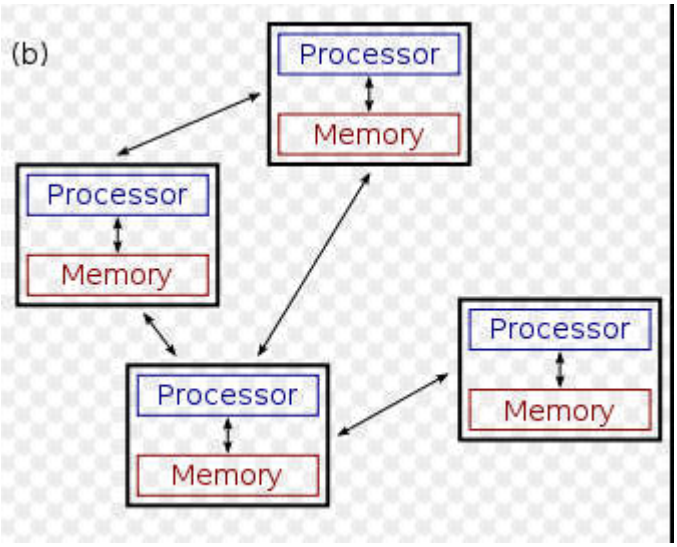
Introduction to Distributed System

Introduction to Distributed System

- A distributed system is a collection of independent computers at networked locations such that they communicate and interact only through message passing that is viewed as a single coherent system by its users.
- Generally, all the components of the distributed system interact so as to obtain a common goal.
- A computer program running in distributed system is called distributed program.
- Distributed programming is used to write distributed programs.
- It differs from parallel system as each computer in a distributed system has its own memory i.e. distributed memory; but in parallel system, all processors have access to shared memory for information exchange.



Examples of Distributed System

Internet

- Internet is the largest collection of interconnected computer networks.

- Program running on the computers interact with each other within the internet by the mechanism of message passing.
- Internet allows users to make use of services like world wide web
- World wide web is an example of distributed system as it contains multiple components under the hood that help browsers to display content but from the user's view it helps to access the web via a browser such that it looks like a single system.
- Basically, the internet is not a distributed system as it only provides essential communication and connectivity channels to the distributed system but the services provided by the internet are the distributed systems.

Characteristics of Distributed System

Concurrency :

- Distributed system allows the programs that share resources to execute concurrently(i.e. At the same time).
- It is a property that enables processes running in different machines to form a common system that is capable of executing codes on multiple machines at the same time.
- Example: A web application is concurrent as it can be used by various users at the same time.
- Concurrency helps to reduce latency and increase throughput as a unit of work can be concurrently done by sub-division.

No Global Clock :

- In other to coordinate to the remote networks it is necessary to synchronize clock in primitive technologies. But as there is no single correct global timing system, it is very difficult to synchronize global clock across all the networks of the system located at different locations.
- The main characteristics of the distributed system is that it does not

need any global clock system to coordinate with the network at different locations because the only way of communication is message passing through the network.

Independent Failures :

- The computer system can fail or crash at any time.
- In primitive centralized system, if the server fails then the whole system fails to operate leaving the negative consequences to the end users.
- With distributed system, even one network fails, it does not hamper the overall system. The workload of the failed network can be overcomed by the other networks within the system situated at different locations.
- This provides reliable system to the end users.

Advantages and Disadvantages of Distributed System

Advantages of Distributed System

- All the computer components work independently dividing the main task. This improves performance of the system and reduce latency time.
- Even if one component of the system crashes or fails, the system as a whole does not crash. This improves availability and reliability of the system.
- As load is distributed among the components of the system, it enhances the total computational power of the overall system.
- Computational power can be grown incrementally by adding small components to the system.
- It allows users to share data and resources.
- The workload is distributed among different components or machines.

- Local database administrator have different degree of local autonomy(retains control to some extent).

Disadvantages of Distributed System

- It is very difficult to implement the distributed system making it more costlier than other systems.
- The exchange of information among the components require coordination which creates processing overheads.
- It is difficult to ensure correctness of algorithms generally when some parts of system is down and is being recovered.

Design Goals

Resource Sharing

- The main goal of distributed system is to allow users to access remote resources and to share them in controlled and efficient manner.
- The resources may be printer, database, data, files and so on.
- By sharing resources among the users, we can minimize the implementational cost of the system.
- It is also necessary for easy collaboration and information exchange among the users of the system.
- Making resource sharing and information exchange easier result in increasing security related problems in the system which must be handled properly and efficiently to protect data from being compromised.

Openness

- A distributed system should be open. It must provides services with the standard rule following some protocols defining syntax and

semantic of that service.
- Services are specified through an interface defined in the Interface Definition Language (IDL).
- IDL must specify complete description of the service including function name, parameters, return values, possible exceptions and what the service is used for.
- The interface specification should be complete (All the necessary things required to implement the interface should be specified) and neutral (The structure of the implementation should not be specified).
- Open distributed system helps to achieve interoperability (Ability for two implementations to work together relying upon each other's services) and portability (Ability of an application designed for a distributed system, to execute correctly on other distributed systems with same implementations without any modification to the application).
- It also helps in adding of new components or replacing current components without affecting the system.

Transparency

- Transparency is the ability to hide the fact that the processes and the resources of the distributed system are physically distributed across multiple computers or machines and makes users to realize that it is a single coherent system.
- Transparency should be considered with other issues like performance.
- For eg: Let a person wants to print a file from the mobile device. Then it would be great if he/she could print it from a nearby printer than from printer located far from his current position. It is against the location transparency but provides reliable outcome to the system.

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

Figure 1-2. Different forms of transparency in a distributed system (ISO, 1995).

Scalability

- A distributed system should be scalable with respect to size (Able to add any numbers of users and resources), geography (Users and resources may be in any locations) and administration (Easy management even if there are many independent administraions).

Resource Sharing and Web Challenges

Resource Sharing

- Users of a system always tends to share resources like hardware (printer), data (files) and specific functioning resources (search engines).
- Users want to share data rather than the disks containing the data
- Service manages a collection of related resources and provides the functionality to the users. Eg: A file sharing is initiated by a file service providing read, write and delete operations or functions on the files.
- Services can be invoked from other computers only by communication (message passing in case of distributed system).
- Requests are sent from clients to a server via a message and the server reply to the clients via a message.
- When clients send a request for an operation to a server, it means

client invokes an operation upon the server.
- Remote invocation means the complete interaction between a client and a server from request by a client to the response given by the server.

Client vs Server

- Client and server are the process rather than the computer.
- The process which requests an operation is called client and the process that provides the requested service is called server.
- Client is active while server is passive.
- Server runs continuously while clients run until the applications lasts.
- An executing web browser is a client while the web server is a server.

World Wide Web

- It is a system to publish and access services and resources throughout the world with the help of Internet.
- Web browsers should be used to access the world wide web.
- It presents data in the form of web pages.
- Each document or web page contains hyperlinks (references to other documents and resources).

- The web is based on following standard components:
a) HTML : It is a language to specify the contents of the web pages.
b) URL : It identifies the documents and other resources stored on the web.
c) Client Server Architecture : It provides standard rules by which browsers can access to the resources from the web servers.

- URL has two top level components, scheme and scheme-specific-identifier. Eg http://www.egnitenotes.com/all_notes/ Here http is a scheme and remaining part is a scheme-specific-identifier.

- HTTP defines the way in which browsers interacts with the web servers. It is a request and reply protocol with only one resource per request.
- A program run by the web server to generate content for the clients is called common gateway interface (CGI).

Web Challenges

- If the resources are deleted or moved, the links to that resource still remains. Such links are called dangling links.
- The response of the web server to the users may be slow.
- The web page is not satisfactory user interface.

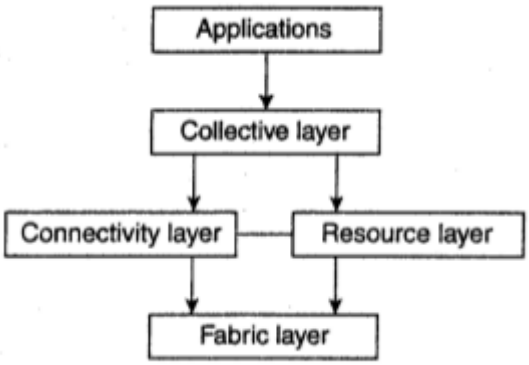
Semantic Web

- The web of linked metadata resources is called semantic web.
- Proper metadata helps the search engines to search the contents in effective and efficient way.

Grid, Cluster and Cloud System

Grid System

- Grid systems have high degree of heterogeneity. It means that the computers in the grid system must not be the same with respect to hardware, operating systems, network, security policy, administrative domain and so on.
- The resources from different organization can collaborate via a virtual organization with access rights with all the members of this virtual organization.



Architecture of Grid System

- The basic grid architecture consists of four layers. They are fabric layer, connectivity layer and resource layer, collective layer and application layer as shown in given figure.
- The fabric layer provides interface to local resources at a specific site which can be shared within a virtual organization. It provides functions to query state and capability of resources, and actual resource management.
- The connectivity layer communication protocols to support grid transactions. It supports delegation of rights from authenticated users to programs running on behalf of those users.
- The resource layer manages a single resource with the help of functions provided by connectivity layer and interfaces of fabric layer. It is responsible for access control.
- The collective layer handles access to multiple resources via resource discovery, allocation and scheduling of multiple resources along with data replication.
- The application layer consists of applications that uses the services provided by the grid computing system.

Cluster System

- Cluster system is generally used for parallel programming in which a single program is run in parallel on multiple computers within the system.

- The feature of cluster computing is homogeneity. It means all the computers within the cluster system must have same operating system and must be operated within the same network.

One well-known example of a cluster computer is formed by Linux-based Beowulf clusters, of which the general configuration is shown in Fig. 1-6. Each cluster consists of a collection of compute nodes that are controlled and accessed by means of a single master node. The master typically handles the allocation of nodes to a particular parallel program, maintains a batch queue of submitted jobs, and provides an interface for the users of the system. As such, the master actually runs the middleware needed for the execution of programs and management of the cluster, while the compute nodes often need nothing else but a standard operating system.

Cloud System

- Cloud computing system is a type of Internet-based computing that provides shared computer processing resources and data to computers and other devices on demand.
- It enables on demand access to a shared pool of configurable computing resources.
- The system can be accessed using a web browser regardless of the location and the devices used.

2. Distributed Objects and File System

Introduction to Distributed Object

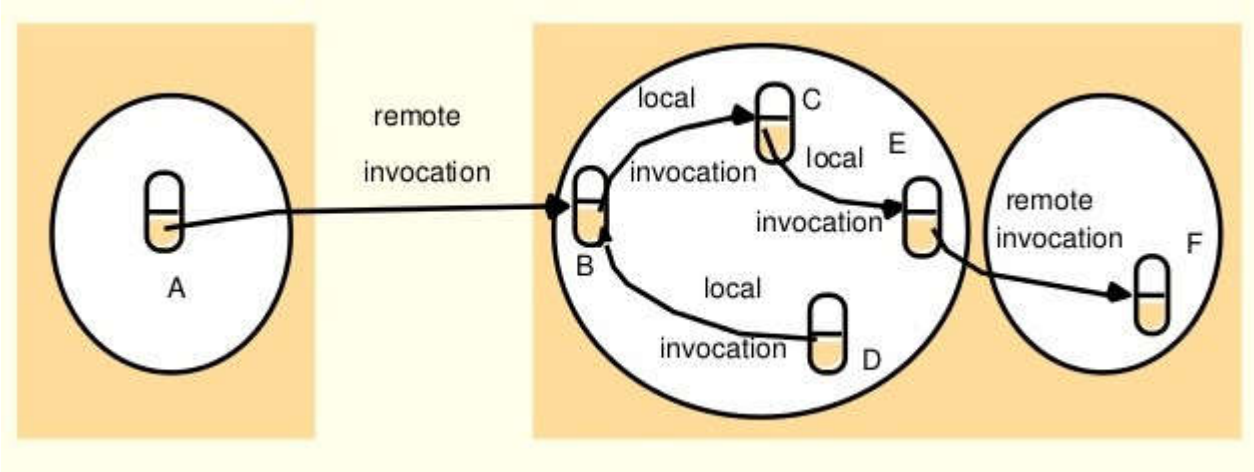
- Distributed objects are the objects with respect to object oriented programming paradigm, that are distributed across different address spaces, which may be on multiple computers within a network or on multiple processes running on a same computer.
- The distributed objects are capable to work together by sharing data and invoking methods.
- It provides location transparency. It means that the remote objects and local objects appear similar within a process or a computer.
- Distributed objects generally communicate with Remote Method Invocation (RMI).
- RMI has message passing mechanism in which one object sends message to another object in a remote machine or process to carry out some task and the results are sent back to the calling object.

Distributed Objects Communication

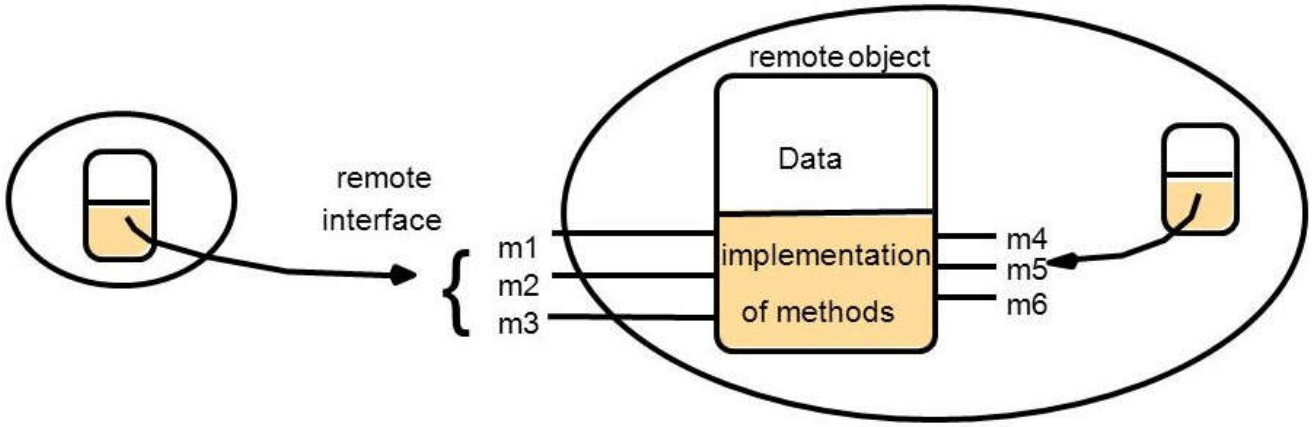
Remote Method Invocation (RMI)

- RMI is the means by which objects in different processes can communicate with one another.
- It allows object in one process to invoke or call the methods of an object in another process.
- Each process consists of a set of objects. These objects may receive local invocation or remote invocation or both.
- The method invocation between the objects in different processes within a computer or different computer is known as remote method invocation.
- The method invocation between the objects in same process is known as local method invocation.
- The objects which can receive remote invocations are called remote objects

- All the objects can receive local invocation from another object within a same process if and only if the invoking object have a reference to the invoked object.



- In the figure, object C can invoke object E locally because C has reference to E.
- The object from other process can invoke a remote object if and only if that object has access to remote object references of the remote object. In the figure, object A can invoke object B remotely; so A must have access to remote object references of B.
- Remote interfaces are the interface contained by a remote object that specifies which of the methods of remote object can be invoked remotely by object from another process.



- An action is initiated by a method invocation which may result in further invocations on methods in other objects and the objects in this chain may be in different processes. In the figure, object A can get remote object reference of F through object B.
- If garbage collection is supported, RMI allow distributed garbage collection by cooperation of local garbage collector and an added module, based on reference counting.
- RMI should be able to raise exceptions if the remote invocation between distributed objects fails.
- The request-reply message structure consists of messageType(int 0:request 1:reply), requestId(int), objectReference(RemoteObjectRef), methodId(int/Method) and arguments(array of bytes).

RMI Design Issues

RMI Invocation Semantics:

- RMI invocation works with request-reply protocol which can be implemented as retry request message(retransmit request until reply is received or server is assumed to fail), duplicate filtering(filter out duplicate requests at the server while retransmitting) and retransmission of results(keep a history of result messages to enable lost results to be retransmitted).
- In local method invocation, the semantics are exactly once (every method is executed exactly once).
RMI invocation semantics can be produced by the choice of above implementation of request-reply protocol.
- In maybe invocation semantics, the remote method may be executed once or not at all. It occurs if no fault tolerance options are adopted. It causes omission failure(message is lost) and crash failure(server fails).
- In at-least-once invocation semantics, the invoker gets result if method was executed at least once and if method was not executed, exception is received by the invoker. It may suffer from crash failure

and arbitrary failure(multiple method execution causing wrong value to be returned).
- In at-most-once invocation semantics, the invoker receives result if method was executed exactly once otherwise receives an exception.

RMI Transparency

- It involves hiding of marshalling, message passing as well as the task of locating and contacting a remote object so as to make both local and remote invocations look similar.

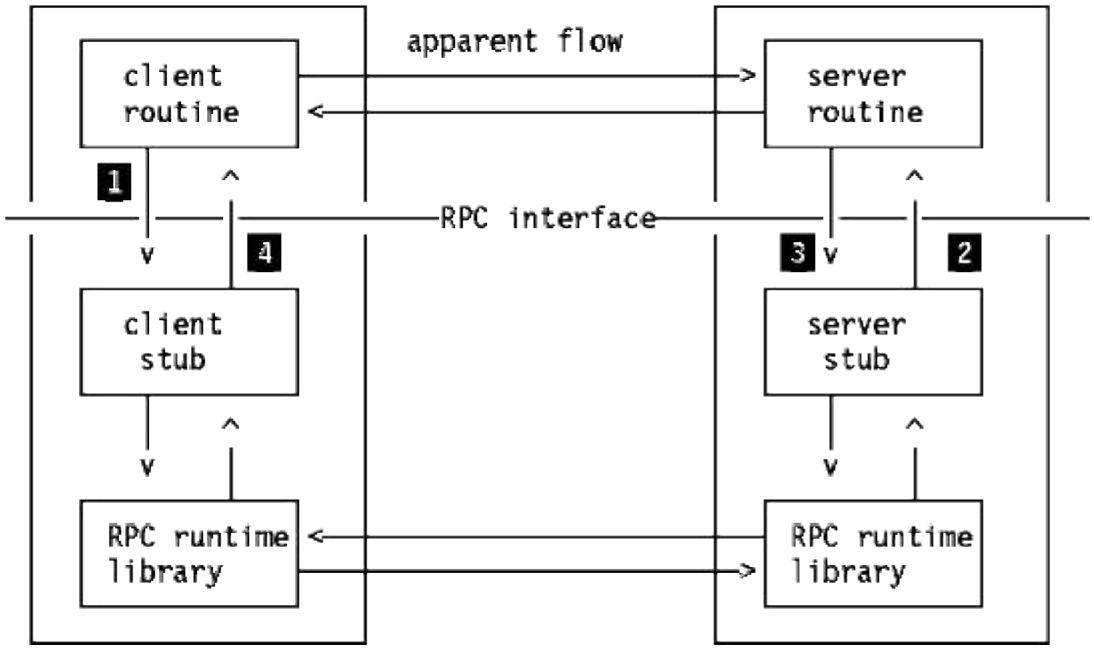
Remote Procedure Call

- RPC is a remote communication medium in which a client program calls a procedure in another program running in a server process.
- The role of server and client is designated based on the work which is temporarily assigned. It means the server may also be clients of other servers creating a chains of RPC.
- A server process consists of a service interface which defines the procedures available for remote calling.
- RPC do not consists of object references.
- RPC also lacks the ability to create new instances of objects.
- It is implemented with maybe, at-least-once and at-most-once invocation semantics.
- It works under request-reply protocol omitting the object reference from request messages.

Working of RPC

1. Client procedure calls client stub in normal way.
2. Client stub builds message, call local OS.
3. Client's OS sends message to remote OS.

4. Remote OS gives message to server stub.
5. Server stub unpacks parameters, call server.
6. Server does work, returns result to the stub.
7. Server stub packs it in message, call local OS.
8. Server's OS sends message to client's OS.
9. Client's OS gives message to client stub.
10. Stub unpacks result, returns to client.



Case Study : Sun RPC

- SUN RPC was designed for client-server communication in the Sun Network File System.
- It can be used over either UDP or TCP. Over UDP, the length of request and reply messages is restricted in length(theoretically 64 kilobytes).
- It uses at-least-once invocation semantics.
- It provides an interface language called XDR and an interface compiler rpcgen used with C language.

Interface Definition Language

- The Sun RPC provides interface language called XDR that is used to define a service interface by specifying a set of procedure definitions and supporting type definitions.
- It supplies a program number and version number rather than interface name. The program number is unique for every program and version number changes if procedure changes. The program number and version number are passed in request message to ensure both client and server are using the same version.
- A procedure definition consists of a procedure signature and a procedure number. The procedure number is passed in request message to identify which procedure is being called.
- Only a single input parameter is allowed. Structure should be implemented if multiple input parameters are required.
- The output parameters of a procedure are returned via a single result.
- The procedure signature contains result type, procedure name and input parameter type.
- The program number is 9999 and the version is 2.
- READ procedure has number 2 while WRITE has number 1.
- The rpcgen is used to generate client stub procedure; server main procedure, dispatcher and server stub procedure; XDR marshalling and unmarshalling procedures from an interface definition.

```
const MAX = 1000;
typedef int FileIdentifier;
typedef int FilePointer;
typedef int Length;
struct Data {
    int length;
    char buffer[MAX];
};
struct writeargs {
    FileIdentifier f;
    FilePointer position;
    Data data;
};
```

```
struct readargs {
    FileIdentifier f;
    FilePointer position;
    Length length;
};

program FILEREADWRITE {
  version VERSION {
    void WRITE(writeargs)=1;
    Data READ(readargs)=2;
  }=2;
} = 9999;
```

Binding

- Binding defines the way of binding server and client together during a remote procedure call.
- It runs a local binding service called port mapper at a known port of a computer.
- Each instance of a port mapper records the program number, version number and port number in use by each service.
- When a server starts, it registers program no, version no and port no with the local port mapper.
- When a client starts, it finds server's port by making remote request to port mapper at server's host specifying program and version numbers.

Authentication

- The messages in Sun RPC provide additional field to pass authentication information between client and server during critical operations like accessing the network file system.
- The supported authentication protocols are none, UNIX style, shared key and Kerberos style.

- RPC header consists of a field which indicates the authentication style used.

Introduction to DFS

- File system is necessary for organization, storage, retrieval, naming, sharing and protection of files.
- Files contain data and attributes.
- Distributed file system is a model of file system that is distributed across multiple machines.
- DFS provides services to the clients of distributed system.

Requirements of DFS

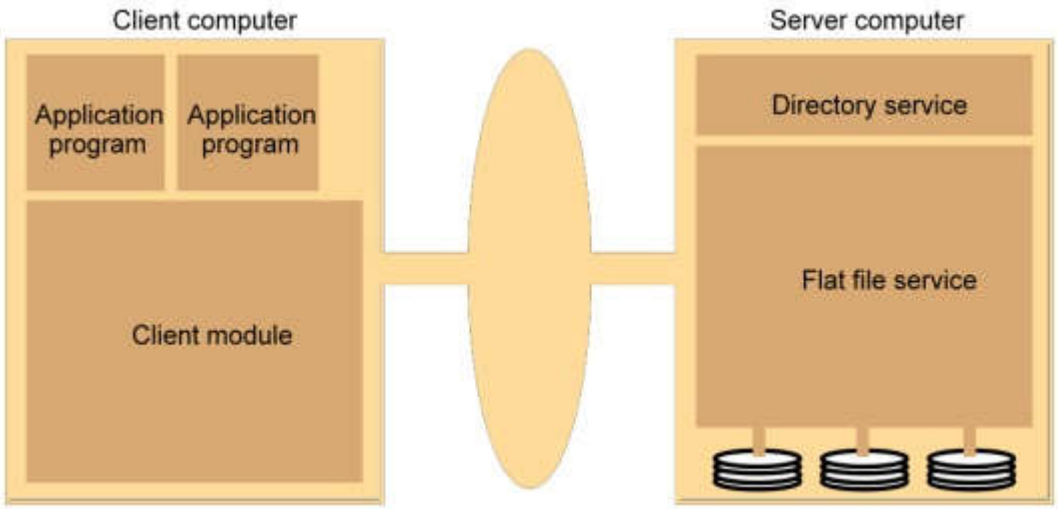
1. The design of DFS must support transparency of access, location, migration , performance, etc.
2. Concurrent file updates should be controlled.
3. It should support file replication to enhance scalability and fault tolerance.
4. The service interfaces should be defined such that they can be implemented in heterogeneous systems.
5. The service should continue to operate even in case of client or server failures.
6. It should maintain consistent states for files.
7. It should provide access control mechanisms to provide security.

Importance of DFS

1. It supports sharing of information in the form of files throughout the Intranet.
2. It enables programs to store and access remote files exactly as they do on local ones.

File Service Architecture

- Distributed file system consists of three components:
a) Flat file service
b) Directory service
c) Client module



Flat file service

- It is a component responsible for implementing operations on the file contents.
- During requests for file service operations, Unique File Identifier (UFID) is used to indicate the files.
- The various flat file service operations are: Read, write, create, delete, getAttributes and setAttributes.

Directory Service

- It is a component that provides mapping between text names for files and their UFID.

- The various directory service operations are: lookup, addName, unName and getNames.

Client Module

- It is a component running in client computer, integrating the file service and directory service operations under a single API.
- With the use of cache, client module helps to achieve high performance.

SUN Network File System

- SUN NFS is a first successful network file system developed by SUN microsystems that focuses on transparancy.

Properties

1. It is both an implementation and a specification of how to access remote files.
2. It focuses on sharing a file system in a transparent way.
3. It uses client server model. A node can act both as client and server.
4. It uses mount to make server file system visible from a client.
5. It is stateless (all client requests must be self contained).
6. It is machine and operating system independent.

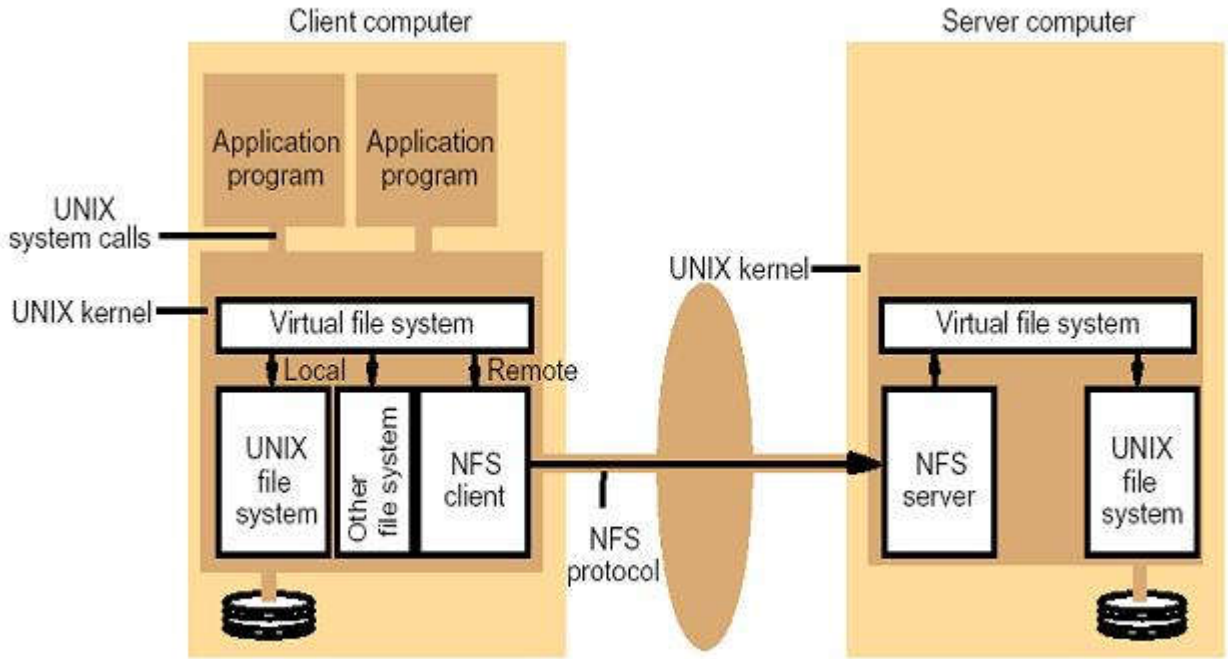
Architecture

1. Protocol:
- It uses SUN RPC mechanism and SUN external data representation (XDR) standard.
- The protocol is stateless. It enhances crash recovery.

- Each procedure call must contain all the information necessary to complete the call.

2. Server Side:
- It provides file handle consisting of:
a) Filesystem id (identify disk partition)
b) I-node number (identify file)
c) Generation number
- File system id is stored in super block.
- Generation number is stored in I-node.

3. Client Side:
- It provides transparent interface to NFS.
- Mapping between remote file name and remote file address is done at server boot time through remote mount.



Operations

1. Search for file within directory
2. Read a set of directory entries
3. Manipulate links and directories
4. Read/write file attributes
5. Read/write file data

Name Services

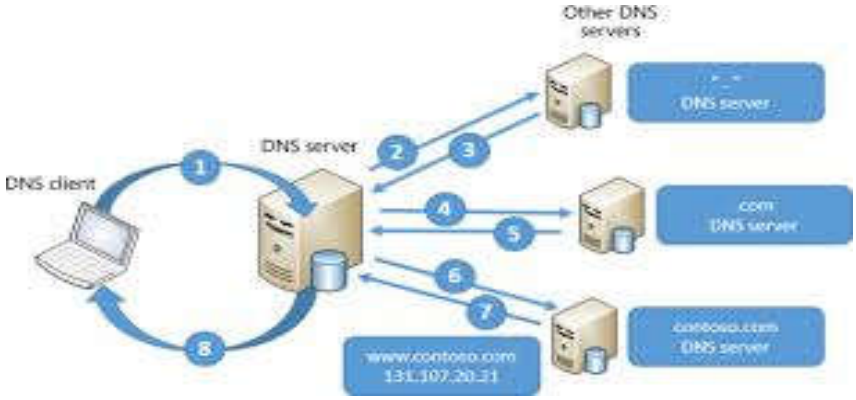
- Names are used to refer various resources.
- Resources are accessed using identifier or references.
- A name is a value that can be resolved to an identifier or an address.
- Names are used because they are human readable.
- Name services are those services that are used to resolve resource names.
- Name service returns the information about the resource when the name of a resource is given.
- Name service is responsible to store a collection of one or more naming contexts.

DNS

-DNS is the Internet naming scheme that allows resources to be accessed by using alphanumeric names.
- A valid DNS names allows; A to Z, a to z, 0 to 9 and Hyphen (-).
- The DNS is a distributed hierarchical database as shown below :
-A ‘.’ is used as a separator.

Working Mechanism:

1. Issue a DNS query to ask for IP address of www.egnitenotes.com
2. Issue a query to root name server.
3. Returns the IP address of TLD.
4. Issue a query and sent to TLD servers.
5. Reply with ns1.egnitenotes.com and IP address.
6. Issue another query and sent to ns1.egnitenotes.com
7. Reply IP address of www.egnitenotes.com
8. Return IP address of www.egnitenotes.com to client.
9. Request for web content to IP address of www.egnitenotes.com



Terminology

- A query is a request for name resolution which is directed to a DNS server.
- A recursive query is sent to DNS server and requires a complete answer.
- A iterative query is directed to DNS server which may be answered with a referral to another DNS server.
- Root hints contain IP address for DNS root servers.

Resource Records

1. A- record = Address
2. CNAME = Alias
3. NS = Authoritative name server
4. MX = Mail exchange
5. PTR = Domain Name Pointer

3. Operating System Support

OS Layer

- The OS layer is present below the middleware layer.
- OS provides problem-oriented abstractions of the underlying physical resources.
- The middleware-OS combination of a distributed system should have good performance.
- Middleware is responsible to provide proper utilization of local resources to implement mechanisms for remote invocations between objects or processes at the nodes.
- Kernel and server processes are responsible to manage resources and present clients with an interface to the resources.
- Kernal, as a resource manager, must provide encapsulation, protection and concurrent processing.
- Clients access resources by making invocations to a server object or system calls to a kernel.

Core OS Functinality

1. Process Manager:
- As a process manager, OS handles the creation of processes and the operations upon them.

2. Thread Manager:
- As a thread manager, OS is responsible for thread creation, synchronization and scheduling.

3. Communication Manager
- OS is responsible for communication between threads attached to different processes on same computer or in remote processes.
- In some OS, additional service is necessary for communication.

4. Memory Manager:
- OS also deals with management of physical and virtual memory to ensure efficient data sharing.

5. Supervisor:
- OS is responsible to dispatch interrupts, system call traps, memory management control, hardware cache and so on.

Protection

- The underlying resources in a distributed system should be protected from illegitimate access.
- To ensure protection of resources, operating system must provides a means to provide clients to perform operations on the resources if and only if they have rights to do so.
- For example: Consider a file with only read or write operations. The illegitimate access would be access of file for write by the client who have read access only.
- Resources can be protected by the use of type-safe programming language like JAVA in which no module can access a traget module without having a reference to it.
- Hardware support can be employed to protect modules from illegitimate invocations, for which kernel should be implemented.

Kernel:

- A kernel is a program that is executed with complete access privileges for the physical resources on its host computer.
- It controls memory management.
- It ensures access of physical resources by the acceptable codes only.
- A kernel process executes in supervisor mode and restricts other processes to execute in user mode.
- It sets up address space for the process. A process is unable to access memory outside its address space.
- A process can switch the address space via interrupt or system call trap.

Monolithic and Micro Kernel:

- Monolithic kernel is a single large process running in a single address space.
- All kernel services executes in the kernel address space.
- It provides faster execution as there is no necessity of address space switching to execute kernel services.
- The device drivers reside in kernel space making it less secure.
- Fault in a single kernel service collapse the whole kernel.
- Adding new features require recompilation of whole kernel.
- Eg: Kernel of Unix and Linux

- Microkernel is a kernel in which it is broken into separate process called servers.
- Servers run in kernel space and user space.
- It provides slower execution.
- Device drivers reside in user space.
- Fault of a single server does not collapse the kernel.
- Eg: Kernel of Mac OS X and Windows NT

Selection of Kernel:

- Micro kernel is suitable for distributed operating system.
- The services of distributed system are complex and segregation of micro kernel makes it easy.
- Micro kernel provides faster communication among the processes with low overhead.

Process and Thread

- Process is an instance of a computer program that is being executed.
- In traditional OS, each process has an address space and a single

thread of control.
- The distributes systems require internal concurrency for which process is multi threaded.
- Thread is a light weighted process that shares the same address space of the corresponding process but runs in quasi-parallel.
- Thread is the operating system abstraction of an activity.
- A process contains execution environment which is local kernel managed resources to which its threads have access.

Importance of Thread in Distributed System

1. Distributed system needs to support multiple users. Such concurrency is impossible without multi-threaded support.
2. Thread is able to block system calls without blocking the entire process. This makes distributed system possible to communicate by maintaining multiple logical connections at the same time.
3. Thread can run in a single address space parallel on different CPU.
4. Threads can share a common buffer which make it possible to implement producer-consumer problem easily.

Comparison of Process and Thread

1. Processes run in separate memory space but threads can run in a same memory space.
2. Process is a self-contained entity while thread depends on process for existence.
3. Process depends on resources heavily while threads require minimal amount of resources.
4. Threads within a process can communicate easily but processes must use IPC for communication.
5. Process has considerable overhead while thread has no overhead.

OS Architecture

Network OS and Distributed OS

- Network OS has networking capability.
- They can be used to access remote resources.
- Each node has its own system image and a user is capable to log in to another computer and run processes there.

- Distributed OS is an operating system that produces a single system image for all the resources in the distributed system.
- Users are never concerned with where their programs run.
- The OS has control over all the nodes in the system.

Preference of NOS over DOS:

1. Users invest in applications to meet their current problem solving needs. So, they do not tend to adapt to a new operating system which is unable to run their applications even if they are more efficient.
2. Users always prefer to have a degree of autonomy for their machines, even in a closely knit organization. This is because they do not want to spoil their process performance due to the process run by other users.

4.Time and State in Distributed System

Time in DS

- Each machine in a distributed system has its own clock providing the physical time.
- The distributed system do not have global physical time.
- Time synchronization is essential to know at what time of day a particular event occurred at a particular computer within a system.

Physical Clock

- Each computer contains an electronic device that counts oscillations in a crystal at a definite frequency and store division of count to frequency in a register to provide time. Such device is called physical clock and the time shown is physical time.
- Since, different computers in a distributed system have different crystals that run at different rates, the physical clock gradually get out of synchronization and provide different time values.
- Due to this, it is very difficult to handle and maintain time critical real time systems.
- Consistency of distributed data during any modification is based on time factor.

Synchronization of Physical Clocks

The algorithms for synchronization of physical clocks are as follows:
1. Christian's method
2. Berkeley's method
3. Network time protocol

Cristian's Method

- It makes use of a time server to get the current time and helps in synchronization of computer externally.
- Upon request, the server process S provides the time according to its clock to the requesting process p.
- This method achieve synchronization only if the round trip times between client and time server are sufficiently short compared to the required accuracy.

Algorithm:
- A process p requests time in a message mr and receives time value t in a message mt. Process p records total round trip time T(round) taken to send request mr and receive reply mt.
- Assuming elapsed time is splitted before and after S placed t in mt, the time estimate to which p should set its clock is t + T(round)/2.
- Assuming min as the earliest point at which S could have placed time mt after p dispatches mr, then:
a) Time by S's clock when reply arrives is in range [t + min, t + T(round) – min]
b) Width of time range is T(round) – 2 * min
c) Accuracy is +- (T(round) /2 – min)

Discussion:
- If a time server fails, the synchronization is impossible.
- To remove this drawback, time should be provided by a group of synchronized time servers.

Berkeley's Algorithm

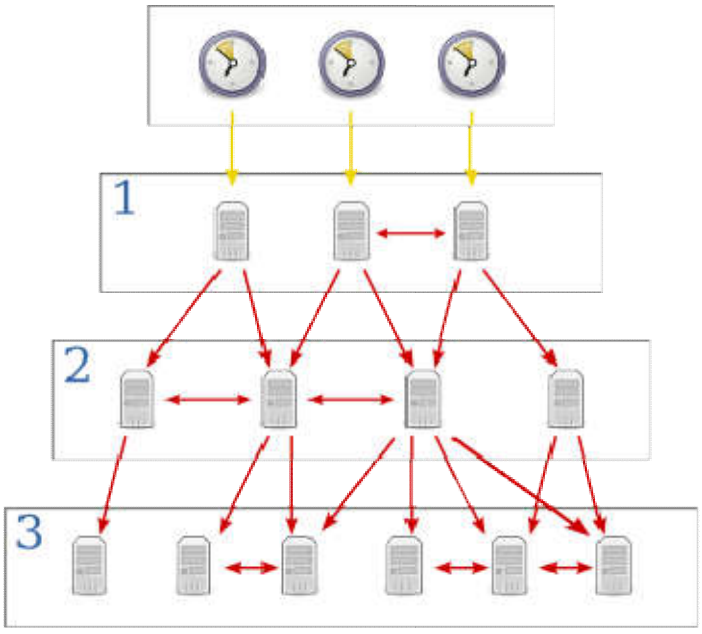
- It is an algorithm for internal synchronization.
- A computer is chosen as a master.
- All other computers are slaves.
- Master periodically polls for the time of slaves and the slaves send back their clock values to master.
- The master estimates local time of each slave by observing the round-trip times.

- Master calculates average of obtained time including its own time.
- While calculating average, it eliminates faulty clocks by choosing a subset of clocks that do not differ from one another by more than a specified amount.
- The master then sends the amount by which each slave should adjust their clock which may be positive or negative.
- If the master fails, one of the slaves can be elected to take the place of master.

Network Time Protocol (NTP)

- It defines an architecture to enable clients, across the Internet to be synchronized accurately to UTC.
- It synchronizes against many time servers.

Design Aims:
- Adjust system clock close to UTC over Internet.
- Handle bad connectivity
- Enable frequent resynchronization
- Security



Hierarchical Structure of NTP:

- NTP is provided by a network of servers located across the Internet.
- Primary servers are connected directly to a time server.
- Secondary servers are synchronized with primary servers.
- The logical hierarchy of server connection is called synchronization subnet.
- Each level of synchronization subnet is called stratum.
- Lowest level executes in user's workstation.
- Server with high stratum numbers are liable to have less accurate clocks.

Server synchronization can be done in following ways:
1. Multicast mode:
- Servers periodically multicasts time to other servers in the network.
- Receivers set their clock assuming small delay.

2. Procedure Call mode:
- One server accepts requests from other computers.
- Server replies with its timestamp.

3. Symmetric mode:
- A pair of servers on higher subnet layers exchange messages to improve accuracy of synchronization over time.

Working of NTP

- It calculates offset o and delay d as:

$$o_j = T(B) - T(A) = [(T_2 - T_1) + (T_4 - T_3)] / 2$$
$$d_j = T(ABA) = [(T_4 - T_1) - (T_3 - T_2)]$$

- The real offset o is :

$$[o_j - d_j / 2] \quad <= \quad o \quad <= \quad o_j + d_j / 2]$$

Logical Clock

- Logical clock is a virtual clock that records the relative ordering of events in a process.
- It is a monotonically increasing software counter.
- It is realized whenever relative ordering of events is more important than the physical time.
- Physical clocks are not needed to be synchronized.
- The value of logical clock is used to assign time stamps to the events.

Lamport Logical Clock

- The ordering of events is based on two situations:
1. If two events within a same process occurred, they occured in the order in which that process observes.
2. Whenever a message is sent between processes, the event of sending message occurred before the event of receiving the message.

- Lamport generalizes the two conditions to form happened-before relation, denoted by -->
i.e a --> b ; meaning that event a happened before event b

- According to Lamport:
1. If for some process p_i: a --> b, then a --> b.
2. For any message m, send(m) --> receive(m)
3. If a, b and c are events such that a-->b and b-->c, then a --> c.
4. If a --> b, event a casually affects event b.
5. If a --> e and e --> a are false, then a and e are concurrent events, which can be written as a || e.

Implementation Rules:
1. C_{pi} is incremented before each event is issued at process P_i.
$C_{Pi} := C_{Pi} + 1$

2. a) When send(m) is a event of process P_i, timestamp $t_m = C_{P(s)}$ is included in m.
b) On receiving message m by P_j, its clock C_{Pj} is updated as:
$C_{Pj} := \max [C_{Pj}, t_m]$
c) The new value of C_{Pj} is used to timestamp event receive(m) by P_j

Problems:
1. Lamport's logical clock impose only partial order on set of events but pairs of distinct events of different processes can have identical time stamp.
2. Total ordering can be enforced by global logical time stamp.

Vector Clock

- Vector clock is a clock that gives ability to decide whether two events are causally related or not by looking at their time stamps.
- A vector clock for a system of N processes is an array of N integers.
- Each process keeps its own vector clock V_i used to time stamp local events.
- The disadvantage is that it takes more amount of storage and message payload proportional to the number of processes.

Rules for clock update

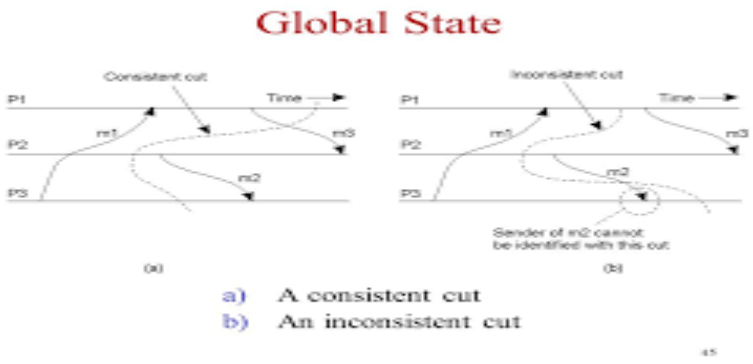
1. Initially $V_i[j] = 0$ for i, j = 1, 2, 3, ... N
2. Just before P_i time stamps an event, it sets $V_i[i] := V_i[i] + 1$
3. P_i includes the value $t = V_i$ in every message it sends.
4. When P_i receives a time stamp t in a message, it sets $V_i[j] := \max(V_i[j] , t_i[j])$, for j = 1, 2, 3, .. N

Global State and State Recording

- In a distributed system, global state consists of local state of each process (message sent and message received) and state of each channel (message sent but not received).
- The challenge is to record the global state.
- The problem is due to lack of global clock because of which the local states are recorded at different time for each process.

Consistent Global State

- A global state is consistent if for any received message in the state the corresponding send is also in the state.
- A cut C is said to be consistent, if for each event it contains, it also contains all the events that happened before that event.



Distributed Snapshot Algorithm

Assumptions:
1. Communication is reliable so that every message sent is eventually received exactly once.
2. Channels are unidirectional and provide FIFO ordered message delivery.
3. The graph of processes and channels are strongly connected.
4. Any process may initiate a global snapshot at any time.

Algorithm:

Marker receiving rule for process P_i :

On P_i's receipt of a marker message over channel c:

if (P_i has not yet recorded its state) it:

records its process state;

records state of c as empty set;

turns on recording of messages over incoming channels;

else

P_i records state of c as set of messages received over c;

end if

Marker sending rule for process P_i:

After P_i has recorded its state, for each outgoing channel c:

P_i sends one marker message over c before it sends any other message over c;

5. Coordination and Agreement

Mutual Exclusion in DS

- Mutual exclusion is a mechanism that prevent interference and ensure consistency when accessing the resources by a collection of processes.
- In distributed system, shared variables i.e. semaphores and local kernel can not be used to implement mutual exclusion. Because of this, distributed mutual exclusion arises which is based only on message passing.
- The basic requirements for mutual exclusion mechanism are:
1. At most one process may execute in the critical section at a time.
2. A process is granted entry to critical section if no any other process is executing within critical section.

Mutual Exclusion Algorithms

Non- Token Based Algorithm

- Each process competes for gaining access to use the shared resources.
- Requests are arbitrated by central control site or by distributed agreement.
- It includes central coordinator algorithm and Ricart-Agrawala algorithm.

Central Coordinator Algorithm

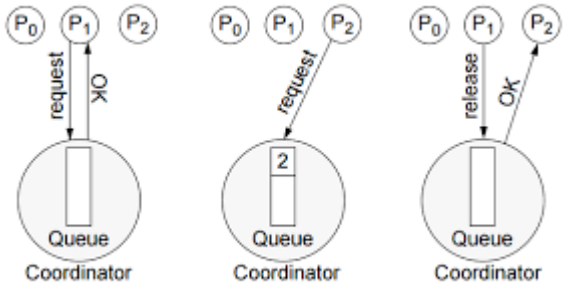
- A central coordinator is present within the distributed system which is responsible to grant access to enter a critical section.
- The coordinator makes a request queue to handle multiple requests.

Algorithm:
1. A process sends a request message to coordinator to enter critical

section and continues its other work while waiting for a reply.
2. The reply from coordinator gives right to access critical section which is based on request queue.
3. After finishing critical section operation, the process notifies coordinator with a release message.

Advantages:
1. It is easy to implement.
2. It requires only three messages per access of critical section.

Disadvantages:
1. The performance of the system may degrade.
2. If a coordinator crashes, a new coordinator must be created using election algorithm.



Ricart-Agrawala Algorithm

- It makes use of distributed agreement to implement mutual exclusion.
- All the processes are assumed to keep Lamport's logical clock.
- The process requiring access to critical section multicast request message to all other process competing for same resource.
- A process enters critical section when all processes reply to the message.
- Each process keeps its state with respect to critical section as released, requested or held.

Algorithm:

```
1. Rule for initialization (performed by each process Pi)
        Statepi := RELEASED
2. Rule for access request to CS
       Statepi := REQUESTED
       Tpi := Value of local logical clock
       Pi sends request message of form ( Tpi, i) to all
processes
       Pi waits until it receives replies from all other
n-1 processes
3. Rule for executing the CS
      statepi := HELD
4. Rule for handling incoming requests
       if statepi = HELD or [(statepi = REQUESTED) and
{(Tpi, i) < (Tpj, j)}] then:
                    queue request from Pj without replying
       else
                reply immediately to Pj
       end if
5. Rule for releasing CS
       statepi :=RELEASED
       Pi replies to all queued requests.
```

Problems
1. It is expensive to handle message traffic [2(n-1) messages]
2. Failure of one involved process blocks the progress.

Token Based Algorithm

- A logical token is passed among the processes.
- A token represents access right to the shared resources.
- The process which holds the token is granted access to critical section.
- It includes Ricart-Agrawala second algorithm and token ring algorithm.

Ricart-Agrawala second Algorithm

- A process can access critical section if it has token.
- To get token, it sends request to all other processes with logical clock and its identifier.
- When P_i leaves critical section, it passes token to the one of the waiting process based on request queue. If no process is in queue, P_i retains the token.
- Each process P_i records time stamp corresponding to last request from P_i in P[j] and token records timestamp of P_i's last holding of token. If request P[j] > token[j]; P_i has pending request.

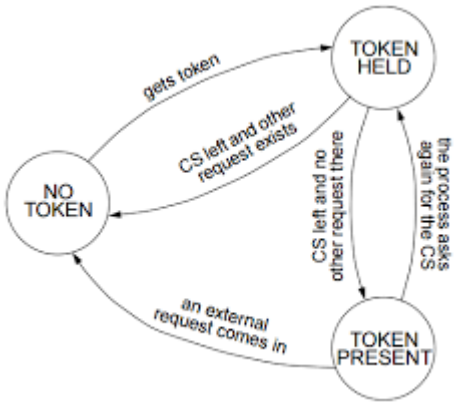
Algorithm:

```
1. Rule for process initialization
         state_Pi := NO-TOKEN for all processes P_i except
one process P_x for which state_Px := TOKEN-PRESENT


         token[k] initialized to 0 for all k =  1 to n
         request_Pi [k] initialized to 0 for all P_i and k
= 1 to n


2. Rule for access request and execution of CS
         if state_Pi = NO-TOKEN then
                  P_i sends request message to all processes
in the form (T_Pi, i)
                  P_i waits until it receives token
        end if
        state_Pi := TOKEN-HELD
        P_i enters CS
```

```
3. Rule for handling incoming requests
        request_Pi[j] := max (request_Pi[j], T_Pi)
        if state_Pi = TOKEN- PRESENT then
                        P_i releases the resource
       end if


4. Rule for releasing CS
       state_Pi := TOKEN-PRESENT
       for k = [i+1, i+2, ....., n, 1, 2,
....., i-2, i-1] do
              if request_Pi[k] > token[k] then
                       state_Pi := NO-TOKEN
                       token[i] := C_Pi
                       P_i sends token to P_k
                       break
               end if
         end for
```

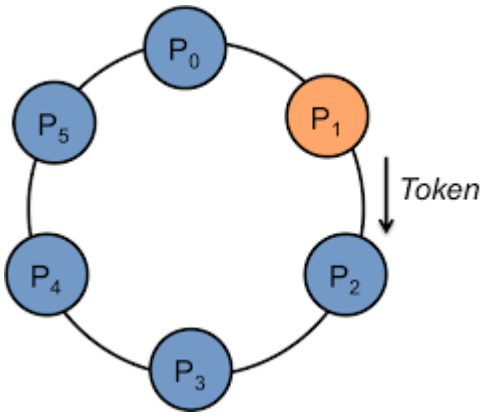
Advantages:
1. It requires only n-1 requests and one reply.
2. Failure of process which is not holding token does not prevent progress.

State Diagram:
The state diagram is shown in given figure:



Token Ring Algorithm

- The n processes are arranged in a logical ring as shown in given figure:



Algorithm:
1. Token is initially given to one process.
2. When a process requires to enter CS, it waits until it gets token from its left neighbor and retains it. After it left CS, it passes token to its neighbor in clockwise direction.
3. If a process gets token but does not require to enter CS, it immediately passes token along the ring.

Problem:
1. It adds load to the network as token should be passed even the process do not need it.
2. If one process fails, no progress is possible until the faulty process is extracted from the ring.
3. Election process should be done if the process holding the token fails.

Distributed Election

- Election algorithm is an algorithm for choosing a unique process to play a particular role.
- Any process can be elected but only one process must be elected and all other processes must agree on it.
- Election started after a failure of current coordinator.
- Election process has two phases:
1. Selection of new leader
2. Information to all processes about the elected process

Bully Algorithm

- Any process can crash during the election.
- It assumes that the system is synchronous.
- It assumes that each process knows identifiers of other processes and it can communicate with process with higher identifier.
- It has three types of messages : election message, answer message and coordinator message.

- When a process P_i detects a failure and election is to be held, it sends election message to all processes with higher identifier and waits for the answer message.
1. If no answer arrives within a time limit, P_i becomes coordinator and sends coordinator message to all other processes.
2. If answer message arrives, P_i knows another process has to be a coordinator and waits for coordinator message. If coordinator message fails to arrive within a time limit, P_i resends the election message.

Algorithm:

```
1. Rule for election process initiator
          state_Pi := ELECTION-ON
          P_i sends election message to processes with
higher identifier
          P_i waits for answer message
          if no answer message before timeout then
                    P_i is coordinator and sends coordinator
message to all processes
          else
                    P_i waits for coordinator message
                    if no coordinator message arrives before
timeout then
                              restart election procedure
                    end if
          end if


2. Rule for handling incoming election message
          P_i replies with an answer message to P_j
          if state_Pi := ELECTION-OFF then
                    start election procedure
          end if
```

Best Case Scenario:
- The process with second highest identifier notices coordinator's failure, it then selects itself as coordinator and sends (n-2) coordinator messages.

Worst Case Scenario:
- The process with lowest identifier indicates election. It sends (n-1) election messages to processes which themselves initiate an election. So, $O(n^2)$ messages are required.

Ring Based Algorithm

- The processes are arranged in a logical ring.
- Each process knows address of one other neighbor process in the clockwise direction.
- It assumes no any failures during election.
- It assumes that the system is asynchronous.
- By default, the state of a process is NON-PARTICIPANT

Algorithm:

```
1. Rule for election process initiator
           state_Pi := PARTICIPANT
           P_i sends election message with message.id := i
to its neighbor


2. Rule for handling incoming election message
           if message.id > j then
                   P_j forwards received election message
                   state_Pj := PARTICIPANT
           else if message.id < j then
                    if state_Pj = NON-PARTICIPANT then
                         P_j forwards election message
with message.id = j
                         state_Pj := PARTICIPANT
                    end if
              else
                    P_j is the coordinator and sends
elected message with message.id := j to its neighbor
                     state_Pj := NON-PARTICIPANT
               end if


3. Rule for handling incoming elected message
              if message.id != i then
                    P_i forwards received message
```

```
                    state_Pi := NON-PARTICIPANT
                end if
```

Average Case:
- n/2 message need to reach maximal node.
- n message to return maximal node.
- n message to rotate elected message.
- Total message : 2n + n/2

Worst Case:
- n-1 message to reach maximal node
- Total message : 3n -1

6. Replication

Replication and its Reasons

- Replication is the mechanism of maintenance of multiple copies of data at multiple computers.
- It helps to make distributed system effective by enhancing performance.

Reasons for Replication

1. Performance enhancement
- The copy of data placed in the proximity of the process using that data decreases the time to access the data.
- This enhances performance of the distributed system.
- Eg: Web browsers store a copy of previously fetched web page locally as a cached data to reduce latency of fetching resources from the server.

2. Increased availability
- Users always want the services to be highly available.
- Replication helps in data redundancy i.e. availability of data even if the server fails.
- If each of n servers has independent probability p of crashing, then the availability of resources at each server is $1-p^n$

3. Fault Tolerance
- Even if one server fails, the data on other servers are provided to the users.

Problems with Replication

Inconsistency:
- If a copy is modified, the copy becomes inconsistent from the rest of the copies.
- To ensure consistency, all the copies should be modified.

Consistency Models

- Consistency model is the contract between processes and the data store.
- Whenever a read operation is done, it should return a value showing the last write operation on data. But due to lack of global clock, it is difficult to determine last write operation.

Types of Consistency
---
1. Strict Consistency
- Any read to shared data returns value stored by the most recent write.
- It makes use of absolute time ordering of all shared resources.
- It is unrealistic for distributed system as it requires absolute global time.

2. Sequential Consistency
- The result of any execution is same as if operations of all processes were executed in some sequential order.

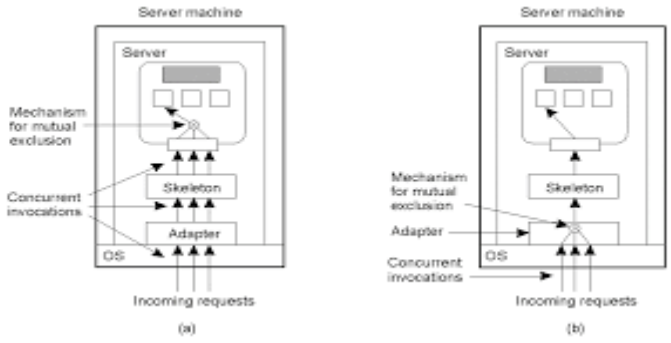
3. Causal Consistency
- Writes that are causally related must be seen by all the processes in the same order.

Object Replication

- Data in the distributed system contains collection of items. These items are called objects.
- An object can be file or object generated by programming paradigm.
- Object replication is the mechanism to form physical replicas of such objects, each stored at single computer and tied to some degree of consistency.

Problem:
- If objects are shared, the concurrent accesses to the shared objects should be managed to guarantee state consistency.

Solution:
- The process of handling concurrent invocations are shown in given figure:



7.Transaction and Concurrency Control

Transactions and Nested Transactions

- Transaction is a sequence of requests to a server by clients that ensures all the objects to remain in consistent state.
- Atomic transaction is necessary when:
1. It must be free from interference by other operations.
2. Either all operations must be completed successfully or they must have no effect in case of server crash.
- A transaction is created and managed by a coordinator, which implements coordinator interface.

```
Operations:
     openTransaction() --> trans;
               //starts new transaction and delivers
unique TID trans.
    closeTransaction(trans) --> (commit, abort);
             // commit return value indicates transaction
has committed.
           // abort return valuue indicates transaction
has aborted.
    abortTransaction(trans);
             //abort the transaction
```

Problems of Concurrent Transaction

1. Lost update problem:
- Assume initial banance of A,B and C be 100$, 200$ and 300$.

Transaction T:	Transaction U:
balance = b.getBalance();	balance = b.getBalance();
b.setBalance(balance*1.1);	b.setBalance(balance*1.1);
a.withdraw(balance/10);	c.withdraw(balance/10);
balance = b.getBalance(); //$200	

	balance = b.getBalance(); //$200
	b.setBalance(balance*1.1); //$220
b.setBalance(balance*1.1); //$220	
a.withdraw(balance/10); //$80	
	c.withdraw(balance/10); //$280

2. Inconsistent Retrieval Problem:
- Initial balance of A and B be 200$ and 200$.

Transaction T:	Transaction U:
a.withdraw(100);	aBranch.branchTotal();
b.deposit(100);	
a.withdraw(100); //$100	total = a.getBalance(); //$100
	total = total + b.getBalance(); //$300
b.deposit(100); //$300	

Nested Transaction

- The transaction which is composed of other transactions as per the requirement is called nested transaction.
- The outermost transaction is called top level transaction and others are called sub-transactions.
- The sub-transactions at same level can run concurrently but their access to common objects is serialized.

Rules for commit

1. A transaction may commit or abort only after its child transactions have completed.
2. When a sub-transaction completes, it makes independent decision either to commit provisionally or to abort.
3. When a parent aborts, all of its sub-transactions are aborted.
4. When a sub-transaction aborts, parent can decide whether to abort or not.
5. If top-level commits, all the sub-transactions that have provisionally committed can commit if none of their ancestors has aborted.

Locks

- The use of exclusive locks helps to serialize access to the objects.
- The server attempts to lock any object that is about to be used by any operation of a client's transaction.
- If client requests access to locked object, it must wait.

```
Consider example of Tand U as before:


Transaction T:
Transaction U:
Operations                                  Locks
Operations                                Locks
-------------------------------------------------------
---------------------------------------
openTransaction
balance = b.getBalance();          Lock B
b.setBalance(balance*1.1);
openTransaction
a.withdraw(balance/10);            Lock A
balance = b.getBalance(); Wait for T's lock on B


closeTransaction                          Unlock A,B

Lock B

b.setBalance(balance*1.1);

c.withdraw(balance/10);          Lock C

closeTransaction                        Unlock B,C
```

- A transaction is not allowed any new locks after it has released a lock.
- First phase is growing phase and second phase is shrinking phase.

- For concurrent transaction reading or single transaction writing but not both; two locks are used as read lock and write lock

---

Lock Compatibility

		Lock Requested	
		Read	**Write**
Lock already set	**None**	Ok	Ok
	Read	Ok	Wait
	Write	Wait	Wait

Deadlock

- The use of locks can lead to deadlock.
- The situation in which two transactions are waiting and each is dependent on the other to release a lock so as to resume is called deadlock.

Deadlock Prevention:
- Lock all the objects used by a transaction when it starts.

Deadlock Detection:
- A wait for graph is analyzed to detect deadlock by finding cycles.
- If deadlock is present, a transaction is selected for abortion.

Optimistic Concurrency Control

- Transactions are allowed to proceed as though there were no possibility of conflict with other clients until it completes its task and issues a closeTransaction request.
- If conflict arises, some transaction will be aborted and should be restarted by the client.
- Each transaction has three phases:
1. Working phase
2. Validation phase
3. Update phase

Working phase

-Each transaction get copy of most recently committed version of the object.
- Read operation is performed immediately.
- Write operation record new values as tentative values.
- A same object can have several tentative values.

Validation Phase

- When closeTransaction is received, the transaction is validated to confirm whether or not there is conflicts.
- On successful validation, transaction can commit.

Update Phase

- If transaction is validated, all the tentative values are made permanent.

Problems

- Starvation prevails.

Validation of Transaction

- Each transaction is assigned a transaction number when it enters validation phase.
- A transaction with number T_i precedes a transaction with number T_j if $i < j$.
- For transaction T_j to be serializable with respect to T_i, their operations must conform the rules as:

1. T_j - write and T_i - read
- T_j must not read objects written by T_i

2. T_j - read and T_i - write
- T_j must not read objects written by T_i

3. T_j - write and T_i - write
- T_j must not write objects written by T_i and vice versa.

Timestamp Ordering

- Each transactions's operation is validated when it is carried out.
- If operation can not be validated, the transaction is aborted.
- Each transaction is assigned a unique timestamp value when it starts.
- A transaction's request to write an object is valid only if that object was last read and written by earlier transactions.
- A transaction's request to read an object is valid only if that object was last written by earlier transactions.

Write Rule

```
if (Tc >= maximum read timestamp on D && Tc > write
timestampon committed version of D ):
        perform write operation on tentative version of
D with write time stamp Tc
else
        abort Tc
```

Read Rule

```
if (Ti write timestamp on committed version of D) {
            Let Dsel be version of D with max write
timestamp <= Ti
            if (Dsel is committed)
                    perform read on Dsel
           else
                   wait or abort then reapply
}
else
          abort Ti
```

Comparison of Concurrency Control Methods

- Timestamp and lock use pessimistic approach.
- Time stamp is better than lock for read only transactions.
- Lock is better when operations are predominantly updates.
- Time stamp aborts transaction immediately.
- Locking makes the transaction wait.
- With optimistic, all transactions are allowed to proceed.

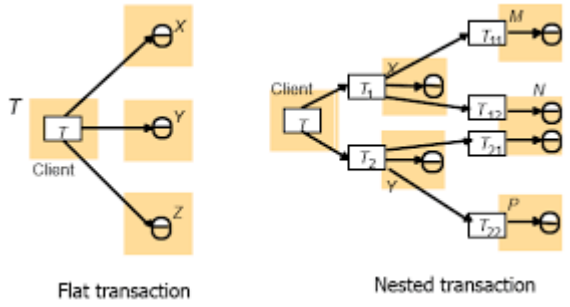
Introduction to Distributed Transaction

- Those transactions that access objects managed by multiple servers is called distributed transaction.
- It requires either all of the servers involved commit the transaction or all of them abort the transaction.
- One of the servers acts as a coordinator which ensure same outcome at all the servers.

Flat and Nested Distributed Transaction

- In flat transaction, a client makes requests to more than one server.
- It completes each of its requests before going on to next one, accessing server objects sequentially.

- In nested transaction, the top level open sub transactions; each of which can further open sub transactions.
- The sub transactions at same level run concurrently.



Atomic Commit Protocol

Two Phase Commit Protocol

- In first phase (voting phase)
1. The coordinator sends 'canCommit?' request to each participants in the transactions.
2. When participants receive 'canCommit?' request, it replies with Yes or No to the coordinator. Before voting Yes, it prepares to commit by saving objects in permanent storage. If the vote is No, the participants abort immediately.

- In second phase (completion phase)
1. The coordinator collects votes including its own vote. If all the votes are Yes, the coordinator sends 'doCommit' to each of the participants. Otherwise the coordinator sends 'doAbort to all the participants that voted Yes.
2. When participants receive 'doCommit' or 'doAbort', they act accordingly. If they receive 'doCommit' request, then they make 'haveCommitted' call as confirmation to the coordinator

8.Fault Tolerance

Introduction to Fault Tolerance

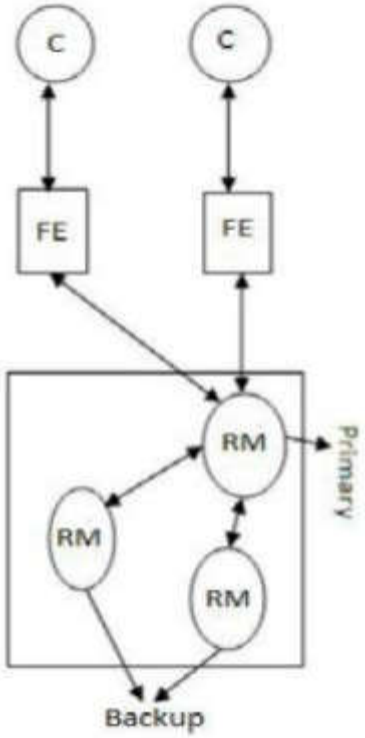
- Fault means defects within hardware or software unit.
- Error means deviation from accuracy.
- Failure is the condition occured when error results system to function incorrectly.
- Fault tolerance means ability to provide services despite of faults occur in some node of the system.
- A system is said to be k-fault tolerant if it is able to function properly even if k nodes of the system suffer from concurrent failures.

Dependabililty requirements of fault tolerant system

1. Availability (system should be available for use at any given time)
2. Reliability (system should run continuously without failure)
3. Safety (temporary failures should not result in a catastrophe)
4. Maintainability (a failed system should be easy to repair)
5. Security (avoidance or tolerance of deliberate attacks to the system)

Types of Faults

1. Node fault
2. Program fault
3. Communication fault
4. Timing fault



Primary backup replication or Passive replication

- At any point of time, there is a single primary replica manager and one or more secondary replica managers or backups.
- The front end of the system is able to communicate only with the primary replica manager to obtain the service.
- The primary replica manager executes the operations and sends updated data to the backups.

Event Sequence:
1. Request : The FE issues request to the primary RM with unique identifier.
2. Coordination : Primary RM takes the request in the order in which it receives it and re sends the response if already executed.
3. Execution : The primary RM executes request and stores response.
4. Agreement : If request is an update, primary RM sends updated state, response and unique identifier to all the backups. The backups

send an acknowledgement.
5. Response : The primary RM responds to front end.

Advantages:
1. It is simple and easy to implement.
2. It can be implemented even if primary RM behaves non-deterministically.

Disadvantages:
1. It provides high overhead.
2. If primary RM fails, more latency is incurred as new view is formed.
3. It can not tolerate byzantine failures.

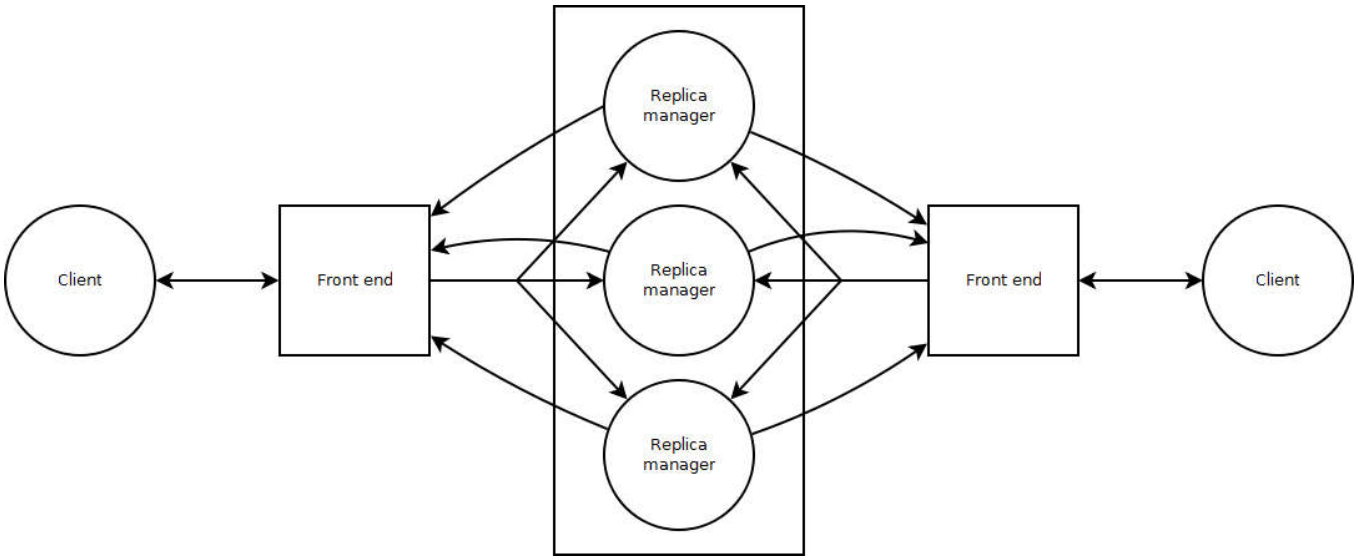
Active Replication

- The replica managers are state machines that play equivalent roles.
- FE multicast requests to the group of RM and all of them process the requests independently but identically and reply.
- If any RM crashes, others RM can responds. So, there is no degradation on performance.
- It can tolerate byzantine failures.

Event Sequence:
1. Request : The FE attaches a unique identifier to the request and multicast it to the group of RM. It does not issue the next request until it has received a response.
2. Coordination : The group communication system delivers the request to every correct RM in the same order.
3. Execution : Every replica manager executes the request.
4. Agreement : No agreement phase is needed.
5. Response : Each replica manager sends its response to the FE. The no of replies that the FE collects depends upon failure assumptions.

Advantages:
1. It can tolerate byzantine failure.
2. If any RM fails, it does not affect system performance.

Disadvantages:
- RM must be deterministic.
- Atomic broadcast protocol must be used.



Process Resilience

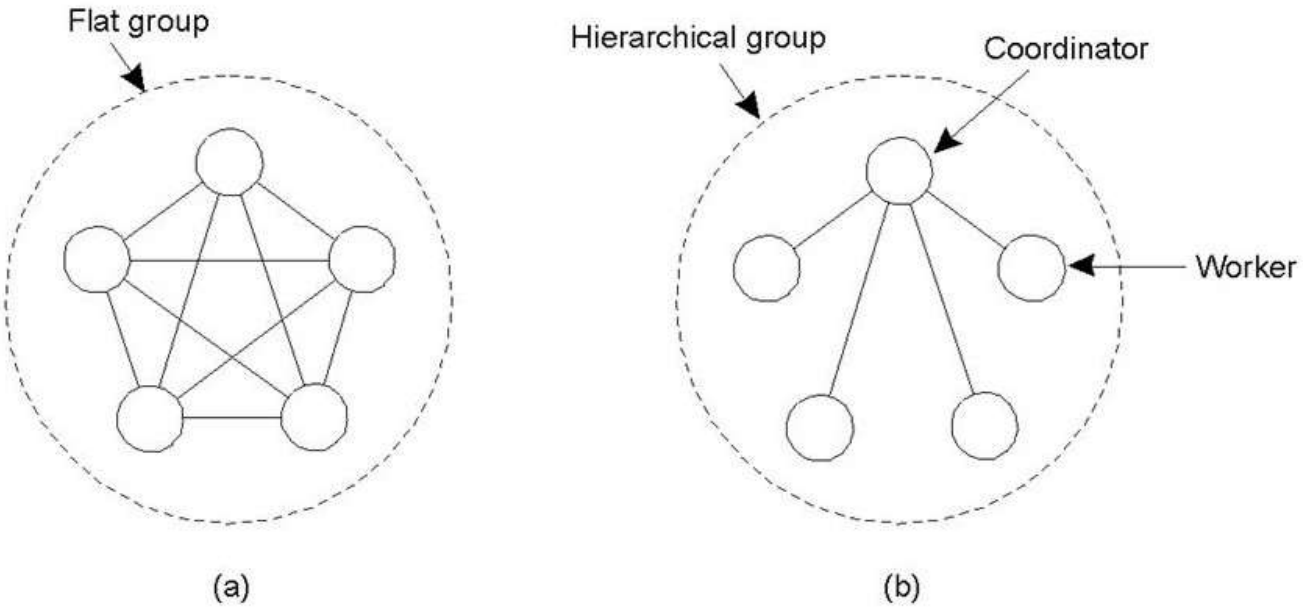
- Process resilience is a mechanism to protect against faulty processes by replicating and distributing computations in a group.

Flat and Hierarchical Groups

- In flat groups, all the processes within a group have equal roles.
- Control is completely distributed to all the processes.
- It is good for fault tolerance as information is exchanged immediately.
- It impose more overhead.
- It is difficult to implement.

- In hierarchical group, all the communications are handled by a single process designated as coordinator.
- It is not completely fault tolerant and scalable.
- It is easy to implement.

a) Communication in a flat group.
b) Communication in a simple hierarchical group



Byzantine Generals Problem

- In this case, the faulty node can also generate arbitrary data, pretending to be a correct one, but making fault tolerant difficult.

Problem:
- Can N generals reach agreement with a perfect channel if M out of N may be traitors?

For N = 4 and M = 1:

- The vector assembled by each general:
1 Got (1, 2, x, 4)
2 Got (1, 2, y, 4)
3 Got (1, 2, 3, 4)
4 Got (1, 2, z, 4)

- Tje vector that each general receives:
1 Got :
(1, 2, y, 4)

(a, b, c, d)
(1, 2, z, 4)

2 Got:
(1, 2, x, 4)
(e, f, g, h)
(1, 2, z, 4)

4 Got:
(1, 2, x, 4)
(1, 2, y, 4)
(i, j, k, l)

Conclusion:
- With m faulty processes, agreement is possible only if 2m + 1 processes function correctly.

Reliable Client Server Communication

Reliable Communication

- During communication between client and server, following may go wrong:
1. Client unable to locate server
2. Client request is lost
3. Server crashes
4. Server response is lost
5. Client crashes

- The solutions are:
1. Report back to client
2. Resend the message
3. Use of RPC semantics
4. Operations should be idempotent.
5. Kill the orphan computation

Reliable Multicasting

Problems:
- Message lost
- Acknowledgement lost
- Receiver suppresses their feedback

Solutions:
- Atomic multicast guarantees all process received message or none at all.

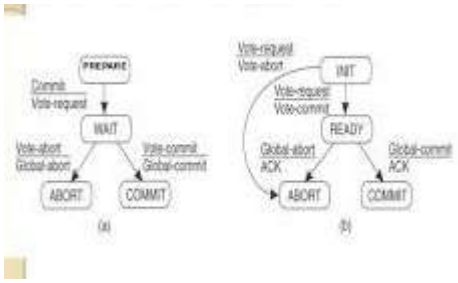
Distributed Commit

- It deals with methods to ensure that either all processes commit to the final result or none of them do.

Two Phase Commit

Model:
- The client who initiated computation acts as coordinator.
- Processes required to commit act as participant.

Phases:
1a - Coordinator sends VOTE_REQUEST to participants
1b - When participant receives VOTE_REQUEST, it reply with YES or NO. If it sends NO, it aborts its local computations.
2a - Coordinator collects all votes. If all are YES, it sends COMMIT to all participants. Otherwise, it sends ABORT.
2b - Each participants waits for COMMIT or ABORT and handles accordingly.



Participant Failure:
1. Initial state = Make transition to ABORT
2. Ready state = Contact another participants
3. Abort state = Make transition to ABORT
4. Commit state = Make transition to COMMIT

Implementation:
1. Coordinator

```
while START _2PC to local log;
multicast VOTE_REQUEST to all participants;
while not all votes have been collected{
      wait for any incoming vote;
      if timeout{
            while GLOBAL_ABORT to local log;
            multicast GLOBAL_ABORT to all participants;
            exit;
      }
      record vote;
}
if all participants sent VOTE_COMMIT and coordinator
votes COMMIT{
      write GLOBAL_COMMIT to local log;
      multicast GLOBAL_COMMIT to all participants;
}
else{
      write GLOBAL_ABORT to local log;
```

```
      multicast GLOBAL_ABORT to all participants;
}
```

2. Participants:

```
write INIT to local log;
wait for VOTE_REQUEST from coordinator;
if timeout{
      write VOTE_ABORT to local log;
      exit;
}
if participant votes COMMIT{
      write VOTE_COMMIT to local log;
      send VOTE_COMMIT to coordinator;
      wait for DECISION from coordinator;
      if timeout{
            multicast DECISION_REQUEST to other
participants;
            wait until DECISION is received;
            write DECISION to local log;
      }
      if DECISION == GLOBAL_COMMIT
            write GLOBAL_COMMIT to local log;
      else if DECISION == GLOBAL_ABORT
            write GLOBAL_ABORT to local log;
}
else{
      write VOTE_ABORT to local log;
      send VOTE_ABORT to coordinator;
}
```

3. Handling Decision Requests:

```
/* Executed in separate thread */
while true{
      wait until any incoming DECISION_REQUEST is
received;
      read most recently recorded STATE from local log;
      if STATE == GLOBAL_COMMIT
            send GLOBAL_COMMIT;
      else if STATE == INIT or STATE == GLOBAL_ABORT
            send GLOBAL_ABORT;
      else
            skip;
}
```

Distributed Recovery

- It is the mechanism to handle failures.
- It helps to recover correct state of the system after failure.

Independent Checkpointing

- Each process periodically checkpoints independent of other processes.
- Upon failure, locate a consistent cut backward.
- It needs to rollback until consistent cut is found.

Coordinated Checkpointing

- The state of each process in the system is periodically saved on stable storage.
- If failure occurs, it rollbacks to previous error free state recorded by the checkpoints of all the processes.

Message Logging

- Combining checkpoint and message logs becomes cheap.

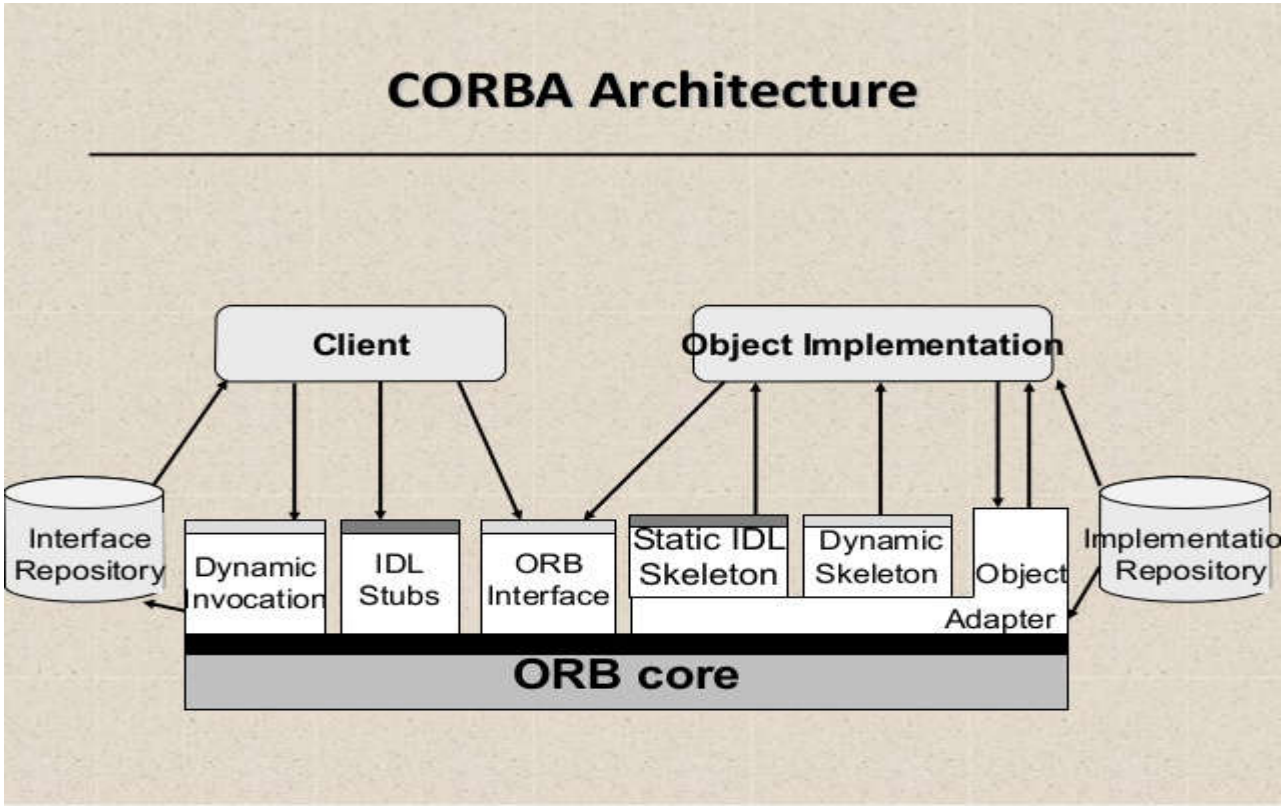
9. Case Study - Distributed System

CORBA

- Common Object Request Broker Architecture
- It is a standard defined by object management group to facilitate the communication of systems deployed on diverse platforms.
- It helps in communication in heterogeneous distributed system.

Architecture of CORBA

The general architecture is shown in given figure:



---

1. Interface Repository:
- It provides representation of available object interfaces of all objects.
- It is used for dynamic invocation.

2. Implementation Repository:
- It stores implementation details for each object's interfaces. (Mapping from server object name to filename to implement service)
- The information stored may be OS specific.
- It is used by object adapter to solve incoming call and activate right object method.

3. Object Request Broker (ORB)
- It provides mechanisms by which objects can interact with each other transparently.

4. Static Invocation:
- It allows a client to invoke requests on an object whose compile time knowledge of server's interface specification is known.
- For client, object invocation is similar to local method of invocation, which automatically forwarded to object implementation through ORB, object adapter and skeleton.
- It has low overhead, and is efficient at run time.

5. Dynamic Invocation:
- It allows a client to invoke requests on object without having compile time knowledge of object's interface.
- The object and interface are detected at run time by inspecting the interface repository.
- The request is then constructed and invocation is performed as it static invocation.
- It has high overhead.

6. Object Adapter:
- It is the interface between server object implementation and ORB.

Services Provided by CORBA

1. Naming service
- It allows clients to find and locate objects based on name.

2. Trading service
- It allows clients to find and locate objects based on their properties.

3. Notification service
- It allows objects to notify other objects that some event has occured.

4. Transaction Service
- It allows atomic transactions and rollback on failures.

5. Security Service
- It protects components from unauthorized access or users.

6. Concurrency control service
- It provides a lock manager that can obtain and free locks for transactions to manage concurrent transactions.

7. Life cycle service
- It defines conventions for creating, deleting, copying and moving CORBA objects.

8. Time service
- It provides interfaces for synchronizing time.

Mach

- Mach is a microkernel that runs on both multiprocessor and uniprocessor computers connected by networks.
- It incorporates sophisticated IPC and virtual memory facilities.
- Port is used to identify individual resources.
- To access a resource, a message is sent to the corresponding port.
- It provides a single system call for message passing : mach_msg
- This system call provides interactions for both asynchronous message passing and request-reply.
- mach_ msg contains msg_header, option, snd_siz, rcv_siz, rcv_name, timeout and notify.

JINI

- It is a coordination based system from Sun Microsystems written in JAVA.
- It uses RMI and Java object serialization to enable Java objects to move around the network.
- Services may be added or removed without configuration.
- It is not based on central control.

Components of JINI

1. Service:
- An entity that another program, service or user can use.

2. Client:
- A component that is capable of using JINI services.

3. Lookup Service:
- A component that keeps track of the services offered in the federation.

TIB/Rendezvous

- Synchronization and data transmission are inseparable activities which must be synchronized to exchange information. This process is called rendezvous.

Implementation of TIB

- One of the two communicating tasks knows the name of the other and names it explicitly.
- The second task knows only that it expects some external interaction.
- It is based on subject based addressing.
- Receiving a message on subject X is possible only if receiver had subscribed to X.

- Publishing a message on subject X means sending message to all subscribers of X.