services:

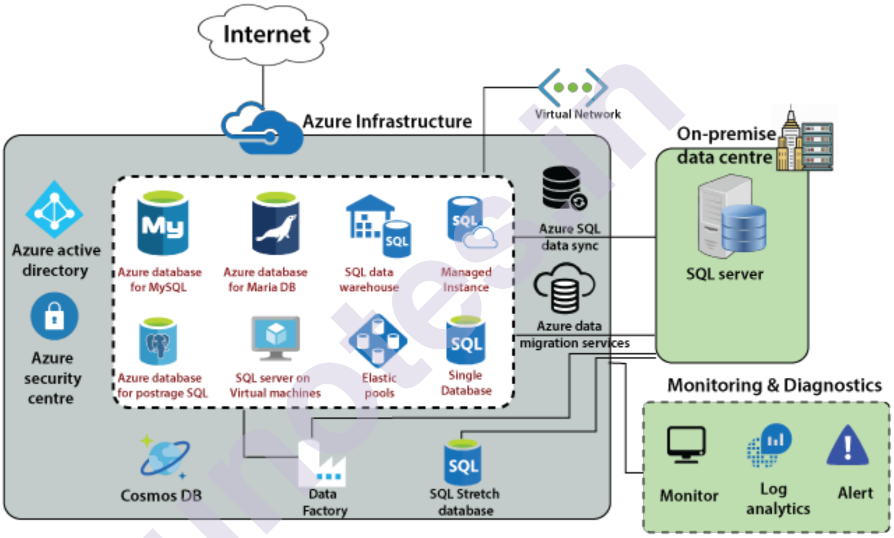
- **Azure Blobs:** A massively scalable object store for text and binary data. Also includes support for big data analytics through Data Lake Storage Gen2.
- **Azure Files:** Managed file shares for cloud or on-premises deployments.
- **Azure Queues:** A messaging store for reliable messaging between application components.
- **Azure Tables:** A NoSQL store for schema-less storage of structured data.
- **Azure Disks:** Block-level storage volumes for Azure VMs.

10.3.3 Database services:

The basic fundamental building block that is available in Azure is the SQL database. Microsoft offers this SQL server and SQL database on Azure in many ways. We can deploy a single database, or we can deploy multiple databases as part of a shared elastic pool.

Azure Database Service Architecture:

Microsoft introduced a managed instance that is targeted towards on-premises customers. So, if we have some SQL databases within our on-premises datacentre and we want to migrate that database into Azure without any complex configuration, or ambiguity, then we can use managed instance. Because this is mainly targeted towards on-premises customers who want to lift and share their on-premises database into Azure with the least effort and optimized cost. We can also take advantage of licensing we have within our on-premises data center.



Microsoft will be responsible for maintenance patching and related services. But, in case if we want to go for the IaaS service for the SQL server, then we can deploy SQL Server on the Azure Virtual machine. If the data have a dependency on the underlying platform and we want to log into the SQL Server, in that case, we can use the SQL server on a virtual machine.

We can deploy a SQL data warehouse on the cloud. Azure offers many other database services for different types of databases such as MySQL, Maria DB, and also PostgreSQL. Once we deployed a database into Azure, we need to migrate the data into it or replicate the data into it.

Azure Database Services for Data Migration:

The services that are available in Azure, which we can use to migrate the data from our on-premises SQL Server into Azure.

Azure Data Migration Service: It is used to migrate the data from our existing SQL server and database within the on-premises data center into Azure.

Azure SQL data sync: If we want to replicate the data from our on-premises database into Azure, then we can use Azure SQL data sync.

SQL Stretch Database: It is used to migrate cold data into Azure. SQL stretch database is a bit different from other database offerings. It works as a hybrid database because it divides the data into different types - hot and cold. A hot data will be kept in the on-premises data center and cold data in the Azure.

Data Factory:

It is used for ETL transformation, extraction loading, etc. Using the data factory, we can even extract the data from our on-premises data center. We can do some conversion and load it into the Azure SQL database. Data Factory is an ETL tool that is offered on the cloud, which we can use to connect to different databases, extract the data, transform it, and load into a destination.

Security:

All the databases that are existing in Azure need to be secured, and also we need to accept connections from known origins. For this purpose, all these database services come with firewall rules where we can configure from which particular IP address we want to allow connections. We can define those firewall rules to limit the number of connections and also reduce the surface attack area.

Cosmos DB:

Cosmos DB is a NoSQL data store that is available in Azure, and it is designed to be globally scalable and also very highly available with extremely low latency. Microsoft guarantees latency in terms of reading and writes with Cosmos DB. For example - if we have any applications such as IoT, gaming where we get a lot of data from different users spread across globally, then we will go for Cosmos DB. Because Cosmos DB is designed to be globally scalable and highly available due to which our users will experience low latency.

Finally, there are two things, and one is we need to secure all the services. For that purpose, we can integrate all these services with Azure Active Directory and manage the users from Azure Active Directory also. To monitor all these services, we can use the security center. There is an individual monitoring tool too, but Azure security center will keep on monitoring all these services and provide recommendations if something is wrong.

Cosmos DB:

Cosmos DB is a NoSQL data store that is available in Azure, and it is designed to be globally scalable and also very highly available with extremely low latency. Microsoft guarantees latency in terms of reading and writes with Cosmos DB. For example - if we have any applications such as IoT, gaming where we get a lot of data from different users spread

across globally, then we will go for Cosmos DB. Because Cosmos DB is designed to be globally scalable and highly available due to which our users will experience low latency.

Finally, there are two things, and one is we need to secure all the services. For that purpose, we can integrate all these services with Azure Active Directory and manage the users from Azure Active Directory also. To monitor all these services, we can use the security center. There is an individual monitoring tool too, but Azure security center will keep on monitoring all these services and provide recommendations if something is wrong.

10.3.4 Additional services:

Azure data services store and manage data on cloud. Microsoft Azure comes with a range of data services: Azure Storage, Azure SQL Database, Azure Document DB, Azure StorSimple, and Azure Redis Cache.

Below is an overview of some most popular Microsoft Azure services, specifically the top 10 azure services and the way you can use them across the entire architecture:

- Azure DevOps
- Azure Blob Storage
- Azure Virtual Machines
- Azure Backup
- Azure Cosmos DB
- Azure Logic Apps
- Azure Active Directory
- API management
- Azure Content Delivery Network
- Azure Site Recovery
- Azure Bots

*****

11

CLOUD PLATFORMS

Unit Structure

11.0 OBJECTIVE

A **computer platform** is a system that consists of a hardware device and an operating system that an application, program or process runs upon. An example of a computer platform is a desktop computer with Microsoft Windows installed on it. A desktop is a hardware device and Windows is an operating system.

The operating system acts as an interface between the computer and the user and also between the computer and the application. So, in order to have a functional device, you need hardware and an operating system together to make a usable computer platform for a program to run on.

The hardware portion of a computer platform consists of a processor, memory, and storage. The processor is a bit like your brain and memory is like a scratchpad for your brain to use while you're working out a problem.

It used to be that people referred to different computer platforms by their physical size, from smallest to largest - microcomputers (smallest), minicomputers (mid-size), and mainframes (largest). The term microcomputer has fallen somewhat out of favor - now most people just refer to these machines as computers or personal computers.

11.1 INTRODUCTION

Cloud computing is the delivery of computing services—including servers, storage, databases, networking, software, analytics, and intelligence—over the Internet ("the cloud") to offer faster innovation, flexible resources, and economies of scale.

How does cloud computing work?:

Rather than owning their own computing infrastructure or data centres, companies can rent access to anything from applications to storage from a cloud service provider.

One benefit of using cloud-computing services is that firms can avoid the upfront cost and complexity of owning and maintaining their own IT infrastructure, and instead simply pay for what they use, when they use it.

In turn, providers of cloud-computing services can benefit from significant economies of scale by delivering the same services to a wide range of customers.

What cloud-computing services are available?:

Cloud-computing services cover a vast range of options now, from the basics of storage, networking and processing power, through to natural language processing and artificial intelligence as well as standard office applications. Pretty much any service that doesn't require you to be physically close to the computer hardware that you are using can now be delivered via the cloud – even quantum computing.

What are examples of cloud computing?:

Cloud computing underpins a vast number of services. That includes consumer services like Gmail or the cloud backup of the photos on your smartphone, though to the services that allow large enterprises to host all their data and run all of their applications in the cloud. For example, Netflix relies on cloud-computing services to run its its video-streaming service and its other business systems, too.

Cloud computing is becoming the default option for many apps: software vendors are increasingly offering their applications as services over the internet rather than standalone products as they try to switch to a subscription model. However, there are potential downsides to cloud computing, in that it can also introduce new costs and new risks for companies using it.

Why is it called cloud computing?:

A fundamental concept behind cloud computing is that the location of the service, and many of the details such as the hardware or operating system on which it is running, are largely irrelevant to the user. It's with this in mind that the metaphor of the cloud was borrowed from old telecoms network schematics, in which the public telephone network (and later the internet) was often represented as a cloud to denote that the location didn't matter – it was just a cloud of stuff. This is an over-simplification of course; for many customers, location of their services and data remains a key issue.

What is the history of cloud computing?:

Cloud computing as a term has been around since the early 2000s, but the concept of computing as a service has been around for much, much longer – as far back as the 1960s, when computer bureaus would allow companies to rent time on a mainframe, rather than have to buy one themselves.

These 'time-sharing' services were largely overtaken by the rise of the PC, which made owning a computer much more affordable, and then in turn by the rise of corporate data centres where companies would store vast amounts of data.

But the concept of renting access to computing power has resurfaced again and again – in the application service providers, utility computing, and grid computing of the late 1990s and early 2000s. This was followed by cloud computing, which really took hold with the emergence of software as a service and hyperscale cloud-computing providers such as Amazon Web Services.

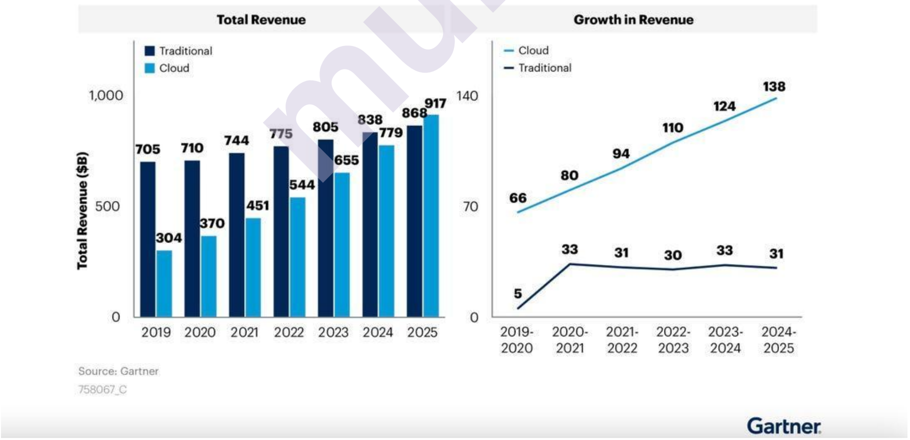
How important is the cloud?:

Building the infrastructure to support cloud computing now accounts for a significant chunk of all IT spending, while spending on traditional, in-house IT slides as computing workloads continue to move to the cloud, whether that is public cloud services offered by vendors or private clouds built by enterprises themselves.

Indeed, it's increasingly clear that when it comes to enterprise computing platforms, like it or not, the cloud has won.

Tech analyst Gartner predicts that as much as half of spending across application software, infrastructure software, business process services and system infrastructure markets will have shifted to the cloud by 2025, up from 41% in 2022. It estimates that almost two-thirds of spending on application software will be via cloud computing, up from 57.7% in 2022.

Figure 1: Sizing Cloud Shift, Worldwide, 2019 – 2025



Source: Gartner
758067_C

11.2 GOOGLE APP ENGINE (GAE)

Google App Engine lets you run (host) your own Web applications on Google's infrastructure. However, by no means is this a "rent a piece of a server" hosting service. With App Engine, your application is not hosted on a single server. There are no servers to maintain: You just upload your application, and it's ready to serve your users. Just as servicing a Google search request may involve dozens, or even hundreds of Google servers, all totally hidden and satisfied in a fraction of a second, Google App Engine applications run the same way, on the same infrastructure. This is the unique aspect of Google's approach. Yes, you cede some control to Google, but you are rewarded by being totally free of the infrastructure, capacity management, and load balancing tasks that enterprise typically have to manage, irrespective of whether they are self-hosting or hosting on someone else's PaaS or IaaS.

You can choose to share your application with the world, or limit access to members of your organization. Google App Engine supports apps written in several programming languages:

With App Engine's Java runtime environment, you can build your app using standard Java technologies, including the JVM, Java servlets, and the Java programming language—or any other language using a JVM-based interpreter or compiler, such as JavaScript or Ruby. App Engine also features a dedicated Python runtime environment, which includes a fast Python interpreter and the Python standard library. The Java and Python runtime environments are built to ensure that your application runs quickly, securely, and without interference from other apps on the system.

As with most cloud-hosting services, with App Engine, you only pay for what you use. Google levies no set-up costs and no recurring fees. Similar to Amazon's AWS, resources such as storage and bandwidth are measured by the gigabyte.

App Engine costs nothing to get started. All applications can use up to 500 MB of storage and enough CPU and bandwidth to support an efficient app serving around 5 million page views a month, absolutely free. When you enable billing for your application, your free limits are raised, and you only pay for resources you use above the free levels.

Application developers have access to persistent storage technologies such as the Google File System (GFS) and Bigtable, a distributed storage system for unstructured data. The Java version supports asynchronous nonblocking queries using the Twig Object Datastore interface. This offers an alternative to using threads for parallel data processing.

"With Google App Engine, developers can write Web applications based on the same building blocks that Google uses," Kevin Gibbs, Google's technical lead for the project, wrote in The Official Google Blog"Google. Twig is an object persistence interface built on Google App Engine's low-level datastore which overcomes many of JDO-GAEs limitations, including full support for inheritance, polymorphism, and generic types. You can easily configure, modify or extend Twig's behavior by implementing your own strategies or overriding extension points in pure Java code. App Engine packages those building blocks and provides access to scalable infrastructure that we hope will make it easier for developers to scale their applications automatically as they grow."

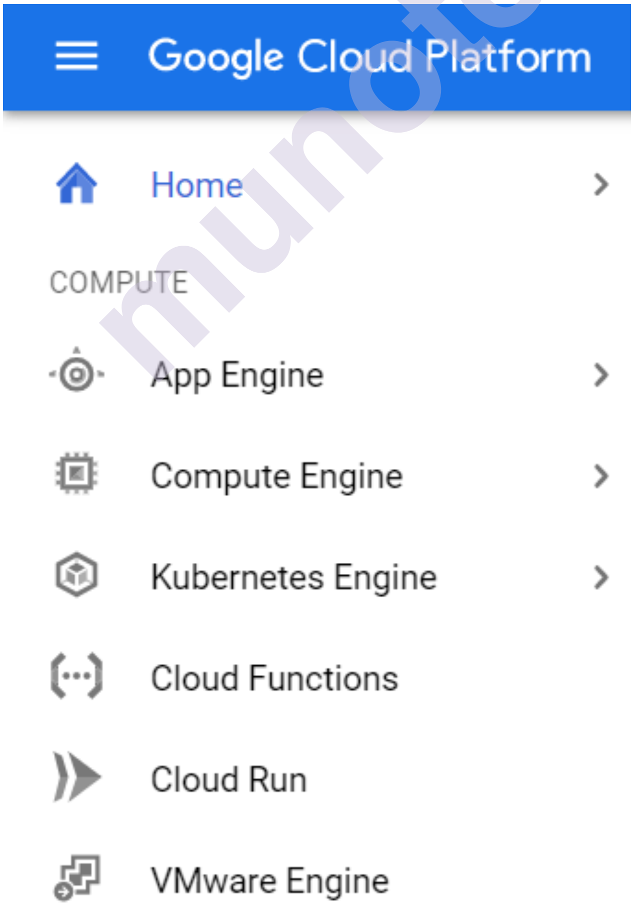
Google App Engine has appeared at a time when an increasing number of tech companies are moving their operations to the cloud; it places Google squarely in competition with Amazon's Elastic Cloud Computing (EC2) and Simple Storage Service (S3) offerings.

Google says its vision with Google App Engine is to offer developers a more holistic, end-to-end solution for building and scaling applications online. Its servers are configured to balance the load of traffic to developers' applications, scaling to meet the demand of an influx of traffic. App Engine also includes APIs for user authentication to allow developers to sign on for services, and for e-mail, to manage

Google Cloud Platform provides a wide range of computing services that target broad categories of user needs.

The Google Cloud Platform provides mainly 6 types of compute options:

1. App Engine
2. Compute Engine
3. Kubernetes Engine
4. Cloud Functions
5. Cloud Run
6. VMware Engine



Now let's talk about some of these services in brief.

Compute Engine:

The Compute Engine service is Google's unmanaged compute service. We can think of Compute Engine as an Infrastructure as a Service (IaaS) offering by Google Cloud. As the service is unmanaged, it is our responsibility to configure, administer, and monitor the system. On Google's side, they will ensure that resources are available, reliable, and ready for you to use. The main benefit in using compute engine is that you have complete control of the systems.

You can do the following when you build on Compute Engine:

- Create Virtual Instances, which is the smallest unit in the GCP project.
- Create instance groups to easily manage multiple instances together.
- Create virtual machine images.

The virtual machine instances running in zones assigned to them. Zones are data center-like resources. They are located within regions which is a geographical location. The zones are within a region are linked with low-latency and high bandwidth network connections.

Pros of Compute Engine:

- It offers the users complete control over the Virtual Machine instances.
- It is easy to set up, you can spin up a server within few minutes.
- The use of preemptive VM's can reduce the cost by up to 80%.
- Set of predefined VM configurations and VM images are available ready to be used according to needs.

Cons of Compute Engine:

- Requires high expertise level, since everything needs to be installed and configured by yourself.
- Autoscaling is slower than App Engine.
- To enable monitoring, you need to install packages into the VM instances. No direct Stackdriver monitoring is possible.

App Engine:

The App Engine is Google's Platform as a Service(PaaS) offering. It is a compute service that provides a managed platform for running applications. As this is a managed service, your focus should be on the application only and Google will manage the resources needed to run the application. Thus App Engine users have less to manage, but you will have less control over the compute resources. The applications hosted on App Engine are highly scalable and run reliably even under heavy load.

The App Engine supports the following languages:

- Python
- Go
- Ruby
- PHP
- Node.js
- Java
- .NET

The App Engine provides two types of runtime environments: standard and flexible.

1. The Standard environment provides a secured and sandboxed environment for running applications and distributes requests across multiple servers to meet the demand. The applications run independently of the hardware, OS, and physical location of the server.
2. The Flexible environment provides more options and control to the developers who want to use App Engine, but without the language constraints of the standard environment. It uses Docker containers as the basic building blocks. These containers can be auto-scaled according to load.

Pros of App Engine:

- You need to focus only on the application code, the rest of everything is managed by Google. Thus reducing management complexities.
- As it provides version management, thus it is easy to maintain and roll out versions of applications.
- It has faster autoscaling as the size of instances is smaller.
- Easy to deploy and monitor.

Cons of App Engine:

- It is more constrained as the instances are smaller, thus enabling fast autoscaling, but there can be cases when large applications require larger instances.
- As it is a fully managed service, the user has no control over the underlying infrastructure that may be required for some complex applications.
- It is expensive in the long run as the cost adds up quickly.

Difference between Compute Engine and App Engine:

	Compute Engine	**App Engine**
Service model	IaaS offering	PaaS offering
Type of Service	Unmanaged Service	Managed Service
Control over resources	More control and flexibility	Less control over computing resources
Costs	Costs less than App Engine over resources	Costs more than Compute Engine
Running Instances	When running application, at least one instance should be running	Can scale down to zero instances when no requests are coming
Use cases	Best for general computing workloads	Best for web-facing and mobile applications
Autoscaling	Slower autoscaling	Faster autoscaling
Security	Less secure than App Engine	Comparatively more secure than Compute Engine

11.3 ANEKA

Aneka includes an extensible set of APIs associated with programming models like MapReduce.

These APIs support different cloud models like a private, public, hybrid Cloud.

Manjrasoft focuses on creating innovative software technologies to simplify the development and deployment of private or public cloud applications. Our product plays the role of an application platform as a service for multiple cloud computing.

Multiple Structures:

Aneka is a software platform for developing cloud computing applications.

In Aneka, cloud applications are executed.

Aneka is a pure PaaS solution for cloud computing.

Aneka is a cloud middleware product.

Manya can be deployed over a network of computers, a multicore server, a data center, a virtual cloud infrastructure, or a combination thereof.

Multiple containers can be classified into three major categories:

1. Textile services

2. Foundation Services

3.Application Services

1. Textile Services:

OAK
"GREEN PROJECT"
Oak is a symbol of strength and chosen as a national tree of many countries like U.S.A., France, G

Fabric Services defines the lowest level of the software stack that represents multiple containers. They provide access to resource-provisioning subsystems and monitoring features implemented in many.

2. Foundation Services:

Fabric Services are the core services of Manya Cloud and define the infrastructure management features of the system. Foundation services are concerned with the logical management of a distributed system built on top of the infrastructure and provide ancillary services for delivering applications.

3. Application Services:

Application services manage the execution of applications and constitute a layer that varies according to the specific programming model used to develop distributed applications on top of Aneka.

There are mainly two major components in multiple technologies:

The SDK (Software Development Kit) includes the Application Programming Interface (API) and tools needed for the rapid development of applications. The Anka API supports three popular cloud programming models: **Tasks, Threads** and **MapReduce**;

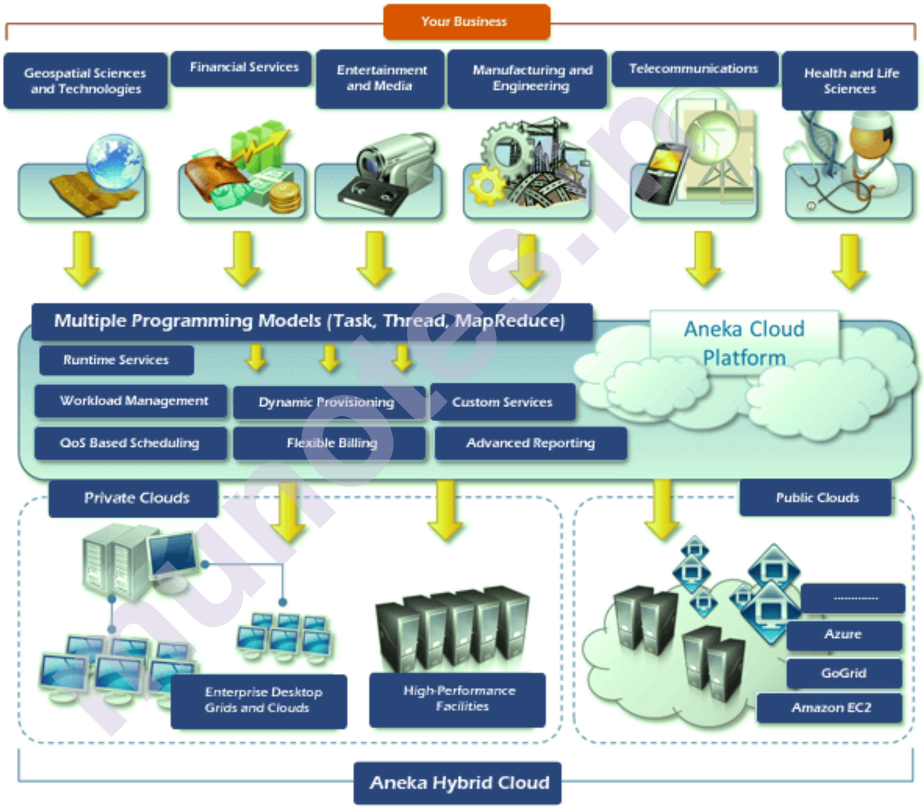
And:

A runtime engine and platform for managing the deployment and execution of applications on a private or public cloud.

One of the notable features of Aneka Pass is to support the provision of private cloud resources from desktop, cluster to a virtual data center using **VMware, Citrix Zen Server**, and public cloud resources such as **Windows Azure, Amazon EC2**, and **GoGrid cloud service**.

Aneka's potential as a Platform as a Service has been successfully harnessed by its users and customers in three different areas, including **engineering, life sciences, education,** and **business intelligence.**

Architecture of Aneka:



Aneka is a platform and framework for developing distributed applications on the Cloud. It uses desktop PCs on-demand and CPU cycles in addition to a heterogeneous network of servers or datacenters. Aneka provides a rich set of APIs for developers to transparently exploit such resources and express the business logic of applications using preferred programming abstractions.

System administrators can leverage a collection of tools to monitor and control the deployed infrastructure. It can be a public cloud available to anyone via the Internet or a private cloud formed by nodes with restricted access.

A multiplex-based computing cloud is a collection of physical and virtualized resources connected via a network, either the Internet or a private intranet. Each resource hosts an instance of multiple containers that represent the runtime environment where distributed applications are executed. The container provides the basic management features of a single node and takes advantage of all the other functions of its hosting services.

Services are divided into clothing, foundation, and execution services. Foundation services identify the core system of Aneka middleware, which provides a set of infrastructure features to enable Anka containers to perform specific and specific tasks. Fabric services interact directly with nodes through the Platform Abstraction Layer (PAL) and perform hardware profiling and dynamic resource provisioning. Execution services deal directly with scheduling and executing applications in the Cloud.

One of the key features of Aneka is its ability to provide a variety of ways to express distributed applications by offering different programming models; Execution services are mostly concerned with providing middleware with the implementation of these models. Additional services such as persistence and security are inverse to the whole stack of services hosted by the container.

At the application level, a set of different components and tools are provided to

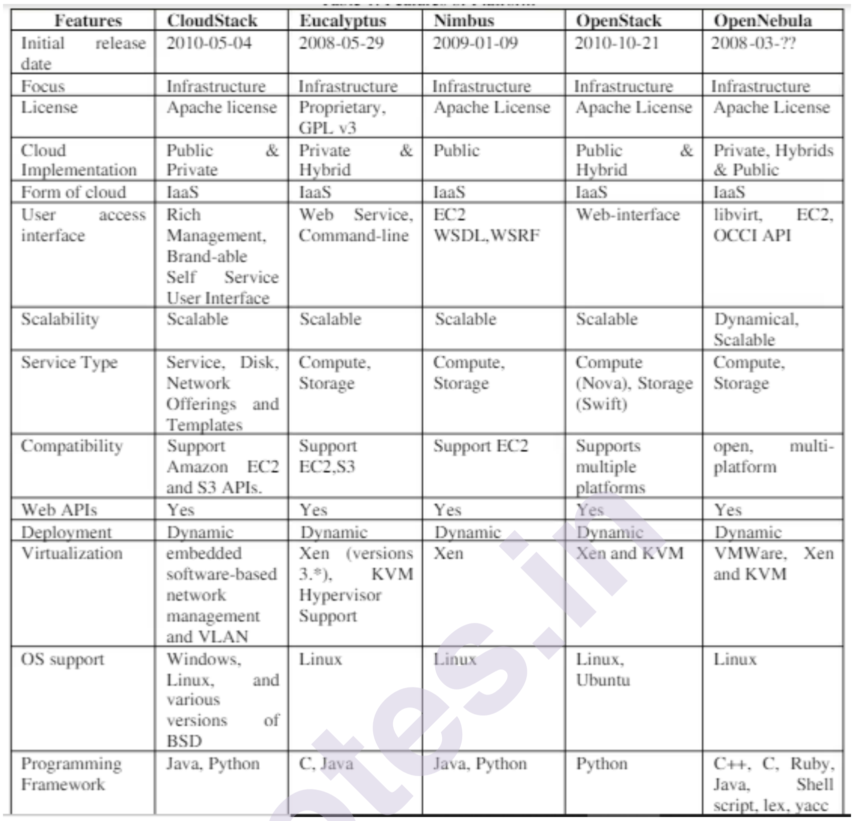
- Simplify the development of applications (SDKs),
- Port existing applications to the Cloud, and
- Monitor and manage multiple clouds.

An Aneka-based cloud is formed by interconnected resources that are dynamically modified according to user needs using resource virtualization or additional CPU cycles for desktop machines. A common deployment of Aneka is presented on the side. If the deployment identifies a private cloud, all resources are in-house, for example, within the enterprise.

This deployment is enhanced by connecting publicly available on-demand resources or by interacting with several other public clouds that provide computing resources connected over the Internet.

11.4 COMPARATIVE STUDY OF VARIOUS CLOUD

Below table shows the comparitive study of various cloud:



Features	CloudStack	Eucalyptus	Nimbus	OpenStack	OpenNebula
Initial release date	2010-05-04	2008-05-29	2009-01-09	2010-10-21	2008-03-??
Focus	Infrastructure	Infrastructure	Infrastructure	Infrastructure	Infrastructure
License	Apache license	Proprietary, GPL v3	Apache License	Apache License	Apache License
Cloud Implementation	Public & Private	Private & Hybrid	Public	Public & Hybrid	Private, Hybrids & Public
Form of cloud	IaaS	IaaS	IaaS	IaaS	IaaS
User access interface	Rich Management, Brand-able Self Service User Interface	Web Service, Command-line	EC2 WSDL,WSRF	Web-interface	libvirt, EC2, OCCI API
Scalability	Scalable	Scalable	Scalable	Scalable	Dynamical, Scalable
Service Type	Service, Disk, Network Offerings and Templates	Compute, Storage	Compute, Storage	Compute (Nova), Storage (Swift)	Compute, Storage
Compatibility	Support Amazon EC2 and S3 APIs.	Support EC2,S3	Support EC2	Supports multiple platforms	open, multi-platform
Web APIs	Yes	Yes	Yes	Yes	Yes
Deployment	Dynamic	Dynamic	Dynamic	Dynamic	Dynamic
Virtualization	embedded software-based network management and VLAN	Xen (versions 3.*), KVM Hypervisor Support	Xen	Xen and KVM	VMWare, Xen and KVM
OS support	Windows, Linux, and various versions of BSD	Linux	Linux	Linux, Ubuntu	Linux
Programming Framework	Java, Python	C, Java	Java, Python	Python	C++, C, Ruby, Java, Shell script, lex, yacc

11.5 COMPUTING PLATFORMS

What Does Platform Mean?:

A platform is a group of technologies that are used as a base upon which other applications, processes or technologies are developed.

In personal computing, a platform is the basic hardware (computer) and software (operating system) on which software applications can be run. This environment constitutes the basic foundation upon which any application or software is supported and/or developed.

Computers use specific central processing units (CPUs) that are designed to run specific machine language code. In order for the computer to run software applications, the applications must be in that CPU’s binary-coded machine language.

Thus, historically, application programs written for one platform would not work on a different platform.

A computer platform — also called digital platform or computing platform — generally refers to the operating system and computer hardware only.

An example of a computing platform is a modern laptop running Windows as an operating system. Another example would be an Apple computer running the Mac OS X operating system.

Platform Standards:

The platform conforms to a set of standards that enable software developers to develop software applications for the platform. These same standards allow owners and managers to purchase appropriate applications and hardware. Thus, to run a bookkeeping program on a computer, one must purchase a bookkeeping software application that was developed for the platform on which it will be used.

Multiple Platforms:

New standards-based interfaces and open interfaces allow application programs to run on multiple platforms. Additionally, software developers have developed software tools that allow applications to run on multiple platforms.

Cross-Platform Software and Multi-Platform Software:

This has given rise to the terms cross-platform software and multi-platform software. A classic example is represented by videogames developed specifically for a certain platform, in this case a console such as the PlayStation or Xbox. Although the same game may exist in different versions to be run on different systems, if that version is built to be run on Microsoft Windows, it won't work if loaded on an Xbox. Each gaming platform will adhere to its own set of standards as well as rules and hardware restrictions. For example, developers may need to lower in-game graphics settings if the game engine is too heavy on a specific console's hardware.

Browsers:

Newer web browsers allow third-party plug-ins to be run as part of the browser. Therefore, some browsers are now spoken of as platforms since they are used as a base on which to run other applications' software programs.

Mobile Platforms:

Today, new mobile devices such as smartphones and tablets possess their own software and hardware. They operate independently of other systems and are capable of running their own apps, tools, and other software, hence they can be effectively considered as platforms.

Digital Platforms:

Software stacks and some applications are also sometimes referred as digital platforms.

For example, SQL is a database application that is often used as an environment to run other tools for CRM, analytics and log management.

Similarly, the collection of the three-open source applications Elasticsearch, Logstash, and Kibana constitutes the ELK Stack, a platform used for logging purposes.

*****

12

CLOUD ISSUES AND CHALLENGES

Unit Structure

12.0 OBJECTIVE

- To study more about cloud issues and challenges.
- To study about cloud security and elasticity.
- To study and understand about Quality of Service (QoS) and resource allocation.

12.1 INTRODUCTION

In Simplest terms, cloud computing means storing and accessing the data and programs on remote servers that are hosted on internet instead of computer's hard drive or local server. Cloud computing is also referred as Internet based computing.

Cloud Computing Architecture:

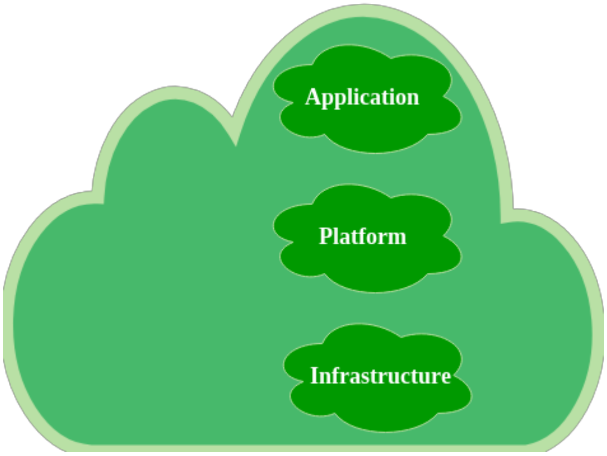
Cloud computing architecture refers to the components and sub components required for cloud computing. These component typically refer to:

1. Front end(fat client, thin client)
2. Back end platforms(servers, storage)
3. Cloud based delivery and a network (Internet, Intranet, Inter cloud).

Hosting a cloud:

There are three layers in cloud computing. Companies use these layers based on the service they provide.

- Infrastructure
- Platform
- Application



Three layers of Cloud Computing:

At the bottom is the foundation, the Infrastructure where the people start and begin to build. This is the layer where the cloud hosting lives.

Now, let's have a look at hosting:

Let's say you have a company and a website and the website has a lot of communications that are exchanged between members. You start with a few members talking with each other and then gradually the numbers of members increase.

As the time passes, as the number of members increases, there would be more traffic on the network and your server will get slow down. This would cause a problem.

A few years ago, the websites are put in the server somewhere, in this way you have to run around or buy and set number of servers. It costs a lot of money and takes lot of time. You pay for these servers when you are using and as well as when you are not using. This is called hosting.

This problem is overcome by cloud hosting. With Cloud Computing, you have access to computing power when you needed. Now, your website is put in the cloud server as you put it on dedicated server. People start visiting your website and if you suddenly need more computing power, you would scale up according to the need.

Benefits of Cloud Hosting:

1. **Scalability:** With Cloud hosting, it is easy to grow and shrink the number and size of servers based on the need. This is done by either increasing or decreasing the resources in the cloud. This ability to alter plans due to fluctuation in business size and needs is a superb benefit of cloud computing especially when experiencing a sudden growth in demand.

2. **Instant:** Whatever you want is instantly available in the cloud.

3. **Save Money:** An advantage of cloud computing is the reduction in hardware cost. Instead of purchasing in-house equipment, hardware

needs are left to the vendor. For companies that are growing rapidly, new hardware can be a large, expensive, and inconvenience. Cloud computing alleviates these issues because resources can be acquired quickly and easily. Even better, the cost of repairing or replacing equipment is passed to the vendors.

Along with purchase cost, off-site hardware cuts internal power costs and saves space. Large data centers can take up precious office space and produce a large amount of heat. Moving to cloud applications or storage can help maximize space and significantly cut energy expenditures.

4. **Reliability:** Rather than being hosted on one single instances of a physical server, hosting is delivered on a virtual partition which draws its resource, such as disk space, from an extensive network of underlying physical servers. If one server goes offline it will have no effect on availability, as the virtual servers will continue to pull resource from the remaining network of servers.

5. **Physical Security:** The underlying physical servers are still housed within data centres and so benefit from the security measures that those facilities implement to prevent people accessing or disrupting them on-site

12.2 CLOUD COMPUTING ISSUES AND CHALLENGES

Let us dive into the challenges of cloud computing, With a multitude of benefits of implementing cloud computing in all-size businesses, cloud computing has become a popular trend in the market. To put in simple words: Cloud computing is nothing but moving on-premises computing to the internet. It saves both time and money. People around the globe get access to an open pool of resources like apps, services, servers, data, and computer networks. It is made possible either by using a privately-owned cloud or a 3rd-party server. It improves the way data is accessed and removes inconsistency in further updates. Also, a minimal amount of administration is required. Cloud computing also ensures data security, better data storage, increased synchronization between employees, and flexibility. Organizations have become capable of making better decisions to scale and grow.

Despite all the development and potential of cloud computing services, there are multiple challenges of cloud computing services that businesses face. Here we have compiled a list of challenges of cloud computing that need to be taken care of, to leverage the maximum capability of the cloud. Let us get started:

1. Security
2. Password Security
3. Cost Management

4. Lack of expertise
5. Internet Connectivity
6. Control or Governance
7. Compliance
8. Multiple Cloud Management
9. Creating a private cloud
10. Performance
11. Migration
12. Interoperability and Portability
13. Reliability and High Availability
14. Hybrid-Cloud Complexity

1. SECURITY:

The topmost concern in investing in cloud services is security issues in cloud computing. It is because your data gets stored and processed by a third-party vendor and you cannot see it. Every day or the other, you get informed about broken authentication, compromised credentials, account hacking, data breaches, etc. in a particular organization. It makes you a little more skeptical.

Fortunately, the cloud providers, these days have started to put efforts to improve security capabilities. You can be cautious as well by verifying if the provider implements a safe user identity management system and access control procedures. Also, ensure it implements database security and privacy protocols.

2. PASSWORD SECURITY:

As large numbers of people access your cloud account, it becomes vulnerable. Anybody who knows your password or hacks into your cloud will be able to access your confidential information.

Here the organization should use a multiple level authentication and ensure that the passwords remain protected. Also, the passwords should be modified regularly, especially when a particular employee resigns and leave the organization. Access rights to usernames and passwords should be given judiciously.

3. COST MANAGEMENT:

Cloud computing enables you to access application software over a fast internet connection and lets you save on investing in costly computer hardware, software, management and maintenance. This makes it affordable. But what is challenging and expensive is tuning the organization's needs on the third-party platform. Another costly affair is the cost of transferring data to a public cloud, especially for a small business or project.

4. LACK OF EXPERTISE:

With the increasing workload on cloud technologies and continuously improving cloud tools, the management has become difficult. There has been a consistent demand for a trained workforce who can deal with cloud computing tools and services. Hence, firms need to train their IT staff to minimize this challenge.

5. INTERNET CONNECTIVITY:

The cloud services are dependent on a high-speed internet connection. So the businesses that are relatively small and face connectivity issues should ideally first invest in a good internet connection so that no downtime happens. It is because internet downtime might incur vast business losses.

6. CONTROL OR GOVERNANCE:

Another ethical issue in cloud computing is maintaining proper control over asset management and maintenance. There should be a dedicated team to ensure that the assets used to implement cloud services are used according to agreed policies and dedicated procedures. There should be proper maintenance and that the assets are used to meet your organization's goals successfully.

7. COMPLIANCE:

Another major risk of cloud computing is maintaining compliance. By compliance we mean, a set of rules about what data is allowed to be moved and what should be kept in-house to maintain compliance. The organizations must follow and respect the compliance rules set by various government bodies.

8. MULTIPLE CLOUD MANAGEMENT:

Companies have started to invest in multiple public clouds, multiple private clouds or a combination of both called the hybrid cloud. This has grown rapidly in recent times. So it has become important to list challenges faced by such organizations and find solutions to grow with the trend.

9. CREATING A PRIVATE CLOUD:

Implementing an internal cloud is advantageous. This is because all the data remains secure in-house. But the challenge here is that the IT team has to build and fix everything by themselves. Also, the team needs to ensure smooth functioning of the cloud. They need to automate maximum manual tasks. The execution of tasks should be in the correct order.

So, at the moment, it sounds quite difficult to set up a private cloud all by yourself. But many organizations are planning to do so in future.

10. PERFORMANCE:

When your business applications move to a cloud or a third-party vendor, so your business performance starts to depend on your provider as well. Another major problem in cloud computing is investing in the right cloud service provider. Before investment, you should look for providers with innovatory technologies. The performance of the BI's and other cloud-based systems are linked to the provider's systems as well. Be cautious about choosing the provider and investigate that they have protocols to mitigate issues that arise in real-time.

11. MIGRATION:

Migration is nothing but moving a new application or an existing application to a cloud. In the case of a new application, the process is pretty straightforward. But if it is an age-old company application, it becomes tedious.

12. INTEROPERABILITY AND PORTABILITY:

Another **challenge of cloud computing** is that applications need to be easily migrated between cloud providers without being locked for a set period. There is a lack of flexibility in moving from one cloud provider to another because of the complexity involved. Changing cloud inventions bring a slew of new challenges like managing data movement and establishing a secure network from scratch. Another challenge is that customers can't access it from everywhere, but this can be fixed by the cloud provider so that the customer can securely access the cloud from anywhere.

13. RELIABILITY AND HIGH AVAILABILITY:

Some of the most pressing issues in cloud computing is the need for high availability (HA) and reliability. Reliability refers to the likelihood that a system will be up and running at any given point in time, whereas availability refers to how likely it is that the system will be up and running at any given point in time. Because most businesses are now reliant on third-party services, cloud systems must be dependable and robust. Cloud providers continue to lack round-the-clock service, resulting in frequent outages. It is critical to use internal or third-party tools to monitor the service being provided. It is critical to have plans in place to monitor SLAs, usage, robustness, performance, and business reliance on these services.

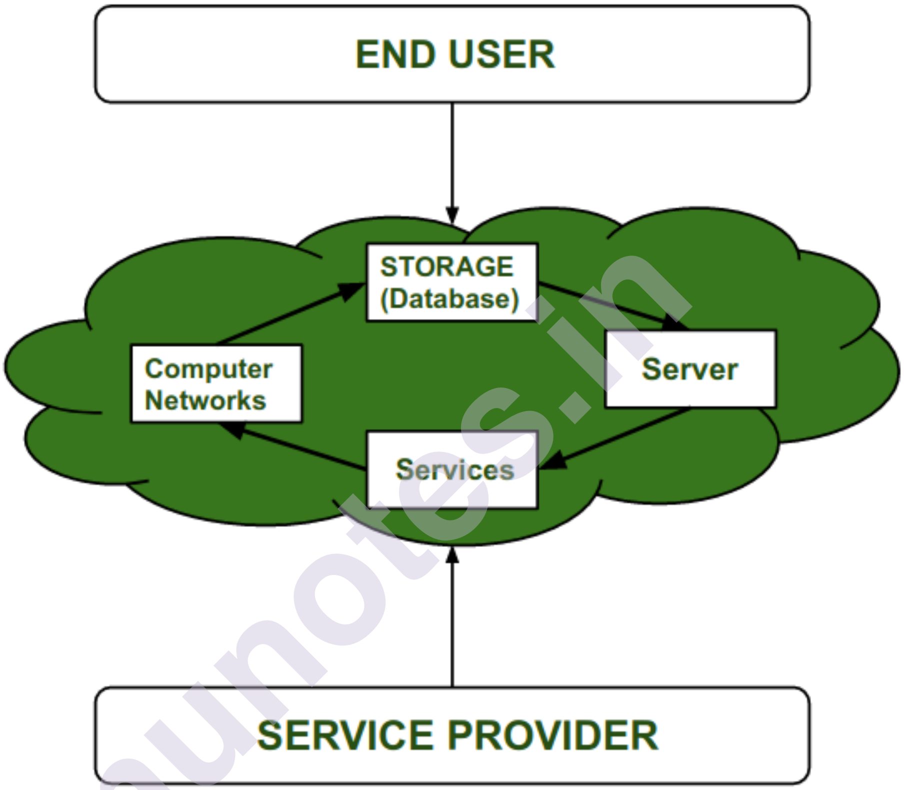
14. HYBRID-CLOUD COMPLEXITY:

For any company, a hybrid cloud environment is often a messy mix of multiple cloud application development and cloud service providers, as well as private and public clouds, all operating at once. A common user interface, consistent data, and analytical benefits for businesses are all missing from these complex cloud ecosystems. **Cloud computing**

challenges such as scalability, integration, and disaster recovery are magnified in a hybrid cloud environment.

12.2.1 Security:

Cloud Computing is a type of technology that provides remote services on the internet to manage, access, and store data rather than storing it on Servers or local drives. This technology is also known as Serverless technology. Here the data can be anything like Image, Audio, video, documents, files, etc.



Need of Cloud Computing:

Before using Cloud Computing, most of the large as well as small IT companies use traditional methods i.e. they store data in Server, and they need a separate Server room for that. In that Server Room, there should be a database server, mail server, firewalls, routers, modems, high net speed devices, etc. For that IT companies have to spend lots of money. In order to reduce all the problems with cost Cloud computing come into existence and most companies shift to this technology.

Security Issues in Cloud Computing:

There is no doubt that Cloud Computing provides various Advantages but there are also some security issues in cloud computing. Below are some following Security Issues in Cloud Computing as follows.

Data Loss:

Data Loss is one of the issues faced in Cloud Computing. This is also known as Data Leakage. As we know that our sensitive data is in the hands of Somebody else, and we don't have full control over our database. So if the security of cloud service is to break by hackers then it may be possible that hackers will get access to our sensitive data or personal files.

Interference of Hackers and Insecure API's:

As we know if we are talking about the cloud and its services it means we are talking about the Internet. Also, we know that the easiest way to communicate with Cloud is using API. So it is important to protect the Interface's and API's which are used by an external user. But also in cloud computing, few services are available in the public domain. An is the vulnerable part of Cloud Computing because it may be possible that these services are accessed by some third parties. So it may be possible that with the help of these services hackers can easily hack or harm our data.

User Account Hijacking:

Account Hijacking is the most serious security issue in Cloud Computing. If somehow the Account of User or an Organization is hijacked by Hacker. Then the hacker has full authority to perform Unauthorized Activities**.**

Changing Service Provider:

Vendor lock In is also an important Security issue in Cloud Computing. Many organizations will face different problems while shifting from one vendor to another. For example, An Organization wants to shift from AWS Cloud to Google Cloud Services then they ace various problem's like shifting of all data, also both cloud services have different techniques and functions, so they also face problems regarding that. Also, it may be possible that the charges of AWS are different from Google Cloud, etc.

Lack of Skill:

While working, shifting o another service provider, need an extra feature, how to use a feature, etc. are the main problems caused in IT Company who doesn't have skilled Employee. So it requires a skilled person to work with cloud Computing.

Denial of Service (DoS) attack:

This type of attack occurs when the system receives too much traffic. Mostly DoS attacks occur in large organizations such as the banking sector, government sector, etc. When a DoS attack occurs data is lost. So, in order to recover data, it requires a great amount of money as well as time to handle it.

12.2.2 Elasticity:

The Elasticity refers to the ability of a cloud to automatically expand or compress the infrastructural resources on a sudden-up and down in the requirement so that the workload can be managed efficiently. This elasticity helps to minimize infrastructural cost. This is not applicable for all kind of environment, it is helpful to address only those scenarios where the resources requirements fluctuate up and down suddenly for a specific time interval. It is not quite practical to use where persistent resource infrastructure is required to handle the heavy workload.

It is most commonly used in pay-per-use, public cloud services. Where IT managers are willing to pay only for the duration to which they consumed the resources.

Example:

Consider an online shopping site whose transaction workload increases during festive season like Christmas. So for this specific period of time, the resources need a spike up. In order to handle this kind of situation, we can go for Cloud-Elasticity service rather than Cloud Scalability. As soon as the season goes out, the deployed resources can then be requested for withdrawal.

Cloud Scalability:

Cloud scalability is used to handle the growing workload where good performance is also needed to work efficiently with software or applications. Scalability is commonly used where the persistent deployment of resources is required to handle the workload statically.

Example:

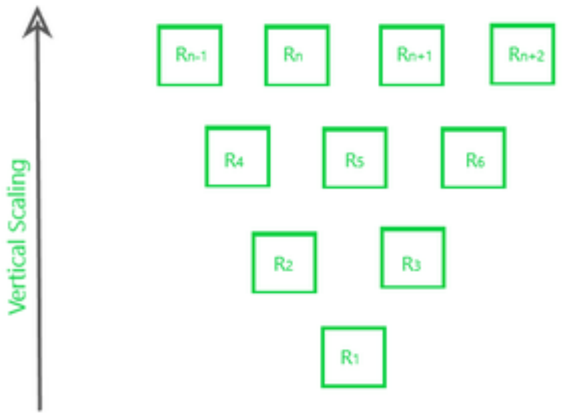
Consider you are the owner of a company whose database size was small in earlier days but as time passed your business does grow and the size of your database also increases, so in this case you just need to request your cloud service vendor to scale up your database capacity to handle a heavy workload.

It is totally different from what you have read above in Cloud Elasticity. Scalability is used to fulfill the static needs while elasticity is used to fulfill the dynamic need of the organization. Scalability is a similar kind of service provided by the cloud where the customers have to pay-per-use. So, in conclusion, we can say that Scalability is useful where the workload remains high and increases statically.

Types of Scalability:

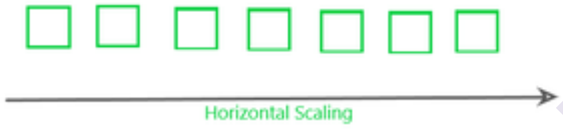
1. Vertical Scalability (Scale-up):

In this type of scalability, we increase the power of existing resources in the working environment in an upward direction.



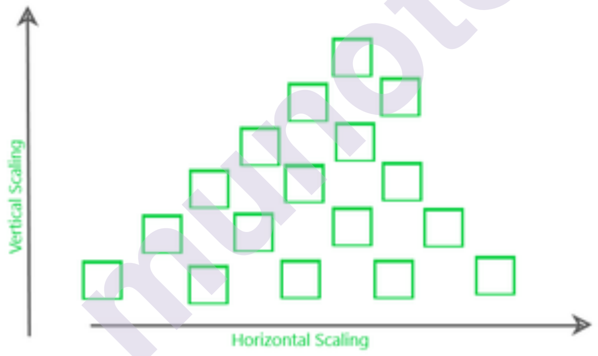
2. Horizontal Scalability:

In this kind of scaling, the resources are added in a horizontal row.



3. Diagonal Scalability:

It is a mixture of both Horizontal and Vertical scalability where the resources are added both vertically and horizontally.



12.2.3 Resource management and scheduling:

Cloud resource management requires complex policies and decisions for multi-objective optimization. Effective resource management is extremely challenging due to the scale of the cloud infrastructure and to the unpredictable interactions of the system with a large population of users. The scale makes it impossible to have accurate global state information and the large user population makes it nearly impossible to predict the type and the intensity of the system workload.

An overview of policies and mechanisms for cloud resource management is followed by a presentation of energy efficiency and cloud resource utilization and the impact of application scaling on resource management.

The policies for CRM can be loosely grouped into five classes:

(1) admission control;

(2) capacity allocation;

(3) load balancing;

(4) energy optimization; and

(5) quality of service (QoS) guarantees.

The explicit goal of an admission control policy is to prevent the system from accepting workload in violation of high-level system policies. A system should not accept additional workload if this would prevent it from completing work already in progress or contracted. Limiting the workload requires some knowledge of the global state of the system.

Capacity allocation means to allocate resources for individual instances. An instance is an activation of a service on behalf of a cloud user. Locating resources subject to multiple global optimization constraints requires a search in a very large search space. Capacity allocation is more challenging when the state of individual servers changes rapidly.

Load balancing and energy optimization are correlated and affect the cost of providing the services; they can be done locally, but global load balancing and energy optimization policies encounter the same difficulties as the capacity allocation. Quality of service (QoS) is probably the most challenging aspect of resource management and, at the same time, possibly the most critical for the future of cloud computing.

Resource management policies must be based on a disciplined approach, rather than ad hoc methods.

To our knowledge, none of the optimal or near-optimal methods to address the five classes of policies scale up, thus, there is a need to develop novel strategies for resource management in a computer cloud. Typically, these methods target a single aspect of resource management, e.g., admission control, but ignore energy conservation; many require very complex computations that cannot be done effectively in the time available to respond.

Performance models required by some of the methods are very complex, analytical solutions are intractable, and the monitoring systems used to gather state information for these models can be too intrusive and unable to provide accurate data. Many techniques are concentrated on system performance in terms of throughput and time in system, but they rarely include energy trade-offs or QoS guarantees. Some techniques are based on unrealistic assumptions; for example, capacity allocation is viewed as an optimization problem, but under the assumption that servers are protected from overload.

Virtually all mechanisms for the implementation of the resource management policies require the presence of a few systems which monitor and control the entire cloud, while the large majority of systems run applications and store data; some of these mechanisms require a two-level control, one at the cloud level and one at the application level. The strategies for resource management associated with IaaS, PaaS, and SaaS will be different, but in all cases the providers are faced with large fluctuating loads.

In some cases, when a spike can be predicted, the resources can be provisioned in advance, e.g., for Web services subject to seasonal spikes. For an unplanned spike, the situation is slightly more complicated. Auto-scaling can be used for unplanned spike loads provided that: (a) there is a pool of resources that can be released or allocated on demand and (b) there is a monitoring system which allows a control loop to decide in real time to reallocate resources.

Policies and mechanisms for resource management

A policy typically refers to the principal guiding decisions, whereas mechanisms represent the means to implement policies. Separation of policies from mechanisms is a guiding principle in computer science.

Cloud resource management policies can be loosely grouped into five classes:

1. Admission control.
2. Capacity allocation.
3. Load balancing.
4. Energy optimization.
5. Quality-of-service (QoS) guarantees.

The explicit goal of an admission control policy is to prevent the system from accepting workloads in violation of high-level system policies; for example, a system may not accept an additional workload that would prevent it from completing work already in progress or contracted. Limiting the workload requires some knowledge of the global state of the system. In a dynamic system such knowledge, when available, is at best obsolete. Capacity allocation means to allocate resources for individual instances; an instance is an activation of a service. Locating resources subject to multiple global optimization constraints requires a search of a very large search space when the state of individual systems changes rapidly.

Load balancing and energy optimization can be done locally, but global load-balancing and energy optimization policies encounter the same difficulties as the one we have already discussed. Load balancing and energy optimization are correlated and affect the cost of providing the services. Indeed, it was predicted that by 2012 up to 40% of the budget for IT enterprise infrastructure would be spent on energy .

The common meaning of the term load balancing is that of evenly distributing the load to a set of servers. For example, consider the case of four identical servers, A,B,C, and D, whose relative loads are 80%,60%,40%, and 20%, respectively, of their capacity. As a result of perfect load balancing, all servers would end with the same load -50% of each server's capacity. In cloud computing a critical goal is minimizing the cost of providing the service and, in particular, minimizing the energy consumption. This leads to a different meaning of the term load balancing; instead of having the load evenly distributed among all servers, we want to concentrate it and use the smallest number of servers while switching the others to standby mode, a state in which a server uses less energy. In our example, the load from D will migrate to A and the load from C will migrate to B; thus, A and B will be loaded at full capacity, whereas C and D will be switched to standby mode. Quality of service is that aspect of resource management that is probably the most difficult to address and, at the same time, possibly the most critical to the future of cloud computing.

As we shall see in this section, often resource management strategies jointly target performance and power consumption. Dynamic voltage and frequency scaling (DVFS)[1] techniques such as Intel's SpeedStep and AMD's PowerNow lower the voltage and the frequency to decrease power consumption.[2] Motivated initially by the need to save power for mobile devices, these techniques have migrated to virtually all processors, including the ones used for high-performance servers.

As a result of lower voltages and frequencies, the performance of processors decreases, but at a substantially slower rate [213] than the energy consumption. Table 6.1 shows the dependence of the normalized performance and the normalized energy consumption of a typical modern processor on clock rate. As we can see, at 1.8 GHz we save 18% of the energy required for maximum performance, whereas the performance is only 5% lower than the peak performance, achieved at 2.2 GHz. This seems a reasonable energy-performance tradeoff!

12.3 QUALITY OF SERVICE (QOS) AND RESOURCE ALLOCATION

In a general sense, Quality of Service (QoS) is defined as a set of quality requirements (i.e., desirable properties) of an application, which are not explicitly formulated in its functional interfaces (1.4.1). In that sense, QoS includes fault tolerance, security, performance, and a set of "ilities" such as availability, maintainability, etc. In a more restricted meaning (considered in this chapter), QoS characterizes the ability of an application to satisfy performance-related constraints. This specific meaning of QoS is specially relevant in areas such as multimedia processing, real-time control, or interactive services for end users.

Performance control is achieved through resource management, the main theme of this chapter. We examine the main abstractions and patterns for managing resources, including the use of feedback control methods.

Introducing Resource Management:

The function of a computing system is to provide services to its users. Each service is specified by a contract between a service provider and a service requester. This contract defines both the functional interface of the service and some extra-functional aspects, collectively known as Quality of Service (QoS), which include performance, availability, security, and need to be accurately specified for each application or class of applications. The part of the contract that defines QoS is called a Service Level Agreement (SLA). The technical expression of an SLA usually consists of a set of Service Level Objectives (SLO), each of which defines a precise objective for one of the specific aspects covered by the SLA. For instance, for an SLA on the performance of a web server, an SLO can specify a maximum response time to be achieved for 95% of the requests submitted by a certain class of users.

Motivation and Main Definitions:

In order to perform its function, a computing system uses various resources such as processors, memory, communication channels, etc. Managing these resources is an important function, and there are several strong reasons for performing it accurately:

- **Maintaining quality of service:** Various indicators related to QoS in user applications, specially performance factors, are directly influenced by resource allocation decisions.
- **Resource accounting:** The users of a shared facility should be charged according to their actual resource consumption. Therefore, any consumed resource must be traced back to a user activity.
- **Service differentiation and resource pricing:** The resource management system may allow users to pay for improved service. The system should guarantee this differentiated form of service, and the pricing scheme should adequately reflect the added value thus acquired.
- **Detecting and countering Denial of Service (DoS):** A DoS attack aims at preventing useful work from being done, by undue massive acquisition of resources such as CPU time, memory, or network bandwidth.
- **Tracking and eliminating performance bugs:** Without accurate monitoring of resource usage, a runaway activity might invisibly consume large amounts of resources, leading to performance degradation in user applications; or a regular activity may reserve unnecessary resources, thus hampering the progress of other activities.

In the traditional view of a computing system, resource allocation, i.e., the sharing of a common set of resources between applications contending for their use, was a task performed by the operating system, and user applications had little control over this process. This situation has changed due to the following causes:

- The increasing number of applications subject to strong constraints in time and space, e.g., embedded systems and applications managing multimedia data.
- The growing variability of the environment and operating conditions of many applications, e.g., those involving mobile communications.
- The trend towards more open systems, and the advent of open middleware.

Thus, an increasing part of resource management is being delegated to the upper levels, i.e., to the middleware layers and to the applications themselves. In this chapter, we examine some aspects of resource allocation in these upper levels.

The term resource applies to any identifiable entity (physical or virtual) that is used by a system for service provision. The entity that actually implements service provision, using resources, is called a resource principal. Examples of physical resources are processors, memory, disk storage, routers, network links, sensors. Examples of virtual resources are virtual memory, network bandwidth, files and other data (note that virtual resources are abstractions built on top of physical resources). Examples of resource principals are processes in an operating system, groups of processes dedicated to a common task (possibly across several machines), various forms of "agents" (computational entities that may move or spread across the nodes of a network). One may define a hierarchy of principals: for example, a virtual machine provides (virtual) resources to the applications it supports, while requesting (physical or even virtual) resources from a hypervisor.

Goals and Policies:

The role of resource management is to allocate resources to the service providers (principals), subject to the requirements and constraints of both service providers and resource providers. The objective of a service provider is to respect its SLA, the contract that binds it to service requesters (clients). The objective of a resource provider is to maximize the utilization rate of its resources, and possibly the revenue it draws from their provision. The relationships between clients, service providers, and resource providers are illustrated in below figure,

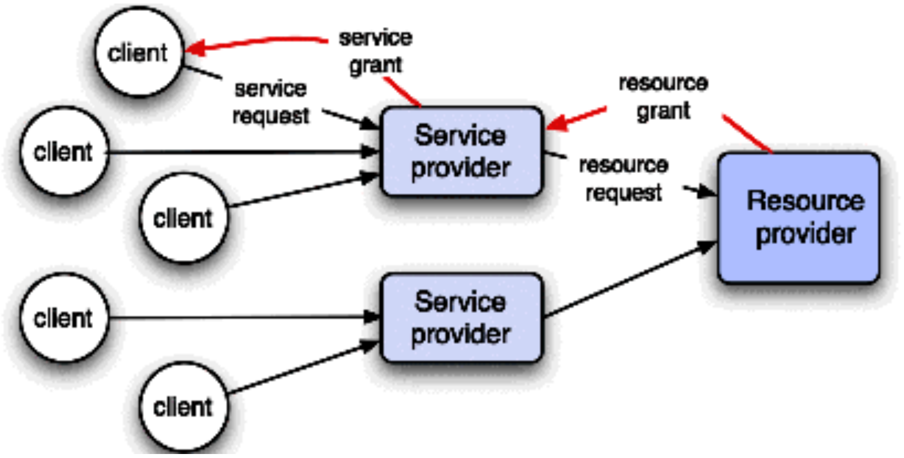


Figure: Service providers and resource providers

The requirements are the following, as seen by a service provider.

- The service provider should offer QoS guarantees to its clients, as specified by an SLA. The SLA may take various forms: strict guarantee, probabilistic (the agreed levels will be reached with a specified probability, or for a specified fraction of the demands), best effort (no guarantee on the results). The SLA is a contract that goes both ways, i.e., the guarantees are only granted if the clients respects some conditions on their requests. These conditions may apply to each client individually or to a population of clients as a whole.

- The service provider allocates resources for the satisfaction of clients' requests. It should ensure equitable treatment (fair share), in the sense that each client needing a resource should be guaranteed a share of that resource proportional to its "right" (or "priority"), as defined by a global policy. The share should be guaranteed in average over a specified period of time. If all clients have the same right, then each should be guaranteed an equal share. Other situations may occur:

    - Service differentiation is a means of specifying different classes of clients with different rights, which may be acquired by purchase or by negotiation.

    - The rights may be time-dependent, e.g., the right of a client may be increased to allow it to meet a deadline.More generally, the guarantee may be in terms of a minimal rate of progress, which prevents starvation, a situation in which a client’s requests are indefinitely delayed.

More generally, the guarantee may be in terms of a minimal rate of progress, which prevents starvation, a situation in which a client's requests are indefinitely delayed.

- If several classes of clients are defined, the service provider should guarantee service isolation: the allocation of resources for a class should not be influenced by that of other classes. This means that a misbehaving client (one that does not respect its side of the SLA) may only affect other clients of its class, not clients from other classes.

The above requirements may be contradictory. For instance, ensuring guarantees through worst case reservation may entail sub-optimal resource utilization. Such conflicts may only be resolved according to a higher level policy, e.g., through priority setting or through negotiation. Another approach is to pool resources, in order to amortize their cost among several applications, provided that peak loads do not occur at the same time for all of them. Also note that the fairness requirement should be met both at the global level (sharing resources among service providers) and for each service provider with respect to its own service requesters.

Resource management policies may be classified using various criteria (based on prediction and/or observation, using open loop or closed loop) and may face various operating conditions: if resources are globally sufficient to meet the global demand, combinatorial optimization is the relevant tool; if resources are insufficient (the common case), control methods are more appropriate. Examples throughout this chapter illustrate these situations.

Resource allocation is subject to a number of known risks, which any resource management policy should consider.

- *Violation of fairness,* examples of which are the above-mentioned starvation, and priority inversion, a reversal of prescribed priorities due to unwanted interference between synchronization and priority setting.
- Congestion, a situation in which the available resources are insufficient to meet the demand. This may be due to an inadequate policy, in which resources are over-committed, or to a peak in the load. As a result, the system's time is essentially spent in overhead, and no useful work can be done, a situation known as thrashing. Thrashing is usually avoided or delayed by an admission control policy
- Deadlock, a situation of circular wait, in which a set of processes are blocked, each of them waiting for a resource that is held up by another member of the set. Deadlock may be prevented by avoiding circular dependencies (e.g., through ordered allocation), or detected and resolved, usually at some cost in progress rate.

In the context of this book, we are specifically interested in resource management for middleware systems; among these, Internet services are the subject of an intense activity, due to their economic importance. We conclude this section with a review of the main aspects of resource management for this class of systems.

12.1.3 Resource Management for Internet Services:

An increasing number of services are available over the Internet, and are subject to high demand. Internet services include electronic commerce, e-mail, news diffusion, stock trading, and many other applications. As these services provide a growing number of functions to their users, their scale and complexity have also increased. Many services may accept requests from millions of clients. Processing a request submitted by a client typically involves several steps, such as analyzing the request, looking up one or several databases to find relevant information, doing some processing on the results of the queries, dynamically generating a web page to answer the request, and sending this page to the client. This cycle may be shortened, e.g., if a result is available in a cache. To accommodate this interaction pattern, a common form of organization of Internet services is a multi-tier architecture, in which each tier is in charge of a specific phase of request processing.

To answer the demand in computational power and storage space imposed by large scale applications, clusters of commodity, low cost machines have proved an economic alternative to mainframes and high-performance multiprocessors. In addition to flexibility, clusters allow high availability by replicating critical components. Thus each tier of a cluster-based application is deployed on a set of nodes (Figure). How the application components running on the servers of the different tiers are connected together depends on the architecture of the application; examples may be found in the rest of this chapter. Nodes may be reallocated between different tiers, according to the resource allocation policy.

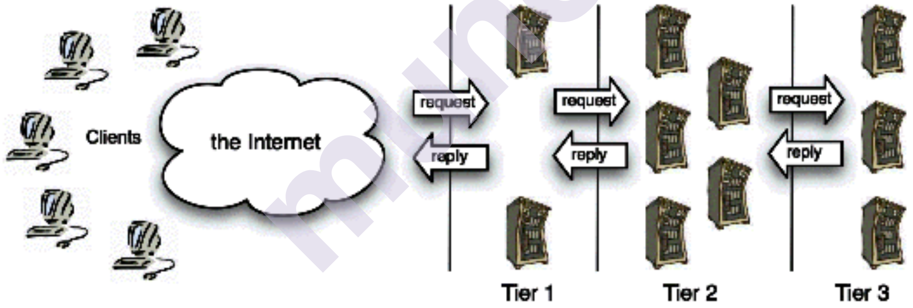


Figure: A cluster-based multi-tier application

A service provider may deploy its own cluster to support its applications. An alternative solution, in increasing use, is for the provider to host the application on a general-purpose platform (or data center) owned by a computing facility provider, and shared with other service providers. The drawback of this solution is that the service provider has less control on the fine-grain tuning of the infrastructure on which its application is running. However, there are several benefits to using shared platforms.

- The service provider is freed from the material tasks of maintaining the infrastructure.

- The fixed cost of ownership (i.e., the part of the cost that is not proportional to the amount of resources) is shared between the users of the common facility.
- Mutualizing a large pool of resources between several applications allows reacting to load peaks by reallocating resources, provided the peaks are not correlated for the different applications.
- Resource sharing improves global availability, because of redundancy in hosts and network connections.

12.4 IDENTITY AND ACCESS MANAGEMENT

Identity and Access Management (IAM) is a combination of policies and technologies that allows organizations to identify users and provide the right form of access as and when required. There has been a burst in the market with new applications, and the requirement for an organization to use these applications has increased drastically. The services and resources you want to access can be specified in IAM. IAM doesn't provide any replica or backup. IAM can be used for many purposes such as you want to secure the control of individual and group access from your AWS resources. With IAM policies, managing permissions to your workforce and systems to ensure least-privilege permissions becomes easier. The AWS IAM is a global service.

Components of IAM

Users

Roles

Groups

Policies

With these new applications being created over the cloud, mobile and on-premise can hold sensitive and regulated information, It's no longer acceptable and feasible to just create an Identity server and provide access based on the requests. In current times an organization should be able to track the flow of information and provide least privileged access as and when required, obviously with a large workforce and new applications being added every day it becomes quite difficult to do the same. So organizations specifically concentrate on managing identity and its access with the help of few IAM tools. It's quite obvious that it is very difficult for a single tool to manage everything but there are multiple IAM tools in the market that help the organizations with any of the few services given below.

Services By IAM:

Identity management

Access management

Federation

RBAC/EM

Multi-Factor authentication

Access governance

Customer IAM

API Security

IDaaS – Identity as a service

Granular permissions

Privileged Identity management – PIM (PAM or PIM is the same)

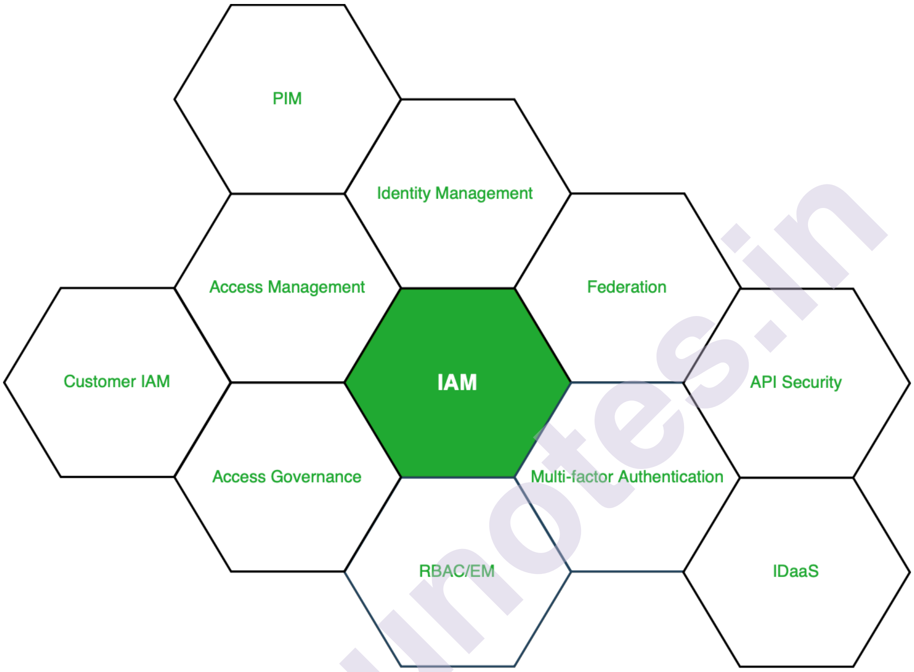


Figure : Services under IAM

More About the Services: Looking into the services on brief, Identity management is purely responsible for managing the identity lifecycle, access management is responsible for the access to the resources, access governance is responsible for access request grant and audits, PIM or PAM is responsible for managing all the privileged access to the resources. The remaining services either help these services or help in increasing the productivity of these services.

Market for IAM: Current situation of the market, there are three market leader (Okta, Saipoint and Cyberark) who master one of the three domains (Identity Management, Identity Governance and Privilege access management), according to Gartner and Forrester reports. These companies have developed solutions and are still developing new solutions that allow an organisation to manage identity and its access securely without any hindrances in the workflow. There are other IAM tools, Beyond Trust, Ping, One login, Centrify, Azure Active Directory, Oracle Identity Cloud Services and many more.

*****