UNIT 1

Introduction to Distributed Computing Concepts

Basic concepts of distributed systems, distributed computing models, software concepts, issues in designing distributed systems, client server model and current case studies of the World Wide Web 1.0 and World Wide Web 2.0.

Introduction to distributed computing concepts

- Using the high performance computer connected by equally high speed communication links, it is possible to build a single system consisting of multiple computers and using it as a single consolidated system.
- In such system, multiple resources work together to deliver the required processing speed and the operating system takes care of maintaining system.
- In distributed system, the computers are not interdependent but interconnected by a high-speed network.
- It is also mean that many computers, be they workstation or desktop system link together, can do the work of high-performance supercomputer.
- The important aspect is controlling the software.
- If a user thinks that the entire interlinked system is single unified computer, there has to be a software envelope that join multiple operating systems together and offers a seamless interface to the user.
- Transparency is the key issue in Distributed computer system.
- Primary requirement of Distributed Computing –
  - Security and Reliability
  - Consistency of Replicated data
  - Concurrent Transaction
  - Fault Tolerance
- Example of Distributed architecture:
  - Nationalized Bank with multiple Branch Offices.

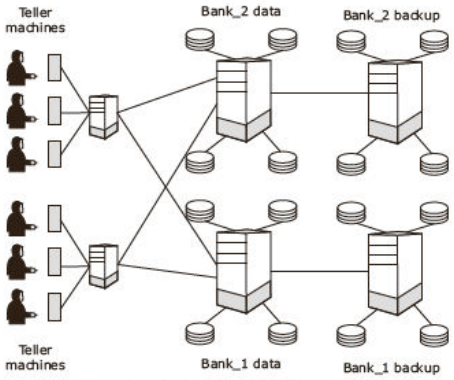


Figure 1-1 Internet connected network representing a bank

Architecture for Distributed System:

- Computer architectures consisting of interconnected, multiple p
  - Shared memory architecture also known as "tightly coupled systems.
  - Distributed memory architecture also known as "loosely coupled" systems.

1. Tightly coupled systems(shared memory):

- As shown in fig, multiple processing elements (PE) are connected by the network through a **bus** or **switch.**
- In these systems, there is a single system wide primary memory (address space) that is shared by all the processors.
- The interconnection network may consist of a single bus or switch connecting different memory modules to each processor as and when required.

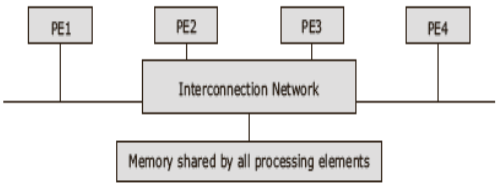


Figure 1-2 Shared memory architecture

- If any processor writes, for example, the value 100 to the memory location x, any otherProcessor subsequently reading from location x will get the value 100.
- Therefore, in these systems, any communication between the processors usually takes place through the shared memory.

2. Loosely coupled systems(Distributed memory):

- In these systems, the processors do not share memory, and each processor has its own local memory.
- In these systems, all physical communication between the processors is done by passing messages across the network that interconnects the processors.
- There is no sharing among the address spaces.
- This architecture is very easy to implement, as it can be built through commodity hardware with the interconnection being established through a standard Ethernet or ATM switches.
- A tightly coupled system suffers from a lack of scalability.
- For truly distributed systems, loosely coupled systems are more suitable.
- They offer scalability at an economical price.

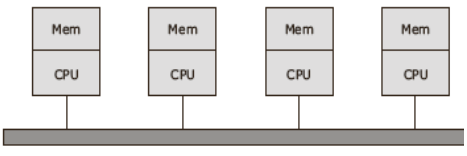


Figure 1-3 Distributed memory architecture

Advantages of Distributed Computing System :

1. **Inherently distributed applications:**
   - Consider the global employee database of a multinational company.
   - Branch offices would create local databases and then link them together for global viewing.
2. **Information sharing among geographically distributed users:**
   - The Email system security actually originated at CERN because scientist wanted to communicate with each other across the world.
   - This can be implemented by interconnecting all nodes over multiple LANs.
   - The node may be situated in member countries and geographically wide apart.
3. **Resource Sharing :**
   - Distributed systems allow sharing of other resources like printers, high-end graphics terminals as well as software and ERP packages.
4. **Better price-performance ratio:**
   - Distributed systems offers a very good price-performance ratio as compared to large centralized computer systems.
   - PCs are becoming more powerful and cheaper. Communication networks have gotten faster.
5. **Shorter response time and higher throughput :**
   - The main idea behind connecting multiple computers through a high-speed network is that a system having multiple resources should have better throughput and shorter response time than a centralized single processor system.
6. **Higher reliability and availability against component failure :**
   - Reliability refers to the degree of tolerance against component failures and errors.
   - A distributed system has multiple processors and storage devices.
   - A critical program can be executed on multiple systems and also replicated so that in case of processor crash, the same program can be completed on another processor.
   - A distributed system also increases the availability of the system. If a hardware component fails, then due to redundancy, other components are available, and the system can function with reduced capacity.
7. **Extensibility and incremental growth :**
   - A system is highly scalable.
   - Each system being a complete personal computer, there is no fear of bandwidth limitation.
   - If the application or the database grows in size or computing power is required, the distributed system can be extended effortlessly by adding more computers.
8. **Better flexibility:**
   - A centralized computer system should give service to many different types of applications.
   - It should handle complex computations requiring high computing power and also cater to complex query-processing demands.

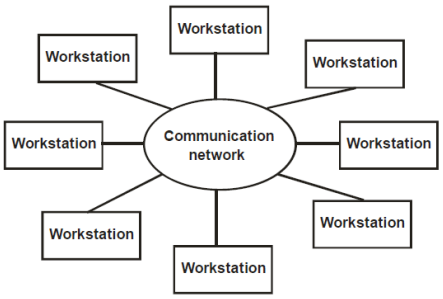
Disadvantages of distributed computing :

1. System Transparency
2. Software inadequacy
3. Data security

Distributed Computing Model :

- There are different types of Distributed Computing system models.
- Some of popular ones include Workstations Model, Mini Computer Model, Workstation Server Model, Processor-pool Model.

1. **Workstation Model :**
   - The workstation model consists of a network of personal computers,each with its own hard disk and local file system and interconnected over the network.
   - These are termed as *diskfulworkstations.*
   - These workstations are connected in a suitable network configuration using star topology.
   - Each workstation can also work as a standalone single-user system.



   - The workstation model is not very easy to implement, since several issues need to be resolved.
   - They are as follows :
     A. How to find an idle workstation?
     B. How to transfer a process from one workstation to another workstation in a manner that is transparent to the user?

C. What happens to the remote process when a user logs on to that workstation and a home process is created?

- The first two issues involve *load-balancing techniques.*
- The third issue can be resolved by using different approaches.
- All work done for the remote process till then will be useless and the file system may be left in an inconsistent state.
- This is waste of system resources, as process migration during execution is very costly.

2. Workstation Server Model :

- The workstation-server model consist of multiple workstations coupled with powerful servers with extra hardware to store the file systems and other software like databases.
- Thus, more number of less expensive workstations with few costly servers makes a fine model for implementing distributed system concepts.
- Server Process can control the functioning of the system.
- Such a system is scalable, reliable, and fault tolerance.
- The workstation can be distributed among users.
- Another advantage is that the local disk may contain temporary files and the really important files can be stored in the main memory.

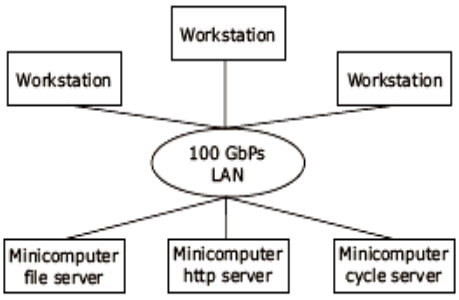
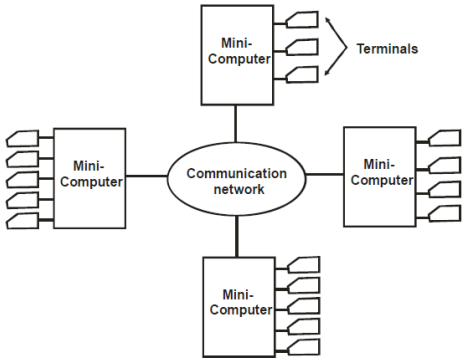


Figure 1-5 Workstation-server model

3. Minicomputer Model :

- The minicomputer model is a simple extension of the centralized timesharing system as shown in Figure distributed computing system based on this model consistsof a few minicomputers interconnected by a communication network.
- Eachminicomputer usually has multiple users simultaneously logged on to it.
- For this, several interactive terminals areconnected to each minicomputer.
- Each user is logged on toone specific minicomputer, with remote access to otherminicomputers.
- The network allows a user to access remoteresources that are available on some machine other than theone on to which the useris currently logged.



- The minicomputer model may be used when resourcesharing with remote users is desired.

4. Process-pool Model :

- A processor-pool model consists of multiple processors, actuallya pool of processor, and a group of workstations.
- Some of the processors in pool have more computing power, while others may be used as file servers.
- The processors are powerful microcomputers and minicomputers.
- Each processor has its own memory, running system program or application programs.
- The processors in the pool have no terminals connected directly.
- The users access the system through their terminals via high-speed networks.
- The terminals can be diskless workstations or specialized wor
- A specialized server called the *run server* handles the scheduling of the processors.

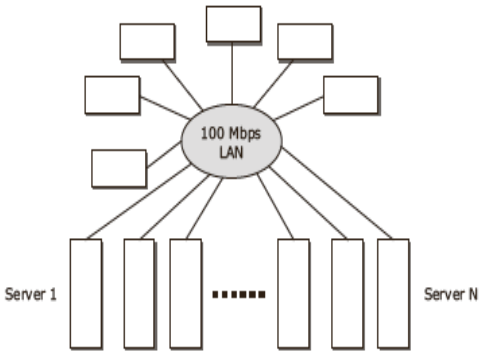


Figure 1-6 Processor pool model

Software concepts

- Distributed computing is also largely dependent on the software.
- Mainly, it is operating system that offers the software envelopes necessary to access distributed resources.
- The distributed operating system lies in between the operating system and the application layer.
- The software which handles the management of the entire distributed system consists of open services such as file services, name services, networking, and electronic mail; support for distributed programming such as remote procedure call and group communication; and the basic operating system microkernel running on each computer in the network.

- Network Operating System (NOS)
- Distributed Operating System (DOS)
- Multiprocessor Time Sharing System

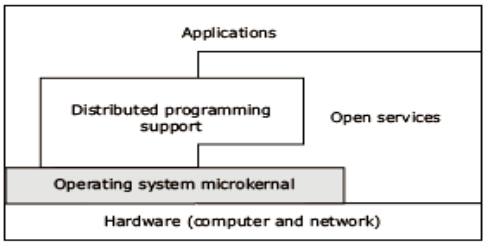


Figure 1-8 Software services

Network Operating System (NOS):

- Build using a distributed system from a network of workstations connected by high speed network.
- Each workstation is an independent computer with its own operating system, memory and other resources like hard disks, file system and databases

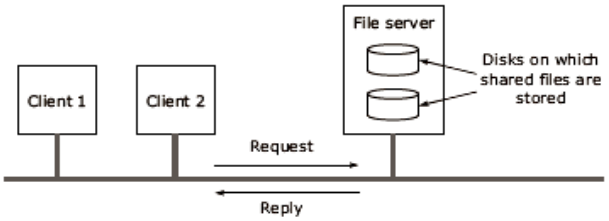


Figure 1-9 Network operating system

Distributed Operating System (DOS)

- Enables a distributed system to behave like a virtual uniprocessor even though the system operates on a collection of machines.
- Some characteristics are discussed below.
- ***Inter-process communication:***This feature allows all processes to communicate globally with each other. A uniformprocess communication schemes is implemented for both local and remote communication.
- ***Uniform process management mechanism:***Every computer executes programs by dividing them into processes. Hence, a computer needs to execute process-related tasks like create, destroy, run, terminate etc.
- ***Identical kernel implementation:***A uniform system call interface should exist across all machines. Hence, we conclude that kernels running on all CPUs must be identical.
- ***Local control of machines:***Each kernel must have control over its own local resources.
- ***Scheduling issues:***A CPU can manage its own scheduling algorithms for processes running on its system.

Multiprocessor Time Sharing System

- Combination of tightly coupled software and tightly coupled hardware with multiple CPUs projecting a uniprocessor image.
- Tasks are queued in shared memory and are scheduled to be executed in time shared mode on available processors**.**

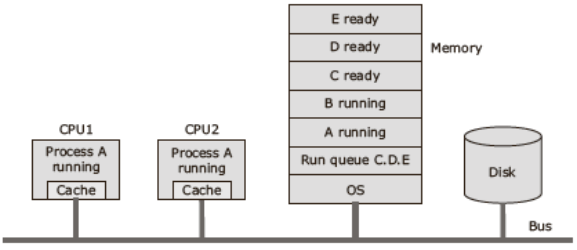


Figure 1-10 Multiprocessor time-sharing system

Issues in designing distributed systems :

I.Transparency:

- A distributed system that is able to present itself to user andapplication as if it were only a single computer system is said to betransparent.
- There are eight types of transparencies in a distributedsystem:

1) **Access Transparency:**
   - It hides differences in data representation and how a resource is accessed by a user.
   - Example, a distributed system may have a computer system that runs different operating systems, each having their own file naming conventions.
   - Differences in naming conventions as well as how files can be manipulated should be hidden from the users and applications.

2) Location Transparency:
- Hides where exactly the resource islocated physically.
- Example, by assigning logical names toresources like yahoo.com, one cannot get an idea of thelocation of the web page's main server.

3) Migration Transparency:
- Distributed system in whichresources can be moved without affecting how the resource canbe accessed are said to provide migration transparency.
- It hidesthat the resource may move from one location to another.

4) Relocation Transparency:
- This transparency deals with the factthat resources can be relocated while it is being accessedwithout the user who is using the application to know anything.
- Example: using a Wi-Fi system on laptops while moving fromplace to place without getting disconnected.

5) Replication Transparency:
- Hides the fact that multiple copiesof a resource could exist simultaneously.
- To hide replication, itis essential that the replicas have the same name.Consequently, as system that supports replication should alsosupport location transparency.

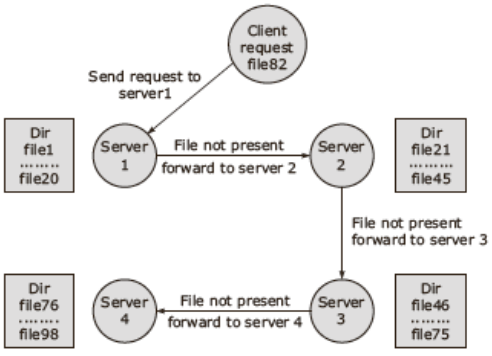


Figure 1-11 Replication transparency

6) Concurrency Transparency:
- It hides the fact that the resourcemay be shared by several competitive users.
- Example, twoindependent users may each have stored their file on the sameserver and may be accessing the same table in a shareddatabase.

7) Failure Transparency:
- Hides failure and recovery of theresources.
- It is the most difficult task of a distributed system and is even impossible when certain apparently realistic assumptions are made.
- Example: A user cannot distinguish between a very slow or dead resource.

8) Persistence Transparency:
- It hides if the resource is inmemory or disk.
- Example, Object oriented database provides facilities for directly invoking methods on storage objects.

II.<u>Flexibility :</u>
- The best way to achieve flexibility is to take a decision whether to use monolithic kernel or microkernel on each machine.
- Kernel is the central control, which provides the basic system facilities.
- The major function of kernel are : memory management, process management, and resource management.
- The kernel is placed above the hardware layer and below the application layer.

a) Monolithic kernel approach :
- The monolithic kernel uses the 'kernel does it all' approach.
- This type of kernel provides all the facilities at the local machine.
- Monolithic kernel is massive and non-modular in structure with all major OS functions performed in the kernel spaces.
- As shown in fig, functions of process management, I/O and device management interprocess communication, and file system management are all managed at the kernel level.
- The system calls are made with a trap to the kernel and they perform the desired tasks and send the results to the user process. Machines manage their own files locally.

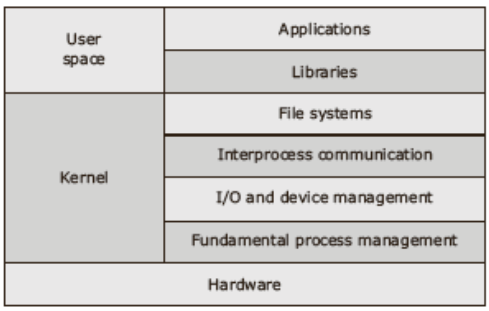


Figure 1-12 Monolithic kernel approach

b) Microkernel approach :
- The microkernel removes many functions from the kernel and retains only minimal essential functions.

- It provides very few functions in the node, while the other functions are available at various servers.
- This type of kernel provides very low-level services like interprocess communication, memory management, and low-level process management.
- The main advantage of this approach is modularity, well-defined interface for each service, and equal accessibility of all services to all the clients irrespective of their locations.
- It is also easier to port, maintain, and extend.

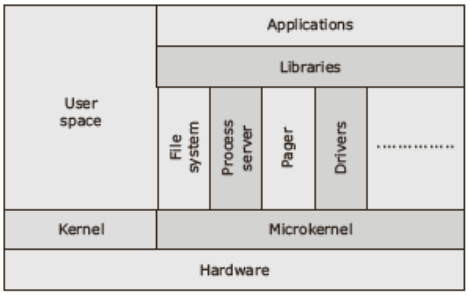


Figure 1-13 Microkernel approach

III.Reliability :

- The users prefer a distributed system where multiple processors are available and the system become reliable.
- So on failure, a backup is available. Reliability in terms of data means that data should be available without any errors.
- In case of replication, all copies should be consistent.

IV.Performance :

- It implies that an application should run just as if it were running on a single processor.
- The metrics are : response time, throughput, system utilization, and the amount of network capacity used.
- Message transmission over a LAN takes some time, about one millisecond. To optimize performance, reduce the number of message transmitted.
- One of the solutions is to batch the message together.

V.**Scalability :**

- Distributed systems are designed to work with a few hundread CPUs.
- Over a period, then may be need to expand the distributed system by adding more CPUs.
- To support more users or resources, there are limitations with centralized services.
- This scalability is related to size. As the number of users grows, the server will face manageability issues.

Table 1-3 Scalability related issues

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

- The solution is to distribute the services across multiple servers, but this is not always feasible.
- Centralized data in a single database ensures security but may lead to communication failure and bottleneck.
- Query handling may also be very slow due to widely spread data.

VI.Security :

- Security consists of three main aspects,, namely,
  - Confidentiality, which means protection against unauthorized access.
  - Integrity, which implies protection of data against corruption
  - Availability, which means protection against failure and always being accessible.

VII.Fault Tolerance :

- Ideally, distributed systems are designed to mask failures.
- In case a system has multiple servers, and if any server break down, the other server takes up the load.
- This process is transparent to the user.

Client-Server Model :

- The structure of the operating system is like a group of cooperating processes called the servers, which offer services to the users called the clients.
- Both run on the same microkernel as the user processes.
- This model uses the connectionless Request Reply protocol thus reducing the overhead of the connection oriented TCP/IP protocol.
- The Request Reply protocol works as follows:
  - The client requesting the service sends a request message to the server.
  - The server completes the task and returns the result indicating that it has performed the task or return an error message indicating that it has not performed the task.
- One advantages of this model is simplicity since it needs to no connection.
- The reply serves as acknowledgement to the request.

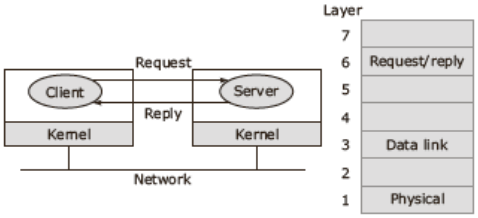


Figure 1-17 The client-server model

[By Nishigandha Dhore]

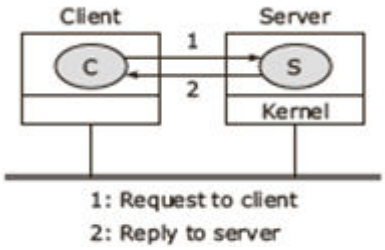
- The other advantages is its efficiency due to shorter stack.
- As seen in above figure, only three layers of the protocol are used.
- The physical and data link layer are responsible for transferring packets between the client and the server through hardware.
- There is no need for routing and establishment connection.
- The request reply protocol defines the set of requests and replies to the corresponding requests.

Client-server Addressing :

- Before the client can send a message to the server, it should know the server address.
- Three main addressing techniques, namely,
  - Machine Addressing
  - Process Addressing
  - Server Addressing

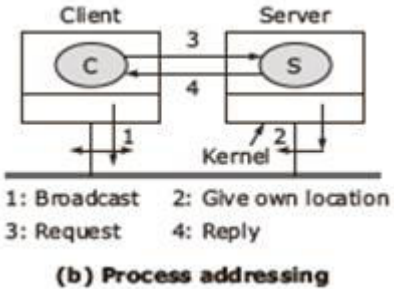
Machine Addressing:

- In this addressing method, the client sends the address as a part of the message, which is extracted art the receiving end by the server.
- This method work well if there is only one process running on the server machine.
- If there are multiple processes running on the server, the process ID should be sent as a part of the message.



Process Addressing:

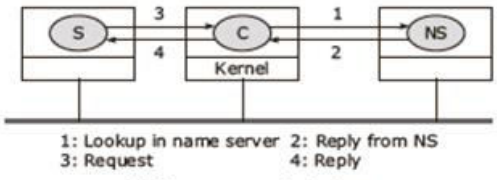
- The other addressing method consists of sending a message to the processes and not the machines.
- A two-part name comprising the machine ID and the process ID is used for addressing.
- The client kernel uses the machine ID to locate the current machine, while the server kernel uses the process ID to locate the process on that machine.
- All machines can start the process addresses from zero.
- This method is not transparent because the user knows the location of the server.
- The sender kernel broadcasts the message which contains the destination process address.
- All kernels check their process addresses, the machines that own the process sends its network address to the sender.



(b) Process addressing

Server Technique:

- Broadcasting puts an extra load on the system.
- We can use an extra machine to the ASCII level names to machine addresses.
- The ASCII strings are embedded in the program.
- This special server is called a *name server.*
- This process addressing techniques are as follows:
  - Hardware the machine number into the client code.
  - Processes pick random addresses and the machine address is located by broadcasting method.
  - Use a two-level naming schemes with mapping carried out by the name server.



(c) Name server technique

Figure 1-19 Addressing techniques

Client-server Implementation :

- All networks have a maximum packet size.
- Therefore, if the message is large, it is split into multiple packets and each packet is transmitted individually.
- We take care of the issue by assigning a sequence number to each packet depending on the order of the packets.
- In case of multiple packets per message, should the receiver acknowledge each packet or just the last packet?
- If an acknowledgement is sent for each packet, then only lost packets are retransmitted, but it increases the message traffic.
- The alternative is to send a single acknowledgement for all packets.
- This result in transmission of fewer acknowledgements.
- The client may not understand which packet is lost and hence may end up retransmitting all packets.
- Different types of packets used in client-server communication :
  - **REQ:** Request packet is used to send the request from the client to the server.
  - **Reply:** This message is used to carry the result from the server to the client.
  - **ACK:** Acknowledgement packet is used to send the correct receipt of the packet to the sender.

- **Are You Alive(AYA)?:** This packet is sent in case the server takes a long time to complete the client's request.
- **I Am Alive(IAA):** The server, if active, replies with this packet. It implies that the client kernel should wait for the reply.

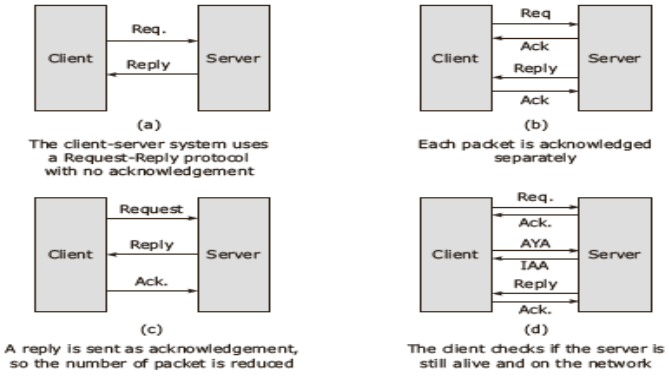


Figure 1-20 Common packet message sequences

Multi-tiered architecture :

- Based on the logical levels, the client-server application is distributed across machines in different ways.
- The client-server application can also be at the server machine that contains the remaining program, which is implemented at the processing level and the database level.
- This organisation is not distributed and server handles all tasks.
- This results in a multi-tiered architecture.
- The program in the application layer is distributed across machines.
- The client and the server form a two-tiered architecture,

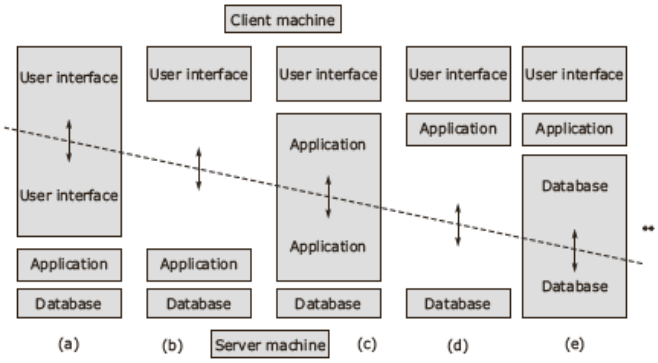


Figure 1-22 Multi-tiered client-server architecture

A. The client machine contains only the terminal-dependent part of the user interface; applications get remote control over presentation of data.
B. The user interface is completely on the server side, and the front-end does no processing except for presenting the application interface.
C. A part of the application can be moved to the front-end.
D. The client is a PC connected across the network to a distributed file system and database. This arrangement is called the three-tiered architecture.
E. The processing is done on a separate server or is distributed across several machines.

Case studies of the World Wide Web 1.0 :

- The WWW as it is popularly known is an excellent example of resource sharing using networked computers.
- The Internet was born in 1983 as a produce of academic and scientific communications.
- Communications were built on standards or protocols for addressingsystems and exchanging data.
- Known as the Transmission Control Protocol/Internet Protocol (TCP/IP), and then included the world Internet that came to identify the huge, global network in use today. By linking their communications, the original users of the Internet were able to exchange electronic mail, use file transferprotocol (FTP) to exchangedata obtain access via telephone lines to computers at other locations,
and converse using newsgroup and bulletin boards.

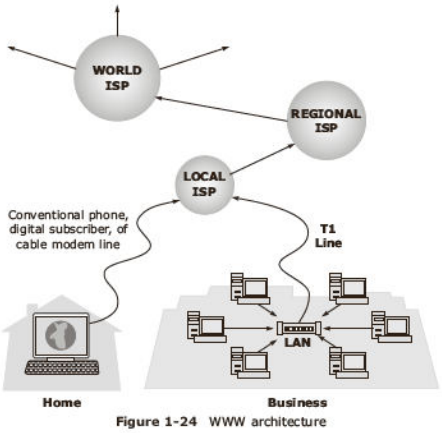


Figure 1-24 WWW architecture

- Today, the Internet is widely used for application and operations like web mail, chat, videoconferencing, photos, social networks, book, planning trips, buying stuff, checking for events, forum and blogs, accepting information on wikis, checking for maps, banks accounts, etc.
- A border definition comes from the organization World Wide Web Consortium (W3C), which was found by Tim Berners-Lee, best known as the inventor of WWW.
- The World Wide Web is the universe of network-accessible information.
- The WWW is basically a network of computers, as shown in fig.
- A home computer may be linked to the Internet using a phone-line modem, DSL, cable modem that interacts with an Internet Service Provider (ISP).
- The business can connect its LAN to an ISP using high-speed phone line like a T1 line.
- A T1 line can handle approximately 1.5 million bits per second. ISPs then connect to larger ISPs, and the largest ISPs maintain fiber-optic 'backbones' for an entire nation or region.
- The web is based on three main standards technological components:
    - The Hypertext Markup Language (HTML) is a language for specifying the contents and layout of pages as they are displayed by web browsers.
    - Uniform Resource Locators (URL) that identify documents and other resources stored as part of the Web.
    - Hyper Text Transfer Protocol (HTTP), client-server architecture with standard rules for interaction by which browsers and other clients fetch documents and other resources from web servers.

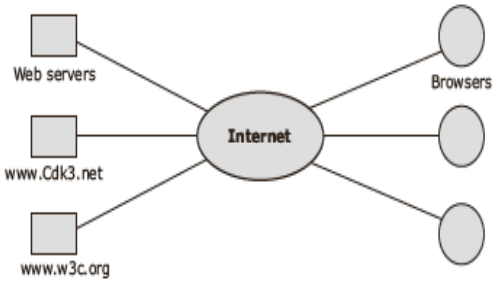


Figure 1-25 Web servers and web browsers

- Below figure shows the Internet scenario with web servers and web browsers.
- The browser formed a connection to a web server, requested a page and received.
- The most popular browsers used to retrieve information from the Web are Netscape and Internet Explorer.

World Wide Web 2.0 :

- The web-based model aims to facilitate communication, secure information sharing, interoperability, and collaboration on the World Wide Web.
- Web 2.0 encapsulates the idea of proliferation of interconnectivity and interactivity of Web-delivered content.
- Typical examples of companies or products which use web 2.0 are;
    - eBay, Dodgeball, AdScese: These exist only on the Internet.
    - Flickr applications which benefit from online operations but can also operate offline.
    - Google Docs, Spreadsheets, iTunes, etc. which operate offline but gain features online.
- With Web-based applications and desktop, it is possible to mimic desktop applications like word processing, spreadsheet, slide show presentations.
- The web server is the preferred model for all Internet-based computing applications, whether personal or commercial.
- Google's architecture is a three-layered stack with the computing platform at the lowermost layer.
- Google maintains over 450,000 servers, arranged in racks located in clusters in cities around world.
- The second layer is the software stack called the distributed system infrastructure.
- The services and applications reside at the topmost layer.
- Google's server infrastructure is divided into various types.
- Google load balancers take the client request and forward it to one of the Google web server via Squid proxy servers.
- These servers take the client request from load balancers and return the result if present in local cache, or else, forward it to Google web server.
- Data-gathering servers are permanently dedicated to spidering the web.
- Spidering similar to crawling means browsing the web in an automated manner.
- Google's web crowler is known as GoogleBot.
- They update the index and document database and apply Google's algorithm to assign ranks to pages.
- Index shards deals with the partitions of data so that the work of building indexes can be parallelized. They return a list of document IDs. Document servers store documents.
- Ad servers manage advertisements offered by services like AdWords and AdSense.

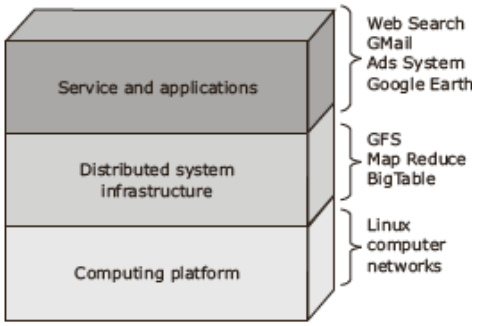


Figure 1-27 Google architecture

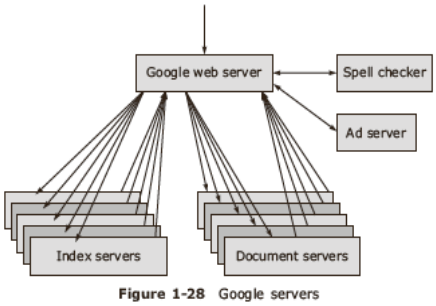


Figure 1-28 Google servers

[By Nishigandha Dhore]

Unit 2

Inter Process Communication

Fundamental concepts related to inter process communication including message-passing mechanism, a case study on IPC in MACH, concepts of group communication and case study of group communication CBCAST in ISIS, API for Internet Protocol

Message-passing mechanism

- A google search engine may send an enquiry to all servers to know their current load and queue status.
- A new request is sent to the least loaded node or the request can be put in a queue based on the responses obtained from the receiver.
- A distributed system hence needs to provide an IPC mechanism to facilitate such an enquiry activity.
- Figure 3-1 depicts the significance of communication network in a distributed system.

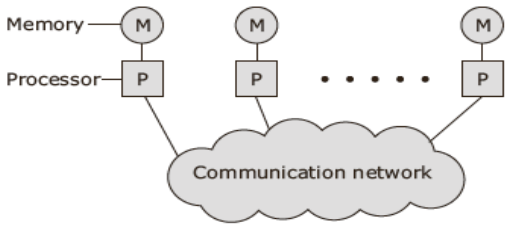


Figure 3-1 Typical message-passing operation

- The two basic methods for information sharing are:
  - Message-passing approach
  - Shared memory approach
- In the message-passing technique, the sender and the receiver processes communication with each other using send and receive primitives.
- Two processes exchange a message by copying it from the sender's address space to the receiver's address space.
- As seen in fig. 3-2, the sender kernel manages message-passing between Process A and Process B.

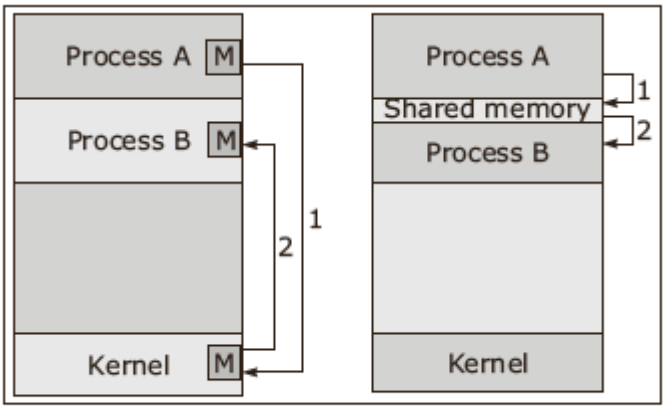


Figure 3-2 Message-passing vs. Shared memory approach

- In shared memory approach, both the sending and receiving process share a common memory area where the information is stored and is really accessible to both the process.
- A message based IPC protocol hides the heterogeneous and complex nature of the hardware and the network from the programmer.
- Simple 'send to' and 'receive' primitives are used to enable processes to communicate by exchanging messages.

Features of message passing:

- The major functions of the message passing system include efficient message delivery and availability of communication progress information.
- The message-passing model advantages are:
  - **Hardware match**: The message passing model fits well on parallel super-computers and clusters of workstations, which are composed of separate processors connected by a communication network.
  - **Functionality:** Message passing offers a full set of functions for expressing parallel algorithms, providing the control which is not found in parallel data and compiler-based models.
  - **Performance:** Effective use of modern CPU's requires management of the memory hierarchy and especially the cache. Message passing achieves this by providing programmers the explicit control of data locality.
- The disadvantage:
  - Responsibility of message passing is on programmer.
  - He has to implement a data distribution schemes, to include inter-process communication and synchronization call in his code.
  - He is also responsible for resolving data dependencies, avoiding dead-lock and race conditions.

Desirable features of message passing:

- **Uniform semantics:** uniform semantics are used for both send and receive operations and for local as well as remote processes.
- **Efficiency:** An optimization technique for efficient message-passing is to avoid repeated costs of setups and termination of connection for multiple message transfers between the same processes.
- **Reliability:** The technique to ensure reliability includes handling lost and duplicate message. Acknowledgement and retransmissions based on the time-out mechanism are used to handle the lost messages.
- **Correctness:** Correctness is related to IPC protocols and is often required for the group communication. Atomicity ensures that the messages are sent to all receivers or to none of them.
- **Flexibility:** The message-passing should be flexible enough to cater to the varied requirements of the application in terms of reliability.
- **Portability:** It should be possible to construct a new IPC facility on another system reusing the basic design of the existing message-passing system.
- **Security:** A good message-passing should provide secure end-to-end communication. Only authenticated users are allowed to access the message in transit.

Message-passing Operations:

- Fig. 3-3 shows a typical message-passing operation.
- The application on the sending hosts wants to carry out a message transfer operation.
- The messaging interface formats the message and passes it to the local buffer.
- The operating system sends it across the local network to the communication server where it is queued.
- The routing program sends the message to the remote communication server based on the destination address specified in the message.
- This server passes the message to the required destination, where it is stored in the local buffer and later accepted by the application.

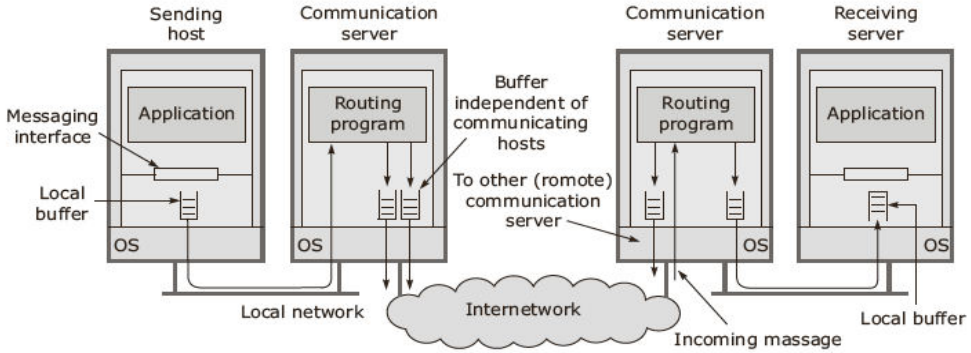


Figure 3-3 Message passing operation

- Very often, the message transfer operation is not so simple.
- There are many issues involved along the communication path.

IPC message format

- A message consists of data and control information, formatted by the sender so that it is meaningful to receiver.
- The sender process adds control information to the data block with header and trailers before transmission.
- Fig. 3-4 shows the typical components of a message.

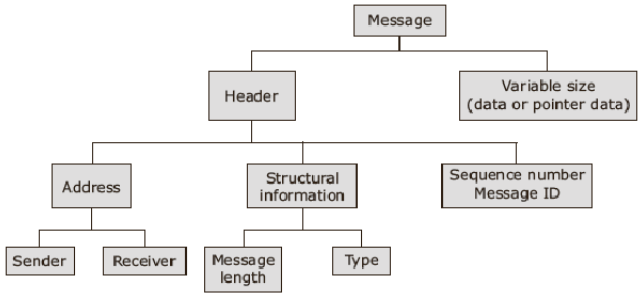
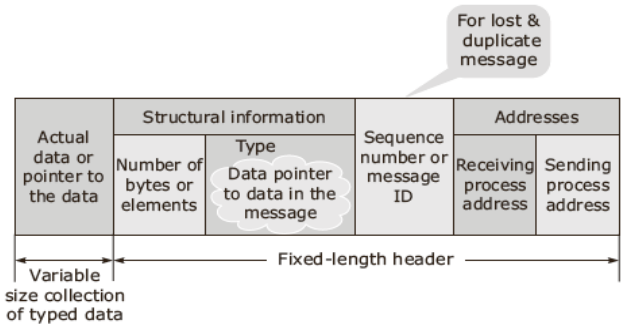


Figure 3-4 Components of an IPC message

- The header information of a message block consists of the following elements, as shown in fig. 3-5:
  - Address: It uniquely identifies the sender and receiver processes in the network by specifying the source and destination address.
  - Sequence number: It is a message identifier which takes care of duplicate and lost message in case of system failure.
  - Structural information: It comprises of: 1)TYPE part specifies whether the message carries data or only the pointers to data 2) LENGTH part specifies the length of the variable sized data in bytes.



Sender and receiver are fully aware of the message formats used in the communication process and the mechanisms used to send and receive messages.

Figure 3-5 A typical IPC message structure

IPC Synchronization

- Message communication techniques are classified based on whether the sender is aware of the message receipt by the receiver
- Two methods are used to ensure the message is received at the buffer as follows:
  - *Polling:* In this method, the receiver periodically checks its buffer for received messages by using a test primitive.
  - *Interrupt method:* In this method, when the buffer receives the message, it sends a software interrupt to notify the receiver process.
- Synchronous communication is incomplete until the sender gets an acknowledgement of the receipt of the message.

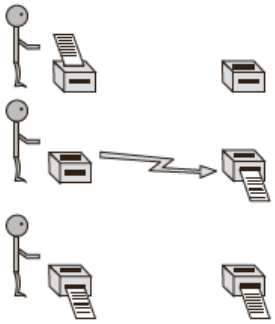


Figure 3-6 Synchronous communication operation

Figure 3-8 Asynchronous communication operation

- A typical synchronous communication in terms of execution and suspension timelines is shown in fig. 3-7.
- The sending process is blocked until the reply is received after which the sender resumes execution.

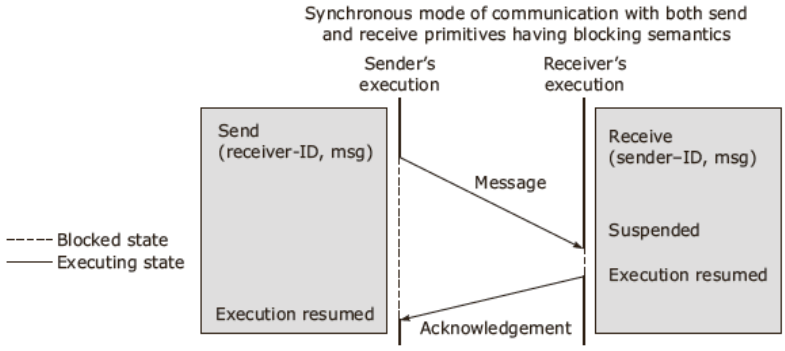


Figure 3-7 Synchronous communication operation

- Asynchronous send is complete as soon as the corresponding message is on its way.
- The sender need not know whether the receiver has received the message or not.
- Synchronous communication is simple and easy to implement.
- It is also reliable because the sender knows that the message is accepted.
- However, the sender will keep on waiting, resulting in a deadlock state, if it does not receive the acknowledgement.
- For distributed systems, the message-passing primitive operations are classified as:
  - **Blocking Type:** A blocking type primitive's invocation blocks the execution of the invoker. The receiver's blocks itself until the message arrives. A blocking operation returns from the subroutine call only when the operation is completed.
  - **Non-blocking:**Non-blocking operations return straight away and allow the sub-program to continue the execution.

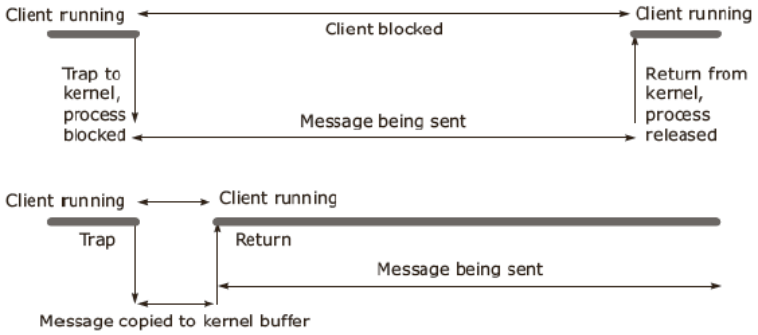


Figure 3-9 Blocking and non-blocking primitives

Message Buffering Strategies:

- Synchronous and Asynchronous modes of communication are two extreme buffering techniques that correspond to ***null buffering***and ***infinite-capacity*** *buffering* respectively.
- The other techniques are ***single***and ***multiple buffering***strategies.
- Fig. 3-10 shows different buffering schemes.

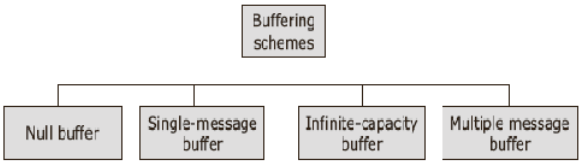


Figure 3-10 Buffering schemes

1) Null buffering

- In this technique, the message is sent directly from the sender's address space to the receiver's address space by a single copy operation as shown in fig. 3-11.

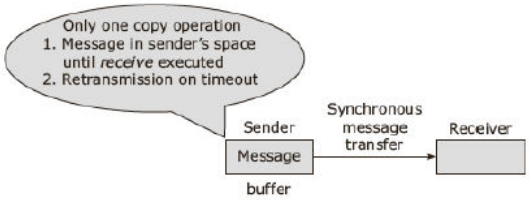


Figure 3-11 Message transfer with no buffering and single copy operation

- Two cases are consider:
  i. If the receiver is not ready to receive the message, the sending process is blocked and the **send** is delayed. When the receiver executes the **receiver** primitive, an acknowledgement is sent to the sender, who can send message. (fig. 3-12)
  ii. In other technique, the sender sends the message and waits for acknowledgement. If the sender does not receives the acknowledgement within a timeout period, the sender retransmit the information.(fig. 3-13). The drawback of this method is that the sender transmits the message multiple times if the receiver is not ready. Secondly, the receiver has to wait until the message transfer is complete.

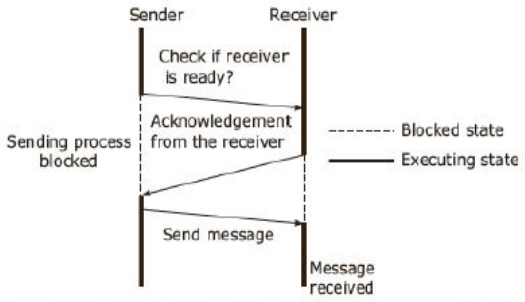


Figure 3-12 Null buffering with effective message-passing blocking mechanism

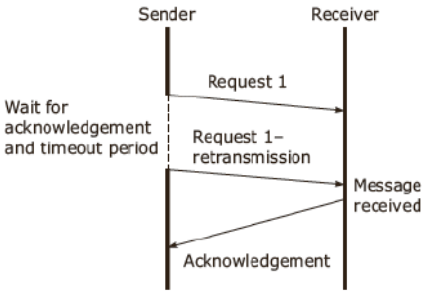


Figure 3-13 Null buffering with effective message-passing

2) Single-message buffering:

- For synchronous communication, if the receiver is not ready to receive the message, it is stored in a buffer.
- This message-passing technique involves two Copy operations, as shown in fig. 3-14.
- The message buffer can be located either in the receiver's address space or the kernel process's address space.
- The drawback of this technique is that the buffer

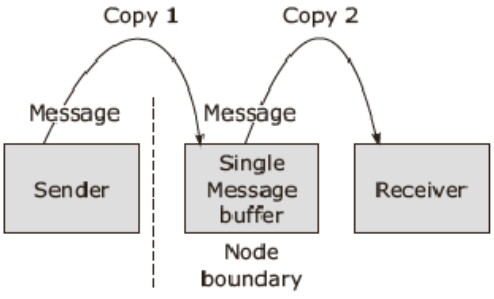


Figure 3-14 Single-message buffering

overflows if there are multiple messages at any point of time.

3) **Infinite-capacity buffer:**
   - In asynchronous communication, the sender need not be aware of whether the receiver is ready for reception or has already received the message.
   - In such case, an infinite-capacity buffer stores all the outstanding messages sent by the sender.

4) **Multiple-message buffer:**
   - This techniques allocates a buffer with a capacity of a few messages on the receiver side.
   - Hence this method avoids the buffer overflow issues.
   - The sender may not be aware that the receiver buffer has overflowed.
   - To overcome this problem, the techniques used are:
     i. ***Unsuccessful Communication Indication:*** It sends message transfer fails due to lack of buffer space.
     ii. ***Flow-controlled technique:*** The sender is blocked sending process is resumed.

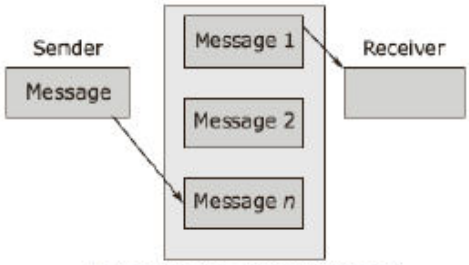


Figure 3-15 Multiple-message buffering

Multidatagram messaging:
- The message-passing technique works well, based on the buffering strategy implemented.
- Users can write a 'create_buffer' system call, which creates a 'buffer/mailbox/' port on the receivers side, of a size specified by the receiver.
- This buffer can be located in the receiver's kernel or the process's address space.
- In kernel's address space, the number of messages and the buffer size can be limited to cater to all users and their processes.
- If the buffer is located in the process's address space, the user is responsible for proper memory allocation, security, and other issues.

Concept of MTU:
- All networks have pre-defined MTU(maximum transfer unit), in which the data is transmitted as a single packet.
- Messages greater than the MTU are broken into chunks or datagrams and transmitted independently (fig. 3-16).

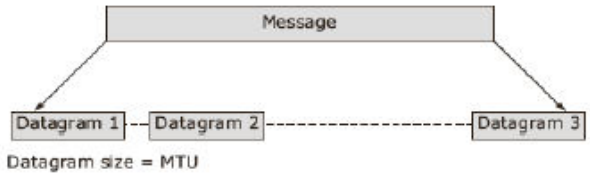


Figure 3-16 Multidatagram message

Message sequencing and reassembly:
- A message-passing system handles the two important functions of **message sequencing** and **reassembly.**

Message contents:
- During message transfer, integer or float data is copied directly. What copy operation should take place, if the message contains only a pointer to the data while the actual data is stored at some other location?
- One solution is to transfer these objects by flattening them on the sender side and regenerating them on the receiver side.(fig. 3-17)

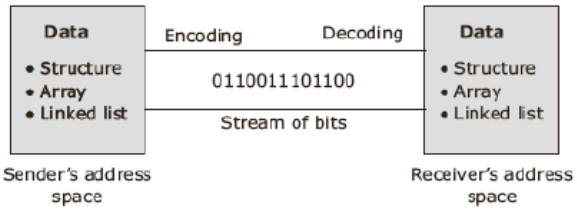


Figure 3-17 Encoding/decoding message data

Tagged representation:
- In this technique, the program object and its type information is encoded in the message.
- This method is simple because the receiving process can know the type of the program object from the message itself.

Untagged representation:

- In untagged representation, the sending and receiving processes decide how the data will be encoded and decoded.
- The message does not carry information about the type of data along with the value.

Process Addressing Techniques:

- A flexible message-passing system supports two process addressing techniques:

1. **Explicit addressing:**
   - In explicit addressing scheme, the primitive directly names the process with which it communicates.
   - The communication primitives used are ***send(process_ID, message)*** and ***receive(process_ID, message).***
2. **Implicit addressing:**
   - In implicit addressing (also called as functional addressing) scheme, the primitive specifies the process with which it has to communicate but not the process ID. Client server communication uses this addressing scheme.
   - The primitives used are ***send_any(process_ID, message)*** and ***receive_any(process_ID, message).***

- After selecting either explicit or implicit process addressing, the next step is to know addressing methods:

1. **Two-level addressing:**
   - The simplest method is the two-level addressing scheme whose syntax is ***machine_ID@local_ID (receiver machine name).***
   - The machine_ID is the address of the receiving machine so that the sending machine's kernel can identify the receiving machine.
   - Local_ID is the process ID of the receiver machine, which forwards the message to the corresponding process.
2. **Three-level addressing:**
   - It supports process migration.
   - This technique is also called the link-based process addressing.
   - The syntax of this address is ***machine_ID@local_ID@machine_ID(node where the process was created @ generated by the first machine@ last known location of the machine).***
   - The first two IDs remain fixed during the life cycle of the process, but the third ID specifies the last known location of the process.(fig 3-18)

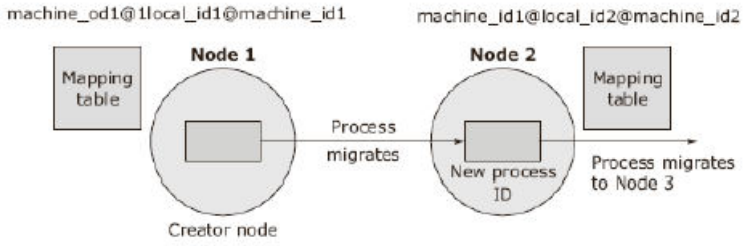


Figure 3-18 Link-based process addressing

- Both process-addressing mechanisms do not provide transparency, because the user decide the process address based on the knowledge of the location.
- Hence, this technique is non-transparent to the user and do not align with the gal of the distributed system.
- The following two techniques are used to maintain transparency:
  - A centralised process_ID allocator maintains a counter, which is incremented by 1 when a new request is made for an ID.
  - The other technique is to use a two-level naming scheme, which consists of a high level machine ASCII string name and a low-level machine dependent name.
  - The syntax used is ***machine_ID@local_ID.***
  - The centralised name server maintains a mapping table for both level names.

Failure handling Mechanism

- A good message-passing IPC protocol should have failure handling mechanism.
- During IPC, the failures may lead to sever problem.
- Failure in the communication link or a receiver node shut down can lead to either loss of the request message or the response message.
- The receiver node may crash during the processing of a request, leading to an unsuccessful execution of request.
- The various IPC problems are shown in fig. 3-19:

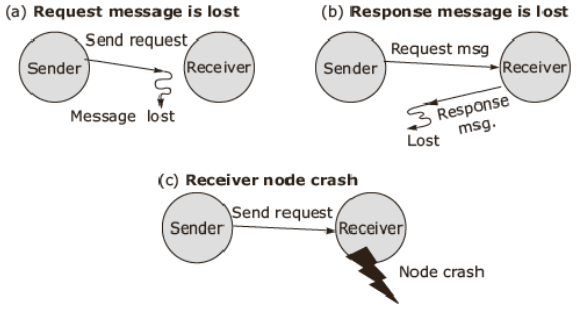


Figure 3.19 IPC problems due to system failures

- A reliable IPC protocol is designed based on the retransmission of message after timeout.

- The timeout period is alittle more than a round-trip message traversal time and the time required to execute the request.
- Based on the number of messages required to complete the message transfer, the IPC protocol are classified as:

1. **4-message reliable IPC protocol:**
   - As shown in fig. 3-20, the communication between two processes commences when client sends a request message to the server.
   - The server kernel sends an acknowledgement to the client.
   - Suppose the client does not receive the acknowledgement in the specific timeout period, it retransmits the message.
   - Once the server kernel completes the execution of the request, it sends the reply message to the client along with the result.
   - Client sends an acknowledgement message to the server.

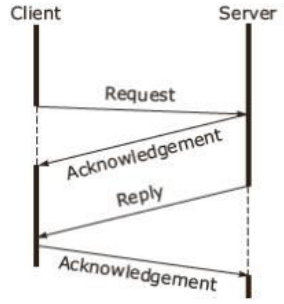


Figure 3-20 4-message reliable IPC protocol

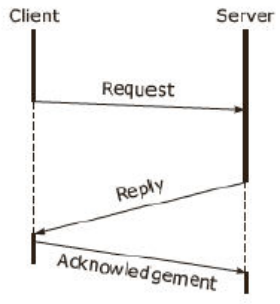


Figure 3-21 3-message IPC protocol

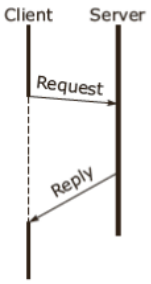


Figure 3-22 2-message IPC protocol

2. **3-message IPC protocol:**
   - In the 3-message IPC protocol shown in fig. 3-21, the client sends a request to the server that processes the request and sends the reply to the client.
   - The client blocks itself until the reply is received or the timeout period is complete, else it retransmits the message.
   - Now the kernel of the client machine sends an acknowledgement to the server.
   - If, due to any failure, the server do not receives an acknowledgement within the timeout period, it retransmits the reply message.
3. **2-message IPC protocol:**
   - This protocol prevents unnecessary retransmission of requests.
   - The client sends a request to the server and remains blocked until the server sends a reply.
   - The server processes the request and returns the reply message to the client.
   - If the reply is not received in the timeout period, the client retransmits the request as shown in fig. 3-22

- Based on the 2-message IPC protocol, fault tolerance communication between the client and the server Is shown in fig. 3-23:
  - Suppose the client sends a request to the server(1).
  - And this request gets lost. Within a predefined timeout period, the client does not receive the reply, so it retransmits the request (2).
  - The server receives the request and starts executing it. But unfortunately it crashes and thus is unable to send the reply to the client. So after a defined timeout period, the client retransmits the request(3).
  - Meanwhile, the server has restarted, is able to execute the request, and sends the reply to the client. However, the response is lost in transit. After another timeout, the client retransmits the request (4).
  - This is successfully received by the server, executed, and the response
  - This is worst case scenario. Failure occurs at each and every point, but carry out IPC.
  - This is also termed as last-one semantics.

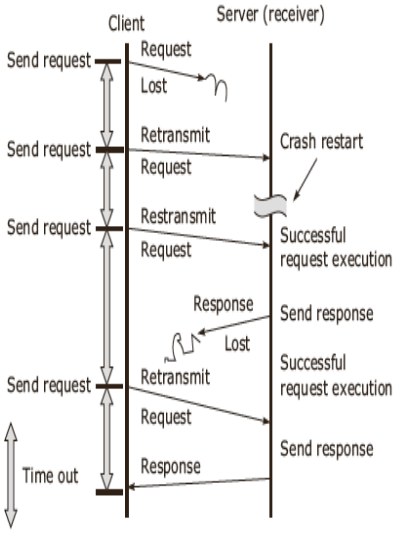


Figure 3-23 Failure handling mechanism

A case study on IPC in MACH

- MACH is a distributed system capable of functioning on heterogeneous hardware.
- It incorporates multiprocessing support.
- It is exceedingly flexible, able to run on everything from shared memory systems to systems with no memory shared between the processors.
- MACH is designed to run on computer systems ranging from one to 1000 of processors.

- MACH sends messages between location-independent ports and these messages contain type data for ease of interpretation.

Components of MACH IPC

- The two components of MACH are:

1. **Message:**
   - Messages are sent to these ports to initiate operations on the objects by the routines that implement the objects.
   - By depending only on ports and messages for all communication, MACH delivers location independence of objects and communication security.
2. **Port:**
   - A port is implemented as a protected, bounded queue within the kernel of the system on which the objects resides.
   - If the queue is full, a sender may abort the send, wait for a slot to become available in the queue.

- The system calls which provide the port functionality do the following:
  - Allocate a new port in a specified task and give the caller's task all access rights to new port.
  - Deallocate a task's access rights to the port.
  - Get the current status of a task's port.
  - Create a backup port, which is given the receive rights for the port.
- When a task is created, the kernel creates several ports for it.
- The function task_self returns the name of the port that represents the task in calls to the kernel.
- For instance, for a task to allocate a new port, it would call port_allocate with task_self as the name of the task that will own the port.
- Another port created for a task is returned by task_notify, and it is the name of the port to which the kernel will send event-notification message.

MACH message format

- A message consists of a fixed length header and a variable number of type data objects.
- The header contains the destination's port name, the name of the reply port to which return message should be sent, and the length of the message.
- The data in the message were limited to less than 8k in MACH 2.5 systems.

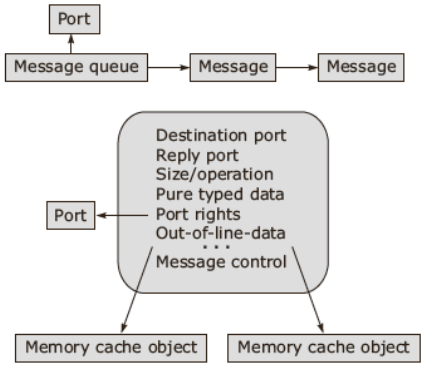


Figure 3-24 Mach message

NetMSgServer

- MACH uses the NetMsgServer to send a message between computers, provide location transparent naming, and transport to extend IPC across multiple computers.
- MACH provides a primitive network-wide name service that allows tasks to register ports for lookup by tasks on any other computer in the network.
- MACH ports can be transferred only in messages, and messages must be sending to ports.
- The NetMsgServer tracks all the rights and out-of-line memory passed in inter-computer messages and arranges for the appropriate transfers.
- The NetMsgServer maintains among themselves a distributed database of port that are transferred between computers and of the ports to which these rights correspond.
- The kernel used the NetMsgServer when a message needs to be sends to a port that is not on the kernel's computer.
- The NetMsgServer on the destination computer uses that kernel's IPC to send the message to the correct destination task.

Concepts of group communication

- The basic form of message-based communication is point-to-point communication which takes place between a single-sender and single-receiver process.
- However, in parallel-distributed applications, a message-passing system should provide group communication facility.

Types of group communication

1. **Unicast communication:**
   - In unicast group communication, as shown in fig. 3-25, individual packets are transmitted to each member of group. It works for small groups.
   - A typical eg. Is a process sending message to another process residing on another node.

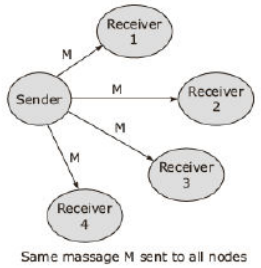
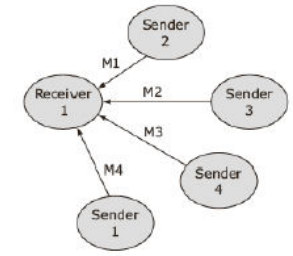


Figure 3-25 Unicast communication

2. **Many-to-one communication:**
   - Many-to-one communication implies many processes residing on different machines sending message to a single process in a node.
   - An eg. Is, multiple file requests coming to the file server from various client (fig. 3-26).



Multiple senders send message to the same receiver
Figure 3-26 Many-to-one communication

3. **Multicast communication**
   - It is a type of on-to-many group communication, also called as multicasting.
   - It is widely used in distributed systems (fig. 3-27).

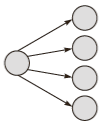


Figure 3.27 One-to-many communication

4. **Broadcast communication**
   - It is a special type of multicast communication (fig. 3-28).
   - One processor sends the same message to several destinations with a single operation.
   - A common eg. Is clock synchronization in a distributed system.

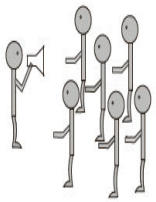


Figure 3-28 A broadcast sends a message to all the recipients

- The other methods of classification depends on how the group can interact with other groups outside the system
- These groups are:
  1. **Closed group:** Parallel processing applications use closed groups where processes have their own goals and do not interact with the outside world.
  2. **Open groups:** In an open group, any outside process can send a message to group as a whole.
- Groups can be classified according to the decision making capability:
  1. **Peer group:** All processes are equal and there is no single point to failure.
  2. **Hierarchical group:** A co-ordinatordecides which worker should carry out the tasks.

Group Management:

- A message-passing system with a group communication facility should provide flexibility for dynamic group management.
- Processes should have the autonomy of joining or leaving a group at any moment of time (fig. 3-29).
- The process of leaving and joining a group should be synchro-nous with message sending.
- It implies that when a member joins the group, it should receive all the messages it is supposed to, and when a member leaves the group, no message should be sent to it.

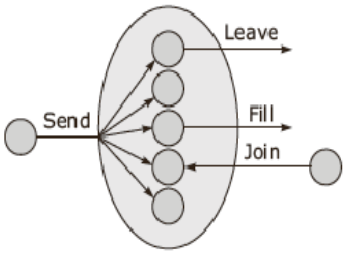


Figure 3-29 Group dynamics

- The following two techniques are commonly used for group manageme
  - **Centralized approach:**
    - A commonly used technique is the use of a centralized server which accepts requests for addition or deletion of members.
    - However, this method suffers from the problem of reliability and scalability.
    - The solution to this problem is replication of the group server.
  - **Distributed approach:**
    - The other technique is to carry out distributed management by using multiple coordinators.
    - Care has to be taken if some members are down and the group is restarted later on.

Group Addressing:

- In a large distributed system spread across multiple LANs, the processors are widely spaced across the network.
- Hence, it is advisable to send individual messages to each member of the group.
- The kernel of the sender process formats the message and contacts the group server to identify the low-level name of the receiver process belonging to the group.
- Based on the message content, the kernel of the receiver process forwards the message to the process identified by the process ID.
- The sender is not aware of the group size or the mechanism for group addressing.
- The operating system is responsible for message delivery to all the members of the group.

Message Delivery:

- Multicast communication is asynchronous in nature because the sender cannot wait until all the receivers are ready to receive the message.
- In addition, the sender process is not aware of the members belonging to the group.
- On the receiver's side, the message is buffered, i.e. the message can be received when the receiver is ready to receive the message.
- There are two methods of group communication semantics for understanding the aspect of reliable communication:
    - **Send-to-all semantics:** As the name implies, the message is sent to all the processes of the multicast group and is buffered until accepted by the receiver process.
    - **Bulletin board semantics:** The message is addressed to the channel and is sent over the network. The bulletin board semantics is advantageous, since it takes into account the receiver process state.

Reliability Mechanism:

- **The reliability mechanism is shown below:**

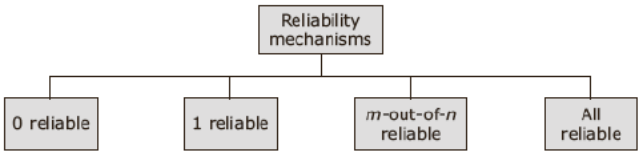


Figure 3-30 Reliability mechanisms

- In the 0-reliable scheme, the sender expects no response from any receiver. A common example is synchronous multicast like a time generator.
- In the 1-reliable scheme, the sender expects response from any on receiver. For eg. ,when the server manager multicasts a message to search for a volunteer, only the first response is considered.
- The m-out-of-n reliable scheme is designed such that a response is expected from m out of n receivers where (1<m<n).
- The all reliable scheme requires a response from all the members of the group. For eg. , an operation such as updating the replicas of a file is multicast to all file servers which have a replica of the file.

Message Ordering:

- In this type of communication, multiple senders send messages to a single receiver.
- A selective receiver specifies a unique sender, i.e. the message transfer takes place only from that sender.
- While in case of non-selective receivers, a set of senders are specified from which the message transfer takes place
- There are 3 types of message ordering:

1. **Absolute ordering:**
    - Absolute ordering semantics ensure that the Sender delivers the entire message to all the Receiver processed in the exact order in which They were sent.
    - This semantic can be implemented by using a global timestamp embedded in the message as an ID.
    - Each machine in the group has its clock synchronized with the system clock.
    - The receiver machine kernel saves all messages from one process into a queue.
    - The semantic of absolute ordering are shown in fig 3-32.
    - S1 sends a message to R1 and R2 at time t1 followed by S2 sending the message to R1 and R2 at time t2.
    - Two timestamp are attached r1 and r2.
    - Based on the absolute ordering semantics, R1 and R2 receive the message m1 and m2 and will order them as m1 followed by m2.

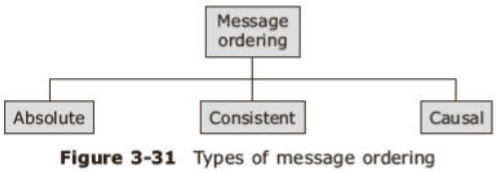


Figure 3-31 Types of message ordering

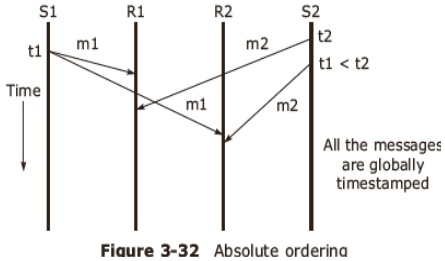


Figure 3-32 Absolute ordering

2. **Consistent ordering:**
    - This semantic is implemented using a single receiver called*sequencer* which assigns a sequence number to all messages and then multicasts them.
    - The receiver kernel saves all the messages, checks the sequence number, orders then and sends them to the receiver process.
    - This technique can suffer failures at sequencer level.
3. **Casual ordering:**

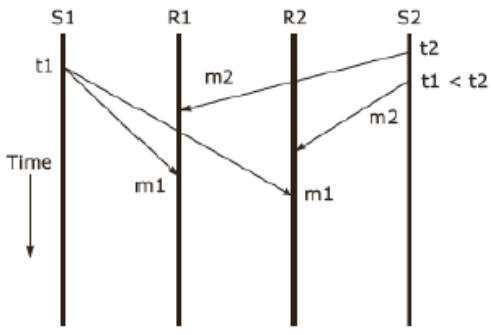


Figure 3-33 Consistent ordering

- Another type of weaker semantic is casual ordering which gives better performance than other techniques.
- Two message-sending events are casually related, if they follow the 'happened before' relation i.e. they are influenced by each other.
- This semantic ensures that the two messages are delivered to all receivers in correct order, if they are casually related to each other (fig 3-34).

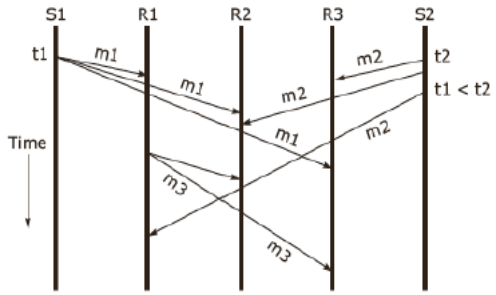


Figure 3-34 Causal ordering

Case study of group communication CBCAST in ISIS

- The CBCAST (casual Broadcast) protocol of the ISIS system implements casual ordering semantics.
- Each member maintains a vector of ***n*** components whose ***ith*** component belongs to the member with a sequence number i.
- The protocol is implemented as follows:
  - The sender process increments its own component value in its own vector by 1 and sends the message with the vector.
  - The runtime system buffers the message to check whether it should deliver or delay the message to ensure casual ordering semantics
  - The runtime system tests the following conditions:

    S[*i*]=R[*i*]+1
  - This test ensures that the receiver has not missed any message from the sender to entire reception of casually related messages

    S[*j*]<=R[*j*] *for j<>i*
  - This test ensures that the sender has not received any message which the receiver has not yet received, i.e. sender's message is not casually related to the message missed by the receiver, where:
    - S is a vector of the sending process attached to the message.
    - R is the vector of the receiving process
    - 'i' is the sequence number of the sender process.
    - After passing both the tests, the message is delivered to the user. Otherwise, the test is repeated later when a new message arrives.

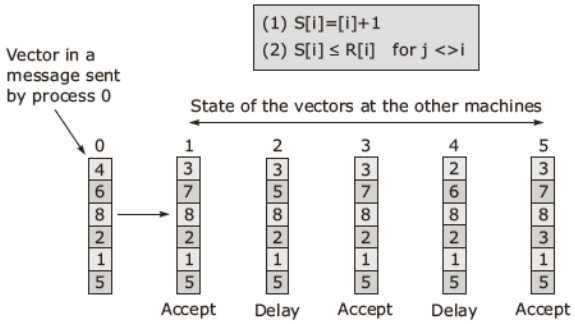


Figure 3-35 CBCAST in ISIS

- Consider eg. Shown in fig 3-35 which illustrate this algorithm.
- In this eg. There are 6 processes, 0,1,2,3,4,5.
- The status of their vectors at processes 0-5 a specific instant of time are(3,6,8,2,1,5), (3,7,8,2,1,5), (3,5,8,2,1,5), (3,7,8,2,1,5), (2,6,8,2,1,5) and (3,7,8,3,1,5) respectively.
- This means that until now, process 0 has sent three messages, 1 has sent seven messages, 2 has sent 8 messages, 3 has sent two messages, 4 has sent 1 message and 5 has sent 5 message.
- Now process 0 sends a new message to other process and attaches a vector to the message (4,6,8,2,1,5).
- The message can be sent to process 1,3 and 5 because it passes both the tests.
- But the message has to be delayed by the runtime system of sites of processes 2 and 4 because one of the tests fails at both processes.
- Thus vector method assists in implementing casual ordering semantics in the CBCAST protocol of the ISIS system.

API for Internet Protocol

1. **Synchronous and Asynchronous Communication:**
   - Communication between two processes usually consists of two operations: Send and Receive.
   - One process sends a stream of bytes to a destination and another process at the destination receive the byte stream.
   - This process may involve synchronization between two processes.
   - We define the send and receive operation in Java API.
   - There is queue attach message destination.

- A sender remotely adds message to the queue and the destination process removes the messages from the local queue.
- In Synchronous communication, both the sending and receiving processes synchronize at every message so that both send and receive primitives are blocking operations.
- When send is issued the sending process is blocked until the receive command is issued and the other process is blocked until the message is received.
- In Asynchronous communication, the send is non-blocking command,
- The issuing process is allowed to continue when the message is stored in a local buffer and then both the transmission of the message and the sender process continues simultaneously.

2. Sockets, UDP and TCP

Sockets:

- Sockets are used in both UDP and TCP types of message communication.
- Sockets abstraction provides an end-point communication.
- A message is transmitted from a socket in one process to a socket in another process.
- The process binds its socket to a local port and to one of the internet addresses of the host computer.
- Then the socket is ready to receive a message sent to that pair.
- Processes may use the same socket for sending and receiving messages.
- Each computer has 2^{16} port numbers to choose from.

UDP and TCP Protocol:

- Both TCP and UDP provide communication facilities at the transport level.
- They establish process-to-process communication with the help of ports.
- UDP provide connectionless communication using a datagram packet.
- TCP protocol provides connection-oriented communication between two processes.

UDP datagram communication:

- A datagram is transmitted from the client process and sent to the server process, which receives it.
- To communicate message to and fro, a process must create a process and bind it to local port and to the internet address of the local host.
- Certain norms must be followed for datagram communication:
    1. The receiving process should clarify the array size in which to receive a message.
    2. Sockets offer non-blocking send and blocking-receive commands for datagram communication. Some implementations also give the option of non-blocking receive.
    3. Usually the receive method can receive a message from any source. So the receive method returns the internet address and the local port of the sender, allowing the receiver to check the origin of the message.
    4. The message communication should be reliable and valid. Checksum is a good way of measuring correctness. Other errors that may occur include failure doe to omission of message. Datagram messages can be dropped due to checksum error of less buffer space.
- UDP protocol may be used for some services where sometimes the messages may be dropped.

TCP datagram communication:

- TCP is connection-oriented protocol.
- It ensures correct delivery of a long sequence of bytes using stream-based programming abstraction.
- I t maintains both of a long the correctness and sequence of the message delivery.
- TCP establishes a bidirectional communication channel before starting the message transfer.
- A virtual connection is set up between the source and the destination to give reliable transmission.
- It is transparent to the intermediate nodes.
- The data packets each the destination in a sequence but may not follow the same route.
- TCP ensures reliable delivery.
- It takes certain precautions such as sequencing, flow control, checksum, retransmission and buffering.

Unit 3

Formal Model Specifications and Remote Communication

Basic concepts of formal model definitions, Different types of communication systems, algorithms for message passing systems, Basic concept of middleware, Remote Procedural Call (RPC), a case study on Sun RPC, Remote Method Invocation (RMI) along with a case study on Java RMI.

Basic concepts of formal model definitions

Different types of communication systems

Algorithms for message passing systems

Basic concept of middleware

- Distributed Applications are difficult to build with IPC protocols tailored only for specific applications.
- RPC is an extension of a local procedural call, which allows client programs to call server programs running on separate processes on remote machines.
- RPC packages the message more like a conventional program and is easy to use.
- Middleware is software which provides a programming model one level above the basic building block of processes and also provides message passing.
- This layer uses protocols like request-reply protocol between processes to provide higher-level abstraction.
- Middleware provides location transparency and independence form the details of communication protocols, operating systems, and hardware.
- RPC and RMI are the examples of middleware.
- The various layers of the middleware are:
    - **Location Transparency:** The clients need not know which server process is running its RPC- whether it is local or remote or where it is located.
    - **Communication protocols:** The processes supporting middleware abstraction are independent of the lower-level transport protocols.
    - **Computer hardware:** RPC uses techniques like external data representation to hide the heterogeneous ordering from the user.
    - **Operating system:** The higher layer abstraction provided by the middleware is independent of the underlying operating system.
    - **Use of programming languages:** Middleware is designed using programming languages like java, CORBA, or IDL.

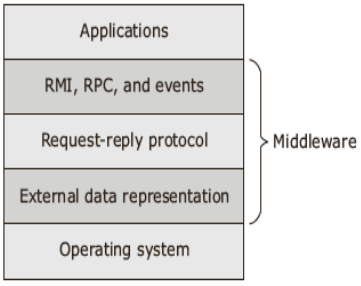


Figure 4-1 Role of middleware in remote communicatior

Remote Procedural Call (RPC)

- The RPC model is similar to the commonly used local procedure call (LPC) model that is used to transfer data and control within program.
- We describe the steps involved in LPC execution and then compare it with RPC.
- In the LPC model, the caller places arguments to a procedure in a specified location-memory or register.
- Then, the caller transfers the control to the procedure.
- The procedure executes in new environment which consists of the arguments given in the calling instruction.
- After completion, the caller regains control, extracts the results of the procedure, and continues its execution.

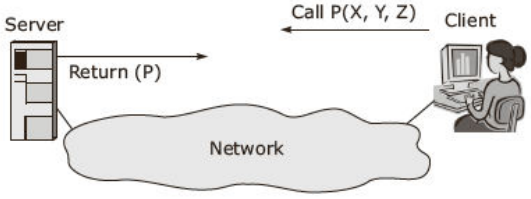


Figure 4-3 Basic RPC model

- In RPC, the caller process and the server process execute on different machines
- First, the caller process sends a call message with procedure parameters to the server process.
- Then, the caller process suspends itself to wait for a reply message.
- A process on the server side extracts the procedure parameters, computes the results, and sends a reply message back to the caller process.
- In fig. 4.3 an RPC process on the client machine calls a procedure P(X, Y, Z) on the server at which point the calling process is suspended and the server initiates the execution of the called process.

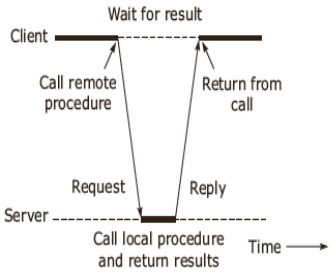


Figure 4-4 A typical RPC

- When the called process receives the parameters X, Y, Z, it computes the result 'p' and sends a reply message to the calling process.
- Figure 4-4 shows hoe only one process is active at a time.
- The client process is in the blocked state and may do other useful work while waiting for reply from the server.
- In RPC, the calling and the called procedures run on different machines and execute in different address spaces.
- Messages passing carries out information exchange between the two processes and is transparent to programmer.
- RPC implementation mechanism involves following steps.
- Client
- Client stub
- RPC runtime
- Server stub
- Server
- The following ten steps explain a complete RPC execution process :

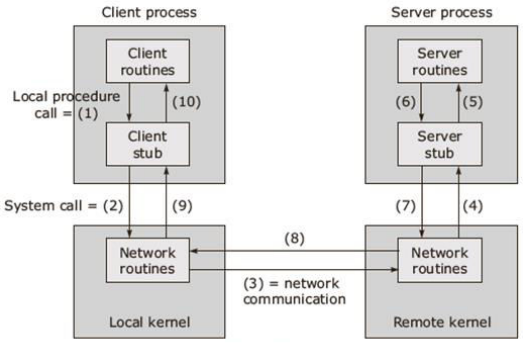


Figure 4-5 RPC execution

1. The client procedure calls the client stub in the normal way.
2. The client stub builds a message and calls the local operating syst
3. The client's as sends the message to the remote as.
4. The remote as gives the message to the server stub.
5. The server stub unpacks the parameters and calls the server.
6. The server does the work and returns the result to the stub.
7. The server stub packs it in a message and calls its local as.
8. The server's as sends the message to the client's as.
9. The client's as gives the message to the client stub.
10. The stub unpacks the result and returns to the client.

Stub generation

- **Manual generation :**
  - It is through a set of standard translation function.
- **Auto generation using Interface Definition Language (IDL):**
  - It is through the IDL(Interface Definition Language).
  - It consists of a list of procedure names and supported arguments and results.
  - The programmer writes the RPC interface using IDL
  - The client imports the interface, while the server program exports the interface.
  - An IDL compiler processes the interface definition in different languages so that the client and server can communicate using RPC.
  - The PRC compilation is shown in the fig:

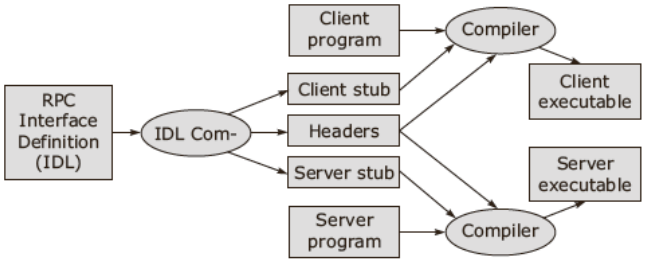


Figure 4-6 Steps for RPC compilation

RPC implementation

- RPC messages:
  - Call / Request
  - Reply

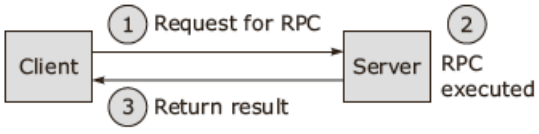


Figure 4-7 RPC messages

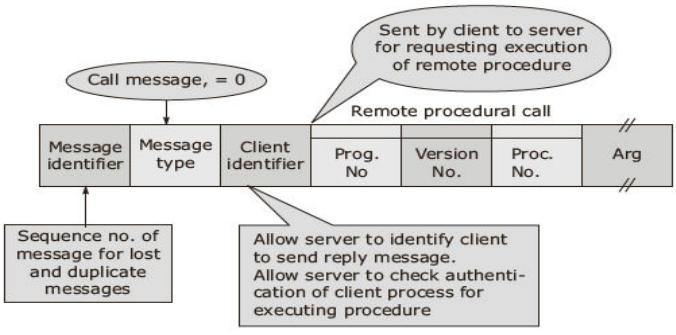


Figure 4-8 RPC call/request message format

Table 4-1 RPC reply message conditions

Condition	Response from the server
Server receives an unintelligible call message, probably because the call message has violated the RPC protocol.	Rejects the call.
Server receives the call messages with unauthorized client ID, i.e. the client is prevented from making the RPC request.	Return reply unsuccessful, and does not execute RPC.
Server does not any receive procedure ID information from the message ID field—program number, version number, or ID.	Return reply unsuccessful, and does not execute RPC.
If all the above conditions are satisfied, the server executes the RPC, but may not be able to decode its arguments due to incompatible RPC interface.	Return reply unsuccessful and does not execute RPC.
Server executes the RPC but an exception condition occurs.	Return reply unsuccessful.
Server executes the RPC successfully without any problems.	RPC is successful and the server returns the result.

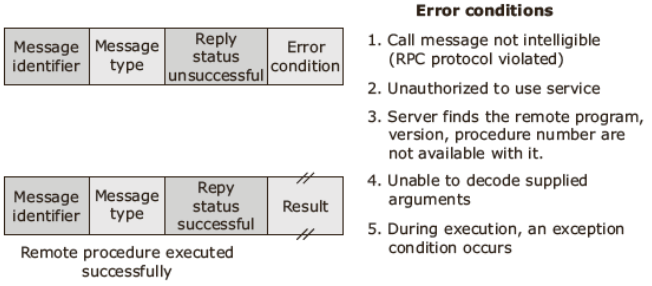


Figure 4-9 RPC reply message format

Issues in RPC implementation:

- Exception handing and security
- Failure handling
- Optimizing RPC execution
- Various types of complicated RPCs

A case study on Sun RPC

- Many RPC systems have been built and are in use today such as Sun, Cedar, Courier etc.
- Sun RPC uses automatic stub generation and also provide the users with the flexibility to write stubs manually.
- An application's interface definition is written in an IDL called RPCL, and extension of Sun XDR.
- It uses Rpcgen compiler which generates the following:
  - A header file containing definitions of common constants and types defined in the interface definition file, external declarations for marshalling and unmarshalling procedures which are automatically generated.
  - An XDR filter file containing XDR marshalling and unmarshalling procedures which are used by client and server stub procedures.
  - A client stub file containing one stub procedure for each procedure in IDL.
  - A server stub file containing main routine, the dispatch routine and a stub procedure for each procedure defined in the interface definition file plus a null procedure.
- The RPC application is created using the files generated by the Rpcgen compiler based on the following steps:
  - The application programmer manually writes the client program as well as the server program.
  - The client program file is compiled to get a server object file.
  - The client stub file and the XDR filters are compiled to get a client stub object file.
  - The server stub file and the XDR filters are compiled to get a server stub object file.
  - Client object file, client stub object file and client stub RPC runtime library are linked together to get a client executable file.
  - Server object file, client stub object file and client stub RPC runtime library are linked together to get a client executable file.
- A sun RPC remote procedure can accept only one argument and return only one result.
- Hence procedures requiring multiple parameters as input or output must include them as components of a single structure.
- To handle differences in data representations, data structures are covered to XDR and back, using marshalling procedures.
- The RPC data structures are converted to XDR and back, using marshalling procedures.
- The RPC runtime library has procedures for marshalling integers of all sizes, character, strings, reals etc.
- Sun RPC supports 'at least once' semantics.
- After sending a requested message, the RPC runtime library waits for a user-defined timeout period forthe server to reply retransmitting the request.
- Each node uses a local binding algorithm called *port-mapper.*
- The server side error handling procedures which process the detected errors sends reply to the client indicating the detected error.
- The client side error handling provides flexibility to choose error reporting mechanism.
- Sun RPC supports asynchronous call-back, broadcast and batch mode RPC.
- A few disadvantages of Sun RPC include location transparency, no general specification of procedure arguments and no supports for network binding service.

Remote Method Invocation (RMI)

Distributed Object Concept:

- Object consists of a set of data and its methods.

- Objects encapsulate data called the state.
- An object invokes another object by invoking its methods i.e. passing arguments and parameters.
- Methods are various operation performed on data, which can be availed through interface.
- Method invocation accesses or modifier the state of the object.
- Users can avail the methods through the object's interface.
- Multiple interfaces can also implement objects.
- Objects are efficiently used in a distributed system by being accessed only through their methods remotely.
- Objects in other processes can invoke methods that only belong to its remote interface.
- As shown in fig. 4-22, local objects are invoked in the remote interface and in other methods implemented by the remote object.

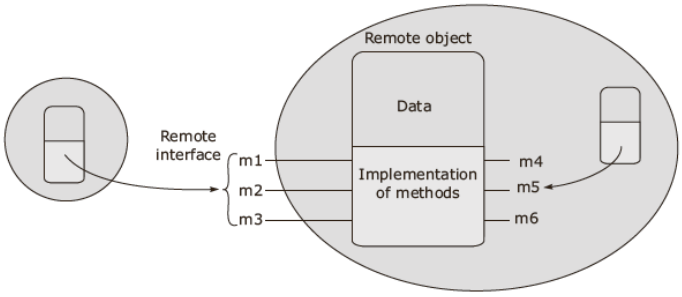


Figure 4-22 Remote object and remote interface

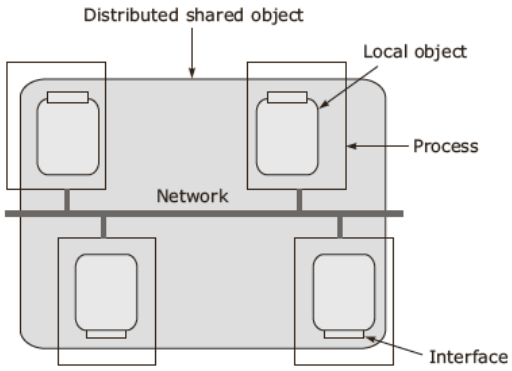


Figure 4-23 Distributed shared object

- The fig 4-23 shows how processes across the network share objects.
- A distributed system uses the client-server architecture.
- The server manages the objects and clients invoke the methods-called the RMI.
- The RMI technique sends the request as a message to the server which executes the method of the object and returns the result message to the client.
- The state of the object is accessed only by the method of the object i.e. unauthorized methods are not allowed to act on the state of the operation.
- As seen in fig 4-24, objects B and F are remote objects.
- All objects can receive local invocation as long as they hold reference to them.
- Two important concept are:
    - *Remote objects reference:* Client can invoke methods of a remote objects interface. B is available to A. It is a unique system-wide ID which refers to a specific remote object.
    - *Remote interface:* Every client has a remote interface which specifies the methods invoked remotely.

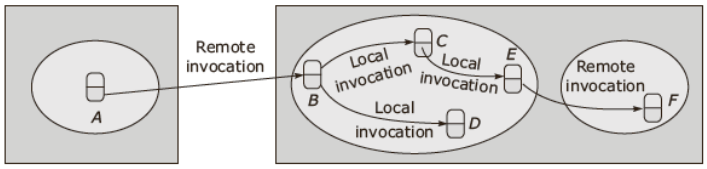


Figure 4-24 RMI and LMI

RMI implementation:-

→<u>Design issues in RMI</u>

Level of transparency

- This involves hiding, marshalling, message passing, locating and contacting the remote object for the client.
- RMI is more prone to failures than LMI.
- The typical RMI flow diagram is shown in figure 4-25.
- As shown in fig 4-26, when the client binds to the distributed object, the object interface called the proxy is loaded in the client address space.
- A proxy is similar to the client-server stub of RPC.
- It marshals the RMI request into the message before sending it to a server and unmarshals the reply message of method invocation when it arrives at the client site.
- The Server-side machine invokes the object and offers the same interface as if it resides on the client machine.

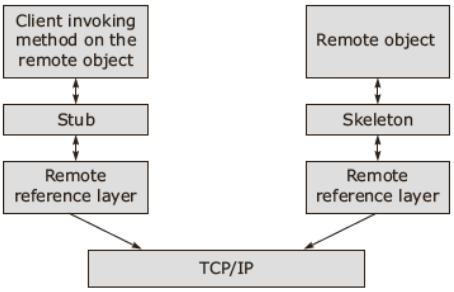


Figure 4-25 RMI flow diagram

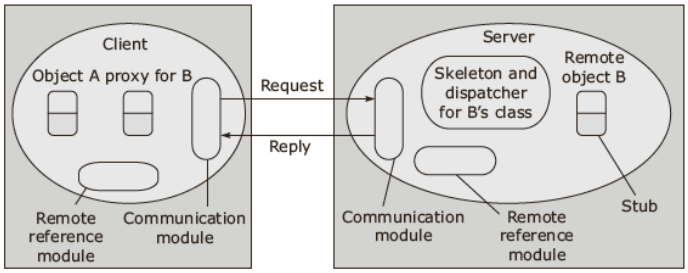


Figure 4-26 RMI components

RMI invocation semantics

- Maybe semantics:
  - With these semantic, the client may be not known whether the remote method is executed once or not at all. This semantic is useful in application where failed invocations are acceptable.
- At-least-once semantics:
  - The invoker receives the result, implying that the server has executed the method at least once, or an exception informing that no result was received.
- At-most-once semantics:
  - The client receives the result, implying that the method executes exactly once, or an exception informing that no result was received so far.

Table 4-2 Invocation semantics

Fault tolerance measures			
Retransmit requestmassage	**Duplicate filtering**	**Re-execute procedure of retransmit reply**	**Invocation semantics**
No	Not applicable	Not applicable	Maybe
Yes	No	Re-execute procedure	At-least-once
Yes	Yes	Retransmit reply	At-most-once

→<u>RMI Execution</u>

- **Communication module:** The client and the server processes form a part of the communication module which uses RR protocol. The format of the request-and-reply message is very similar to the RPC message.
- **Remote reference module:** It is responsible for translating between local and remote object reference and creating remote object references. It uses remote object table which maps local to remote object references. This module is called by the component of RMI software for marshalling and unmarshalling remote object references.
- **RMI software:** This is the middleware layer and consists of the following:
  - **Proxy:** It makes RMI transparent to the client and forwards the message to the remote object. It marshals the arguments and unmarshalls the results and also sends and receives messages from the client.
  - **Dispatcher:**A server consists of one dispatcher and a skeleton for each class which represents a remote object. This unit receives the request message from the communication module, selects the appropriate method in the skeleton and passes the request message.
  - **Skeleton:**It implements the method in the remote interface. It unmarshalls the arguments in the request message and invoke the corresponding method in the remote object,

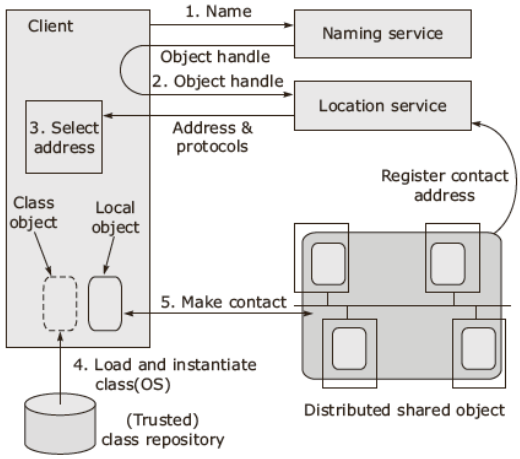


Figure 4-27 RMI implementation

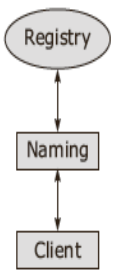


Figure 4-28 Locating remote objects

- **Server program:** The server program contains classes for dispatcher, skeleton and the remote object it support.

- **Client program:** The client program contains classes of processes for the entire remote object, which it will invoke.
- **Binder:** An object A requests remote object reference for object B. The binder in a distributed system is a service which maintains a table of textual names to be remote object-referenced.(Fig 4-28).

→Types of Objects

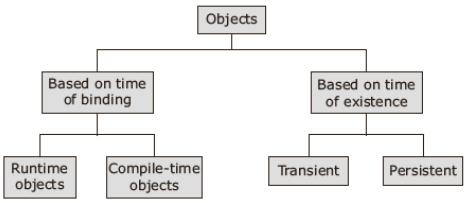


Figure 4-29 Types of objects

- **Runtime object:**Distributed applications are built easily using compile-time objects. These types of objects are dependent on the programming language. To avoid this dependency, they are bound at runtime.
- **Compile-time object:**The object adapter acts as a wrapper around the implementation details to give it the appearance of an object. An interface is defined to implement the object which can be later registered. The interface provides an image of remote object to the client, as in fig. 4-30.
- **Persistent:**These objects exist even if they are not contained in the server process address space. It implies that the client manages the persistent object and can store its state on any secondary storage and exit.
- **Transient:**They exist only for the time when the server manages
- the object. When the server ceases to exit, the object gets vanished.

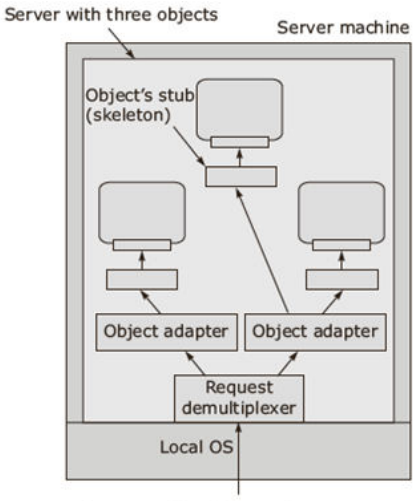


Figure 4-30 Object adapter

→Binding a Client to an Object

- As compared to RPC, RMI provides system-wide object reference so that objects can be freely associated between processes on different machines.
- A proxy placed in the process address space implements the interface containing the method, which the process can invoke.
- Fig. 4-31 explains the process of binding a client to an object.
- Binding can be classified as *explicit* and *implicit.*
- **Implicit Binding**: The client is transparently bound to the object when the reference is resolved to the actual object.
- **Explicit Binding:** The client first calls a specific function to bind the object before the method invocation.

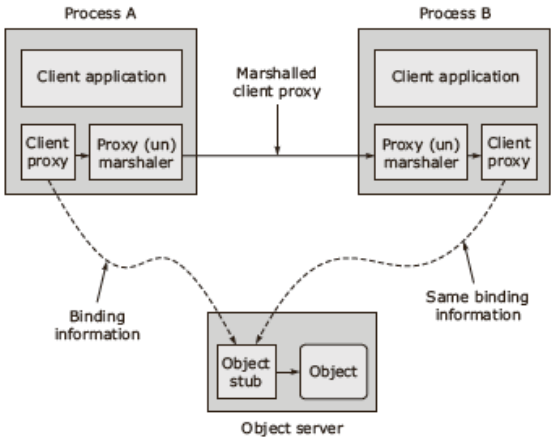


Figure 4-31 Binding

→RMI parameter passing:

- **Pass by value :**
  - References are passed by value, i.e. copied from machine to machine.
  - When the object reference is given to the process, it binds to the referred object when needed.
  - However this method is inefficient for very small distributed object.
- **Pass by reference:**
  - When the method is invoked with the object reference as a parameter, this reference is copied and passed as a value if it refers to the remote object.
  - In fact, the objects are passed as reference.
  - However, if the object is local, i.e. in the same address space as the client, it is copied as a whole and passed with the invocation.

Case study: Java RMI.

- Java hides the difference between local and remote method invocation from the user.
- After marshalling the type of object, it is passed as a parameter to the RMI in java.

- Fig. shows how the Java RMI is implemented.
- The reference to remote object consists of the network address and endpoint of server.
- It also contains the local ID for the actual object in the server address space.
- Each object in Java is an instance of a class, which contains the implementation of one or more interface.

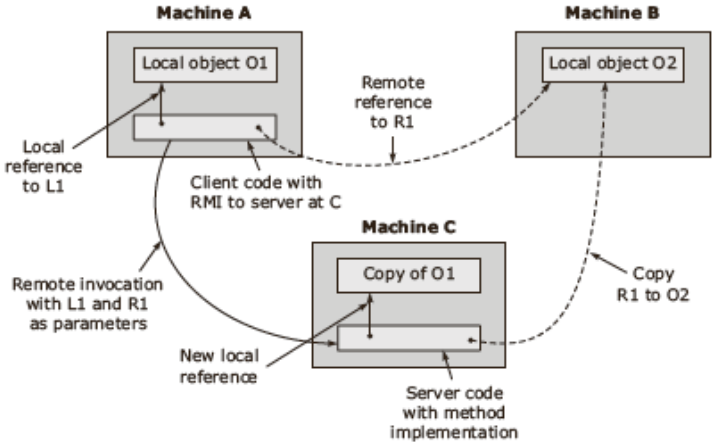


Figure 4-32 Java remote object

- A Java remote object is built from two classes:
  1. Server class:
     - This class contains the implementation of server-side code, i.e. objects which run on the server.
     - It also consists of a description of the object state and the implementation of methods which operates on that state.
     - The skeleton on the server-side stub is generated from the interference of the object.
  2. Client Class:
     - This class contains the implementation of client-side code and proxy.
     - It is generated from the object interface specification.
     - The proxy basically converts each call into a message that is sent to the server-side implementation of the remote object.

- Communication is set up to the server and is cut down when call is complete.
- The proxy state contains the server's network address, end to the server and local ID of the object.
- Hence, a proxy consists of sufficient information to invoke the methods of a remote object.
- Proxies are serializable in Java, marshalled and sent as a se unmarshalled, and can invoke methods in remote objects.
- Java RMI layer is shown in fig 4-33:

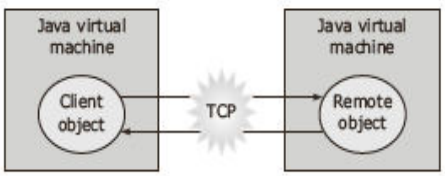


Figure 4-33 Java RMI layer

Marshalling proxies in Java:

- The implementation –handle specification classes are required to construct the proxy.
- Marshalling code is replaced with the implementation-handle for remote object reference.
- In Java, RMI reference to objects is a few hundred bytes.
- Java RMI allows object specification solution, which is flexible.
- Objects whose state rarely changes are made truly distributed.
- This is possible by copying its entire state to the client at binding time.
- The client invokes the methods on a local copy. At each invocation, the client checks the state at binding to ensure consistency.
- If the state changes, the local copy is refereshed.

Unit 4

Clock synchronization

Clock synchronization, physical and logical clocks, global state mutual Exclusion algorithms, election algorithms.

Clock synchronization

- Clock synchronization is an important aspect of any distributed system operations.

Physical clocks

- Every uniprocessor system needs a timer mechanism to keep track of time for process execution and accounting for the time spent by the process for using various resources, such as CPU, I/O, or even memory.
- In distributed system, application will have several processes running on different machines.
- Ideally, a global clock is required.
- In real systems, this is not possible.
- Each CPU has its own clock and it is required that all the clocks in the system display the same time.
- Problems with un-synchronized clocks:
    - Consider a distributed online reservation system in which the last available seat may get booked from multiple nodes if their local clocks are not synchronized.
    - Synchronized clocks enable measuring the time duration of distributed activities which start on one node and terminate on other node. For eg., there is need to calculate the time to transmit a message from one node to another node. This becomes difficult if the sender and receiver clocks are not synchronized with each other.

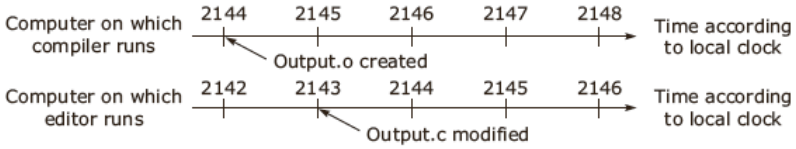


Figure 5-1 Problem with unsynchronized clocks

Implementing computer clocks

- A computer clock consists of quartz crystal, counter register, holding register, and an interrupt handler.
- The various components of the physical clock functions as follows:
    - The crystal oscillates at a predictable frequency.
    - The counter register keeps tracks of the oscillations of the quartz crystal and it decrements by one for every oscillation. When it reaches to 0, an interrupt is generated and the value is reinitialized to that of the holding register.
    - The holding register is set to an initial value, which is decided based on the frequency of oscillations of the crystal. By adjusting the holding register value and knowing the crystal oscillation frequency, the number of interrupt per second can be controlled. This is the granularity of the system clock.
- The following operations are carried out to enable the computer clock to function as ordinary clock:
    - The holding register value is chosen to be 60 clock ticks per second.
    - The computer clock is synchronized with the real-time clock by storing two values in the system: a fixed starting date and time and the number of ticks.
- The start date and time is entered during the initial booting.
- The system converts this entered value to the number of ticks after start date and time.
- The interrupt handler increments the system clock by one unit on every interrupt, called a clock tick, to keep the clock running.
- Thus, the computer clock resembles and functions as a physical clock.

Drifting of computer clocks:

- Since a quartz crystal oscillates at a constant frequency, the clock always runs at a constant rate.
- But there may be difference in the oscillation periods of two clocks running at different rates.
- For clocks based on quartz crystal, the drift rate is approx. 10^{-6} or a difference of 1 second for every 11.6 days.
- So the computer clock needs to be repeatedly resynchronized with the real-time clock to maintain perfect.
- Assume that when the real time is t, the time value of the clock is $C_p(t)$.
- If all the clocks across the world are synchronized, $C_p(t)=t$, for all p and all t.
- If C denotes the time value of the clock, then ideally dC/dt must be equal to 1.
- A clock is non-faulty of there is a limit on the amount of drift from real-time for any finite-time interval.

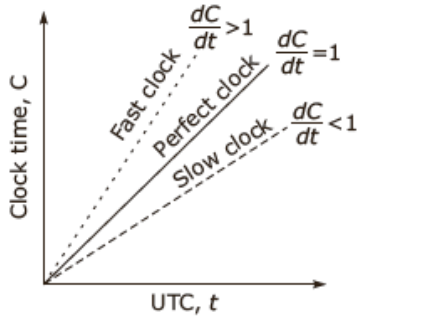


Figure 5-2 Clocks tick at different rates

[By Nishigandha Dhore]

- A faulty clock may not always maintain perfect time.
- The maximum drift allowable is p and a clock is said to be non-faulty if the following condition holds true:

$$(1-\rho) \leq dC/dt \leq 1+\rho$$ for some small constant drift.

- As shown in fig. 5-2, after synchronization with a perfect clock, slow and fast clocks drift in opposite direction from the perfect clock.
- Hence
  - ***dC/dt < 1*** for slow clocks
  - ***dC/dt >1*** for fast clocks
- unlike a centralized system where the computer clock will be synchronized with the real-time clock, a distributed system requires the following types of clock synchronization:
  - Synchronization using real-time clocks
  - Mutual synchronization among clocks within the system.

1. Synchronization using real-time clocks

- This type of synchronization, external in nature, is basically used for real-time applications.
- It allows the system to exchange timing-related information with other system and also users.
- Real-time clocks are synchronized to the official time of the day.
- After synchronization period is reached, still there is a need to resynchronize the clock periodically as seen in fig. 5-3.
- With a linear compensating function (1) the slope may any time reach dC/dt < 1. Hence, successive application of a second linear compensating function (2) can bring the time closer to the true slope.

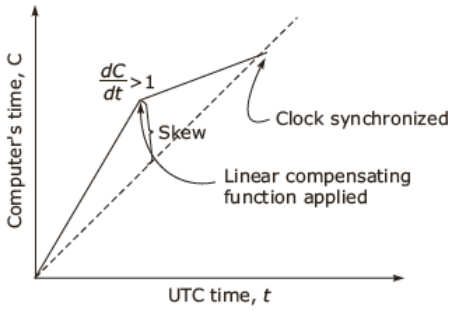


Figure 5-3 Compensating for a fast clock

2. Mutual synchronization among clocks within the system:

- Mutual synchronization method is used with application that need a uniform time across all the nodes in the system and for measurement of duration of distributed activities which are initiated and terminated on different nodes.
- Externally synchronized clocks are also internally synchronized but the converse is not true.
- This is because internally synchronized clocks may drift from external time with passage of time.

Issues in clock synchronization:

- The first issue in clock synchronization is the Ability for each node to read the other node's clock value.
- Many algorithms are available to read the clock value but apart from the actual mechanism, a node can obtain only approximate value of clock skew with respect to the clock of the other node in the system.
- It is important to compute the minimum delay between nodes by counting the time needed to prepare, transmit,and receive an empty message in absence of transmission errors and the system load.
- Another issue in synchronization is that time must never run backwards since it may lead to repetition of some operations creating havoc in the system.
- During synchronization, a fast clock has to be slowed down and a slow clock has to be speeded up.
- If the fast clock is readjusted to its actual time in one go, it may lead to running the time backwards for that clock.
- Hence, clock synchronization algorithms are designed to gradually introduce a change in the fast running clock instead of readjusting at on go. (fig. 5-4).

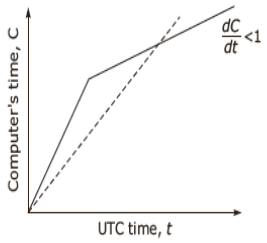


Figure 5-4 Need for clock resynchronization

Clock synchronization algorithms:

- The Clock synchronization algorithms can be classified as:

1. **Centralized algorithms:**
   - In DS using centralized clock synchronization algorithm, one node has a real-time receiver.
   - The clock time of this node is regarded as correct and is used as the reference time.
   - The goal of the clock synchronization algorithm is to keep the clock of all other nodes synchronized with the clock time of the time server node.

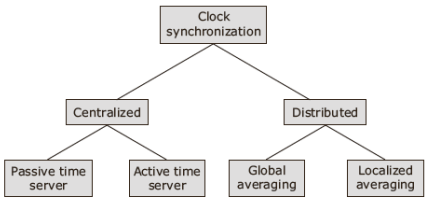


Figure 5-6 Classification of clock synchronization algorithms

a. Passive time server:

i. Each node periodically queries the time server by sending a message called 'time=?'.
ii. The period for querying is chosen to be less than or equal to twice the network latency.
iii. When the timer server node receives the message from the node, it responds with 'time=T' message. (fig,5-7).
iv. Assume that the client node has a clock time of T0 when it sends 'time= ?' message and the time T1 when it receives the 'time=T' message.
v. T0 and T1 are measured using the same clock.
vi. The best estimate of the time required for message propagation of the message 'time=T' from the time server node to the client node is (T1-T0)/2.
vii. Hence when the client node receives the reply from the time server, the client node is readjusted to:

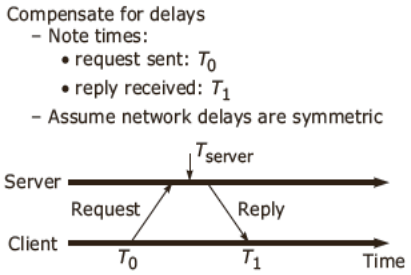


Figure 5-7 Passive time server algorithm

$$\text{Client sets time to: } T_{new} = T_{server} + \frac{T_1 - T_0}{2}$$

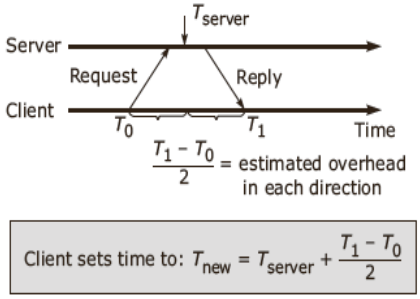


Figure 5-8 Time approximation using passive time server algorithm

b. Active time server:

i. In this algorithm, the time server periodically sends a message 'time=?' to all computers in the system.
ii. When each computer receives the message it sends back it time value to the time server.
iii. It is assumed that the time server has an idea of the time required for propagation of message from each node to its own node.
iv. Based on this value, it readjusts the clock values of the reply message.
v. Then it takes an average of a subset of these clock values including its own.
vi. This method avoids reading from unreliable clocks whose value, if chosen would have modified the actual value.

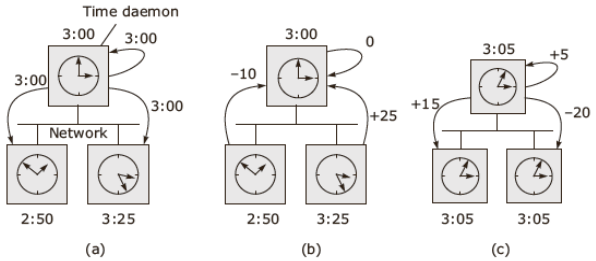


Figure 5-11 Berkeley algorithm (a) the time daemon asks all the other machines for their clock values; (b) the machines answer; (c) the time daemon tells everyone how to adjust their clock.

2. Distributed Algorithms:

- Characteristics
  - Relevant information is distributed across machines
  - Processes make decisions based only on local information
  - Single points of failure must be avoided
  - No common or global clock is available

a. Global averaging distributed algorithm:

i. In this algorithm, the clock process at each node broadcasts its local clock time in the form of a special 'resync' message when its local time is equal to T0+IR for some integer I.
ii. T0 is fixed time in the past, agreed upon by all nodes in the system, and R is system parameter which depends on the factors such as total no. of nodes in the system.
iii. The resync message is broadcast from each node at the different nodes may run at slightly resynchronization interval.

iv. But since the clocks of different nodes may run at slightly different intervals, these broadcast may not be initiated at the same time from all nodes.

b. Localized averaging distributed algorithm"

i. This algorithm overcomes the scalability limitation of the global averaging distributed algorithm.
ii. The nodes of the distributed system are logically arranged in a specified pattern such as a ring or grid.
iii. Periodically, each node exchanges, its clock time with its neighbor in the logical pattern and then, set its clock time to the average of its own clock time and the clock time of its neighbours.

Network time Protocol

- The Network Time Protocol(NTP) is an Internet standard with version 3 being used for clock synchronization across the internet.
- It enables clients across the internet to be accurately synchronized to UTC, despite of message delays.
- It provide the reliable service by surviving lengthy losses of connectivity, redundant paths and redundant servers.
- Synchronization Modes
  - **Multicast mode:**
    - It is preferred for high speed LANs.
    - It has lower accuracy but it is efficient.
  - **Procedural call mode**
    - It is similar to Cristian's algorithm.
  - **Symmetric mode**
    - In this mode, the pair of servers exchange message and retain data to improve synchronization over time and all messages are delivered unreliably via UDP.

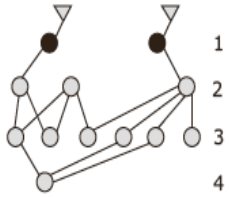


Figure 5-12 Synchronization subnet

- NTP messages are exchanged in pairs in both procedure call and symmetric mode.
- NTP calculates the offset for each pair of message and the offset between the two clocks.
- The Simple Network Time Protocol (SNTP) is based on the Unicast mode of NTP.
- It operates in multicast or procedure call and is recommended for environments where the server is the root node and the client is the leaf of the synchronization subnet.

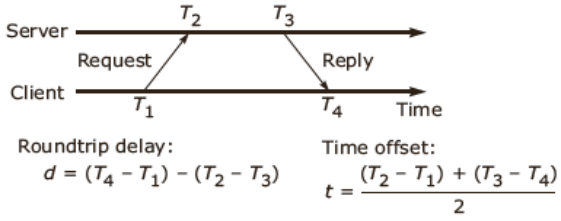


Figure 5-13 SNTP

Use of synchronized clocks

- At-most-once message delivery semantics
- Clock-based file system cache consistency

Logical clocks

- **Event Ordering:**
  - Happened-before relation:
  - It is denoted by → on a set of events that satisfies the following conditions:
    - If a and b are events in the same process and a occurs before b, then a→b
    - If a is the event of sending message by one process and b is the event of receiving the same message by another process, then a→b.
    - If a→b and b→c , then a→c.
  - Happened before relation is used to understand the concept of logical clocks.
  - The concept of logical clocks is based on the fact that clock synchronization need not be absolute.
  - If 2 processes do not interact, it is not necessary that their clocks be synchronized.

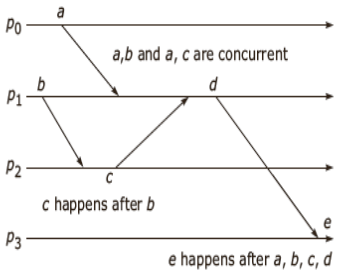


Figure 5-15 Happened-before relationships and concurrent events

Implementing of logical clocks:

- The logical clocks of the system are correct if the events of the system, which are related to each other by the happened before relation, can be properly ordered using these clocks.
- The timestamp C(e) is attached to each event e, satisfying the following properties:

- C1: if a and b are two events in the same process and a→b, then we demand that C(a)<C(b).
- C2: if a corresponds to sending a message m, and b corresponds to receiving that message, then also C(a)<C(b).
- C3: A clock C associated with the process P must always go forward, never backwards.
- Hence logical clock must always be made by adding a positive value, never subtracting from it.

a. Lamport's Timestamp:

- Consider 3 processes P1, P2, P3 running on different machines, each having its own clock running at its own speed.
- When the clock ticks4 times in process P1, it ticks 6 times in P2 and 8 times in process P3.
- Each clock runs at constant rate are different due to difference in crystal and P3's clock is running faster than others.
- At time 4, P1 sends message A to P2 which is received by P2 at 12.
- Since the message timestamp is 4, and P2's clock says 12, it means that it took 8 ticks to reach P2, which is possible.
- Similarly, the message B from P2 with timestamp of 18 reaches P3 at 40, again a possible value. This is because P2 and P3 are faster than P1.
- When P3 with timestamp 48 sends a message to P2, the time there is only 42. This is not possible and this situation should be prevented.
- A similar problem would occur when P2 sends a message D to P1 with timestamp 54 and the time there is 36.
- To avoid these situation, we use Lamport's solution which states that If C is left at 48, it must arrive at anything later than 48, the receiver's clock is forwarded to show a value say 50.
- Similarly , the message D carries a timestamp of 56 instead of 48 and P1's clock is corrected to 58.

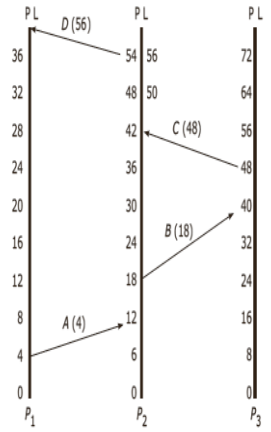


Figure 5-17 Lamport's clocks (a) Three processes, each with its own clock. The clocks run at different rates. (b) Lamport's algorithm corrects the clocks.

b. Vector Timestamp

- A vector timestamp VT(a) assigned to an event 'a' has a property that if VT(A)<VT(b) for some event b, then event 'a' is known to casually precede event 'b'.
- Each process Pi has an array V1[1..n], where Vi[j] denotes the number of events that process Pi knows have taken place at process Pj.
- When Pi sends a message m, it adds 1 to Vi[j], and sends Vi along with m as vector timestamp VT(m).
- The result is that on arrival, each process knows Pi's timestamp.
- Vector properties :
  - **VCi [i]** is the number of events that have occurred so far at **Pi.**
  - If **VCi [j] = k** then **Pi** knows that k events have occurred at **Pj.**
- The actual steps carried out to accomplish the above properties are:
  - Before executing an event, Pi executes **VCi[i]←VCi[i]+1**
  - When process Pi sends a message m to **Pj**, it sets m's(vector) timestamp **ts(m)** equal to **VCi** after having executed the previous step.
  - On the receipt of a message **m**, process **Pj** adjust its own vector by setting **VCj[k]← max{VCj[k],ts(m)[k]}** for each **k**, after which it executes the first step and delivers the message to the application.

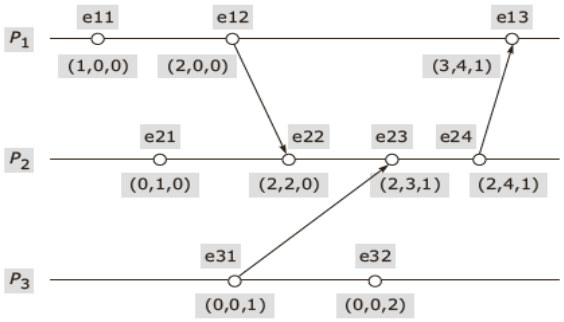


Figure 5-20 Example of vector timestamp

Global state mutual

- The global state of a distributed system consists of the local state of each process, together with the messages which are in transit.
- A local state for a distributed database system consists of only those records which form a part of the database and exclude temporary records used for computations.

[By Nishigandha Dhore]

- An effective way of recording the global state of a distributed system was proposed by Chandy and Lamport, who introduce the notion of distributed snapshot..
- The notion of the global state can be represented by a cut shown in fig. 5-21.
- A consistent cut fig 5-21(a), is shown with a dashed line, crossing the time axis of three processes, P1, P2 and P3
- The cut represents the last event recorded for each process.
- All recorded message receipts have a corresponding recorded send event.
- Fig. 5-21(b) shows an inconsistent cut, where receipt of message m2 by process P3 is recorded but the snapshot contains no corresponding send event.

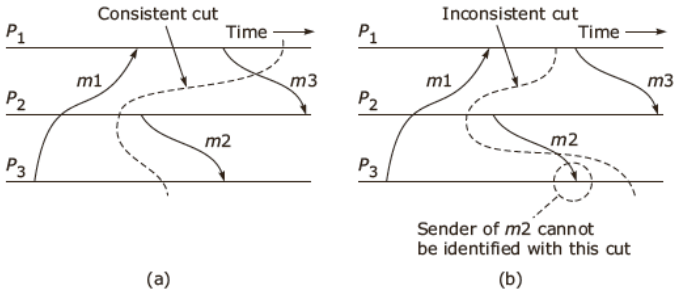


Figure 5-21 Global state (a) consistent cut (b) inconsistent cut

- To understand the algorithm for taking a distributed snapshot, we assume that the distributed system can be represented as a collection of processes connected to each other through point-to-point communication channels.
- Working of algorithm:
  - ***P*** starts by recording its own local state.
  - ***P*** subsequently sends a marker along each of its outgoing channels.
  - When Q receives a marker through channel C, its action depends on whether it has already recorded its local state:
    - Not yet recorded: It records its local state and sends the marker along each of its outgoing channels.
    - Already-recorded: The marker on C indicates that the channel's state should be recorded: all the messages received before this marker and the time Q recorded its own state.
  - Q is finished when it has received before this marker along of its incoming channels.

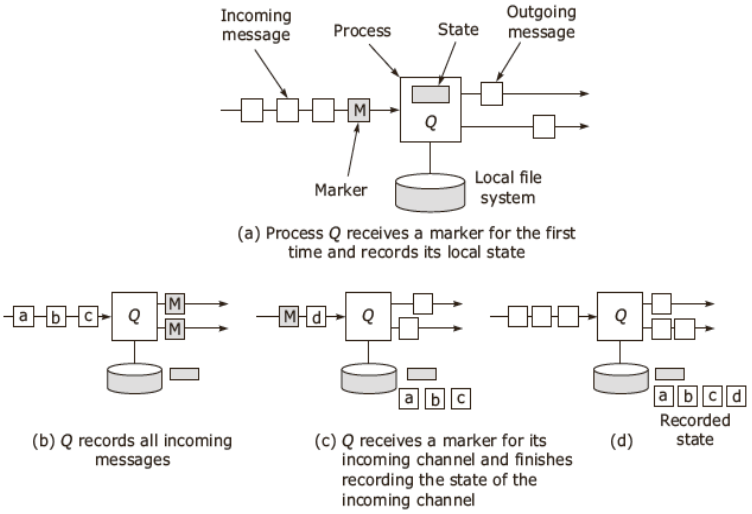


Figure 5-22 Organization of a process and channels for a distributed snapshot

- A process is said to have completed its part of algorithm when it has received a marker along each of its incoming channels and processed each one.
- At this point, its recorded state and the state which was recorded for each incoming channel are collected and send to the process that initiated the snapshot.
- There are various applications of using the global state.
- A few of them are replica management, monitor distributed computations, simplifying distributed algorithms etc.
- On arrival of a message at a process, delay the delivery of the message until the message immediately preceding it is delivered.

Exclusion algorithms

- Mutual Exclusion (ME) can be achieved with shared memory, such as semaphores, lock variable and monitors.
- In distributed systems, ME is more complex due to following reasons:
  - No shared memory
  - Delays in propagation of information which are unpredictable
  - Synchronization issues with clocks

- Need for ordering of events.

- Algorithms:
  - **Centralized Algorithm**
  - **Distributed Algorithm**
  - **Token Ring Algorithm**

1. Centralized Algorithm:

- One of the simplest approaches to achieve ME is to use the centralized algorithm.
- As shown in fig 5-23, the master holds a list of processes requesting the CS and grants requests in some order
- The single master-central process is essentially a monitor; it is also called a *semaphore server.*
- This algorithm of mutual exclusion used three messages per use of CS: Request-R, Grant-G, and Release-R.
- The centralized algorithm shown in fig. 5-24 works as follows.
- At the start, one of the processes is elected to be the coordinator.
- For eg, the process that as the highest network address.
  a) Process 1 asks the coordinator for permission to enter a critical region. Permission is granted.
  b) Process 2 then asks permission to enter the same critical region. The coordinator does not reply.
  c) When process 1 exits the critical region, it tells the coordinator, which then replies to Process 2.

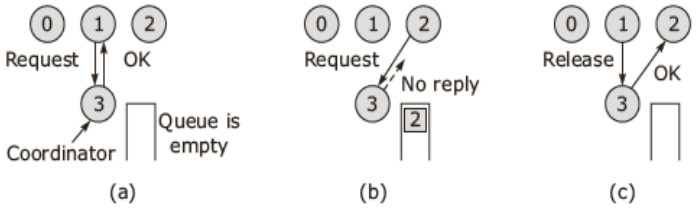


Figure 5-24 Centralized algorithm

- When a process wants to enter a critical region, it sends a Request message to the coordinator stating which critical region it wants to enter.
- If no other process is currently in that CS, the coordinator sends a reply.
- If some other process is already inside the CS, the coordinator will queue the request and sends a 'permission denied' message back or no reply at all.
- When a process exits the CS, it sends a message to the coordinator, releasing its exclusive access.
- The coordinator will then take the first item off the queue of deferred request and send that process a 'grant' message.
- If the process is still blocked, it enters the CS.
- If a 'denied' message has been sent, the process will have to poll for incoming traffic. Either way, on receiving the grant, it enters the CS.
- Advantage:
  - It is simple to implement.
  - There is fairness and no starvation and it requires only three types of message: request, release and grant.
- Disadvantage:
  - It leads to a single point of failure
  - The single coordinator becomes the performance bottleneck.

2. Distributed algorithm:

- A single point of failure is unacceptable in a distributed system, hence various approaches are proposed.
- One of them is Ricart and Agarwala's algorithm.
- The distributed algorithm for mutual exclusion works as follows:
  - When a process wants to enter a CS, it builds a message with the name of the CS, its PID, and the current time, and sends the message to all other processes including itself.
  - The sending of message is assumed to be reliable, which implies that every message is acknowledge.
  - A reliable group communication can be used, if available, instead of sending individual message.
    a) Two processes want to enter the same critical region at the same moment.
    b) Process 0 has the lowest timestamp, so it wins.
    c) When Process 0 is done, it sends an OK, so Process 2 can now enter the critical region.
- As shown in fig 5-25, when a process receives a request message from another process, one of the following 3 cases may occur:
  1. If the receiver is not in the CS and does not want to enter it, it sends back an OK message to the sender.
  2. If the receiver is already in the CS, it does not reply but queues the request.

3. If the receiver wants to enter the CS but has not yet done so, it compares the timestamp it receives with its own message, and the lower one wins. That is, if the incoming message has a lower timestamp, the receiver sends an OK to the sender. Otherwise, the receiver queues the request and sends nothing.

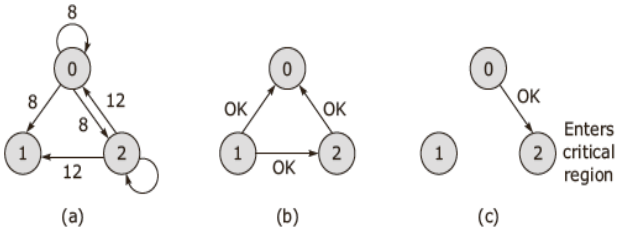


Figure 5-25 Distributed algorithm

- After sending the request, the process waits until everyone has answered.
- As soon as the permissions are in, it enters the CS.
- When it exits the CS, it sends OK message to all processes in its queue and deletes them all from the queue.
- Eg: Process 0 sends everyone a request with the timestamp 8, while Process 2 sends a request with the timestamp 12.
- Process 1 is not interested in entering the CS, so it sends an OK message to both Process 0 and 2.
- Process 0 and 2 both see the conflict and compare the timestamp, Process 2 sees that it has lost, so it grants permission to Process 0 by sending it an OK.
- Process queues the request from Process 2 and enters the CS.
- When it finishes, it removes the request from the Process 2 from its queue and sends an OK to Process 2, allowing Process 2 to enter the CS.
- The algorithm works because in case of a conflict, the lowest timestamp wins and everyone agrees on the ordering of the timestamp.

3. Token Ring algorithm:

- The token ring algorithm presents a different approach to implement CS and ME.
- It is suitable for bus-based system with no inherent ordering of processes(fig 5-26).

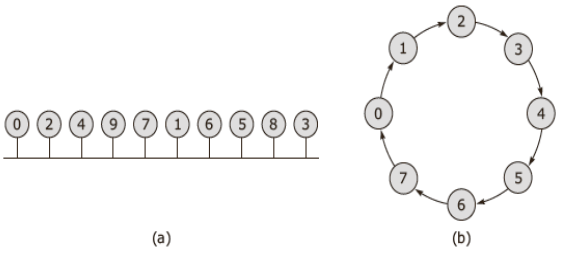


Figure 5-26 Token ring algorithm (a) An unordered group of processes on a network (b) A logical ring constructed in software

- A logical ring is constructed over a bus network in which each process is assigned a position in the ring.
- The ring position can be allocated, say, in a numerical order of network address.
- So a process knows who is next in line.
- When a ring is initialized Process 0 is given a token.
- The token circulates around the ring, passing from a process to process in point-to-point messages.
- Only when a process has a token, it can enter a CS.
- When it exits a CS, it passes the token to the next-in-line process.
- A process cannot enter a second CS by the same token.
- If a token arrives and the process is not going to enter a CS, it passes the token to the next process.
- Hence, if no process is entering a CS, the token keeps on circulating in the ring.
- Only one process has the token at any instant, so only one process can actually be in a critical section.
- Since the token circulates in a well-defined order, starvation does not occur.
- Once a process decides to enter a CS, at worst, it will have to wait for every other process to enter and leave the CS.
- Problem:
    - If the token is lost, it must be regenerates.
    - Detecting the loss of a token is difficult, because the amount of time between successive appearances of the token is unbounded.
    - Also , if a process crashes, the ring configuration must be maintained.

Election algorithm:

- In the distributed system, many algorithms are required to carry out specific task and keep the system in control.
- Irrespective of which process does this function, one of them in the system needs to do it.
- The election algorithm elects a coordinator from among the currently running processes.
- This algorithms have two major goals:
  - They attempt to locate the process with the highest process number and designate it as the coordinator and inform all the active processes about this coordinator.
  - The second goal of an election algorithm is to allow a recovered leader to re-establish control.
- A few assumptions are carried out while carrying out the election process in DS:
  - Provide each process with a unique system/global number.
  - Elect a process using a total ordering on the required set.
  - All processes know the process number of members.
  - All processes agree on the new coordinator.
  - All processes hold on election to determine if the new coordinator is up or crashed.
- Two election algorithms are:

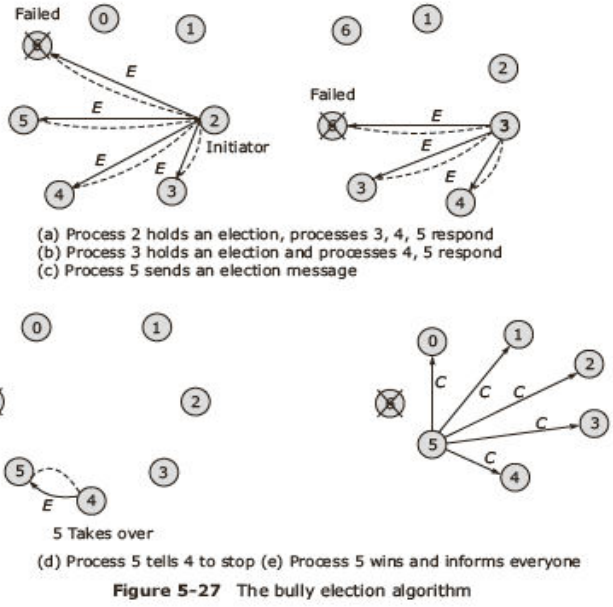
1. **BULLY ALGORITHM:**
   - This algorithm is initiated whenever any process finds that the coordinator cannot be located.
   - This algorithm involves three types of messages:
     - Election message(E): announce election
     - Reply Message(R): acknowledge election message.
     - Coordinator message(C): announce new coordinator.

The bully algorithm works as follows:

- A process begins an election when it notices that the coordinator has failed.
- Hence it sends election message to all processes with the higher priority and waits for a reply message.
- If none arrives in a certain time, it declares itself as the coordinator and sends a coordinator message to all processes with the lower priority.
- If a reply message does arrive, it waits a certain amount of time for a coordinator message to arrive from the new coordinator and if none arrives, it begins another election.
- If the process receives an election message, it sends back an answer message and begin another election, unless it has already begin one.

Example:

- Suppose process 2 notice that the coordinator is no longer responding to responds it initiates an election. It holds an election as follows:
- Process 2 sends E message to all processes with higher number (3,4,5,6). Process 6 is down, hence it does not respond. Process 3,4,5, exists and hence they respond with an R message.
- Process 2's job is done. Of no one would have responded, then process 2 would have won the election. This election is now initiated by processes 3,4,5 and above steps are repeated.
- Eventually, all process gives up except 5, and it becomes the coordinator. It announces its victory by sending all processes C message. When process 6 that was previously down comes back up, it holds an election.
- Since it happens to be the highest-numbered process running, it will win the election and take over the coordinators job.



(a) Process 2 holds an election, processes 3, 4, 5 respond
(b) Process 3 holds an election and processes 4, 5 respond
(c) Process 5 sends an election message
(d) Process 5 tells 4 to stop (e) Process 5 wins and informs everyone

Figure 5-27 The bully election algorithm

2. RING ALGORITHM

- It is a non-token algorithm, in which the processes are arranged in a logical ring, ordered, and each process knows its successor.
- This algorithm has two types of message:
  - Election (E)—announce an election, is token message
  - Coordinator (C) — announce new coordinator
- Any process Pi, noticing that the coordinator is not responding, sends an election message to its successor.
- If its successor is down, the message is sent to next member.
- The receiving process adds its number to the message and passes it along.
- When the election message gets back to the election initiator, it recognizes its own process number and changes the message to C.
- Now it circulates this message to all members to inform everyone as to who the coordinator is and who the members of the ring are.
- The coordinator is the highest process in the total order, all processes know the order and thus all will agree no matter how the election started.
- The message is circulated one, then it is removed and everyone goes back to work

Eg:

- Consider a DS having 7 active processes where the previous coordinator process 7 has crashed and process 2 initiates the election.
- It builds the lection message and circulates to its successor and this process continues.
- Process 5 sends the message to process 6, which adds its number and sends the message to process 7.
- Process 7 has crashed, it does not reply within a timeout period and the message is send from process 5 to process 0.
- Finally after the message circulates across the ring, it is received at process 2

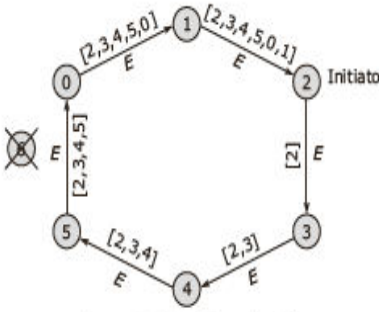


Figure 5-28 Ring algorithm

Table 5-2 Comparison of Bully and Ring election algorithms

Criteria	Bully election algorithm	Ring election algorithm
Types of messages used	Election, Reply, Coordinator	Election, Coordinator
Method of election		
System Structure	No structure	Logical ring
Who can initiate	Anyone who notices that coordinator is not responding sends election message	Anyone who notices that coordinator is not responding sends election message
How it works	Higher number process takes over and initiates election	Each process adds its number and circulates the election message till it reaches the initiator
Who is elected	Highest-numbered process	Highest-numbered process
Strength	Highest-numbered coordinator bullies all into submission	Only one coordinator is chosen
Weakness	Scalability and takes time because multiple elections have to be held.	Scalability, more than one election results in message storm, all containing same information
Best case number of messages	$n - 2$	$n - 1$
Worst case number of messages	$O(n^2)$	$3n - 1$

Unit 5

Distributed System Management
Resource management, process management, threads, and fault tolerance

Resource management

- A distributed system consists of multiple resources interconnected by a communication network.
- **The techniques for scheduling a process are:**
  - Task assignment approach
  - Load balancing approach
  - Load sharing approach
- **Task assignment approach:** Each process submitted by the user is divided into a set of tasks which are scheduled on suitable nodes to improve performance. A typical eg. Is that of incoming request which is allocated to an idle processor in the processor pool model.
- **Load balancing approach:** The processes submitted by the user are distributed among the nodes to equalize the workload among processors.
- **Load sharing approach:** It relies on heavy loaded processors to transfer their load to idle or lightly loaded processor, thus improving the response time for users.

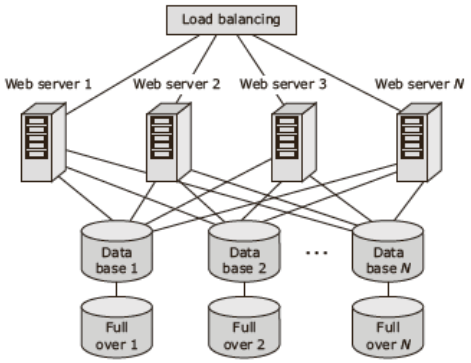


Figure 6-1 Web server farm

Desirable Features of a global Scheduling Algorithm:

- **No apriori knowledge about processes to be executed:**A good global process scheduling algorithm should operate with no a priori knowledge of the process to be executed, since it places extra burden on the user to specify this information before execution.
- **Ability to make dynamic scheduling decisions:** The algorithm should be able to make scheduling decision dynamically based on the current system status.
- **Flexible:** The algorithm should be flexible enough to migrate the process multiple times in case there is a change in system load.
- **Stable:** The algorithm must be stable such that processor do useful work, reduce thrashing overhead and minimize the time spent in unnecessary migration of the process.
- **Scalable:** The algorithm should be scalable and able to handle workload inquiry from any number of machines in the network.
- **Unaffected by system failures:** The algorithm should not be disabled by the system failures, such as node or link crash, and it should have decentralized decision-making capability.

Task Assignment Approach:

- In this method, a process is divided into multiple tasks and the objective is to find an optimal policy for the tasks of an individual process.
- A set of typical assumptions are made in these approach.
- The task assignment algorithm along with these assumptions achieve the goals of minimizations of IPC costs, less turnaround time for process completion, high degree of parallelism and efficient usage of all system resources.

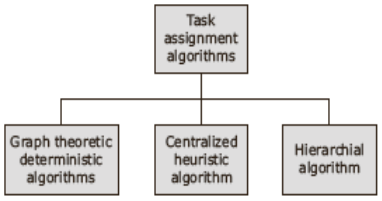


Figure 6-2 Task assignment algorithms

1. **Graph theoretic deterministic algorithm:**
   - To find an optimal assignment of tasks to nodes is to minimize the total execution and communication costs and create a static assignment graph, this algorithm is used.
   - A system with m CPUs and n processes has any of the following three cases:
     - m=n: Each process is allocated to one CPU
     - m<n: Some CPUs may remain idle or work on earlier allocated processes
     - m>n: There is a need to schedule processes on CPUs, and several processes may be assigned to each CPU.
   - The main objective of performing CPU assignment is to minimize IPC cost, guarantee quick turnaround time to complete process execution, and achieve a high degree of parallelism for efficient utilization of system resources and to minimize network traffic.

The graph theoretic deterministic algorithm works as follows:

- First, the system is represented as a weighted graph where each node is a process and each arc represents the flow of message between two processes.
- Eg. 8 processes to be mapped to two processors.
- Fig 6-3 depicts a problem.
- The processes are represented as nodes A,B,C,D,E,F,G and H.
- The arcs between the sub-graphs represent network traffic and their weights represent IPC costs.
- The graph is divided into partitions, subject to constraints such as CPU and memory requirements.
- The partitions are made such that arcs within the sub-graphs denote IPC on the same CPU.

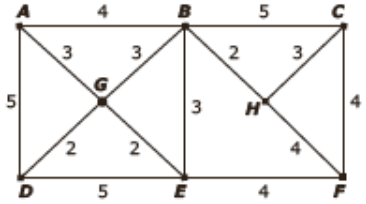


Figure 6-3(a) Task assignment problem

- Since there are no costs incurred while executing processes on the same processor, they can be ignored.
- Next, the objective is to allocate processes and CPUs and identify partitions which minimize network traffic, meeting all constraints.
- Once all the weighted graphs are drawn, they are partitioned in different ways with each one having a different network load.
- Optimal task assignment is shown in fig 6-3 (a).
- How 8 processes are allocated to 2 processors is shown in fig. 6-3(b) and 6-3(c).
- The graph can be partitioned in two ways:

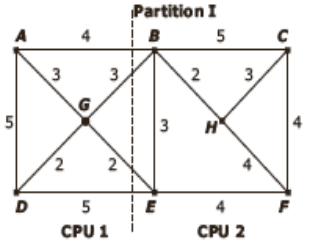


Figure 6-3(b) Optimal task assignment with partition

- **Partition I:**
  - CPU 1 runs processes A, D and G.
  - CPU 2 runs processes B, E, F, H and C.
  - IPC costs=4+3+2+5=14 units.
- **Partition II:**
  - CPU 1 runs processes A, D, G and B.
  - CPU 2 runs processes H, C and F.
  - IPC costs=5+2+4=11units.

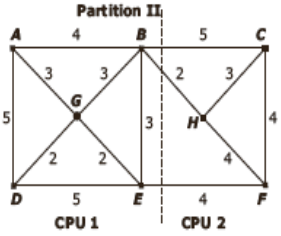


Figure 6-3(c) Optimal task assignment with partition II

- The network traffic differs based on the partitioning scheme used.
- Partition II communication incurs less network traffic as compared to partition I.

2. Centralized heuristic algorithm:

- The centralized heuristic algorithm does not require advance information and is also called *top-down algorithm*.
- A coordinator maintains the usage table with one entry for every user and this is initially zero.
- When an event occurs, messages are sent to the coordinator and the table is updated.
- The algorithm gives each user a fair share of computing power and does not try to maximize CPU utilization.
- The usage table entries can either be zero, positive or negative.
- A zero value indicate a neutral state, a positive value implies that the machine is the user of system resources and the negative value means that the machine needs resources.

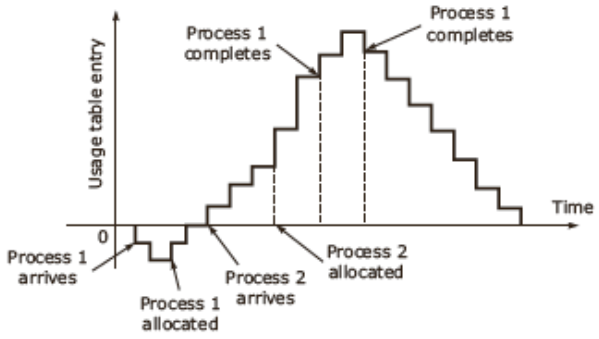


Figure 6-4 Centralized heuristic task assignment algorithm

- Fig, 6-4, shows how algorithm works:
    - In the first step, a process is created on the machine.
    - If the machine becomes overloaded, the machine decides to run the process elsewhere.
    - Next, the machine asks the usage table coordinator to allocate a processor to it.
    - The request is granted if the processor is available and no one else wants it; else the request is denied temporarily, and a note is made if no processors are free.
    - The heuristic used for processor allocation is that when the CPU becomes free, pending requests whose owners have the lowest score win.
    - As a result, a user who has a request pending for a long time will always be allocated a processor first.
    - This is the objective of the algorithm-to allocate processing capacity fairly.
    - It always favours a light user than a heavy one, thus ensuring that the load is balanced across processors.

3. **Hierarchical algorithm:**
    - A major problem with the centralized algorithm is that it does not scale to large systems because the central node could become a bottleneck.
    - One of the solutions is to design a hierarchical algorithm where the processes are organized into a logical hierarchy independent of the physical network structure such as managers and workers.
    - As shown in fig. 6-5, process hierarchy is modeled like an organization hierarchy.
    - As the number of processor maintains communication with one superior and a few subordinates.
    - The information stream is manageable, but the system could fail if any middle-level machine fails.
    - In this case, the next subordinate takes over, but how to choose this subordinate could be decided by the subordinate themselves or by the superior.
    - To make the decision-making easier and manageable, the top of the tree is truncated as shown in the fig. and replaced with a committee.
    - In case of failure of any one manager, the other managers in the committee continue the operation.

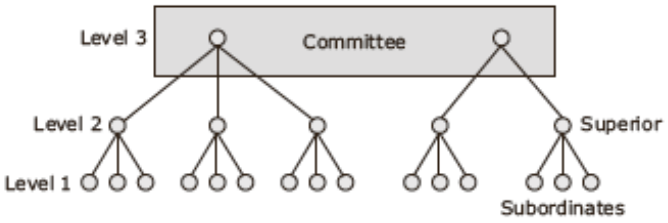


Figure 6-5 Hierarchical task assignment algorithm

- The allocation algorithm works between two levels in a group.
- The situation is complex because at any instant of time, multiple requests are in various stages of allocation leading to outdated available worker estimates, race conditions, deadlocks etc.

<u>Load Balancing Approach:</u>

- The scheduling algorithms which uses the load-balancing approach are based on the fact that an even load distribution helps in better resource utilization.

Load balancing Taxonomy:

- The taxonomy of load-balancing algorithm is show in fig. 6-6

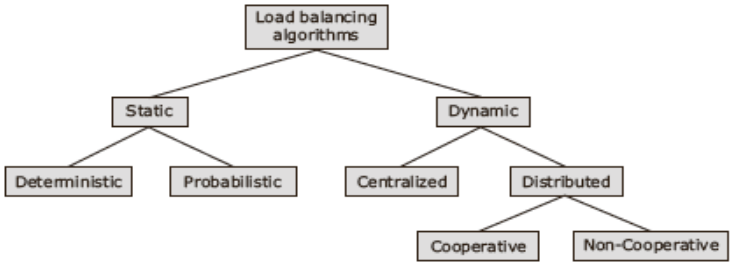


Figure 6-6 Taxonomy of load balancing algorithms

- **Static vs Dynamic:**
    - **Static algorithms** are fixed prior to process execution and do not respond to the current state of the system.

- **Dynamic algorithms** make scheduling decisions based on the current system and they perform better, but are more complex because they rely on information from all the machines.

- **Static: deterministic vs heuristic:**
  - **Deterministic algorithms** are suitable when the process behaviour is known in advance. If all the details, such as the list of processes, computing requirements, file requirements and communication requirements are known prior execution, then it is possible to make a perfect assignment.
  - In case the load is unpredictable or variable from minute-to-minute or hour-to-hour, **a heuristic processor** allocation is preferred because it is an ad hoc technique of allocation.
- **Dynamic: centralized vs distributed:**
  - **A centralized** dynamic scheduling algorithm means that the scheduling decision is carried out at one single node called the centralized server node. This approach is efficient, since all the information is available at a single node, but the drawback is that it leads to a bottleneck and performance degradation and also system is not reliable.
  - **In dynamic distributed scheduling algorithm,** the task of processor assignment is physically distributed among the nodes. Dynamic distributed scheduling algorithms are more effective than centralized algorithm, since the scheduling decisions are made on individual nodes.
  - Dynamic is further divided into:
    - **Cooperative algorithm:**here distributed entities cooperate with each other to make scheduling decision. These algorithms are complex and involve large overhead.
    - **Non-cooperative algorithm:** In these, the individual entities make independent scheduling decision and hence they involve minor overhead. These algorithms may not be stable.

Issues in designing in load balancing algorithms:

1. **Policies for Load estimation:**
   - The major objective of a load-balancing algorithm is to balance the workload on all the nodes of the system.
   - All load-balancing algorithms assume that each machine knows its own load and also whether it is under or overloaded.
   - The machine can also communicate its current load or state to other machines.

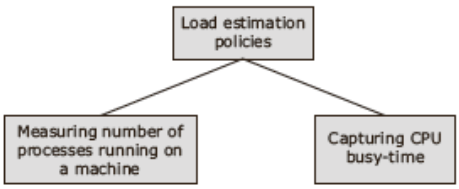


Figure 6-7 Classification of load estimation policies

   - Hence, it is important to measure the workload of a particular node by some measurable parameters that can be time-dependent or node-dependent.
   - Load estimation policies are classified based on measuring number of processes running on a machine or capturing CPU busy time
     - **Measuring number of processes running on a machine:**One way to estimate the load is to calculate the number of processes running on the machine. But as you are aware, machines can have many processes running both in foreground and background. Load can be taken as equal to number of processes running or ready to run. It is quite obvious that every running or runnable process puts some load on the CPU, even when it is a background process.
     - **Capturing CPU busy time:**The other technique of load estimation is to capture the CPU busy-time. CPU utilization can be measured by allowing a timer to interrupt the machine to periodically observe the CPU state and find the fraction of idle time.
2. **Policies for Process transfer:**
   - The load-balancing strategy involves transferring some processes from heavily loaded nodes to lightly loaded nodes.
   - Hence there is a need to decide a policy which indicates whether a node is heavily of lightly loaded called the *threshold policy.*

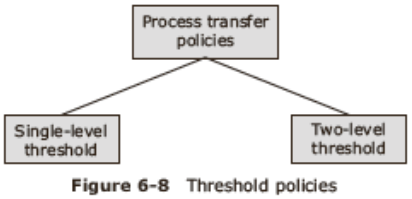


Figure 6-8 Threshold policies

   - This threshold is a limiting value, which decides whether a new process, ready for execution, should be executed locally or transferred to a lightly loaded node.
   - The threshold policy is divided into:
     - **Single-level threshold policy:**The threshold policy evolved as a single threshold value. So a node accepts a new process as long as its load us below the threshold, else it rejects the process as well as the request for the remote execution. The use of single threshold value may lead to useless process transfer, leading to instability in scheduling decisions.

- **Two-level threshold policy:**A two-level threshold policy is preferred to avoid instability. It has two threshold levels: high and low marks. The load state of a node can be divided into three regions: overloaded, normal and under-loaded.

3. **Location policies:**
   - **Location Policies are classified into:**
     - **Threshold policy:** In the threshold policy, the destination node is selected at random and a check is made to verify whether the remote process transfer would load that node. If not, the process transfer is carried out; else another node is selected at random and probed.
     - **Shortest location policy:**In this policy, nodes are chosen at random and each of these nodes is polled to check for load. The node with the lowest load value is selected as the destination node. Once selected, the destination node has to execute the process irrespective of its state at the time the process arrives.
     - **Bidding location policy:**The bidding location policy transforms the system into market scenario with buyers and sellers of services. Each node is assigned two roles, namely the manager and the contractor. The manager broadcasts a request for a bid message to all nodes and the contractors send their bids to the manager node. A bid contains a quoted price based on the processing power, memory size and resource availability of the node.
     - **Pairing policy:**This policy focuses on load balancing between a pairs of nodes. Load balancing operation is carried out between the nodes belonging to the same pair by migrating processes from the heavily loaded node to the lightly loaded node.

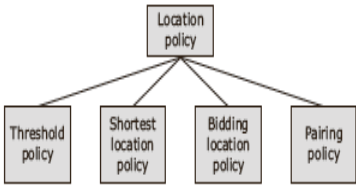


Figure 6-9 Classification of location policies

4. **State information exchange:**
   - **The state information exchange is divided into:**
     - **Periodic Broadcast:** In this policy, each node broadcasts its state information at a specific time interval *t.* This method generates heavy traffic and unnecessary message transmitted. Scalability is also an issue.
     - **Broadcast when state Changes:**In this method, a node broadcasts its state information only when a node's state changes. An extension of this method is to broadcast only when the state changes result in threshold values.
     - **On-demand exchange of state information:**The method of on-demand exchange of state information is based on the fact that a node needs to know the state of other nodes, only when it is either under-loaded or over-loaded.
     - **Exchange by polling:**This method is based on the fact that there is no need to exchange state information with all nodes in the system. Only when a node needs cooperation from another for load balancing, it should search for a suitable partner by polling all the nodes one-by-one and exchanging information.

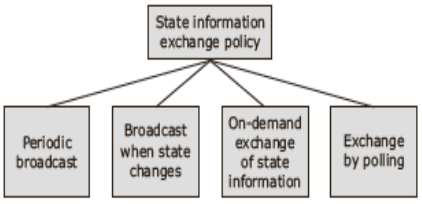


Figure 6-10 State information exchange policies

5. **Priority assignment:**
   - The priority assignment policy is to schedule local and remote proce
     - **Selfish priority assignment policy:** In this policy local processes are given higher priority as compared to remote processes. Typically, the priority of a process depends on the number of local processes and the number of remote processes at a specific node.
     - **Altruistic priority assignment policy:**This policy has the best response-time performance. In this policy, remote processes incur lower delay that local processes, even though local processes are the principal workload at each node, whereas remote processes are secondary workload.
     - **Intermediate priority assignment policy:**The performance of intermediate priority assignment policy falls between selfish and altruistic priority assignment policies.

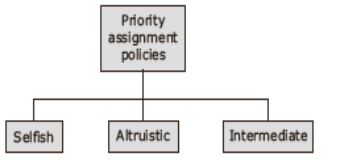


Figure 6-11 Classification of priority assignment policies

[By Nishigandha Dhore]

6. **Migration limiting policies:**
    - **Two types:**
        - **Uncontrolled**: In this policy, a remote process arriving at a node is treated as a local process. Hence in this policy, the process can be migrated any number of times, but may result in system instability.
        - **Controlled:** In this policy, the local and remote processes are treated differently

Load Sharing Approach:

- For load balancing, it is necessary and sufficient to prevent the nodes from being idle or heavily loaded.
- This is termed as *Dynamic Load Sharing.*

Issues in designing load sharing algorithms:

- **Load estimation policies:**
    - Load sharing algorithms are based on the fact that it is sufficient to know whether a node is idle or heavily loaded.
    - The load estimation policy used is a count of total number of processes on a node.
    - But this estimation policy may not be suitable for current distributed systems since several processes may be permanently be in execution even on an idle node.
    - Therefore CPU utilization may be used for load estimation.
- **Process transfer policies:**
    - All load-sharing algorithms employ an all or nothing strategy with a single threshold value being fixed at 1.
    - A node can accept a remote process only when it has no process and a node can transfer a process as soon as it has more than one process.
    - This strategy may not be good because an idle node will have to wait before it can acquire a new process, resulting in wastage of CPU power.
    - A probable solution would be to carry out anticipatory transfer to nodes which are not idle but may soon become idle.
- **Location policies:**
    - In the load-sharing algorithm, the location policy decides the sender and receiver nodes which are involved for load sharing.
    - Depending on who takes the initiative to perform a global search for a suitable node, the location policies can be classified as sender-initiated and receiver-initiated policies.(fig. 6-12)
        - **Sender-initiated location policy:**This policy uses the sender of the process to decide where to send the process. The heavy loaded nodes search for lightly loaded nodes where the process can be transferred.
        - **Receiver-initiated location policy:**In the receiver-initiated policy, lightly loaded nodes search for heavily loaded nodes from which processes can be accepted for execution. When the load on a node falls below a threshold value, it broadcast a probe message to all nodes or probes node one by one to search for heavily loaded node.
- **State information exchange policies:**
    - Load-sharing algorithms do not focus on equalizing load on all nodes, so it is not necessary for them to periodically exchange state information.
    - The common policies used for state information exchange are *broadcast* and *poll* when the state changes.
        - **Broadcast policy**: In the broadcast method, a node broadcasts a *Stateinformationrequest*message when it becomes either underloaded or overloaded. In the sender-initiated policy, the node broadcasts this message when it is underloaded. The receiver-initiated policy broadcasts when idle, which is also called broadcast-when-idle policy.
        - Poll policy: In this method, when a node's state changes, it randomly polls the other nodes and exchanges state information with the polled nodes.

Process Management

- In DS environment, the main goal of process management is to make the best possible use of existing resources by providing mechanism and policies for sharing them among all processes.
- Process management goals are achieved by providing processor allocation, process migration and thread facility.

- Process migration involves moving a process, either before or during execution, to a new node for better utilization of the process.
- Threads provide a mechanism for a fine-grained parallelism for better utilization of the processor capability.
- Distributed process management involves two major functions: 1) processor allocation and 2) processor migration.
- Processor allocation decides which process should be assigned to which processor.
- Process migration involves moving a process, either before or during execution, to a new node for better utilization of the process.

Functions of distributed process management:

- One of the key function involved in distributed process management is process migration is that, it is defined as the change of location and execution of a process from the current processor to the destination processor.
- Depending on when a process is migrated, it is classified into non-pre-emptive and pre-emptive process migration.

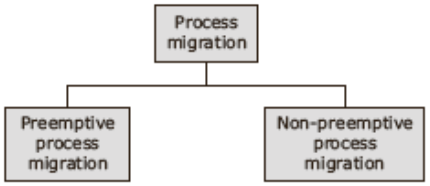


Figure 6-14 Types of process migration

- Pre-emptive method is more complex and expensive than the non-pre-emptive method, because the processed environment has to be sent along with the process code and data.

Desirable features of Process Migration:

- **Transparency:** It should be provided at the object access level. The object access level is a minimum requirement for the system to support non-pre-emptive process migration facility. Transparency is also provided at the system call and IPC level.
- **Minimal interference:** Migration should cause minimum interference to the migratory process and to the system as a whole. The solution is to decrease the freezing time(the time for which the process execution is stopped) when it is transferred to the destination node.
- **Minimal residual dependencies:** Also, a good process migration mechanism must leave minimal residual dependencies on the source node from where it has migrated. Once the process starts execution on the new node, it should not depend on the source node where the process was started
- **Efficiency:** The main issue that hampers the efficiency during process migration are process migration time, the cost of object location and the migration process, and the cost of supporting execution after the process has migrated to the destination node.
- **Robustness:** A desirable process migration mechanism must be robust, that is, it should be able to handle failures.
- **Ability to communicate between co processes of the job:** The system should support parallel processing among the processes of a single job which is distributed over many nodes. Coprocesses should communicate with each other directly so that the communication cost is reduced

<u>PROCESS MIGRATION:</u>

- The main step in process migration are:
  - Freezing process on the source node
  - Starting process on the destination node
  - Transporting process address space on destination node
  - Forward the messages addressed to migrated processes

Mechanism for Process Migration:

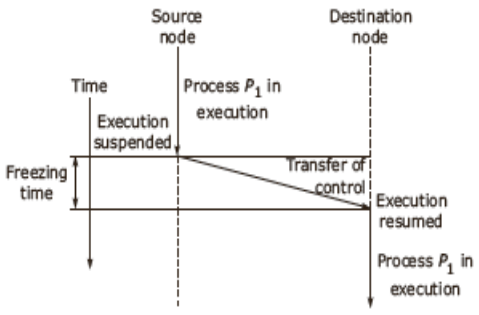


Figure 6-15 Process migration mechanism

1. **Freezing the process on the source node:**
   - The first and foremost step in process migration is to freeze the process on the source node by blocking it.
   - The various steps involve in blocking are:

     <u>Blocking Sequence:</u>
     - **Blocking the process immediately:** The process is i system call, or if it is executing a system call but is sleeping on an interruptible priority waiting for a kernel event to occur.
     - **Wait for I/O operations to complete and then block the process:** The process cannot be blocked immediately if some I/O operations are not completed.

Tracking information about open files:

- The information about opened files includes information, such as file ID and the current positions of a file pointer.

2. **Creating an empty process on the destination node:**
   - The next step is to create an empty process on the destination node which has a different process ID that is different from the source process ID.
   - In the initial phase, both copies exist; the snapshot is transferred to the destination node and the process ID is changed to the source process ID.

3. **Transferring the migrant process and address space onto the destination node:**
   - After the entire migrant process is transferred, the new copy is unfrozen and the old copy is deleted.
   - Then the process is restarted on the destination node.
   - This sequence of operations look simple, but there could be situation like a process that may be executing a system call when the snapshot is taken.
   - In such case, it may be required to perform the system call again on the destination node.

4. **Restarting the process on the destination node:**
   - The next step is to transfer the process address space to the destination node.
   - The process can now start executing on the destination node.
   - The method by which the messages arriving at the source node after migration are transferred to the destination node where the process is currently executing.

5. **Forwarding the message addressed to the migrant process to the destination node where the process started execution:**
   - The next step in process migration is forwarding messages, addressed to the migrant process to the destination node.

Address space transport mechanism:

- After freezing the process on the source node, the next step is to transfer the address space to the destination node.

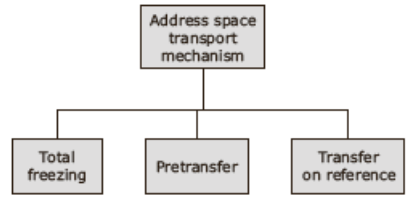


Figure 6-16 Address space transport mechanisms

- **Total freezing:**
  - Process execution stopped during address space transfer.
  - But freezing is not suitable for migration of interactive process.
  - The time-out occurs during migration, and the delay will be noticed by the user.

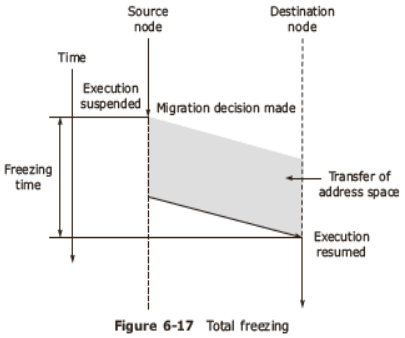


Figure 6-17 Total freezing

- **Pre transfer:**
  - Address space is transferred while process continues to run on source node.
  - Pre-transfer mechanism transfers the complete address space followed by subsequent transfer of pages modified later.
  - As compared to total freezing, pre-transfer allows the process execution to continue while a part of address space is being transferred.
  - Highest priority in scheduling.
- **Transfer-on –reference:**
  - Process state is transferred while address space is transferred on demand.

- The transfer-on –reference mechanism is based on the principle that most of the time, processes tends to use only a small part of the address space during execution.
- The process status is transferred, while the address space is left on the source node.
- When a reference is made to a memory page, the desired blocks are transferred from the source node.

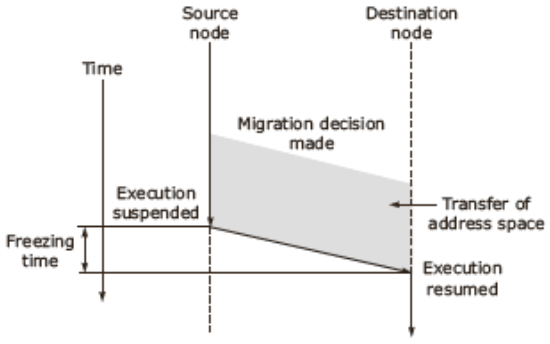


Figure 6-18 Pretransfer

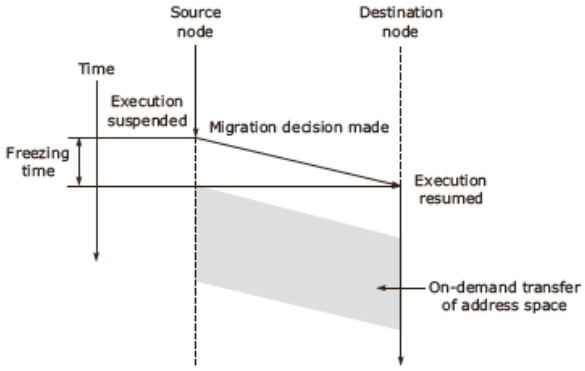


Figure 6-19 Transfer-on-reference

Message forwarding:

- **The commonly used message forwarding mechanism are:**
  - **Returning message to the sender as undelivered:**
    - Once the process has stopped executing on the source node all messages received at the source node are returned to the sender as undeliverable or dropped assuming that the sender node has buffered all the copies and will retransmit when needed.
  - **Origin site mechanism:**
    - This mechanism is used in Sprite system.
    - A message is sent first to the origin site, which forwards it to the current location where the process is migrated.
    - This process is prone to failure of origin site and put continuous load on the origin site even after the process has migrated from that node.
  - **Linking traversal mechanism:**
    - In this mechanism, a message queue is built on the source node for message received on that node after the process execution has stopped there, but has not yet started on the destination node.
    - Once the process has settled at the destination node, the messages from the source node queue are forwarded to the migrant process on the destination node.
  - **Linking update mechanism:**
    - While transporting the migrant process, the source node sends a link update message to the kernels controlling all migrant process communication partners.
    - This message is acknowledged for synchronization purpose.

Advantages of process migration:

- Reduce average response time of heavily loaded nodes
- Speed up of individual jobs
- Better utilization of resources
- Improve reliability of critical processes
- Improving system security

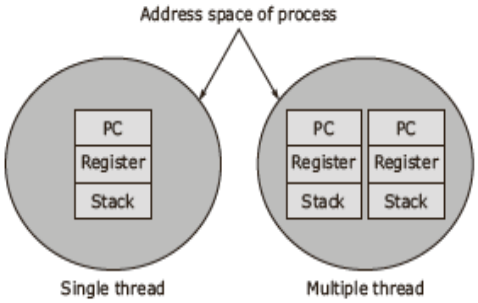


Figure 6-22 Process address space

THREADS

- Threads belong to a process share the same address space.
- They are not entirely independent of each other because they may share global variables.
- There is no protection between threads belonging to the same process.
- Threads share the same set of OS components.
- Threads share CPU in the time sharing mode and create a child threadsand block themselves while waiting for system calls to complete.
- They can also change state during execution between run, block, ready

or terminated states.

- A running thread utilizes the CPU and is active whereas a blocked thread waits for another thread or some session call to complete in order to unblock itself.
- Threads have many advantages over processes and it is advantageous to create threads within the same process than to create new processes.
- Threads allow parallelism, improve system performance and when combined with blocking system call, make programming easier.
- Threads can be concurrently executed, increasing the system throughput of the system.
- To conclude, a set of threads using a shared address space make software design simple, easy to program and efficient.

Table 6-1 Comparison of processes versus threads

Criteria	Process	Thread
Control block	Process Control Block (PCB): program counter, stack, and register states; open files, child processes, semaphores, and timers	Thread Control Block (TCB): program counter, stack, and register states
Address space	Separate for different processes, provides protection among processes	Share process address space, no protection between threads belonging to the same process
Creation overhead	Large	Small
Context switching time	Large	Small
Objective of creation	Resource utilization, to be competitive	Use pipeline concept, to be cooperative

Thread models

- **Dispatcher worker model**
  - The example of the dispatcher model is a server process such as a file server which accepts requests from the client, checks for access permission, and accordingly services the request.
  - The server maintains incoming requests in a queue and carries out the current task to completion.
  - Since a queue is maintained and there is a single server in the system, very few requests are processed, thus increasing the response time for clients

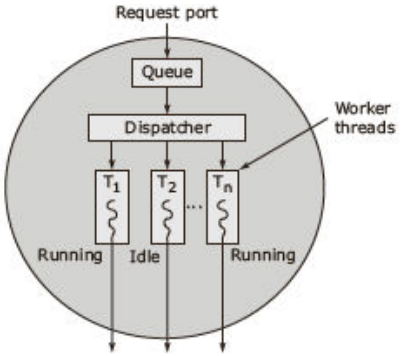


Figure 6-23 Dispatcher-workers model

- **Team model**
  - In team model all threads are treated equal, such that one handles request on its own,
  - In case threads are capable of performing specific functions, a queue can be maintained.
  - When a thread changes state from running to idle, it takes a new request from the job queue and the starts execution.

- **Pipeline model**
  - This model uses pipelining concept used in CPU instruction execution.
  - The tasks assigned to the threads are completed and the results generated by the first threads are passed on to the next thread.
  - It takes this as input and start running.
  - Data passes across multiple threads with each one processing it partly and the last thread giving the final result.

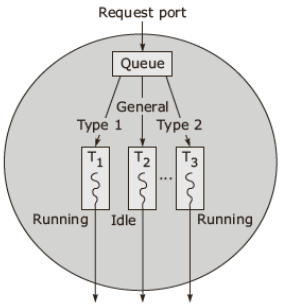


Figure 6-24 Team model

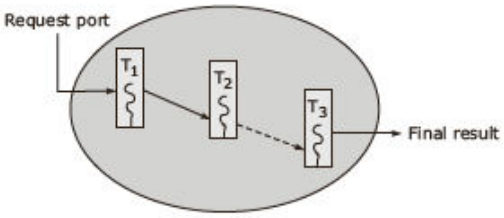


Figure 6-25 Pipeline model

Design issues in threads:

- **Thread semantics**
  - The 1st step to be carried out before using thread is *thread creation.*
  - Threads can be created either *statically or dynamically.*
  - In static approach, the number of threads to be created is fixed when the program is written or when it is compiled.
  - In dynamic approach, threads are created as and when needed during the process lifecycle and they exit when the task is completed.
  - Threads follow the same steps for termination as processes.
- **Thread synchronization:**
  - Since threads belonging to a process share the same address space, thread synchronization mechanism are required to ensure that multiple threads do not access the same data simultaneously.
  - A critical region is implemented using mutex variable, which is a binary semaphore having two states: locked and unlocked.
  - Various operations are defined on mutexes such as lock, unlock and trylock.
  - Fig 6-26 shows an example to explain how mutex and condition variable are used together to provide mutually exclusive access to shared data and thus implement a critical region.
  - Thread package provides another synchronization feature called the condition variable, which is associated with a mutex at the time it is created.

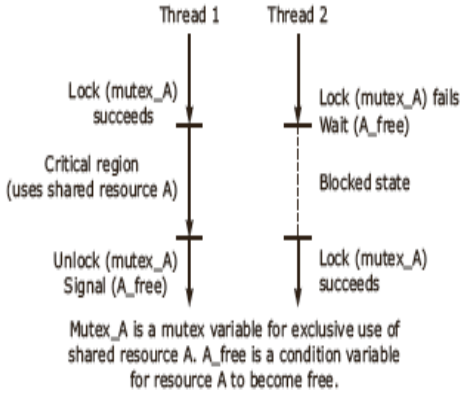


Figure 6-26 Implementation of a critical region

- **Thread scheduling:**
  - **Priority assignment facility:**
    - It uses on of the simplest scheduling algorithms, either FIFO or round-robin policy to use a time-slice form the CPU.
    - In this case, all the threads have equal priority.
    - To give preference to some thread, a scheduling policy can be based on priorities assigned to the threads by the application programmer.
    - This policy ensures that important threads run on a high priority basis.
    - The priority-based scheduling policy can either be pre-emptive of non-pre-emptive.
  - **Choice of dynamic variation of quantum size:**
    - The thread package normally supports dynamic variation of quantum size.
    - In case of fixed-length quantum technique, a fixed time quantum is assigned to each thread for using CPU.
    - It is an ideal algorithm for short requests, even on heavily loaded systems, but may prove inefficient in lightly loaded systems.
  - **Handoff scheduling scheme:**
    - In this scheme, a thread can name its successor so that the queue of runnable processes can be bypassed.
    - This method is useful in enhancing the system performance in case a set of threads need to run in sequence.
  - **Affinity scheduling scheme:**

- This technique directs the thread to run on the CPU where it ran fast, hoping that the thread's address space is still in the CPU's cache.
- This scheduling policy is used for better performance.

Implementing thread package:

1. **User level approach:**
   - In user-level, the user space stores the processes, threads, and a runtime system which is a collection of thread management routines.
   - This runtime system also maintains the status information table to keep track of the current status of each thread.
   - All calls of the thread package are implemented as calls to the runtime system procedures.
   - They also perform thread switching if the thread which made a call needs to be suspended.
   - The user-level implementation uses two-level scheduling.
   - At one level, the kernel scheduler allocates the quantum to processes and at levelstwo, the runtime system divides this quantum of the process among its threads.
2. **Kernel level approach:**
   - In kernel-level approach, the threads are managed and maintained by kernel.
   - The thread status information table is stored in the kernel address space.
   - Calls which may block a thread are system calls that trap the kernel.
   - The major function of the kernel is to select the order in which the threads should run, based o which process they belong to.
   - Since this scheduling of threads is managed entirely by the kernel, it is called single-level scheduling

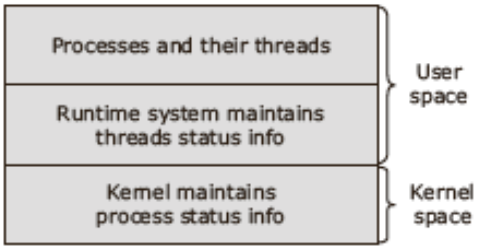


Figure 6-27 User-level thread package implementation

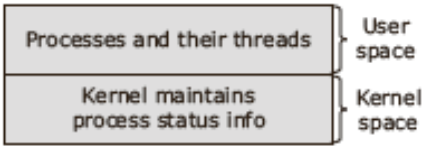


Figure 6-28 Kernel-level thread package implementation

Table 6-2 User-level vs. Kernel-level thread implementation

Criteria	User-level approach	Kernel-level approach
Thread package implementation	Can be implemented even on the OS which does not support threads.	Can be implemented only in the OS which supports threads because it needs to be integrated into the kernel design.
Flexibility to use customized scheduling algorithms	Users can design the algorithms which suit the application because of the use of two-level scheduling.	Users can only specify priorities for selecting a new thread because only one-level scheduling is used.
Context switching	Faster because it is managed by the runtime system.	Slower because the trap has to be made to the kernel.
Scalability	Scalable since the status information table is maintained by the runtime system.	Poorly scalable because this status information table is maintained by the kernel.
Issue with round-robin scheduling policy to share CPU cycles among threads of a process	Round-robin scheduling policy cannot be used because there are no clock interrupts in a process. So the thread given to the CPU continues to run till the CPU gives it up. The runtime system can be modified slightly to receive clock interrupts from the kernel; scheduler can now control thread allocation to the CPU.	Clock interrupts occur periodically. Kernel tracks the CPU time per thread and hence can interrupt the thread execution. So the CPU can be allocated to another thread.
Blocking system call implementation	If a thread makes a blocking system call, all threads of the process will be trapped. The kernel schedules another process to run, the objective of the thread will be lost. The solution is to use a jacket routine; extra code before a blocking system call. It checks if the call causes a trap to the kernel. Call is allowed to be made if it is safe; else the thread is suspended. The entire operation is done atomically.	Easy to implement if a thread makes a blocking system call. The sequence of operations are: • Thread makes a call • Trap to kernel • Thread is suspended • Kernel starts a new thread

Threads and Remote execution:

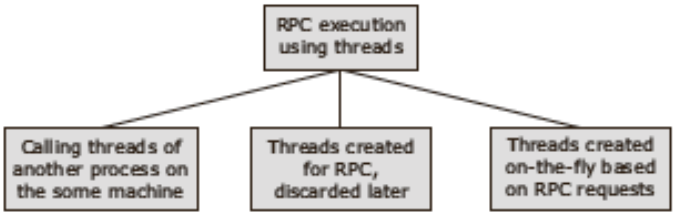


Figure 6-29 Types of RPC execution in distributed system

1. **Calling threads of another process on some machine:**
   - The first method refers to calling threads of another process on the same machine during RPC execution.
   - In DS, more RPCs are executed on the same machine as the caller than on a remote machine.
   - In such cases, it is possible for a process to call the thread in another process on the same machine following this sequence of steps.(fig, 6-30)

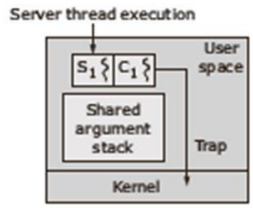


Figure 6-30 Calling threads of another process during RPC

2. **Threads created for RPC discarded later:**
   - In the second method, threads are created for RPC execution and discarded later.
   - RPC execution can be speeded up based on the observation that when the server threaded blocks waiting for a new request, it does not have any important context information.
   - Hence, when a thread has finished carrying out the request, it vanishes, and its context-related information can be discarded.
3. **Threads created on-the-fly based on RPC requests:**
   - In the third method, threads are created on-the-fly based on RPC requests at the server.(fig 6-31)
   - When a new message arrives at the server machine, the kernel creates a new thread to service this request.
   - The message is mapped to the server's address space.
   - A stack is set up, so the new thread can access the message.
   - This scheme is called an 'implicit receive'.
   - The thread created to handle the incoming local RPC is called pop-up thread.

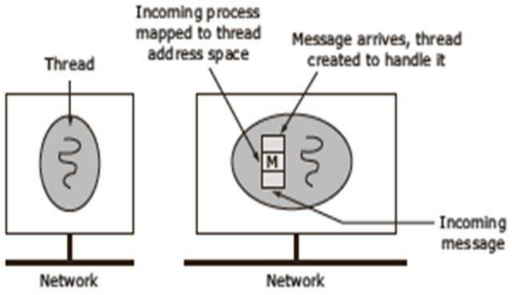


Figure 6-31 Implicit receive method of RPC execution

FAULT TOLERANCE

- A fault is a malfunction caused by errors due to designed, programming etc.
- All faults do not lead to immediate system failure, but may degrade the system performance by giving an unexpected output.
- Faults are classified as :
  - **Transient:**
    - These faults occur suddenly, disappear, and may not occur again of operation is repeated.
  - **Intermittent:**
    - These faults recur often, but may not be periodic in nature.
  - **Permanent:**
    - These faults can be easily identified and the component can be replaced.

- The main objective of designing a fault-tolerant system is to ensure that the system operates correctly even in the presence of faults.
- Faults and failures go hand in hand and can occur at all level.
- Mean time of fault is calculated as:

$$\text{Mean time to failure} = \sum_{k=1}^{\infty} kp\,(1-p)^{k-1}$$

$$\text{Mean time to failure} = 1/p$$

System failures:

- **Fail silent faults / fail stop faults :**
  - In fail-silent faults, a faulty processor stops responding to any input and does produce further outputs.
  - It intimates to the system that it is no longer functioning.
  - These faults are also called *fail-stop faults.*
- **Byzantine faults:**
  - The term Byzantine came from the Byzantineempire infested with endless conspiracies, intrigues.
  - The system may work maliciously with faulty processors giving an impression of correct operations.
  - Software bugs lying undetected in the system my exhibit these faults.
  - These failures are difficult to handle because detection itself a painstaking task.

Use of redundancy:

- **Information redundancy:**
  - Extra bits are added with the data transmitted to allow recovery from garbled bit detect and correct error.
- **Time redundancy:**
  - An action performed once is repeated if needed after specific time period.
- **Physical redundancy:**
  - Extra component are added to enable the system to tolerate faults due to loss or malfunction of some components.
  - **Active replication:**
    - In this method all processors are up all the time in parallel and they hide the faults completely.
  - **Primary backup methods**
    - In this method, one server is designated as primary and it does all work.
    - If primary server fails, the backup server takes over.

Unit 6

- **Distributed Shared Memory**

Fundamental concepts of DSM, types of DSM, various hardware DSM systems, Consistency models, issues in designing and implementing DSM systems.

- A DSM provides a logical abstraction of shared memory which is built using a set of interconnected nodes having physically distributed memory.

DSM Architecture:

- The advantages offered by DSM include ease of programming and portability,achieved through the shared memory programming paradigm.
- DSM systems are claimed to be scalable, and achieve very high computing power.
- A DSM system can generally be viewed as set of nodes or clusters, which are connected by an interconnection network and having view of a single logical shared memory.

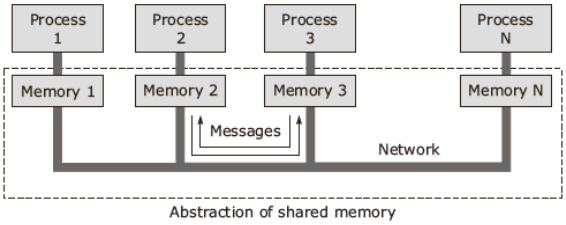


Figure 7-1 DSM architecture using nodes

- As shown in fig. , DSM is a collection of workstations connected by a LAN sharing of virtual address space.
- Processes running on different nodes can exchange message through the implementation of a simple message-passing system. Conceptually, the DSM abstraction exists only virtually.
- The DSM cluster-based architectureis built using interconnected clusters and poses a single virtual address space.
- The memory accesses are managed by either DSM hardware or software.
- A cluster can be uniprocessor or multiprocessor system.
- Each cluster in the system contains a physical memory module, v global address space.

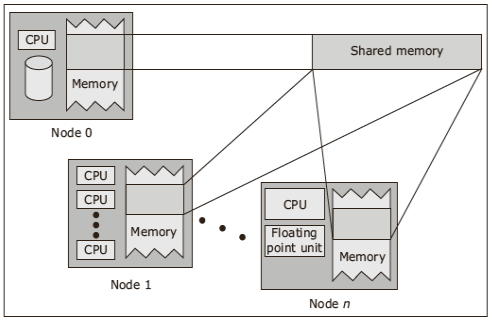


Figure 7-2 DSM architecture using clusters

- This DSM abstraction presents a large, shared memory space to all processors of all nodes.
- Data caching is used in DSM systems to reduce n/w latency.
- The unit of caching is a memory block.

Message Passing vs Shared Memory :

- In Message Passing paradigm, which uses two basic primitives for IPC, namely send & receive.
- The sending process puts the data to be shared in the message and sends it to the correctly addressed receiver.
- The DSM paradigm provides processes with common shared address space which they can use just like a local memory with standard memory semantics, such as read & write.

Criteria	Message Passing	DSM
Programming Model	Variables have to be marshalled, transmitted,& unmarshalled at the receiving process.	Variables & pointers to shared Variables can be accessed from the shared memory directly. No other communication process is required.

Synchronization process	Achieved using message passing primitives such as locks and semaphores.	Achieved using shared memory primitives, such as aquire and release.
Timing	communication processes have to execute at the same time.	communication processes can execute in non-overlapping lifetimes.

Types of DSMs:

1. **Hardware-level DSM:**

- Hardware-level implementation of DSM is a natural extension of cache coherence mechanism, which are used in shared-memory multiprocessors with private caches.
- These implementations use a smaller unit of sharing, cache blocks & hence; they are particularly superior for applications that have a high level of fine-grain sharing.
- HDSM is often used in high-end machines where performance is more important than cost, but these implementations are not scalable to more than 10 systems.

2. **Software- level DSM:**

- If a processor does not find the page in its local memory, it triggers a page fault to the DSM runtime software.
- It locates the page 7 transfers it to the local memory of the requesting processor.
- The DSM runtime software is responsible for maintaining consistency among replicas of pages available on different machines.
- Software support for DSM is more flexible, convenient, and scalable than Hardware implementations, but in many cases, it cannot compete with the hardware-level DSM in performance.
- Fig shows a DSM system implemented in software.
- If a processor does not find a page in its local memory, it triggers a page fault to the DSM runtime software.
- It locates the page and transfers it to the local memory of the requesting processor.

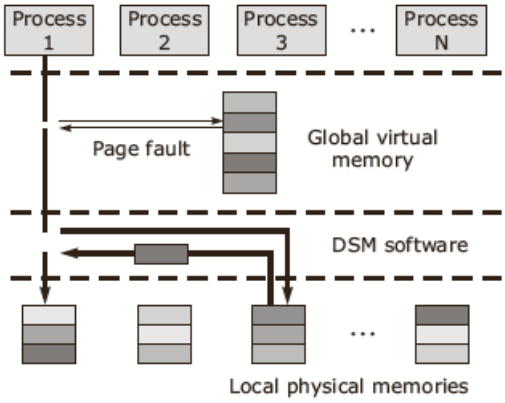


Figure 7-3 Page access in an SDSM system

3. **Hybrid-level DSM:**

- As the name implies, hybrid-level DSM lies between Hardware & Software DSM systems.
- Example is Non-uniform Memory Access (NUMA) machines.
- Like a multiprocessor, each NUMA processor can access each word in the common virtual address space by reading or writing to it.
- Here caching is controlled by software and not through hardware like in multiprocessor system.

Advantages of DSM:

- DSM is a better way of IPC than message-passing which relievers the implementation programmer from writing programming constructs.

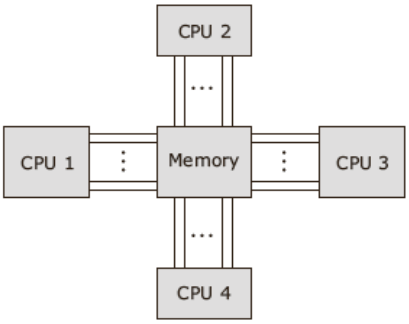
1. **Simple abstraction**
   - The first advantage is that DSM provides a simple abstraction.
   - Direct programming using message passing is tedious and error prone, since programmers have to be aware and keep track of data movement across the processor.
   - Also, programmers have to use explicit communication primitives, channels or ports.
   - RPC is used to overcome this problem, but it faces difficulty in case the caller wants to pass context-related data or complex data structure.
   - DSM provides a simple abstraction to application programmers.

2. **Improved portability of distributed application programs**
   - The second advantage is improved portability of distributed application programs.
   - It is easy to transition from sequential to distributed application because both of them use consistent access protocols.
   - This implies that the solution application programs for shared memory multiprocessors can be directly executed on DSM systems without making any changes.

3. **Provides better performance in some applications**
   - DSM also provides better performance in some application.
   - It uses services of the lower-level message-passing system.
   - DSM implementations take advantages of the locality of the locality of data in application programs.
   - DSM algorithms allow movements of data between nodes and application and exhibit a reasonable degree of locality of access.
4. **Large memory space at no extra cost**
   - DSM provides a large memory space at no extra cost, which the application developer does not usually expect.
   - With DSM, the total memory size is the sum of the memories of all nodes.
   - Hence, paging and swapping is decreased.
   - DSM also provides a flexible communication environment.
5. **Better than message passing systems**
   - In a message-passing system, identification and co-existence of both the sender and receiver processes are essential.
   - Otherwise, the message is undelivered.
   - This is not the case in DSM system because the sending process places the data in the shared memory and other process accesses this data at its own time.
   - Hence, the lifetimes of both the sending and receiving processes are independent of each other.
   - Process migration becomes easier in DSM implementation.
   - Migration is a tedious and time-consuming process because it involves the transfer of address-space from the old to the new node.
   - DSM provides on-demand migration by detaching the Process Control Block (PCB) from the old node and attaching it to the new node.

Hardware DSM

1. **On-chip Memory DSM:**



- Most computers have an external memory, while some CPUs with little functionality and on-chip shared memory is widely used in cars, appliances, and toys.
- The address and data lines are directly connected from all the CPUs to the single shared memory.
- Practically, it is very expensive and impossible to build a set of hundred CPUs with a single shared on-chip memory.

2. **Bus-based Multiprocessor:**

- A bus is set of parallel wires carrying either address or data information. In a computer, buses are used to transfer information between CPU, memory, and I/O controllers.
- To perform a read or write operation, CPU sends memory address on the bus and asserts the bus control line, which indicates that it is going to use the bus.
- The requested word is fetched from the memory, transferred over the bus, and read by the requesting CPU.
- *Bus arbitration mechanism* is used to avoid multiple CPUs from accessing the bus at same time.
- This mechanism is either centralized or de centralized.

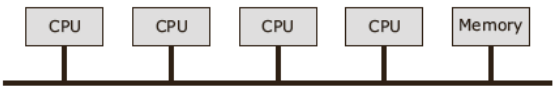


Figure 7-5 Bus-based multiprocessor

Figure 7-6 Bus-based multiprocessor with caching

- The main disadvantage is that more than three or four CPUs immediately overload the bus.
- Network traffic on the buses can be reduced by using caches with each CPU apart from shared memory, as shown in following. Fig 7-6.
- A simple protocol used to maintain consistency among cache is the *write-through cache consistency protocol.*
- When the CPU first reads a word from the memory, the word is fetched and stored in the cache of the requesting CPU.
- Later the word is needed again, it is taken from the cache itself and thus bus traffic is avoided.

Operation	Action taken by cache for local CPU operations	Action taken by cache for remote CPU operations
Read miss	Fetch data from memory and store in cache	No action
Read hit	Fetch data from local cache	No action

Write miss	Update data in memory & store in cache	No action
Write hit	Update data in memory & cache	Invalidate cache entry

- The cache consistency protocol used in the bus-based multiprocessor systems has three important properties:
  - Consistency is achieved, since all caches snoop.
  - Protocol is built into MMU.
  - The algorithm is performed in one memory cycle.

3. Ring-based Multiprocessor:

- Ring-based multiprocessors are implemented in Memnet DSM.
- A single shared memory is divided into private and shared areas.
- The private space further allocated per machine to accommodate unshared data 7 codes.
- Hardware is used to keep the data in shared spaces consistent, which is common to all machines.
- The Memnet device consists of the ring interface, MMU, cache, & a part of memory.
- This sytem does not have any centralized global memory but each block has a home machine.
- Data blocks can be cached either in a home machine or a remote machine.
- The global memory in reality is physically distributed across all the machines.
- A read-only block may be present on multiple machines or a read-write block can be present on only one machine.
- In both cases, the block may not be present on its home machine.
- If a block has no definite place, then it will get a definite place on home machine.

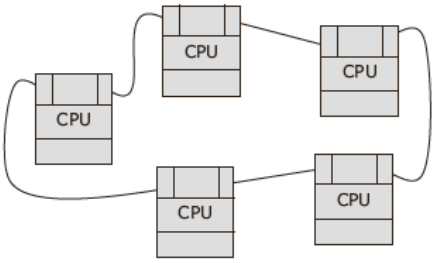


Figure 7-7 Memnet ring

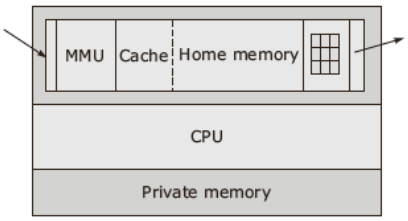
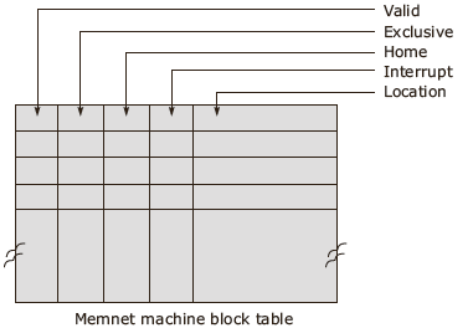


Figure 7-8 Memnet single machine memory

- As shown in above fig, the Memnet device on each machine contains a table which contains entries per block in the shared address space.
  - *Valid* bit tells us whether the block is present in cache and is up to date.
  - *Exclusive* bit tells us whether it is local copy & whether it is only one copy.
  - *Home* bit is set if it is the data blocks home machine.
  - *Interrupt* bit is used to force interrupt.
  - *Location* field tells whether the block is located in the cache, and whether it is present and valid.

- The Memnet protocol used for the read-write operations works like this.
- When the CPU wants to read a word from the shared memory, the memory address is passed to the Memnet device which checks the block table for the block's presence.
- Then the request is satisfied immediately; else the Memnet device waits to capture the circulating token, puts the request, and suspends the CPU.



4. Switched Multiprocessor:

- Both the bus & ring –based multiprocessor systems work well for small systems but do not scale well beyond 64 CPUs.
- As CPUs are added, both the bus and ring bandwidth saturated and reduce system performance.
- To overcome this problem, we can either reduce the amount of communication by caching or increase the communication capacity by adding more than one bus, or by using tree or grid structure.

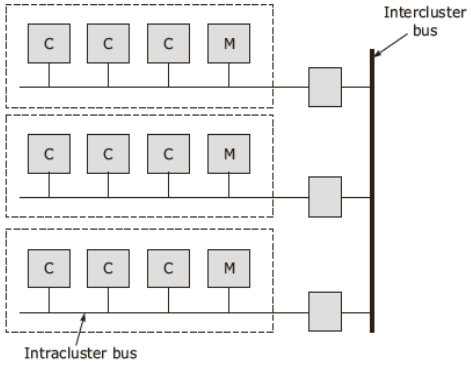


Figure 7-10 Switched multiprocessor

- As shown in above fig, multiple clusters can be interconnected using an inter-cluster bus.
- The amount of inter-cluster traffic is reduced in case the CPUs communicate within their own cluster. If a large bandwidth is needed, it is possible to connect multiple inter-clusters to form a super-cluster using the grid or the tree-structured bus.
- Dash is a typical example of a switched multiprocessor, built as a research project at Stanford and it uses a hierarchical design-based grid of clusters.
- It consists of 16 clusters with each cluster consisting of four CPUs, 16 MB of global memory, and some I/O devices.
- Each CPU can snoop only on the local bus.
- The total address space available in Dash Prototype is 256 M divided into 16 regions.

Consistency Models:

- Memory consistency model refers to the time frame, i.e., how recent the shared memory updates are visible to all the others processes running on different machines.
- A read to a shared data items Y returns the value stored by the most recent write operation on Y.
- Several, consistency models are strict, sequential, casual, PRAM, processor, weak, and release, entry and scope consistency models.
- To explain various consistency models, we have used a special notation.
- Processes are represented as P_1, P_2, etc.
- The operations carried out by each process are shown horizontally with the time increasing towards right.
- The symbols used are *W(x)a&R(y) b* which means write to x with value a & read from y which returns a value b.
- The initial value of all variables is assumed to be 0.

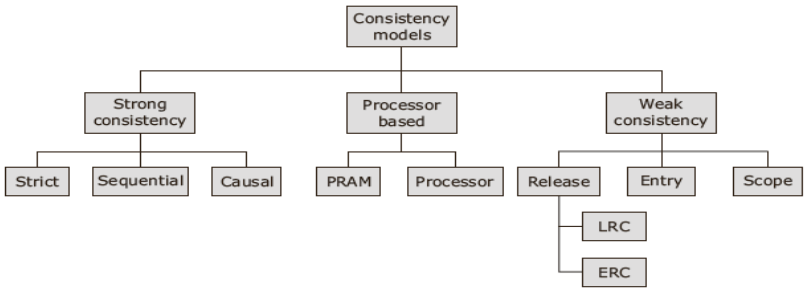


Figure 7-13 Classification of consistency models

1. Strict consistency:

- Strict consistency is the strongest form of consistency model.
- A shared memory system supports strict consistency model if the value read is the last value written irrespective of the location of processes performing the read or write operations.
- It is impossible to implement the strict consistency model in a distributed system.
- When the memory is strictly consistent, all writes are instantaneously visible to all processes and an absolute global time order is maintained.

Figure 7-14 Strict consistency

Example:

- Suppose P_1 does a write operation to a location storing a value 5.
- Later when P_2 reads x, it sees the value 5.
- This implies strictly consistent memory.
- But if the value 5 is not written immediately, and suppose P_2 does a read, it would get a stale value.

2. Sequential consistency:

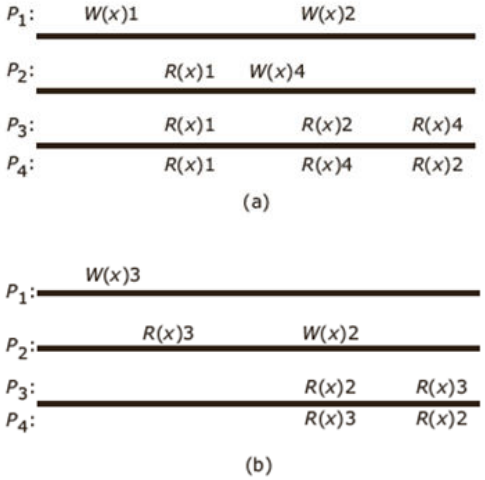
- A shared memory supports the Sequential consistency model, when all processors in the system observe the same ordering of reads & writes, which are issued in sequence by the individual processors.
- The result of any execution is the same as if the operations of all processes were executed in some sequential order, and the operations of each individual process appear in the sequence in the order specified by its program.
- This consistency model can be implemented on a DSM that replicates writable pages by ensuring that no memory operation is started until the earlier ones have been completed.

Two possible cases of running the same program

Figure 7-15 Sequential consistency

3. Casual consistency:

- The casual consistency model represents a weakening of sequential consistency for better concurrency.
- All processes see only those reference operations in the same correct order that are casually related to each other.
- A casually related operation is the one which has influenced another operation.
- If two processes simultaneously write two variables, they are not casually related.
- However, in case a read is followed by a write, the two operations are casually related.
- A read is casually related to a write if it takes data from a write operation.
- Operations which are not casually related are said to be *concurrent.*



- Fig 8-16(a) shows an example that violates the definition of casual consistency, while fig 8-16(b) depicts the correct sequence of casual memory.
- W(x)3 depends on W(x)2 which in turn could be a result of computation a read by R(x)3.
- W(x)3 & W(x)2 are casually related.
- The value should be seen in the same order by all the processes, i.e., first value of x=3 and the x=2; but P_4 sees it in a different order.
- Thus, it violates the casual memory concept.

4. PRAM Consistency:

- Pipelined Random Access Memory (PRAM) consistency model provides weaker consistency semantics as compared to the consistency models described so far.
- This consistency model is simple, easy to implement, and provides good performance.
- The write operations are sequenced at each node, independent of the write operation on other nodes.
- Performance is better, since the process need not wait for a write operation performed by it to complete before starting the next one and all write operation of a single process are pipelined.

Figure 7-17 PRAM consistency

5. Processor consistency:

- Memory coherence implies that all processes agree on the same order of all write operations to that location.
- This model assumes that the order in which the memory operations are seen by two processors need not be identical, but the order of write issued by each processor must be preserved.
- PRAM consistency & processor models are among the stronger consistency models & they involve large amount of network communication which turns out to be expensive.
- PRAM consistency & processor consistency can give better performance than the stronger models.
- Weaker consistency models are designed to take advantages of specific characteristics of many applications.
- The first characteristic is that there is no need to show the change in memory done by every memory write operation to other processes.
- Multiple write operation results can be combined & then sent to other processes when needed. Secondly, isolated accesses to shared variables are rare.
- The main problem arises when there is a need to show the changes performed by a process to other processes, since time is different for different applications.

6. Weak consistency:

- All DSM system that supports the week consistency model use a special variable called the *synchronization variable.*
- When this variable is accessed by a process, the shared memory is synchronized by making all changes visible to all other processes.

- Weak consistency model require that the memory addresses.
- Weak consistency model requires that the memory be made consistent only on synchronization accesses which are divided into 'acquire' & 'release' pair.
- DSM does not propagate the updates till the process leaves the critical section where it is accessing a shared variable.

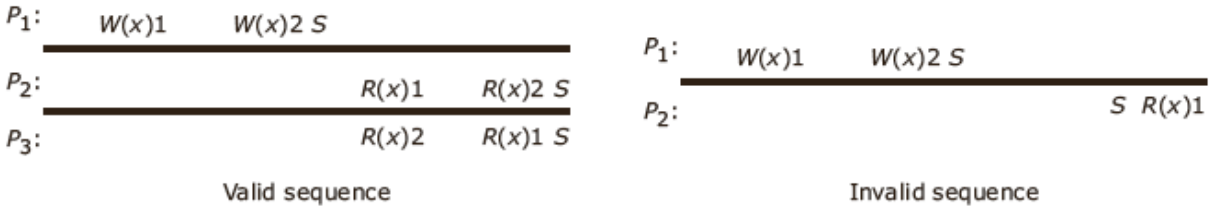


Figure 7-18 Weak consistency

- The requirements to be met for supporting weak consistency specify that all accesses to synchronization variables must obey sequential consistency semantics.
- The three properties of the weak consistency model are as follows:
  - Access to synchronization variable is sequential consistent.
  - Access to synchronization variable is allowed only when all previous writes are completed everywhere.
  - Until all previous accesses to synchronization variable are performed, no read-write data access operations will be allowed.
- Weak consistency has a problem, because it uses a single synchronization variable.
- When accessed, the memory does not know it, because the process has finished writing the shared variables, or is about to start reading them.

7. <u>Release consistency (RC):</u>

- RC mechanism tells the system whether a process is entering or existing a critical section.
- So the system decides whether to perform the first or second operation when synchronization variable is accessed by a process.

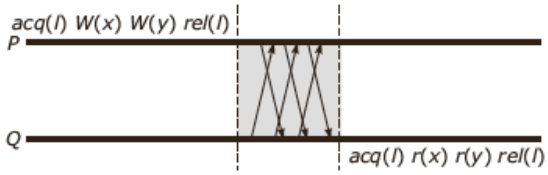


Figure 7-19 Relaxed memory consistency model

- Two synchronization variables are used for this purpose, namely acquire & release.
- Acquire tells the system that it is about to enter a critical section, so the system perform only second operation when the variable is accessed.
- The release variable is used by a process to tell the system that it has exited the critical section, so that the system performs only the first operation when this variable is accessed.
- RC can also be implemented using a synchronization mechanism called *barrier* which defines the end of execution of a group of concurrently executing processes.
- Unless all processes reach the barrier, others which reach the barrier are blocked, and none of the processes are allowed to proceed with their execution beyond the barrier.
- when all process reach the barrier, the shared variables are synchronized and then all processes resume execution.

8. <u>Eager Release consistency (ERC):</u>

- Assume that x is replicated at two processors.
- With ERC, a message has to be sent to other nodes at the time of release, informing the changes to x and y.
- However, only the next processor that acquires the lock can access x and y.
- The release operation is blocked until acknowledgments have been received from all other cache.

9. <u>Lazy Release consistency (LRC):</u>

- With LRC, the modifications are not sent to other nodes at the time of release.
- Apart from reduction in message traffic, LRC also allows the notification of modification to be piggybacked on the*lock grant* message going from the releasing to the acquiring process.

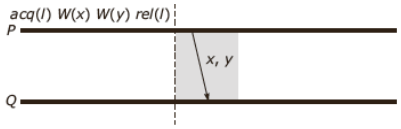


Figure 7-20 ERC: Changes propagated to all nodes at the time of release

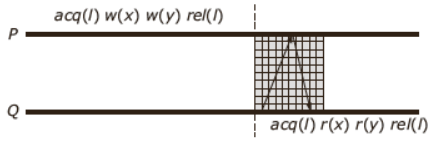


Figure 7-21 LRC: Synchronization of memory occurs upon successful acquire

10. Entry consistency (EC) :

- EC requires to programmer to use acquire and release at the start and end of the each critical section, respectively.
- However, unlike RC, EC requires ordinary shared variable to be associated with a synchronization variable, such as a lock or a barrier.
- EC is similar to LRC but is more relaxed.

11. Scope consistency (ScC) :

- ScC provides high performance like EC & good programmability like LRC.
- This is achieved by use of scope concept, which reduces the amount of data updates sent among processors, and fits naturally into thesynchronization provided by the lock mechanism.
- In ScC, a scope is limited view of memory with respect to which memory references are performed.
- The acquire operation open a scope, while the release operation closes the scope. ScC is more efficient than LRC.

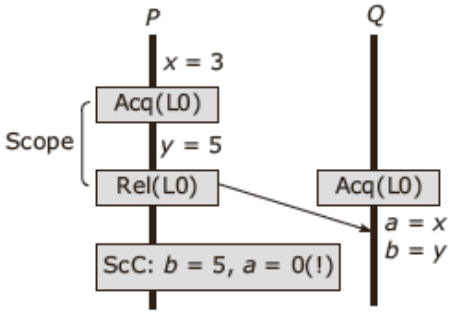


Figure 7-22 Scope consistency

Design Issues in DSM System :

1. Granularity:

- Granularity of sharing or block size is the most visible parameter chosen in the designing of the DSM system.
- The shared memory can be word, page, cache block, complex data structure.

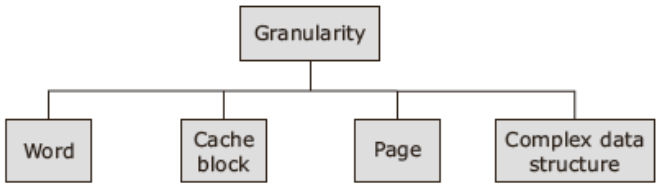


Figure 7-11 DSM granularity

- The unit determines the size of the data blocks which are shared and transferred across the network on a memory fault.
- Hardware DSM system use smaller units, while software DSMs based on virtual memory mechanisms organize data in larger physical blocks(pages), counting on coarse-grain sharing.
- The use of larger blocks results in saving space for directory storage, but it also increase the probability that multiple processors will require access to the same block simultaneously, even if they actually access unrelated parts of that block, referred to as *false sharing*.

2. Structure:

- The structure of shared data represents the abstract view of shared memory space which is represented to the DSM application programmer.
- It also refers to the organization of data items in the shared memory.
- Structure & granularity of data are closely related to each other.
- The shared memory may appear either as a storage of a word or as a storage of data objects.
- There are three types of structuring of shared memory: no structuring, structuring as data type, and structuring as database.

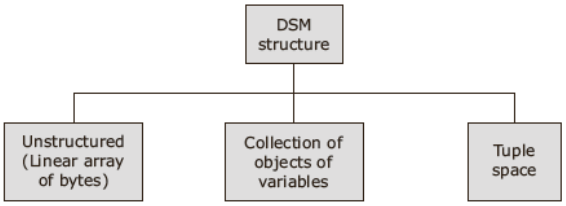


Figure 7-12 DSM structures

- The simplest type of DSM does not have any structure; it is just linear array of words.
- The main advantage is that any grain size can be fixed for sharing and is easy to design.
- The shared memory space can also be structured based on data type as either a collection of objects or a collection of variables in source language.
- The third method is to structure the DSM system as a database called a *tuple space*. Primitives are used to access the data in shared memory.
- Process on multiple machines share an abstract space consisting of shared objects.
- The DSM runtime system handles the location and management of object automatically.

3. Coherence Protocols:

- A coherence protocol specifies how the rules set by the memory consistency model are to be implemented.

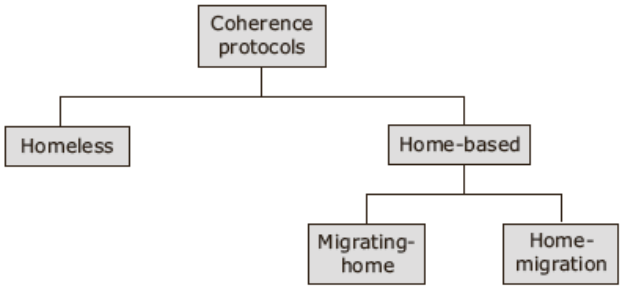


Figure 7-25 Types of coherence protocols

1. Homeless protocol :

- The homeless protocol, does not assign any processor to hold the most updated copy of a page.
- Serving a page fault in a homeless protocol can be complicated.
- A processor may have to contact multiple processors which have modified the page to collect the updates, and then apply them to collect the clean page.

2. *Home-based protocol:*

- In home-based protocol, a processor is fixed to hold the most up-to-date copy of every page in shared memory.
- This processor is known as home of the page. Under the home-based protocol, the updates made by every processor on a page are propagated to the home processor at synchronization time.
    - *Migrating-home protocol:*
        - If the home processor itself never accesses the page, then the updates made by other processors must be propagated through the network at synchronization time.
        - If the home can be migrated to the processor which accesses the page, then the updates made by new home need not be sent anywhere.
        - Hence, Migrating-home protocol is used, which allows the home location of a page to be migrated from a processor when serving a page fault.
    - *Home-migration protocol:*
        - It allows change of home location if there is only one writer to a page between two barriers.
        - However, instead of performing home migration when serving a page request, the migration is done at the barrier synchronization.

- Various algorithms are available which decides the type access allowed.
- They are classified are as follows:

Single reader/ single writer algorithm :

- This algorithm allows only a single copy of the page to exist in the system.

Multiple reader/ single writer algorithm :

- This algorithm is commonly used in DSM systems and allows replicas to exist for read access but only one copy exist for write operation.

Multiple reader/ Multiple writer algorithm :

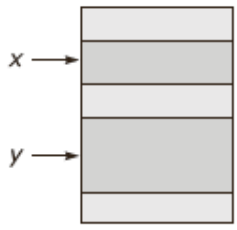
- This algorithm allows existence of multiple copies of pages for reading as well as writing.

Issues in Implementing DSM system :

- Various issues involved in implementing DSM systems are thrashing, responsibility of DSM management, replication vs migration strategies.

1. **Thrashing :**

- False sharing occurs when two or more processes want to access different variables x and y which reside on the same page.
- Even though the original variables are not shared, it appears as if the page is shared by all the processes.
- The larger the page size, the probability of false sharing is higher, since the page, may contain many variables which can be used independently by different processes.
- False sharing of a page leads to thrashing.
- Thrashing occurs in DSM systems which allow migration of block betv
- System spends large amount of time transferring data among the nodes.



Page

Figure 7-29 Acc ess to different variables on the

- Thrashing occurs when interleaved data access is made by more than one node on the same data block within a short interval of time, also called as the *ping-pong-effect.*
- Thrashing can also occur when block with read-only permissions are repeatedly invalidated after replication.
- Thrashing can be resolved or reduced using various techniques.
- The first one is to provide an *application-controlled lock,* so the other nodes can be prevented from accessing the block for a short time period.
- The second technique is to pin the block to a node for a specific time period.

2. **Responsibility for DSM Management :**

- The following methods are used for data location and DSM consistency management :

Centralized-manager algorithm:

- In a centralized-manager algorithm, all read and write requests are sent to the server(S), which keep the identity of the owner node (N) of a particular data
- The server forwards the request for data to the owner, and site indicating that it receives the copy of the block from the owner.
- The algorithm working is shown in fig.
- This algorithm has two disadvantages:
  1. It serializes requests and reduce parallelism.
  2. Results in system failure in case of server crash.

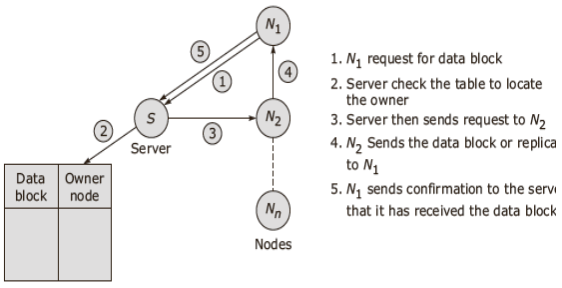


Figure 7-30 Centralized-manager algorithm

Broadcast Algorithm:

- In a broadcast algorithm, each node (N) maintains an owned blocks tables (T) containing entries for each block(B) currently owned by a node.
- When a fault occurs, the fault handler broadcasts a read-write request on the network.
- The node currently having the request block responds to the request message by sending the block.
- The main problem with this algorithm is scalability, since the nodes not having the block also have to process the request.

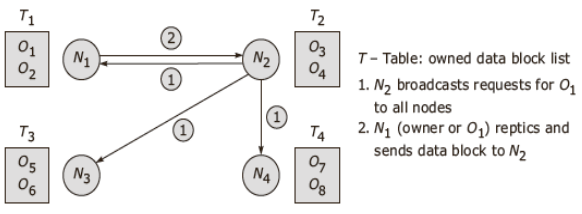


Figure 7-31 Broadcast algorithm

Fixed distributed-manager Algorithm:

- In this method, instead of centralizing the management, each server (S) is predetermined to manage a subset of data block (B).
- It overcomes the problem of the centralized server schemes by distributing the role of the centralized server to many nodes (N).
- Each server node is given a predetermined set of data blocks to manage, based on mapping function.

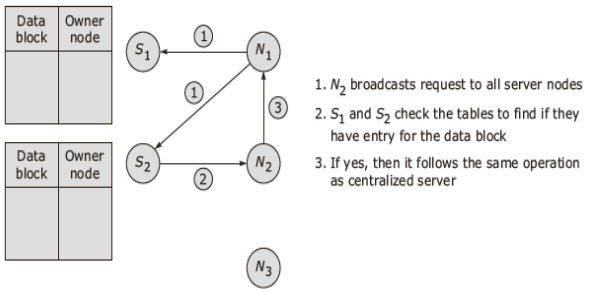


Figure 7-32 Fixed distributed-manage algorithm

Dynamic distributed-manager Algorithm:

- This method, keep track of ownership information of all blocks in each node. Each node (S/N) is a mini server and manages few blocks.
- The node table per block (T) keeps the ownership information for all blocks in the shared memory space.
- The ownership information kept in the table may not always be correct but provides a beginning for traversal to the true owner node of block.
- This field is also called as the *probable owner*.
- In case of block fault, the faulting node extracts the probable owner entry in its table and sends a request for the block to the node.

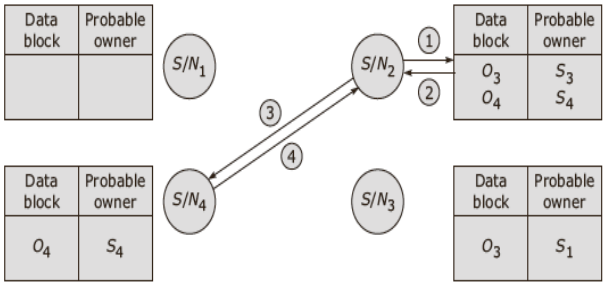


Figure 7-33 Dynamic distributed-manager algorithm

3. **<u>Replication VS Migration Strategies:</u>**

- The protocols used for implementing a consistency model in a DSM system depend on whether the DSM system allows replication and/or migration of shared data blocks.

No Replication:

- It is possible that a DSM system does not replicate any data.
- Then a single copy of each block exists in the entire system and its location is fixed.
- This information can be mapped into a directory structure.

Replication:

- This strategy helps in increasing parallelism and almost all DSM systems replicate blocks.
- Read operations can be carried out in parallel at more than one node by accessing local copies of data.
- The communication cost is incurred reduce, thus leading to reduction of average cost of read operations.

No Migration:

- In this strategy, the location never changes.
- The location information can be mapped into a directory structure, and on a page fault, the directory is used to locate the faulting block address.
- Data location is fast and depends on the directory size.

Migration:

- Unlike non-migration, the owner node of a block changes, since the block is allowed to be migrated across the entire DSM system.
- When it is migrated, its new address has to be immediately updated in the directory.
- This method is advantageous because it assists in reducing network traffic.

Non-replicated, non-migrating block:

- It can be implemented in places where each block of the shared memory has a single copy and whose location is always fixed.
- All access request are sent to the owner node which has only one copy of block.

Non-replicated, migrating block:

- This scheme has a single copy of block in the entire system.
- Access request from any node other than owner node causes to block to migrate to the requesting node, and the ownership of the block changes.
- When the block is migrated, it is removed from the local address space where it was mapped. Here, only one node can read or write on data block at a time.

Replicated, migrating blocks:

- Most DSM systems replicate blocks to improve performance, since read operations can be carried out in parallel.
- This results in a decrease in average cost of read operations, since replicas exists at a local node.
- Replication increases the cost of write operations because all replicas must be either invalidated or updated to maintain consistency.

Replicated, non-migrating block:

- This strategy allows replication at multiple nodes, but the location of replica is fixed.
- A read or a write access request is carried out by sending the access request to any one of the nodes having a replica of the block.
- All replicas have to be kept consistent by updating them, using write-update protocol.

Unit 7

Distributed File System Concepts of a Distributed File System (DFS), file models, issues in file system design , naming transparency and semantics of file sharing, techniques of DFS implementation,

Introduction:

- Files are used as repository to store information.
- A file system is a part of the OS that performs file management functions, such as organization, storage, retrieval, naming, sharing, and protection of file.

Functions of DFS :

- The first feature is *remote information sharing*, which implies that any file should be transparently accessed from any node irrespective of the location of the file.
- The second feature is *user mobility,* i.e., the system should be flexible such that one can work from any node at any instance of time without the need to relocate any storage device.
- The third feature is *availability,* which implies that the files should always be available in spite of any temporary failure. The DFS hence maintain multiple copies on different nodes, which are called *replicas*.
- DFS should *support diskless workstations*. These workstations are economical, less noisy, and generated less heat because they allow disks to be separatedfrom the workstations.

Component of DFS :

- *Storage service* is related to the allocation & management of space on the secondary storage. It provides a logical view of storage to the users. This is possible by providing operations for storage & retrieval of data.
- *True file service* provides operations on individual files, such as access, modification, creation, deletion, etc.
- *Name service* is another component which enables the users to identify file easily with text names which are mapped to internal file IDs used to locate the files. This service is also called *directory service,* and performs operations, such as create, delete, add, etc.

Features of DFS :

- **Transparency :**
  - *Structure Transparency* – It is useful to achieve performance, scalability, and reliability.
  - *AccessTransparency* – It enables access to local or remote files to be provided in same way.
  - *Naming Transparency* – It implies that the name should not indicate the location of the file. File movement should be possible without changing the name of the file.
  - *Replication Transparency*– When file is replicated on multiple nodes, the number of copies & location should be hidden from the users.

- **User mobility :**
  - User should be able to view or access the file system from any node at any time & the system should exhibit the same performance.
- **Performance:**
  - It is measured as the average amount of time needed to satisfy a user's request for access to a file.
  - In a conventional system, it is equal to the summation of the time required to access secondary storage & the CPU processing time.
- **Ease to use :**
  - A good DFS system should be simple and easy to use.
  - The semantics and the interface should be easy, user friendly, & should support a large number of applications.
- **Scalability :**
  - Most distributed systems span across locations.
  - Hence, a good DFS should be scalable & should the growth of nodes & users, without disruption of service or loss of performance.
- **Availability :**
  - This implies that the system should continue to function even in case of partial failure. Some degradation may be allowed, but the entire system should not break down.
- **Reliability:**
  - It is important to minimise the loss of stored data, thus reducing the load on users to create their own backups.
  - The system should create backups automatically, which will prove useful in the event of loss of original files.

- **Integrity of data :**
  - A good DFS should ensure the integrity of data. Concurrency control mechanism must be used to allow multiple users to access files.
  - Atomic transactions should also be used to maintain the data integrity of files.
- **Security :**
  - Appropriate security mechanisms must be implemented so that the users are confident of the privacy of their data.
  - Security mechanisms must be implemented to prevent unauthorized access.
- **Support for heterogeneous systems :**
  - A distributed system comprising heterogeneous machines provides flexibility to the users to work on different computing platforms and applications.

File Models:

- File models decide the performance of a DFS, since they are based on how the data is stored in the file & what method is used for modification.

Structured & Unstructured Files:

- In the formal model, as the name suggests, the file server understands the file as pure sequence of bytes.
- Modern operating systems use this structure.
- The other file model is the structuredFile model.
- Here, the file appears as an orderedsequence of records each possibly of variable sizes.
- In the indexed file system, a record has akey field, & the file is accessed by specifying thiskey.

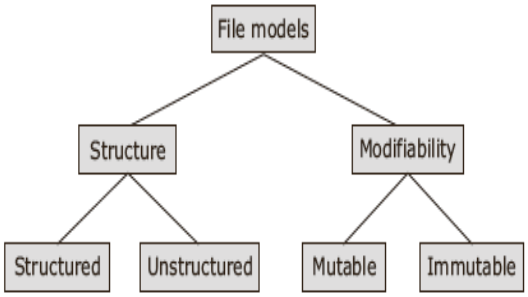


Figure 8-1 File models

Mutable & Immutable Files:

- Files have attributes. These attributes can be modified by user. Based on how file can be modified, they are classified as mutable & immutable files.
- Most current operating system use the mutable model where updates on a file overwrite the old contents, producing a new file.
- In immutable file model, file cannot be modified once created, but can only be deleted.
- Each file can have multiple versions, and can be maintained separately.
- Practical implementation of such model increases the disk space & disk allocation activity.
- Advantage of immutable files is that file caching & replication become easier.

Issues in DFS Design:

- Major components of DFS are true file service & directory service.

File Service Interface :

- File service interface enables the users to access file in a distributed environment.
- File accessing model decide how the user's request for file access is serviced.
- The file access model is depends on two factors: the *method* for remote file access, & the *unit* of data access.

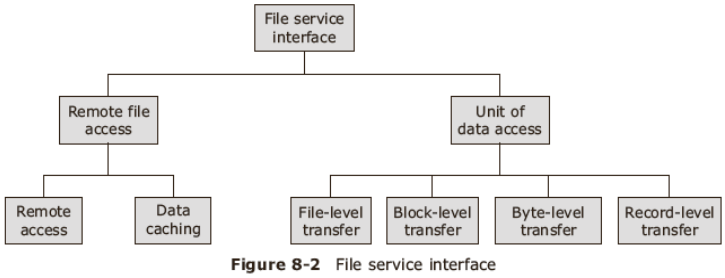


Figure 8-2 File service interface

1. **Based on method of Remote file access :**

The model can be classified as *remote access model* &*data caching model*.

Remote access model:

- In this model, the user's request is performed on server node,

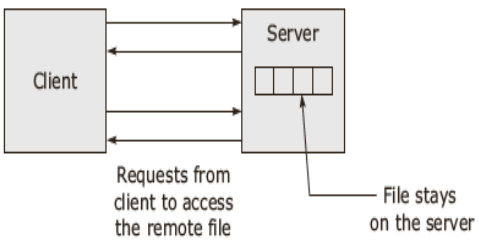


Figure 8-3 Remote file access

& a copy of the file is returned to the user.

- The request & response message are transferredacross the network as data packets along with communication overheads.
- Therefore, a remote file service model'sInterface & the communication protocols must be designed carefully to minimize the overheads attached to be number of messages required to satisfy the request.

Data caching model:

- This is also called as upload/download model.
- This model uses the locality feature of data access.
- On the first request for a file from the user, the datais brought to the user's node,cached, & subsequent requests are satisfied locally.
- This access model is implemented in Sprite distribute
- system.

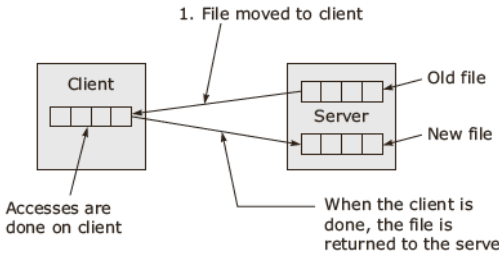


Figure 8-4 Data caching model

2. **Unit of data access :**

File level transfer :

- In this model, the entire file is moved when a user makes a request for a file.
- It is efficient to transmit the file page-by-page for multiple requests made for the same file system, since the protocol overhead is required only once.
- This model assists in scalability because fewer accesses are made to the file server, resulting in decreasing network traffic & server load.
- This disk scheduling routine can be optimized because it is known that the entire file has to be transferred.
- This model supports heterogeneous workstations.
- The drawback of this model is that more storage space is required on the user's node.
- This model is also inconvenient for large files, especially in diskless workstations.

Block-level transfer :

- In this model, blocks are transferred on user request.
- If the virtual memory page size tallies with the file block size, it is called page-level transfer.
- Advantage is, it does not require large storage space, because the entire file is not copied when small portion are required.
- Hence, this is suitable for diskless workstations.
- For large files, there is needed to make multiple requests for accessing the same file.
- It increases network traffic.
- The results in poor performance as compared to the file transfer model.

Byte-level transfer :

- In this model, the unit of data transfer is byte.
- This level of transfer provides maximum flexibility because storage & retrieval is possible for any range of file.
- Cache management is difficult due to the variable size of access requests.

Record-level transfer :

- It is structured file model where the unit of transfer is a record.
- Files can be protected using the capabilities & access control lists.
- Each user has the capability for each object to enable access.
- The capability describes the type of permissible access.
- An access control list specifies the list of users who can access the file & how they can obtain access.

Comparison of the file transfer models :

Table 8-1 Comparison of file transfer models based on unit of data access

Type of transfer	Unit of transfer	Advantages	Disadvantages
File-level	File	Fewer accesses to file server and reduces network traffic. Good for small-sized files. Supports heterogeneous environments.	Not suitable for diskless work-stations. Not suitable for large-sized files. Network bandwidth and storage space are wasted if only a small part of the file is needed.
Block-level	Block	Storage space is saved. Suitable for diskless workstations.	For large files, there is a need to make multiple requests for accessing the same file. Increases network traffic.
Byte-level	Byte	Flexibility for any range of data storage and retrieval.	Cache management is difficult.
Record-level	Record	Easier to protect data. Ideal for database environment.	Increases network traffic in case large number of records have to be accessed.

Directory Service Interface :

- The directory service provides operations for creating & deleting directories, naming & renaming files, & moving across directories.
- It is independent of the file service implementation, where files are transferred in one piece or accessed remotely.
- In distributed file system, user can combine related files together by using directories & subdirectories.
- The file name consists of letters, numbers, & special characters.
- The file name has extension of maximum three characters such as doc, txt etc., which identify the type of file.

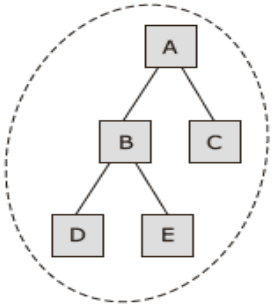


Figure 8-5 Hierarchical file system

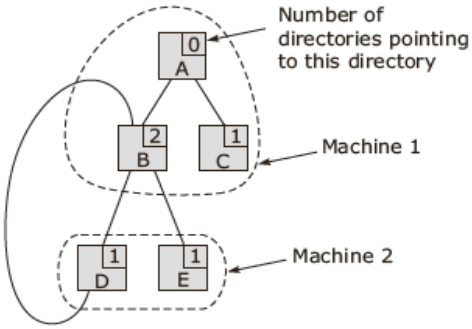


Figure 8-6 Linked directory structure

- Various operations are performed on directorysuch a
- Directory is divided into subdirectories & then subdirectories are furtherdivided into next level, leading to a tree structure called the *hierarchy file system.*
- In a distributed system, files on different machines can be linked together.
- Links or pointers to arbiter directories can be created, resulting in tree & directory graphs.
- There is distinct difference between tree & graphs in a distributed system.
- Fig 8-6 depicts various directories, A,B,C, which are linked either to B,C,D, or E.
- Apart from the hierarchical tree structured, directory D also links to directory B.
- In a tree graph, link A to link B can be removed if directory B is empty. But in a directed graph, link A to link B can be removed as long as any one link remains.
- The number of links can be tracked & maintained by a *reference count.*
- In centralized systems, all file activity is stopped, & the graph is traversed from the root to all the reachable directories.
- This problem is become critical in distributed systems because it is expensive to discover orphan directories which are distributed across machines.
- This is key issue in DFS- whether all the machines have the same view of the directory hierarchy.

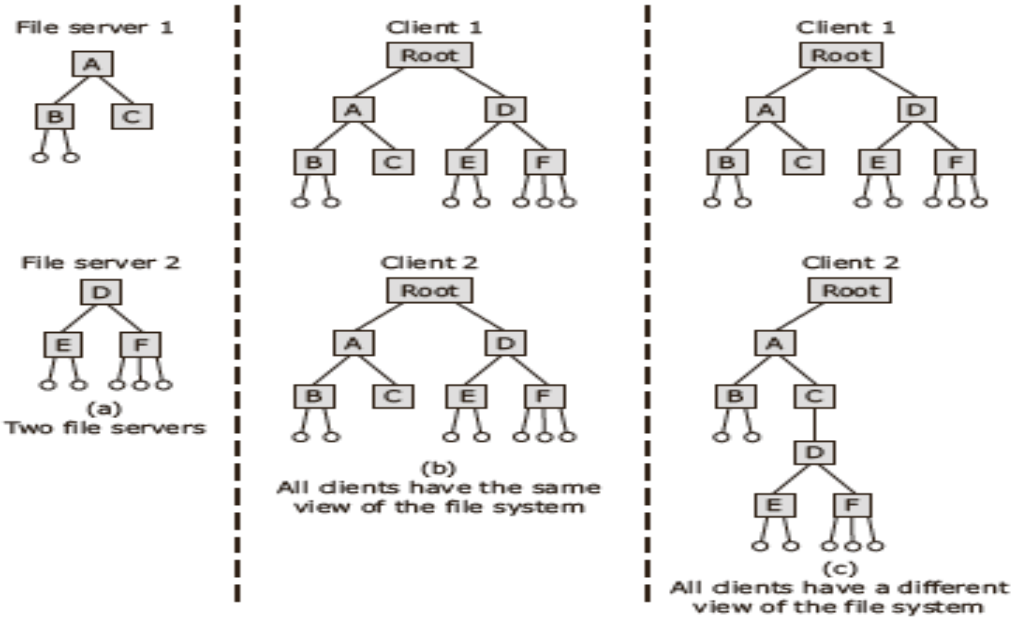


Figure 8-7 View of directory hierarchy

- Fig 8-7 (a), which consists of two servers, each holding the directories & some files.
- The squares are directories & the circles are files.
- Fig 8-7 (b) depicts all clients having the same view of the file system.
- It is easy to program & understand.
- Fig 8-7(c) represents a system where different clients have different view of the file system. Such a file system resides on machines that manage multiple file servers by remote mounting.
- This method is flexible, easy to implement, but does not behave like a time-sharing system.

Naming Transparency :

- The principle transparency issue related to naming is *location* transparency which implies that the file name does not indicate the location of file.
- Example is */server4/dir1/dir4/y* tells that the directory is located on server 4.
- The server can be moved without changing the path name. This cannot be done automatically because the first component of all path name is the server name.
- */server1/dir1/dir4/y* become new path name of the file y.
- If the file y is large & cannot be accommodate on server 4, it can be located on server 1 directly.
- The system in which files can be moved without changing the names are said to have location independence.

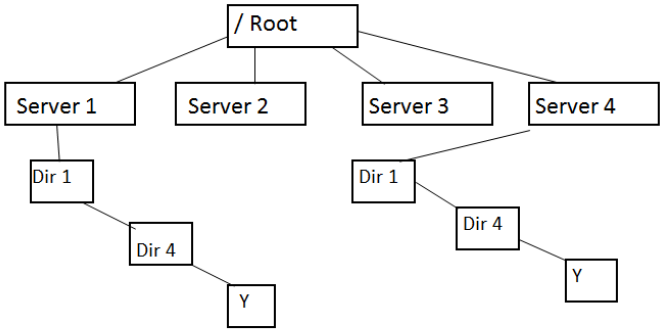


Fig 8.8 Naming Transparency

- There are three approaches to file & directory naming in a distributed system, *two-level naming,mounting remote file system on to a local file,* & a*single name specification* that looks the same on all machines.
- To achieve naming transparency, a two level naming scheme can be used, where level one is a symbolic name used by people

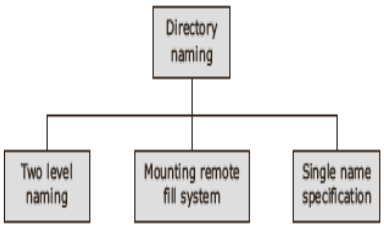
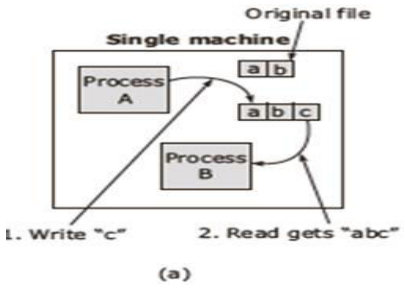


Figure 8-9 Approaches to directory naming

[By Nishigandha Dhore]

, where leveltwo is binary name is to be used by the systemitself.

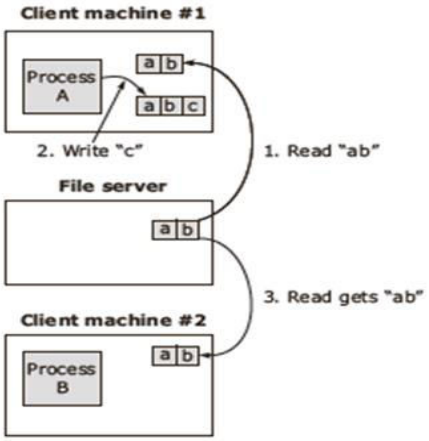
- Directories are used to provide a mapping between these two levels.
- When user references a symbolic name, the system maps it with the appropriate directory to get the binary name which is used to locate the file.
- The easier way to decide the binary name is that it should indicate the server & the specific file on the server.
- This scheme allows a directory on one server to hold the file which is stored on another server.
- In a distributed system, it is quite likely that looking-up ASCII names provide several binary names which are replicas and/or backups.
- From these names, any one file can be located.
- This method provides fault tolerance through redundancy.



(a)

Semantic of File Sharing :

- A shared file can be simultaneously accessed by multiple users.
- It is necessary to define semantics of read & write, i.e., when modifications made by the user to a file will be made visible to other users.
- Fig 8-10 (a)-In single machine, Read operation follows the Write operation, the Read returns the value just written.
- Sending a file sequentially to a single server is practically not desirable because of poor performance, poor scalability, & poor reliability.
- This problem can be solved by allowing clients to maintain local copies of frequently used,& large-sizedfiles in their cache.
- Fig 8-10 (b)- Client machine 1 modifies a local copy of a file & then close it.
- Shortly after client machine 2 reads the file from the server, it will get an obsolete copy from the server.



(b)

Types of file sharing semantics :

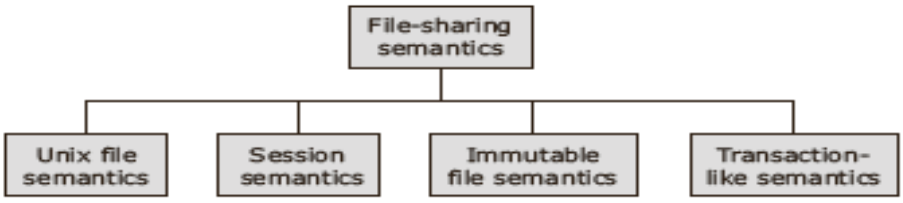


Figure 8-11 File sharing semantics

Unix File Semantics :

- In a single processor system, the file sharing semantics enforces absolute time ordering, & ensures that when a Read operation follows a Write operation, the Read always returns the value last written.
- This model is ideal for single processor systems because it is easy to serialize all Read/Write requests.
- In a distributed system, the UNIX semantics is achieved if it uses only one file server, & the clients do not cache files.
- The read or write requests from the client go directly to the server, & it processes them sequentially.
- Network delays may cause the Read that occurred after the Write to reach the server first, thus giving an older value to the server.
- This model is difficult to achieve in a distributed file system.

Session Semantics:

- A new rule can be used, namely changes made to an open file are visible only to the process which modified that file. Once the file is closed, the changes will be made visible to all the other processes.
- This does not alter the sequence of operations, & the behaviour of the file is redefined, & the subsequent reads get new values.
- This rule is known as *session semantics.*

- It is derived from the name *session,* which is a series of file accesses done between the open & close operations. Other open instances of the file do not reflect these changes.
- Using session semantics, multiple clients can perform both Read & Write operation concurrently on the same file.
- Each client works with an image of the file. When client closes the file, others are still accessing an older copy of the file.
- When a session is closed, the image is sent back to the server for update operation.
- Session semantics is used with file systems built using the file transfer model.

Immutable Shared-file Semantics:

- This model is based on immutable file model, with implies that a file can never be open for writing.
- If changes are to be made to the file, they are updated in the new version of the file which is treated as a new file.
- File sharing is allowed only in the Read mode.
- It is also possible to create a new file & save it along the same path with the same name. Now earlier file become inaccessible to the user.

Transaction-like Semantics :

- A transaction is asset of operations enclosed between the 'begin' & 'end' operations.
- Execution of transaction-like semantics ensures that partial changes are not visible to concurrent users until the entire transaction is complete, without any interference from other transactions.
- If more than one transaction starts at the same time, the system ensures that the final result is the same as if they were all executed in some sequential order.

Table 8-2 Features of file sharing semantics

Method	Feature
Unix semantics	Operations on files are instantaneously visible to all processes.
Session semantics	Changes are not visible to other processes till the file is closed.
Immutable files	Updates are not allowed. Simplifies sharing and replication.
Transaction-like semanties	Changes have the all-or-nothing property.

DFS Implementation :

Various structure related issues in DFS are:

DFS System Structure :

- In some systems, the machine can act either as a client or a server, depending on the service it offers or the service it receives.
- In other systems, there may be no distinction between client & servers. The file & directory server may also be user programs, so the system can be configured to run as either client or server software on the same machine.
- In a distributed environment , the mapping of a symbolic name to a binary name on a directory server & then accessing the file on a file server can be done on same machine or on a different machine..
- Keeping both on different machines is more flexible because both functions are unrelated, which makes the software simpler.
- On the other hand, two servers would definitely increase network traffic.
- In case of directory hierarchy is split on multiple servers, either an iterativelookup or an automatic lookup can be used.

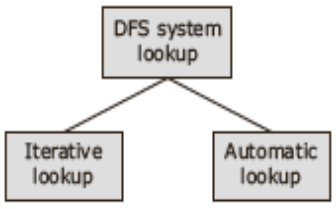


Figure 8-12 DFS system lookups

Iterative lookup method :

- The entries for the directory are current directory server 1, *a* on directory server 1 which point to entry *b* on server 2, pointing to *c* on server 3.
- This file contains an entry for *myfile.*
- To look up this entry, the client has to send messages in turn to server 1,2 & 3, & each message comprises of a set of reply-request messages.
- Access to the last server locates the file name.
- Then the file is sent to the client.

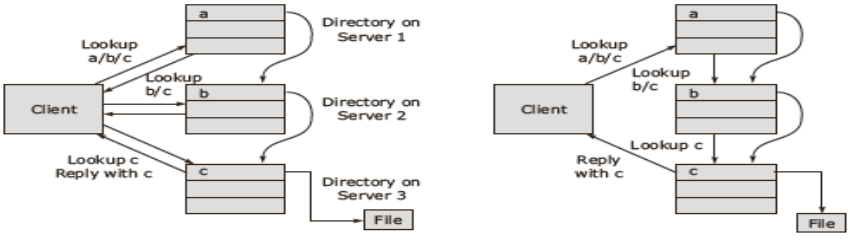


Figure 8-13 (a) Iterative lookup and (b) Automatic lookup

Automatic lookup method :

- The client sends request to the directory server 1, which forwards the request to server 2, which further sends request to server 3, from where it locates the file.
- Automatic lookup is more efficient as compared to iterative lookup, but it cannot be done using RPC because the request is made to server1, & the reply is obtained from server 3.
- As the number of directory servers is increased, the cost of file location also increases.
- To improve lookup performance, the binary names accessed earlier can be cached locally.

Stateful & stateless servers :

- The final structural issue is whether the directory & file servers should maintain the state information of clients.
- In stateful server, the state information of all clients is maintained on the server.
- Following sequence of operations when a file is opened :
- File server opens a file.
- Server maintains information about which client has which file open.
- Client is given a file ID for future references.
- Subsequent requests come with the file ID.
- Server uses a file ID to determine which file is requested.
- Maintain the state information table which maps file IDs to the file.
- Stateless servers do not maintain such state information. Each request must contain the full name of the file & offset within the file, so that server knows what to do.
- This increase the length of the message. In case of server crashes, the entire information is lost.
- When the server is rebooted, it is unaware of which clients have which files open.
- Stateful servers also have their own benefits. They use short length messages & hence utilize less network bandwidth.
- Performance is better because information about open files is available in the main memory.
- This decreased delay & duplicate requests on time out from the clients can be tracked from the state table.

Table 8-3 Relative advantages of stateful and stateless servers

Stateless servers	Stateful servers
Fault tolerance	Short request messages
No need for OPEN/CLOSE calls	Improved performance
No server space wasted on tables	Possible to read ahead
No limitation on the number of files open	Idempotency easier
No issues if the client crashes	Possible to lock files

Unit 8

Advances in Distributed Computing (SOA & Cloud Computing)

Service-Oriented Architecture, Elements of Service-Oriented Architectures, RPC versus Document Orientation, Major Benefits of Service- Oriented Computing, Composing Services, Goals of Composition, Challenges for Composition, Spirit of the Approach

Service-Oriented Architecture

- A service-oriented architecture is basically a collection of services that communicate with each other.
- The communication can involve either simple data passing or it could involve two or more services coordinating some specific activity.
- The first SOA for many people in the past was with the use of DCOM or Object Request Brokers(ORBs).
- A style of building reliable distributed systems
- SOA delivers functionalities as services emphasizing loose coupling between interacting services

Architecture:

- It is a formal description of system, defining its purpose, functions , externally visible properties and interfaces.
- It also includes the description of the system's internal components and their relationship, along with the principles governing itsdesign, operation and evolution.
- In fig 13-14, depicts a simple service interaction cycle, which starts with a service advertising itself through a well-known registry service(step 1).

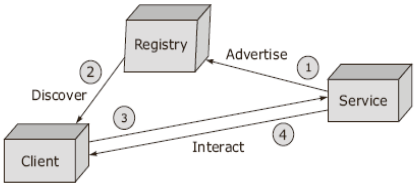


Figure 13-14 Service interaction cycle

- A potential client, who may or may not be another service, enquires with the registry(step 2) to search for a service that meets its needs.
- The registry returns a list of suitable service, and the client selects the most suitable one and passes a request message to it, using a mutually recognised protocol (step 3).
- In this eg, the service responds (step 4) either with the result of the requested operation or with a fault message.

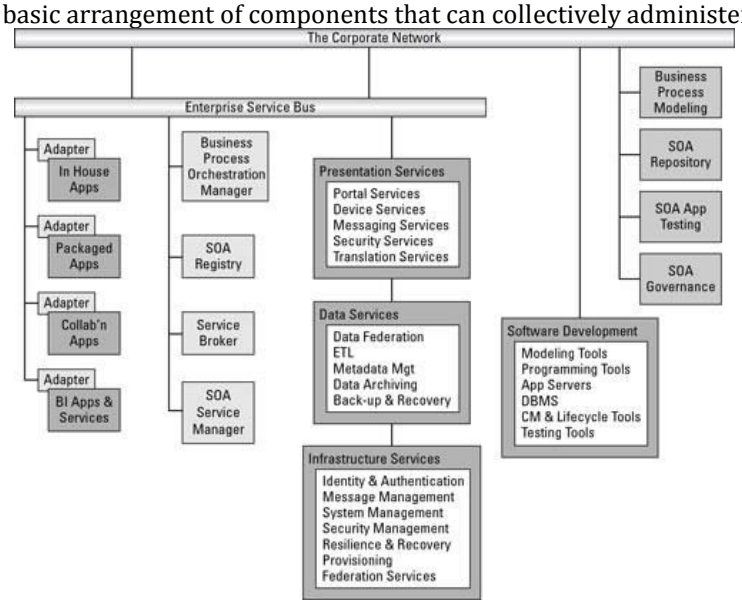
Elements of Service-Oriented Architectures

- All of the elements of service-oriented architecture (SOA) are arranged to connect through business processes to deliver a precise level of service.
- SOAs impose the following requirements:
- **Loose coupling:**
  - No tight transactional properties would generally apply among the components.
  - In general, it would not be appropriate to specify the consistency of data acrossthe information resources that are parts of the various components. However, it wouldbe reasonable to think of the high-level contractual relationships through which theinteractions among the components are specified.
- **Implementation neutrality:**
  - The interface is what matters. We cannot depend on the detailsof the implementations of the interacting components. In particular, the approach cannotbe specific to a set of programming languages.
- **Flexible configurability:**
  - The system is configured late and flexibly.
  - In other words, thedifferent components are bound to each other late in the process.
  - The configurationcan change dynamically.
- **Long lifetime:**
  - We do not necessarily advocate a long lifetime for our components.
  - However,since we are dealing with computations among autonomous heterogeneous parties indynamic environments, we must always be able to handle exceptions.
  - This means thatthe components must exist long enough to be able to detect any relevant exceptions, totake corrective action, and to respond to the corrective actions taken by others.
  - Componentsmust exist long enough to be discovered, to be relied upon, and to engendertrust in their behavior.
- **Granularity:**
  - The participants in an SOA should be understood at a coarse granularity.
  - Thatis, instead of modeling actions and interactions at a detailed level, it would be better tocapture the essential high-level qualities that are (or should be) visible for the purposesof business contracts among the participants.
  - Coarse granularity reduces dependenciesamong the participants and reduces communications to a few messages of greatersignificance.
- **Teams:**

- Instead of framing computations centrally, it would be better to think in terms ofhow computations are realized by autonomous parties.
- In other words, instead of aparticipant commanding its partners, computation in open systems is more a matter ofbusiness partners working as a team.
- That is, instead of an individual, a team of cooperatingparticipants is a better modeling unit.
- A team-oriented view is a consequenceof taking a peer-to-peer architecture seriously.

Components:

- SOA develops a basic arrangement of components that can collectively administer an intricate business service.



1. **Adapter**: A software module added to an application or system that allows access to its capabilities via a standards-compliant services interface.
2. **Business Process Modeling:** A procedure for mapping out what the business process does both in terms of what various applications are expected to do and what the human participants in the business process are expected to do.
3. **Enterprise Service Bus:** The enterprise service bus is the communications nerve center for services in service oriented architecture. It tends to be a jack-of all-trades, connecting to various types of middleware, repositories of metadata definitions (such as how you define a customer number), registries (how to locate information), and interfaces of every kind (for just about any application).
4. **Service Broker:** Software in a SOA framework that brings components together using the rules associated with each component.
5. **SOA Governance:** SOA governance is an element of overall IT governance and as such lays down the law when it comes to policy, process, and metadata management. (Metadata here simply means data that defines the source of the data, the owner of the data, and who can change the data.)
6. **SOA Repository:** A database for all SOA software and components, with an emphasis on revision control and configuration management, where they keep the good stuff, in other words.
7. **SOA Service Manager:** Software that orchestrates the SOA infrastructure — so that the business services can be supported and managed according to well-defined Service Level Agreements.
8. **SOA Registry:** A single source for all the metadata needed to utilize the Web service of a software component in a SOA environment.

RPC versus Document Orientation

- There are two main views of Web services. Services can be understood in terms of the*RPC-centric view* or the *document-centric view*.
- The former treats services as offering a setof methods to be invoked remotely, i.e., through remote procedure calls.
- The latter treatsservices as exchanging documents with one another.
- In both views, what is transmitted areXML documents and what is computed with are objects based on or corresponding to theXML documents.
- However, there is a significant conceptual difference.
- The RPC view sees the XML documents as incidental to the overall distributed computation.
- The documents are merely serializations of the business objects on which the maincomputation takes place.
- The document-centric view considers the documents as the mainrepresentations and purpose of the distributed computation.
- Each component reads, produces,stores, and transmits documents.
- The documents are temporarily materialized into businessobjects to enable computing, but the documents are the be all and end all of the computation.

- The RPC view thus corresponds to a thin veneer ofWeb services over an existing application.
- The application determines what functionality the services will support. The documentview more naturally considers Web services as a means of implementing business relationships.
- The documents to be processed (and their relationships) determine the functionality ofthe services.
- The business objects, such as there are, on either side of a relationship are local,and should not be exposed to the other side.
- For this reason, the document-centric view coheres better with our primary use case ofapplying services in open environments.
- The RPC view is more natural for the use case ofmaking independently developed applications interoperate.
- What happens is that applicationdevelopers expose their application interface in the form of Web services, which can then bebound to in the usual manner.
- If the applications are designed for method integration, thenthe RPC view of services is natural for such interoperation.
- However, if the applications aredesigned—as they should be—to function as independent components, then the document centricview would be natural even for application interoperation.

Major Benefits of Service- Oriented Computing

- Service-Oriented Computing (SOC) is a new computing paradigm that utilizes services as the basic constructs to support the development of rapid, low-cost and easy composition of distributed applications even in heterogeneous environments.
- It is worth considering the major benefits of using standardized services here.
- Clearly anythingthat can be done with services can be done without. So what are some reasons for usingservices, especially in standardized form? The following are the main reasons that stand out.
    - Services provide higher-level abstractions for organizing applications in large-scale,open environments. Even if these were not associated with standards, they wouldbe helpful as we implemented and configured software applications in a manner thatimproved our productivity and improved the quality of the applications that we developed.
    - Moreover, these abstractions are standardized. Standards enable the interoperation ofsoftware produced by different programmers. Standards thus improve our productivityfor the service use cases described above.
    - Standards make it possible to develop general-purpose tools to manage the entire systemlifecycle, including design, development, debugging, monitoring, and so on. Thisproves to be a major practical advantage, because without significant tool support, itwould be nearly impossible to create and field robust systems in a feasible manner.Such tools ensure that the components developed are indeed interoperable, because tool vendors can validate their tools and thus shift part of the burden of validation fromthe application programmer.
    - The standards feed other standards. For example the above basic standards enable further standards, e.g., dealing with processes and transactions.

Composing Services

- Although there can be some value in accessing a single service through a semantically wellfoundedinterface, the greater value is clearly derived through enabling a flexible *composition*of services.
- Composition leads to the creation of new services from old ones and can potentiallyadd much value beyond merely a nicer interface to a single pre-existing service. Thenew services can be thought of as *composite services*.
- Service composition concepts involve enough intricacy so as to attract considerable interestand to demand a careful analysis of the underlying principles.
- The need for principles isgreater as the basic infrastructure forWeb services becomes more common.
- We address theseprinciples herein.
- Sometimes, the term *composition* is taken to mean a particular approach toachieving composition, for example, by invoking a series of services.
- In the present usage,however, composition refers to any form of putting services together to achieve some desiredfunctionality.
- Composed Web services find application in a number of practical settings.
- For example,portals aggregate information from a number of sources and possibly offer programmaticfacilities for their intended audience.
- The challenge to making an effective portal is to be ableto personalize the information presented to each user.
- Electronic commerce is another majorscenario where users would like to aggregate product bundles to meet their specific needs.
- Virtual enterprises and supply-chain management reflect generalizations of the consumerorientede-commerce scenarios, because they include more subtle constraints among a largernumber of participants.

Goals of Composition

- Most of the applications touted for Web services are simple and straightforward client-serverinteractions.

- For example, an airline's flight-schedule database could interact directly with aPC user's personal information and appointment software to book a flight; or software for apersonal database of contacts could automatically query a distant series of phone databasesto add missing numbers to its list.
- This sort of a scenario is not far-fetched at all.
- Even in theearly days of Web service standards, Southwest Airlines and Dollar Rent-A-Car developed aprototype system that used SOAP to link South-west'sWeb site to Dollar's reservation system,so that airline customers could reserve a car along with their airline tickets [Metz, 2001].
- Although useful, such applications are insufficient to drive the strong development anddeployment ofWeb services.
- The fruitful, and also the more challenging, applications requireservices to be combined in ways that yield more powerful and novel uses.
- Service composition has been studied in the research literature for quite some time, butit is now becoming an important theme in practical Web system development.
- The basicidea behind service composition is simple.
- Web sites can be thought of as not only offeringcontent, but also providing services.
- For example, Yahoo! provides a news service and Amazonprovides a book selection service.
- We typically invoke these services by hand through aWeb browser, but a program could invoke them directly.
- Service composition on the Web isabout taking some existing services and building new customized services out of them.
- Forinstance, you might find the latest news headlines and search for books that match those headlines.
- Another example is where you might take the news from one service, filter it througha service that selects news based on a given user's interests, and pass the selected news itemsthrough a transcoding service to create a personalized Web page that a user could reviewthrough a handheld device.
- Or, more conventionally, you could create a travel service thatinvokes hotel, airline, and car rental services.
- In other words, you would create a workflowover the existing services.

Challenges for Composition

- The main advantage of Web services arises when we can compose them to create new services.
- Unfortunately, much of the attention on Web services has been focused on the lowerlevel,infrastructural matters, often down to encoding syntaxes and unnecessarily narrowmeans of invoking services. For Web services to be composed effectively requires an understandingof deeper concepts. These concepts have been developed in diverse parts of computerscience, especially heterogeneous databases, distributed computing, artificial intelligence,and multi-agent systems.
- Email is typically used for people to communicate with each other, so in using email, theserver is behaving like an intelligent agent.
- The current specifications forWeb services do not address transactions or specify a transactionmodel. The Organization for the Advancement of Structured Information Standards(OASIS) is developing one, but the view of most implementers is that SOAP will managetransactions—somehow. Without guidance from a standard or an agreed-upon methodologyby the major vendors, transactions will be implemented in an *ad hoc* fashion, thus defeatingthe hopes for interoperability and extensibility.
- Some of the other problems for composed services are
    - Security will be more difficult, because more participants will be involved and thenature of their interactions and their needs might be unanticipated by the designers ofthe services.
    - There will be incompatibilities in vocabularies, semantics, and pragmatics among theservice providers, service brokers, and service requesters.
    - As services are composed dynamically, performance problems might arise that werenot anticipated.
    - Dynamic service composition will make it difficult to guarantee the quality of service(QoS) that applications require.
- Two fundamental styles for delivering Web services are emerging, characterized as RPCstyle(favoured by Sun) and document-style (favoured by Microsoft, and supported by Sun).
- In the latter style, the body of a SOAP message would not have the call-response semanticsof most programming languages, but rather would consist of arbitrary XML documents thatuse WSDL to describe how a service works.
- In the long term the document-style is likely to prevail, because it is more declarative (rather than procedural), more asynchronous, and moreconsistent with the document-exchange underpinnings of the Web.

Spirit of the Approach

- Figure 2.1 shows the generic architecture for Web services.
- Although this is a simple picture, it radically alters many of the problems that must be solved in order for the architecture to become viable on a large scale.
    - To publish effectively, we must be able to specify services with precision and with greater structure. This is because the service would eventually be invoked by parties that are not from the same administrative space as the provider of the service and differences in assumptions about the semantics of the service could be devastating.

- From the perspective of the registry, it must be able to certify the given providers so that it can endorse the providers to the users of the registry.
- Requestors of services should be able to find a registry that they can trust. This opens up challenges dealing with considerations of trust, reputation, incentives for registries and, most importantly, for the registry to understand the needs of a requestor.

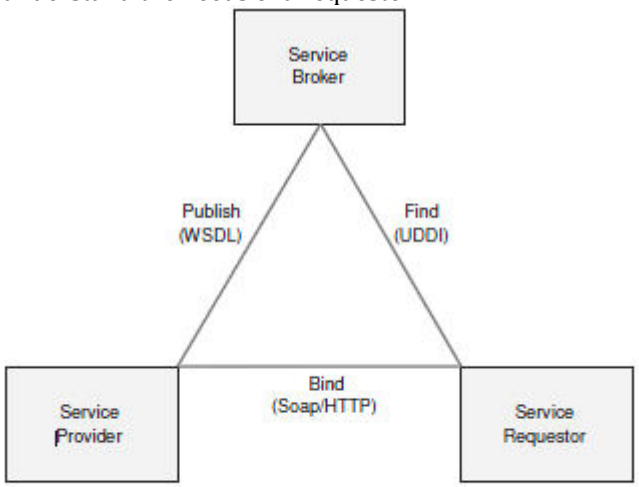


Figure 2.1: The general architectural model for Web services

- Once a service has been selected, the requestor and the provider must develop a finergrainedsharing of representations. They must be able to participate in conversations toconduct long-lived, flexible transactions. Related questions are those of how a servicelevel agreement (SLA) can be established and monitored. Success or failure with SLAsfeeds into how a service is published and found, and how the reputation of a provideris developed and maintained.

- The keys to the next-generation Web are *cooperative services, systemic trust, and understandingbased on semantics, coupled with a declarative agent-based infrastructure*.
- The size and dynamism of theWeb presents problems, but it fortuitously provides a meansfor solving its own problems.
- For example, for a given topic there might be an overloadof information, with much of it redundant and some of it inaccurate, but a system can usevoting techniques to reduce the information to that which is consistent and agreed upon.
- For another example, there might be many potential service providers competing for manypotential clients, and some of the providers might not be trustworthy, but a system can use aWeb-based reputation network to assess credibility.
- Finally, different sites might use differentontologies, but a multiplicity of ontologies can yield global, dynamically formed, consensusontology.
- The subsequent chapters of this book address each of these concepts.

Unit 9

Fundamentals of Cloud computing

Evolution of Cloud Computing ,cluster computing, Grid computing, Grid computing versus Cloud Computing, Key Characteristics of cloud computing

Evolution of Cloud Computing

- The trend toward cloud computing started in the late 1980s with the concept of grid computing when, for the first time, a large number of systems were applied to a single problem, usually scientific in nature and requiring exceptionally high levels of parallel computation.
- In Europe, long distance optical networks are used to tie multiple universities into a massive computing grid in order that resources could be shared and scaled for large scientific calculations.
- Grid computing provided a virtual pool of computation resources but it's different than cloud computing.
- Grid computing specifically refers to leveraging several computers in parallel to solve a particular, individual problem, or to run a specific application.
- Cloud computing, on the other hand, refers to leveraging multiple resources, including computing resources, to deliver a unified "service" to the end user.
- In grid computing, the focus is on moving a workload to the location of the needed computing resources, which are mostly remote and are readily available for use.
- Usually a grid is a cluster of servers on which a large task could be divided into smaller tasks to run in parallel.
- From this point of view, a grid could actually be viewed as just one virtual server. Grids also require applications to conform to the grid software interfaces.
- In a cloud environment, computing and extended IT and business resources, such as servers, storage, network, applications and processes, can be dynamically shaped or carved out from the underlying hardware infrastructure and made available to a workload.
- In addition, while a cloud can provision and support a grid, a cloud can also support non-grid environments, such as a three-tier Web architecture running traditional or Web 2.0 applications
- In the 1990s, the concept of virtualization was expanded beyond virtual servers to higher levels of abstraction—first the virtual platform, including storage and network resources, and subsequently the virtual application, which has no specific underlying infrastructure.
- Utility computing offered clusters as virtual platforms for computing with a metered business model.
- More recently software as a service (SaaS) has raised the level of virtualization to the application, with a business model of charging not by the resources consumed but by the value of the application to subscribers.
- The concept of cloud computing has evolved from the concepts of grid, utility and SaaS.
- It is an emerging model through which users can gain access to their applications from anywhere, at any time, through their connected devices.
- These applications reside in massively scalable data centers where compute resources can be dynamically provisioned and shared to achieve significant economies of scale.
- The strength of a cloud is its infrastructure management, enabled by the maturity and progress of virtualization technology to manage and better utilize the underlying resources through automatic provisioning, re-imaging, workload rebalancing, monitoring, systematic change request handling and a dynamic and automated security and resiliency platform.
- As more enterprises add cloud computing the level of applications is migrating toward more missions critical and SaaS will become a mainstay of IT strategies.
- This ability of SaaS to deliver expensive applications at affordable will continue to accelerate.

Cluster computing

- Network clustering connects otherwise independent computers to work together in some coordinated fashion.
- Because clustering is a term used broadly, the hardware configuration of clusters varies substantially depending on the networking technologies chosen and the purpose (the so-called "computational mission") of the system.
- Clustering hardware comes in three basic flavours: so-called "shared disk," "mirrored disk,"and "shared nothing" configurations.

Shared Disk Clusters

- One approach to clustering utilizes central I/O devices accessible to all computers ("nodes") within the cluster.
- We call these systems *shared-disk clusters* as the I/O involved is typically disk storage for normal files and/or databases.
- Shared-disk cluster technologies include Oracle Parallel Server (OPS)and IBM's HACMP.
- Shared-disk clusters rely on a common I/O bus for disk access but do not require shared memory.
- Because all nodes may concurrently write to or cache data from the central disks, a synchronization mechanism must be used to preserve coherence of the system.
- An independent piece of cluster software called the "distributed lock manager" assumes this role.
- Shared-disk clusters support higher levels of system availability: if one node fails, other nodes need not be affected.

- However, higher availability comes at a cost of somewhat reduced performance in these systems because of overhead in using a lock manager and the potential bottlenecks of shared hardware generally.
- Shared-disk clusters make up for this shortcoming with relatively good scaling properties: OPS and HACMP support eight-node systems, for example.

Shared Nothing Clusters

- A second approach to clustering is dubbed *shared-nothing* because it does not involve concurrent disk accesses from multiple nodes. (In other words, these clusters do not require a distributed lock manager.)
- Shared-nothing cluster solutions include Microsoft Cluster Server (MSCS).
- MSCS is an atypical example of a shared nothing cluster in several ways.
- MSCS clusters use a shared SCSI connection between the nodes,that naturally leads some people to believe this is a shared-disk solution.
- But only one server (the one that owns the quorum resource) needs the disks at any given time, so no concurrent data access occurs.
- MSCS clusters also typically include only two nodes, whereas shared nothing clusters in general can scale to hundreds of nodes.

Mirrored Disk Clusters

- *Mirrored-disk cluster* solutions include Legato's Vinca.
- Mirroring involves replicating all application data from primary storage to a secondary backup (perhaps at a remote location) for availability purposes.
- Replication occurs while the primary system is active, although the mirrored backup system -- as in the case of Vinca -- typically does not perform any work outside of its role as a passive standby.
- If a failure occurs in the primary system, a failover process transfers control to the secondary system.
- Failover can take some time, and applications can lose state information when they are reset, but mirroring enables a fairly fast recovery scheme requiring little operator intervention.
- Mirrored-disk clusters typically include just two nodes.

Grid computing

- Grid computing is the collection of computer resources from multiple locations to reach a common goal.
- The grid can be thought of as distributed system with non-interactive workloads that involve a large number of files.
- Grid computing is distinguished from conventional high performance computing systems such as cluster computing in that grid computers have each node set to perform a different task/application.
- Grid computers also tend to be more heterogeneous and geographically dispersed (thus not physically coupled) than cluster computers.
- Although a single grid can be dedicated to a particular application, commonly a grid is used for a variety of purposes.
- Grids are often constructed with general-purpose grid middleware software libraries.
- Grid size varies a considerable amount. Grids are a form of distributed computing whereby a “super virtual computer” is composed of many networked loosely coupled computers acting together to perform large tasks.
- For certain applications, “distributed” or “grid” computing, can be seen as a special type of parallel computing that relies on complete computers (with onboard CPUs, storage, power supplies, network interfaces, etc.) connected to a network (private, public or the Internet) by a conventional network interface, such as Ethernet.
- This is in contrast to the traditional notion of a supercomputer, which has many processors connected by a local high-speed bus.
- Grid computing combines computers from multiple administrative domains to reach **a common goal**, **to solve a single task**, and may then disappear just as quickly.
- One of the main strategies of grid computing is to use middleware to divide and apportion pieces of a program among several computers, sometimes up to many thousands.
- Grid computing involves computation in a distributed fashion, which may also involve the aggregation of large-scale clusters.
- The size of a grid may vary from small—confined to a network of computer workstations within a corporation, for example—to large, public collaborations across many companies and networks.
- "The notion of a confined grid may also be known as an intra-nodes cooperation whilst the notion of a larger, wider grid may thus refer to an inter-nodes cooperation".
- Grids are a form of distributed computing whereby a “super virtual computer” is composed of many networked loosely coupled computers acting together to perform very large tasks.
- This technology has been applied to computationally intensive scientific, mathematical, and academic problems through volunteer computing, and it is used in commercial enterprises for such diverse applications as drug discovery, economic forecasting, seismic analysis, and back office data processing in support for e-commerce and Web services.
- Coordinating applications on Grids can be a complex task, especially when coordinating the flow of information across distributed computing resources.

[By Nishigandha Dhore]

- Grid workflow systems have been developed as a specialized form of a workflow management system designed specifically to compose and execute a series of computational or data manipulation steps, or a workflow, in the Grid context.

Comparison of grids and conventional supercomputers

- "Distributed" or "grid" computing in general is a special type of parallel computing that relies on complete computers (with onboard CPUs, storage, power supplies, network interfaces, etc.) connected to a network (private, public or the Internet) by a conventional network interface producing commodity hardware, compared to the lower efficiency of designing and constructing a small number of custom supercomputers.
- The primary performance disadvantage is that the various processors and local storage areas do not have high-speed connections.
- This arrangement is thus well-suited to applications in which multiple parallel computations can take place independently, without the need to communicate intermediate results between processors.
- The high-end scalability of geographically dispersed grids is generally favorable, due to the low need for connectivity between nodes relative to the capacity of the public Internet.
- There are also some differences in programming and deployment.
- It can be costly and difficult to write programs that can run in the environment of a supercomputer, which may have a custom operating system, or require the program to address concurrency issues.
- If a problem can be adequately parallelized, a "thin" layer of "grid" infrastructure can allow conventional, standalone programs, given a different part of the same problem, to run on multiple machines.
- This makes it possible to write and debug on a single conventional machine, and eliminates complications due to multiple instances of the same program running in the same shared memory and storage space at the same time.

Design considerations and variations of grid computing

- One feature of distributed grids is that they can be formed from computing resources belonging to multiple individuals or organizations (known as multiple administrative domains).
- This can facilitate commercial transactions, as in utility computing, or make it easier to assemble volunteer computing networks.
- One disadvantage of this feature is that the computers which are actually performing the calculations might not be entirely trustworthy. The designers of the system must thus introduce measures to prevent malfunctions or malicious participants from producing false, misleading, or erroneous results, and from using the system as an attack vector. This often involves assigning work randomly to different nodes (presumably with different owners) and checking that at least two different nodes report the same answer for a given work unit. Discrepancies would identify malfunctioning and malicious nodes.
- Due to the lack of central control over the hardware, there is no way to guarantee that nodes will not drop out of the network at random times. Some nodes (like laptops or dialup Internet customers) may also be available for computation but not network communications for unpredictable periods. These variations can be accommodated by assigning large work units (thus reducing the need for continuous network connectivity) and reassigning work units when a given node fails to report its results in expected time.
- The impacts of trust and availability on performance and development difficulty can influence the choice of whether to deploy onto a dedicated cluster, to idle machines internal to the developing organization, or to an open external network of volunteers or contractors. In many cases, the participating nodes must trust the central system not to abuse the access that is being granted, by interfering with the operation of other programs, mangling stored information, transmitting private data, or creating new security holes. Other systems employ measures to reduce the amount of trust "client" nodes must place in the central system such as placing applications in virtual machines.
- Public systems or those crossing administrative domains (including different departments in the same organization) often result in the need to run on heterogeneous systems, using different operating systems and hardware architectures. With many languages, there is a tradeoff between investment in software development and the number of platforms that can be supported (and thus the size of the resulting network). Cross-platform languages can reduce the need to make this trade off, though potentially at the expense of high performance on any given node (due to run-time interpretation or lack of optimization for the particular platform). There are diverse scientific and commercial projects to harness a particular associated grid or for the purpose of setting up new grids. BOINC is a common one for various academic projects seeking public volunteers; more are listed at the end of the article.
- In fact, the middleware can be seen as a layer between the hardware and the software. On top of the middleware, a number of technical areas have to be considered, and these may or may not be middleware independent. Example areas include SLA management, Trust and Security, Virtual organization management, License Management,

Portals and Data Management. These technical areas may be taken care of in a commercial solution, though the cutting edge of each area is often found within specific research projects examining the field.

Market segmentation of the grid computing market

- For the segmentation of the grid computing market, two perspectives need to be considered: the provider side and the user side:

The provider side

- The overall grid market comprises several specific markets. These are the grid middleware market, the market for grid-enabled applications, the utility computing market, and the software-as-a-service (SaaS) market.
- Grid middleware is a specific software product, which enables the sharing of heterogeneous resources, and Virtual Organizations. It is installed and integrated into the existing infrastructure of the involved company or companies, and provides a special layer placed among the heterogeneous infrastructure and the specific user applications. Major grid middlewares are Globus Toolkit, gLite, and UNICORE.
- Utility computing is referred to as the provision of grid computing and applications as service either as an open grid utility or as a hosting solution for one organization or a VO. Major players in the utility computing market are Sun Microsystems, IBM, and HP.
- Grid-enabled applications are specific software applications that can utilize grid infrastructure. This is made possible by the use of grid middleware, as pointed out above.
- Software as a service (SaaS) is "software that is owned, delivered and managed remotely by one or more providers." (Gartner 2007) Additionally, SaaS applications are based on a single set of common code and data definitions. They are consumed in a one-to-many model, and SaaS uses a Pay As You Go (PAYG) model or a subscription model that is based on usage. Providers of SaaS do not necessarily own the computing resources themselves, which are required to run their SaaS. Therefore, SaaS providers may draw upon the utility computing market. The utility computing market provides computing resources for SaaS providers.

The user side:

- For companies on the demand or user side of the grid computing market, the different segments have significant implications for their IT deployment strategy. The IT deployment strategy as well as the type of IT investments made are relevant aspects for potential grid users and play an important role for grid adoption.

Grid computing versus Cloud Computing

	Grid computing	**Cloud computing**
What?	Grids enable access to **shared** computing power and storage capacity from your desktop	Clouds enable access to **leased** computing power and storage capacity from your desktop
Who provides the service?	Research institutes and universities federate their services around the world through projects such as EGI-Inspire and the European Grid Infrastructure.	Large individual companies e.g. Amazon and Microsoft and at a smaller scale, institutes and organizations deploying open source software such as Open Slate, Eucalyptus and Open Nebula.
Who uses the service?	Research collaborations, called "Virtual Organizations", which bring together researchers around the world working in the same field.	Small to medium commercial businesses or researchers with generic IT needs
Who pays for the service?	Governments - providers and users are usually publicly funded research organizations, for example through National Grid Initiatives.	The cloud provider pays for the computing resources; the user pays to use them
Where are the computing resources?	In computing centres distributed across different sites, countries and continents.	The cloud providers private data centres which are often centralized in a few locations with excellent network connections and cheap electrical power.
Why use them?	You don`t need to buy or maintain your own large computer centre. You can complete more work more quickly and tackle more difficult problems. You can share data with your distributed team in a secure way.	You don`t need to buy or maintain your own personal computer centre - You can quickly access extra resources during peak work periods

[By Nishigandha Dhore]

What are they useful for?	Grids were designed to handle large sets of limited duration jobs that produce or use large quantities of data (e.g. the LHC and life sciences)	Clouds best support long term services and longer running jobs (E.g. facebook.com)
How do they work?	Grids are an **open source** technology. Resource users and providers alike can understand and contribute to the management of their grid	Clouds are a **proprietary** technology. Only the resource provider knows exactly how their cloud manages data, job queues, security requirements and so on.
Benefits?	- **Collaboration:** grid offers a federated platform for distributed and collective work. - **Ownership:** resource providers maintain ownership of the resources they contribute to the grid - **Transparency:** the technologies used are open source, encouraging trust and transparency. - **Resilience:** grids are located at multiple sites, reducing the risk in case of a failure at one site that removes significant resources from the infrastructure.	- **Flexibility:** users can quickly outsource peaks of activity without long term commitment - **Reliability:** provider has financial incentive to guarantee service availability (Amazon, for example, can provide user rebates if availability drops below 99.9%) - **Ease of use:** relatively quick and easy for non-expert users to get started but setting up sophisticated virtual machines to support complex applications is more difficult.
Drawbacks?	- **Reliability:** grids rely on distributed services maintained by distributed staff, often resulting in inconsistency in reliability across individual sites, although the service itself is always available. - **Complexity:** grids are complicated to build and use, and currently users require some level of expertise. - **Commercial:** grids are generally only available for not-for-profit work, and for proof of concept in the commercial sphere	- **Generality:** clouds do not offer many of the specific high-level services currently provided by grid technology. - **Security:** users with sensitive data may be reluctant to entrust it to external providers or to providers outside their borders. - **Opacity:** the technologies used to guarantee reliability and safety of cloud operations are not made public. - **Rigidity:** the cloud is generally located at a single site, which increases risk of complete cloud failure. - **Provider lock-in:** there's a risk of being locked in to services provided by a very small group of suppliers.
When?	The concept of grids was proposed in 1995. The Open science grid (OSG) started in 1995 The EDG (European Data Grid) project began in 2001.	In the late 1990`s Oracle and EMC offered early private cloud solutions . However the term cloud computing didn't gain prominence until 2007.

Key Characteristics of cloud computing

- There are several key characteristics of a cloud computing environment.
- Service offerings are most often made available to specific consumers and small businesses that see the benefit of use because their capital expenditure is minimized.
- This serves to lower barriers to entry in the marketplace, since the infrastructure used to provide these offerings is owned by the cloud service provider and need not be purchased by the customer.
- Because users are not tied to a specific device (they need only the ability to access the Inter-net) and because the Internet allows for location independence, use of the cloud enables cloud computing service providers' customers to access cloud-enabled systems regardless of where they may be located or what device they choose to use.
- Multi-tenancy enables sharing of resources and costs among a large pool of users.
- Chief benefits to a multi-tenancy approach include:
    - Centralization of infrastructure and lower costs
    - Increased peak-load capacity
    - Efficiency improvements for systems that are often underutilized
    - Dynamic allocation of CPU, storage, and network bandwidth
    - Consistent performance that is monitored by the provider of theservice

[By Nishigandha Dhore]

- Reliability is often enhanced in cloud computing environments because service providers utilize multiple redundant sites.
- This is attractive to enterprises for business continuity and disaster recovery reasons.
- The drawback, however, is that IT managers can do very little when an outage occurs.
- Another benefit that makes cloud services more reliable is that scalability can vary dynamically based on changing user demands.
- Because the service provider manages the necessary infrastructure, security often is vastly improved.
- As a result of data centralization, there is an increased focus on protecting customer resources maintained by the service provider.

Unit 10

Cloud models

Benefits of Cloud models, Public Cloud, Private Cloud, Hybrid Cloud, Community Cloud, Shared Private Cloud, Dedicated Private Cloud, Dynamic Private Cloud, Savings and cost impact

Web services delivered from cloud, Platform as a service, Software as a service, and Infrastructure as a service.

Benefits of Cloud models:

1. **Achieve economies of scale** – increase volume output or productivity with fewer people. Your cost per unit, project or product plummets.
2. **Reduce spending on technology infrastructure.**-Maintain easy access to your information with minimal upfront spending. Pay as you go (weekly, quarterly or yearly), based on demand.
3. **Globalize your workforce on the cheap**-People worldwide can access the cloud, provided they have an Internet connection.
4. **Streamline processes-**Get more work done in less time with less people.
5. **Reduce capital costs-**There's no need to spend big money on hardware, software or licensing fees.
6. **Improve accessibility-**You have access anytime, anywhere, making your life so much easier!
7. **Monitor projects more effectively-**Stay within budget and ahead of completion cycle times.
8. **Less personnel training is needed-**It takes fewer people to do more work on a cloud, with a minimal learning curve on hardware and software issues.
9. **Minimize licensing new software-**Stretch and grow without the need to buy expensive software licenses or programs.
10. **Improve flexibility-**You can change direction without serious "people" or "financial" issues at stake.

Public Cloud

- Public cloud services are offered by third-party datacentre provider to end-user consumers over the internet.
- Public cloud offers resources pooling, self-service, service accounting, elasticity, multi-tenancy to manage the solutions, deployment and securing the resources and applications.
- Companies can use it on-demand and with the pay-as-you-use option, it is much like utilize consumption.
- Enterprises are able to offload commodity applications to third-party service providers
- The term 'public' does not mean that:
    - It is free, even though it can be free or fairly inexpensive to use.
    - That a user's data is publically visible-public cloud vendors typically provide an access control mechanism for their users.

Private Cloud

- Private clouds are deployments made inside the company's firewall and traditionally run on-site servers.
- Private clouds offer some of the benefits of a public cloud computing environment, such as elastic on-demand capacity, self-service provisioning, and service-based access.
- Private cloud is suitable when he traditional requirements, such as control, security and resiliency are more emphasized by an organization with the restricted and designated user access and authorization.
- Services in private cloud are:
    - Virtualization
    - Government and management
    - Multi-tenancy
    - Consistent deployment
    - Security and access control.
- Services consumed from a public cloud are:
    - Security and data privacy
    - Ease of access
    - Discovery of services.
    - Restful interface support
    - Lower cost.
- The features and benefits of private clouds therefore are:
    - Higher security and privacy
    - More control
    - Cost and energy efficiency
    - Improved reliability

Hybrid Cloud

- A Hybrid cloud is a combination of an interoperating public and private cloud.
- This is the model when consumer takes the non-critical application or information and compute requirements to the public cloud while keeping all the critical information and application data in control.
- The hybrid model is used by both public and private cloud simultaneously.
- It is an intermediate step in the evolution process, providing business an on-ramp from their current IT environment into the cloud.

- It offers the best of both cloud worlds-the scale and convenience of a public cloud and the control and reliability of on-premises software and infrastructure- and let them move fluidly between the two basis of their needs.
- This model allows the following:
    - Elasticity, which is the ability to scale capacity up or down within minutes.
    - Pay-as-you-go pricing.
    - Network isolation and secure connectivity as if all the resources were in a privately owned datacentre.
    - Gradually move to the public cloud configuration, replicate an entire datacentre or move anywhere in between.

Community Cloud

- This is the cloud managed by groups of people, communities and agencies especially government to have the common interest such as maintaining the compliance, regulation, and security parameters working on same mission.
- The members of the community share access to the data and application in the cloud.

Shared Private Cloud

- This is a shared compute capacity with variable usage-based pricing to business units that are based on service offerings, account datacenters.
- It is required as internal profitcenter to take over or buy infrastructure made available through account consolidations.

Dedicated Private Cloud

- Dedicated private cloud has IT service Catalog with dynamic provisioning.
- It depends on standardized service-oriented architecture (SOA) architectural assets that can be broadly deployed into new and existing accounts and is a lower cost model.

Dynamic Private Cloud

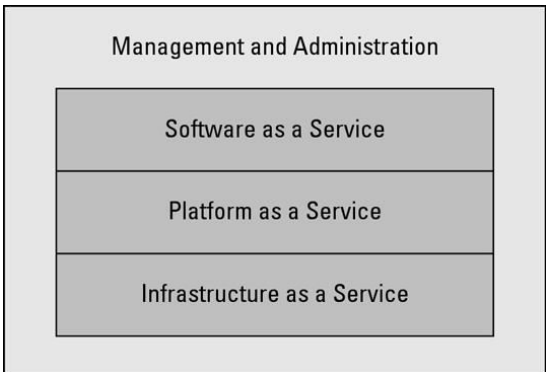
- Dynamic private cloud allows client workloads to dynamically migrates to and from the compute cloud as needed.
- This model can be shared and dedicated.
- It delivers on the ultimate value of clouds.
- This is very low-management model with reliable SLAs and scalability.

Savings and cost impact

- The use of virtualization in cloud computing consolidates systems, which drives reduction in hardware costs.
- This is often the initial appeal of funding virtualization projects.
- Saving on manual efforts are also bigger.
- Today enterprises are involving human resources to provision the compute resources, which require longer cycles sometimes even week, hence they suffer cost.
- In this way, highly skilled resources are doing the administration work and not focusing on important higher value jobs.
- This can be avoided by the automation of the tasks that are highly repetitive, enhancing agile and improving quality.
- Cloud computing features two delivery models, namely private and public.
- Private cloud exists behind the firewall, while public cloud computing is accessed through internet.
- Cloud vendors believe these 3 models traditional IT, private cloud services and public cloud services can co-exist as the part of an overall strategy, based on the application type and the business need that would dictate which model to use.
- Hybrid cloud services, delivered to the end-user, are composed of both private and public cloud computing elements.

Web services delivered from cloud

- Web service offerings often have a number of common characteristics, such as a low barrier to entry, where services are offered specifically for consumers and small business entities.
- Often, little or no capital expenditure for infrastructure is required from the customer.
- While massive scalability is common with these types of offerings, it not always necessary. Many cloud vendors have yet to achieve massive scalability because their user base generally does not require it.



Platform as a service

- Cloud computing has evolved to include platforms for building and running custom web-based applications, a concept known as Platform-as-a-Service.
- PaaS is an outgrowth of the SaaS application delivery model.
- ThePaaS model makes all of the facilities required to support the complete lifecycle of building and delivering web applications and services entirely available from the Internet, all with no software downloads or installationfor developers, IT managers, or end users.
- Unlike the IaaS model, wheredevelopers may create a specific operating system instance with home-grown applications running, PaaS developers are concerned only with web-based development and generally do not care what operating system issued.
- PaaS services allow users to focus on innovation rather than complex infrastructure.
- Organizations can redirect a significant portion of theirbudgets to creating applications that provide real business value instead of worrying about all the infrastructure issues in a roll-your-own delivery model.
- The PaaS model is thus driving a new era of mass innovation.
- Now,developers around the world can access unlimited computing power.
- Any-one with an Internet connection can build powerful applications and easily deploy them to users globally.
- PaaS offerings may also include facilities for application design, application development, testing, and deployment as well as services such as team collaboration, web service integration, and marshalling, database integration, security, scalability, storage, persistence, state management, application versioning, application instrumentation, and developercommunity facilitation.
- Platform as a Service allows users to create software applications using tools supplied by the provider.
- PaaS services can consist of preconfigured features that customers can subscribe to; they can choose to include the features that meet their requirements while discarding those that do not.
- Consequently, packages can vary from offering simple point-and-click frameworks where no client side hosting expertise is required to supplying the infrastructure options for advanced development.
- Below are some of the features that can be included with a PaaS offering:
    - Operating system
    - Server-side scripting environment
    - Database management system
    - Server Software
    - Support
    - Storage
    - Network access
    - Tools for design and development
    - Hosting

Below are some of the benefits of PaaS to application developers:

- **They don't have to invest in physical infrastructure;** being able to 'rent' virtual infrastructure has both cost benefits and practical benefits. They don't need to purchase hardware themselves or employ the expertise to manage it.
- **Makes development possible for 'non-experts';** with some PaaS offerings anyone can develop an application. They can simply do this through their web browser utilising one-click functionality. Salient examples of this are one-click blog software installs such as Word Press.
- **Flexibility;** customers can have control over the tools that are installed within their platforms and can create a platform that suits their specific requirements. They can 'pick and choose' the features they feel are necessary.
- **Adaptability;** Features can be changed if circumstances dictate that they should.
- **Teams in various locations can work together;** as an internet connection and web browser are all that is required, developers spread across several locations can work together on the same application build.
- **Security;** security is provided, including data security and backup and recovery.

Software as a service

- The traditional model of software distribution, in which software is purchased for and installed on personal computers, is sometimes referred to asSoftware-as-a-Product.
- Software-as-a-Service is a software distributionmodel in which applications are hosted by a vendor or service provider andmade available to customers over a network, typically the Internet.
- SaaS isbecoming an increasingly prevalent delivery model as underlying technologies that support web services and service-oriented architecture (SOA)mature and new developmental approaches become popular.
- SaaS is alsooften associated with a pay-as-you-go subscription licensing model.
- Mean- while, broadband service has become increasingly available to support useraccess from more areas around the world.
- **Software as a service** (**SaaS**) is a software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted.It is sometimes referred to as "on-demand software".
- SaaS is typically accessed by users using a thin client via a web browser.

[By Nishigandha Dhore]

- SaaS has become a common delivery model for many business applications, including office & messaging software, DBMS software, management software, CAD software, development software, gamification, virtualization, accounting, collaboration, customer relationship management (CRM), management information systems (MIS), enterprise resource planning (ERP), invoicing, human resource management (HRM), content management (CM) and service desk management.
- SaaS is most often implemented to provide business software functionality to enterprise customers at a low cost while allowing those customers to obtain the same benefits of commercially licensed, internally operated soft-ware without the associated complexity of installation, management, sup-port, licensing, and high initial cost.

SaaS Implementation Issues

- Many types of software components and applications frameworks may beemployed in the development of SaaS applications.
- Using new technology found in these modern components and application frameworks can drastically reduce the time to market and cost of converting a traditional on-premises product into a SaaS solution.
- According to Microsoft,SaaSarchitectures can be classified into one of four maturity levels whose key attributes are ease of configuration, multitenant efficiency, and scalability.

Key Characteristics of SaaS:

- Deploying applications in a service-oriented architecture is a more complex problem than is usually encountered in traditional models of software deployment.
- The key characteristics are:
    - Network-based management and access to commercially available software from central locations rather than at each customer's site, enabling customers to access applications remotely via the Inter-net.
    - Application delivery from a one-to-many model (single-instance, multitenant architecture), as opposed to a traditional one-to-one model.
    - Centralized enhancement and patch updating that obviates any need for downloading and installing by a user. SaaS is often used in conjunction with a larger network of communications and collaboration software, sometimes as a plug-in to PaaS architecture.

Benefits of SaaS model:

- SaaS also helps to increase the availability of applications to global locations.
- SaaS also ensures that all application transactions are logged for compliance purposes.
- The benefits of SaaS to the customer are very clear:
    - Streamlined administration
    - Automated update and patch management services
    - Data compatibility across the enterprise (all users have the same version of software)
    - Facilitated, enterprise-wide collaboration
    - Global accessibility

Infrastructure as a service.

- According to the online reference Wikipedia, Infrastructure-as-a-Service(IaaS) is the delivery of computer infrastructure (typically a platform virtualization environment) as a service.
- IaaS leverages significant technology,services, and data center investments to deliver IT as a service to customers.
- Unlike traditional outsourcing, which requires extensive due diligence,negotiations ad infinitum, and complex, lengthy contract vehicles, IaaS iscentered around a model of service delivery that provisions a predefined,standardized infrastructure specifically optimized for the customer's applications.
- Simplified statements of work and à la carte service-level choicesmake it easy to tailor a solution to a customer's specific application requirements.
- IaaS providers manage the transition and hosting of selected applications on their infrastructure.
- Customers maintain ownership and management of their application(s) while off-loading hosting operationsand infrastructure management to the IaaS provider.

The following are salient examples of how IaaS can be utilized by enterprise:

- **Enterprise infrastructure;** by internal business networks, such as private clouds and virtual local area networks, which utilize pooled server and networking resources and in which a business can store their data and run the applications they need to operate day-to-day. Expanding businesses can scale their infrastructure in accordance with their growth whilst private clouds (accessible only by the business itself) can protect the storage and transfer of the sensitive data that some businesses are required to handle.
- **Cloud hosting;** the hosting of websites on virtual servers which are founded upon pooled resources from underlying physical servers. A website hosted in the cloud, for example, can benefit from the redundancy provided

by a vast network of physical servers and on demand scalability to deal with unexpected demands placed on the website.

- **Virtual Data Centers (VDC);** a virtualized network of interconnected virtual servers which can be used to offer enhanced cloud hosting capabilities, enterprise IT infrastructure or to integrate all of these operations within either a private or public cloud implementation.

A typical Infrastructure as a Service offering can deliver the following features and benefits:

- **Scalability;** resource is available as and when the client needs it and, therefore, there are no delays in expanding capacity or the wastage of unused capacity
- **No investment in hardware**; the underlying physical hardware that supports an IaaS service is set up and maintained by the cloud provider, saving the time and cost of doing so on the client side
- **Utility style costing;** the service can be accessed on demand and the client only pays for the resource that they actually use
- **Location independence;** the service can usually be accessed from any location as long as there is an internet connection and the security protocol of the cloud allows it
- **Physical security of data centre locations;** services available through a public cloud, or private clouds hosted externally with the cloud provider, benefit from the physical security afforded to the servers which are hosted within a data centre
- **No single point of failure;** if one server or network switch, for example, were to fail, the broader service would be unaffected due to the remaining multitude of hardware resources and redundancy configurations. For many services if one entire data center were to go offline, never mind one server, the IaaS service could still run successfully.

There are three types of IaaS Cloud offerings:

1. **Public IaaS Cloud**: In the public cloud, provider(s) rent out hardware resources in a multi-tenant method to the general public using virtualization technology. This allows multiple users to share server resources. The public cloud is a prime example of the cloud computing model: easy to set up, highly scalable and elastic, where users will only pay for the resources that they use.

2. **Private IaaS Cloud**: The private cloud employs virtualization technology and delivers cloud-computing services to a single organization. The services are provisioned privately and sit behind the firewalls managed by the individual business. Servers and resources are specifically dedicated to the individual business, and cannot be used by others. The private cloud is best suited for businesses that have large CAPEX budgets and rely on their own data center professionals and security experts, who are needed to provide more control over the computing environment(s).

3. **Hybrid IaaS Cloud**: A hybrid cloud is generally considered the conjoining of physical and virtual infrastructure in a public or private cloud. For example, a company may opt to manage some physical servers in a private cloud, while outsourcing other servers to a public cloud. The hybrid cloud allows companies to take advantage of scalability with cloud technologies, all while managing sensitive company data or applications not suitable or licensable in the cloud.

Unit 11

Cloud Security Fundamentals

Privacy and security in cloud, Security architecture, Data security, Identity and access management, security challenges

Privacy and security in cloud

- A risk assessment and gap analysis of controls and procedures must beconducted.
- Based on this data, formal privacy processes and initiativesmust be defined, managed, and sustained.
- As with security, privacy controls and protection must an element of the secure architecture design.
- Depending on the size of the organization and the scale of operations,either an individual or a team should be assigned and given responsibility for maintaining privacy.
- A member of the security team who is responsible for privacy or a corporate security compliance team should collaborate with the company legal team to address data privacy issues and concerns.
- As with security, a privacy steering committee should also be created to help make decisionsrelated to data privacy.
- Typically, the security compliance team, if one evenexists, will not have formalized training on data privacy, which will limitthe ability of the organization to address adequately the data privacy issuesthey currently face and will be continually challenged on in the future.
- The answer is to hire a consultant in this area, hire a privacy expert, orhave one of your existing team members trained properly.
- This will ensurethat your organization is prepared to meet the data privacy demands of itscustomers and regulators.

Security architecture

- A security architecture framework should be established with considerationof processes (enterprise authentication and authorization, access control,confidentiality, integrity, nonrepudiation, security management, etc.), operational procedures, technology specifications, people and organizationalmanagement, and security program compliance and reporting.
- A security architecture document should be developed that defines security and privacy principles to meet business objectives.
- Documentation is required formanagement controls and metrics specific to asset classification and control,physical security, system access controls, network and computer management, application development and maintenance, business continuity, andcompliance.
- A design and implementation program should also be integrated with the formal system development life cycle to include a businesscase, requirements definition, design, and implementation plans.
- Technology and design methods should be included, as well as the security processesnecessary to provide the following services across all technology layers:
    1. Authentication
    2. Authorization
    3. Availability
    4. Confidentiality
    5. Integrity
    6. Accountability
    7. Privacy
- The creation of a secure architecture provides the engineers, data centeroperations personnel, and network operations personnel a common blue-print to design, build, and test the security of the applications and systems.
- Design reviews of new changes can be better assessed against this architecture to assure that they conform to the principles described in the architecture, allowing for more consistent and effective design reviews.

Data security

- The ultimate challenge in cloud computing is data level security, and sensitive data is the domain of the enterprise, not the cloud computing provider.
- Security will need to move to the data level so that enterprises can besure their data is protected wherever it goes.
- For example, with data-levelsecurity, the enterprise can specify that this data is not allowed to go out-side of the United States.
- It can also force encryption of certain types of data, and permit only specified users to access the data.
- It can provide compliance with the Payment Card Industry Data Security Standard (PCIDSS).
- True unified end-to-end security in the cloud will likely require anecosystem of partners.

Identity and access management

- Identity and access management is a criticalfunction for every organization, and a fundamental expectation of SaaScustomers is that the principle of least privilege is granted to their data.
- The principle of least privilege states that only the minimum access necessary to perform an operation should be granted, and that access should begranted only for the minimum amount of time necessary.
- However,business and IT groups will need and expect access to systems and applications.
- The advent of cloud services and services on demand is changing the identity management landscape.
- Most of the current identity managementsolutions are focused on the enterprise and typically are architected to work in a very controlled, static environment.
- User-centric identity management solutions such as federated identity managementalso make some assumptions about the parties involved andtheir related services.
- In the cloud environment, where services are offered on demand andthey can continuously evolve, aspects of current models such as trustassumptions, privacy implications, and operational aspects of authentication and authorization, will be challenged.
- Meeting these challenges willrequire a balancing act for SaaS providers as they evaluate new models andmanagement processes for IAM to provide end-to-end trust and identity throughout the cloud and the enterprise.
- Another issue will be finding theright balance between usability and security.
- If a good balance is notachieved, both business and IT groups may be affected by barriers to completing their support and maintenance activities efficiently

Security challenges

- With the cloud model, you lose control over physical security. In a public cloud, you are sharing computing resources with other companies.
- In a shared pool outside the enterprise, you don’t have any knowledge or controlof where the resources run.
- Exposing your data in an environment shared with other companies could give the government “reasonable cause” to seizeyour assets because another company has violated the law.
- Simply becauseyou share the environment in the cloud, may put your data at risk of seizure.
- Storage services provided by one cloud vendor may be incompatible with another vendor’s services should you decide to move from one to theother.
- Vendors are known for creating what the hosting world calls “sticky services”—services that an end user may have difficulty transporting fromone cloud vendor to another.
- Data integrity means ensuring that data is identically maintained during any operation (such as transfer, storage, or retrieval).
- Put simply, data integrity is assurance that the data is consistent and correct.
- Ensuring theintegrity of the data really means that it changes only in response to authorized transactions.
- This sounds good, but you must remember that a common standard to ensure data integrity does not yet exist.
- Using SaaS offerings in the cloud means that there is much less need forsoftware development.

- Cloud applications undergo constant feature additions, and users mustkeep up to date with application improvements to be sure they are protected.
- The speed at which applications will change in the cloud will affectboth the SDLC and security.
- Even worse, a secure SLDC will not be able toprovide a security cycle that keeps up with changes that occur so quickly.
- This means that users must constantly upgrade, because an older versionmay not function, or protect the data.
- Security needs to move to the data level, so that enterprises can be sure theirdata is protected wherever it goes.
- Sensitive data is the domain of the enter-prise, not the cloud computing provider.
- One of the key challenges in cloudcomputing is data-level security.
- Outsourcing means losing significant control over data, and while thisisn't a good idea from a security perspective, the business ease and financialsavings will continue to increase the usage of these services.
- In the rush to take advantage of the benefits of cloud computing, notleast of which is significant cost savings, many corporations are likely rushing into cloud computing without a serious consideration of the security implications.

Unit 12

Implementation of Cloud Technologies

Introduction to Cloud Technologies, Hypervisor, Web services, AJAX , MASHUP, Hadoop, Map reduce, Virtualization Technologies, Virtual Machine Technology Cloud data centre, Case studies : Google, Microsoft, Amazon

Introduction to Cloud Technologies

Hypervisor:

- A **hypervisor** or virtual machine monitor (VMM) is a piece of computer software, firmware or hardware that creates and runs virtual machines.
- A computer on which a **hypervisor** is running one or more virtual machines is defined as a host machine.
- Each virtual machine is called a guest machine.
- A computer on which a hypervisor is running one or more virtual machines is defined as a *host machine*.
- Each virtual machine is called a *guest machine*.
- The hypervisor presents the guest operating systems with a virtual operating platform and manages the execution of the guest operating systems.
- Multiple instances of a variety of operating systems may share the virtualized hardware resources.

Classification-:

Hypervisor types

In their 1974 article "Formal Requirements for Virtualizable Third Generation Architectures" Gerald J. Popek and Robert P. Goldberg classified two types of hypervisor:

- ***Type 1*** (or *native*, bare metal) hypervisors run directly on the host's hardware to control the hardware and to manage guest operating systems. A guest operating-system thus runs on another level above the hypervisor. This model represents the classic implementation of virtual-machine architectures; IBM developed the original hypervisors as bare-metal tools in the 1960s: the test tool SIMMON, and CP/CMS. CP/CMS was the ancestor of IBM's z/VM. Modern equivalents include Oracle VM Server for SPARC, Oracle VM Server for x86, the Citrix XenServer, VMware ESX/ESXi and Microsoft Hyper-V 2008/2012.
- ***Type 2*** (or *hosted*) hypervisors run within a conventional operating-system environment. With the hypervisor layer as a distinct second software level, guest operating-systems run at the third level above the hardware. VMware Workstation and Virtual Box exemplify Type 2 hypervisors.

Web services????(AWT)

AJAX

- **Ajax** short for **asynchronous JavaScript + XML**) is a group of interrelated Web development techniques used on the client-side to create asynchronousWeb applications.
- With Ajax, Web applications can send data to and retrieve from a server asynchronously (in the background) without interfering with the display and behavior of the existing page. Data can be retrieved using the XMLHttpRequestobject.
- Despite the name, the use of XML is not required; JSON is often used instead (AJAJ), and the requests do not need to be asynchronous.
- Ajax is not a single technology, but a group of technologies.
- HTML and CSS can be used in combination to mark up and style information.
- The DOM is accessed with JavaScript to dynamically display – and allow the user to interact with – the information presented.
- JavaScript and the XMLHttpRequest object provide a method for exchanging data asynchronously between browser and server to avoid full page reloads.

Technologies

- The term *Ajax* has come to represent a broad group of Web technologies that can be used to implement a Web application that communicates with a server in the background, without interfering with the current state of the page.
- In the article that coined the term Ajax,[3] Jesse James Garrett explained that the following technologies are incorporated:
  - HTML (or XHTML) and CSS for presentation
  - The Document Object Model (DOM) for dynamic display of and interaction with data
  - XML for the interchange of data, and XSLT for its manipulation
  - The XMLHttpRequest object for asynchronous communication
  - JavaScript to bring these technologies together.
- Since then, however, there have been a number of developments in the technologies used in an Ajax application, and the definition of the term Ajax.
- XML is not required for data interchange and, therefore, XSLT is not required for the manipulation of data.

- JavaScript Object Notation (JSON) is often used as an alternative format for data interchange, although other formats such as preformatted HTML or plain text can also be used.
- Asynchronous HTML and HTTP (AHAH) involves using XMLHTTP Request to retrieve (X)HTML fragments, which are then inserted directly into the Web page.

Drawbacks

- Dynamic Web page updates also make it difficult to bookmark and return to a particular state of the application.
- Depending on the nature of the Ajax application, dynamic page updates may interfere disruptively with user interactions, especially if working on an unstable Internet connection.
- Most Web crawlers do not execute JavaScript code, so in order to be indexed by search engines, a Web application must provide an alternative means of accessing the content that would normally be retrieved with Ajax.
- Any user whose browser does not support JavaScript or XMLHttpRequest, or simply has this functionality disabled, will not be able to properly use pages which depend on Ajax.
- Similarly, some Web applications that use Ajax are built in a way that cannot be read by screen-reading technologies.
- Screen readers that are able to use Ajax may still not be able to properly read the dynamically generated content.

MASHUP

- A **mashup**, in web development, is a web page, or web application, that uses content from more than one source to create a single new service displayed in a single graphical interface.
- For example, you could combine the addresses and photographs of your library branches with a Google map to create a map mashup.
- The term implies easy, fast integration, frequently using open application programming interfaces (open API) and data sources to produce enriched results that were not necessarily the original reason for producing the raw source data.
- The term mashup originally comes from British - West Indies slang meaning to be intoxicated, or as a description for something or someone not functioning as intended.
- In recent English parlance it can refer to music, where people seamlessly combine audio from one song with the vocal track from another—thereby mashing them together to create something new.
- The main characteristics of a mashup are combination, visualization, and aggregation.
- It is important to make existing data more useful, for personal and professional use.
- To be able to permanently access the data of other services, mashups are generally client applications or hosted online.
- In the past years, more and more Web applications have published APIs that enable software developers to easily integrate data and functions the SOA way, instead of building them by themselves.
- Mashups can be considered to have an active role in the evolution of social software and Web 2.0.
- Mashup composition tools are usually simple enough to be used by end-users.
- They generally do not require programming skills and rather support visual wiring of GUI widgets, services and components together.
- Therefore, these tools contribute to a new vision of the Web, where users are able to contribute.

Types of mashups:
There are many types of mashup, such as business mashups, consumer mashups, and data mashups.
The most common type of mashup is the consumer mashup, aimed at the general public.

- ***Business* (or *enterprise*) *mashups*** define applications that combine their own resources, application and data, with other external Web services.They focus data into a single presentation and allow for collaborative action among businesses and developers.
- ***Consumer mashups*** combine data from multiple public sources in the browser and organize it through a simple browser user interface.
- ***Data mashups*,** opposite to the consumer mashups, combine similar types of media and information from multiple sources into a single representation. The combination of all these resources creates a new and distinct Web service that was not originally provided by either source.

Hadoop

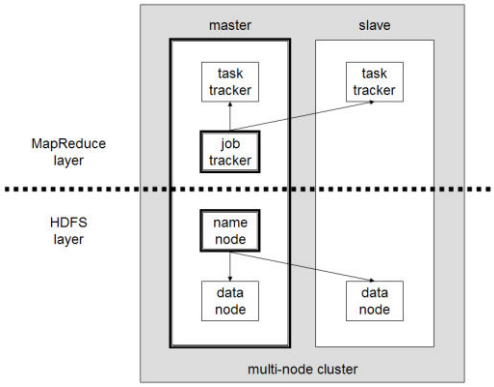
- **Hadoop** is an open-sourcesoftware framework for distributed storage and distributed processing of Big Data on clusters of commodity hardware.
- Its Hadoop Distributed File System (HDFS) splits files into large blocks (default 64MB or 128MB) and distributes the blocks amongst the nodes in the cluster.
- For processing the data, the Hadoop Map/Reduce ships code (specifically Jar files) to the nodes that have the required data, and the nodes then process the data in parallel.
- This approach takes advantage of data locality, in contrast to conventional HPC architecture which usually relies on a parallel file system (compute and data separated, but connected with high-speed networking).

- Since 2012, the term "Hadoop" often refers not to just the base Hadoop package but rather to the **Hadoop Ecosystem**, which includes all of the additional software packages that can be installed on top of or alongside Hadoop, such as Apache Hive, Apache Pig and Apache Spark.
- The base Apache Hadoop framework is composed of the following modules:
  - *Hadoop Common* – contains libraries and utilities needed by other Hadoop modules.
  - *Hadoop Distributed File System (HDFS)* – a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster.
  - *Hadoop YARN* – a resource-management platform responsible for managing compute resources in clusters and using them for scheduling of users' applications.
  - *Hadoop MapReduce* – a programming model for large scale data processing.

- All the modules in Hadoop are designed with a fundamental assumption that hardware failures (of individual machines, or racks of machines) are common and thus should be automatically handled in software by the framework.
- Apache Hadoop's MapReduce and HDFS components originally derived respectively from Google's MapReduce and Google File System (GFS) papers.

Architecture

- Hadoop consists of the *Hadoop Common* package, which provides file system and OS level abstractions, a Map Reduce engine (either Map Reduce/MR1 or YARN/MR2) and the Hadoop Distributed File System (HDFS).
- The Hadoop Common package contains the necessary Java ARchive (JAR) files and scripts needed to start Hadoop.
- The package also provides source code, documentation, and a contribution section that includes projects from the Hadoop Community.
- For effective scheduling of work, every Hadoop-compatible file system should provide location awareness: the name of the rack (more precisely, of the network switch) where a worker node is.



- Hadoop applications can use this information to run work on the node where the data is, and, failing that, on the same rack/switch, reducing backbone traffic.
- HDFS uses this method when replicating data to try to keep different copies of the data on different racks.
- The goal is to reduce the impact of a rack power outage or switch failure, so that even if these events occur, the data may still be readable.

A multi-node Hadoop cluster

- A small Hadoop cluster includes a single master and multiple worker nodes.
- The master node consists of a JobTracker, TaskTracker, NameNode and DataNode.
- A slave or *worker node* acts as both a DataNode and TaskTracker, though it is possible to have data-only worker nodes and compute-only worker nodes.
- These are normally used only in nonstandard applications.
- Hadoop requires Java Runtime Environment (JRE) 1.6 or higher.
- The standard startup and shutdown scripts require that Secure Shell (ssh) be set up between nodes in the cluster.
- In a larger cluster, the HDFS is managed through a dedicated NameNode server to host the file system index, and a secondary NameNode that can generate snapshots of the namenode's memory structures, thus preventing file-system corruption and reducing loss of data.

Map reduce

- **MapReduce** is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster.
- A MapReduce program is composed of a **Map()**procedure that performs filtering and sorting (such as sorting students by first name into queues, one queue for each name) and a **Reduce()** procedure that performs a summary operation (such as counting the number of students in each queue, yielding name frequencies).
- The "MapReduce System" (also called "infrastructure" or "framework") orchestrates the processing by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance.

- The model is inspired by the map and reduce functions commonly used in functional programming, although their purpose in the MapReduce framework is not the same as in their original forms.

[By Nishigandha Dhore]

- The key contributions of the MapReduce framework are not the actual map and reduce functions, but the scalability and fault-tolerance achieved for a variety of applications by optimizing the execution engine once.
- As such, a single-threaded implementation of MapReduce (such as MongoDB) will usually not be faster than a traditional (non-MapReduce) implementation, any gains are usually only seen with multi-threaded implementations.
- Only when the optimized distributed shuffle operation (which reduces network communication cost) and fault tolerance features of the MapReduce framework come into play, is the use of this model beneficial.
- MapReduce libraries have been written in many programming languages, with different levels of optimization.
- A popular open-source implementation is Apache Hadoop.
- The name MapReduce originally referred to the proprietary Google technology, but has since been genericized.
- MapReduce is a framework for processing parallelizable problems across huge datasets using a large number of computers (nodes), collectively referred to as a cluster (if all nodes are on the same local network and use similar hardware) or a grid (if the nodes are shared across geographically and administratively distributed systems, and use more heterogenous hardware).
- Processing can occur on data stored either in a filesystem (unstructured) or in a database (structured).
- MapReduce can take advantage of locality of data, processing it on or near the storage assets in order to reduce the distance over which it must be transmitted.
    - **"Map" step:** Each worker node applies the "map()" function to the local data, and writes the output to a temporary storage. A master node orchestrates that for redundant copies of input data, only one is processed.
    - **"Shuffle" step:** Worker nodes redistribute data based on the output keys (produced by the "map()" function), such that all data belonging to one key is located on the same worker node.
    - **"Reduce" step:** Worker nodes now process each group of output data, per key, in parallel.
- MapReduce allows for distributed processing of the map and reduction operations.
- Provided that each mapping operation is independent of the others, all maps can be performed in parallel – though in practice this is limited by the number of independent data sources and/or the number of CPUs near each source.
- Similarly, a set of 'reducers' can perform the reduction phase, provided that all outputs of the map operation that share the same key are presented to the same reducer at the same time, or that the reduction function is associative.

Virtualization Technologies

- **Virtualization**, in computing, refers to the act of creating a virtual (rather than actual) version of something, including but not limited to a virtual computer hardware platform, operating system (OS), storage device, or computer network resources.
- Virtualization began in 1960s mainframe computers as a method of logically dividing the mainframes' resources for different applications. Since then, the meaning of the term has broadened.
- *Hardware virtualization* or *platform virtualization* refers to the creation of a virtual machine that acts like a real computer with an operating system.
- Software executed on these virtual machines is separated from the underlying hardware resources.
- For example, a computer that is running Microsoft Windows may host a virtual machine that looks like a computer with the Ubuntu Linux operating system; Ubuntu-based software can be run on the virtual machine.
- In hardware virtualization, the *host machine* is the actual machine on which the virtualization takes place, and the *guest machine* is the virtual machine.
- The words *host* and *guest* are used to distinguish the software that runs on the physical machine from the software that runs on the virtual machine.
- The software or firmware that creates a virtual machine on the host hardware is called a *hypervisor* or *Virtual Machine Manager*.

Different types of hardware virtualization include:

1. **Full virtualization:** Almost complete simulation of the actual hardware to allow software, which typically consists of a guest operating system, to run unmodified.
2. **Partial virtualization:** Some but not all of the target environment is simulated. Some guest programs, therefore, may need modifications to run in this virtual environment.
3. **Paravirtualization:** A hardware environment is not simulated; however, the guest programs are executed in their own isolated domains, as if they are running on a separate system. Guest programs need to be specifically modified to run in this environment.

- Hardware-assisted virtualization is a way of improving the efficiency of hardware virtualization.
- It involves employing specially designed CPUs and hardware components that help improve the performance of a guest environment.
- The usual goal of virtualization is to centralize administrative tasks while improving scalability and overall hardware-resource utilization.
- With virtualization, several operating systems can be run in parallel on a single central processing unit (CPU).

- This parallelism tends to reduce overhead costs and differs from multitasking, which involves running several programs on the same OS.
- Using virtualization, an enterprise can better manage updates and rapid changes to the operating system and applications without disrupting the user.
- "Ultimately, virtualization dramatically improves the efficiency and availability of resources and applications in an organization.
- Instead of relying on the old model of "one server, one application" that leads to underutilized resources, virtual resources are dynamically applied to meet business needs without any excess fat" (ConsonusTech).
- Hardware virtualization is not the same as hardware emulation.
- In hardware emulation, a piece of hardware imitates another, while in hardware virtualization, a hypervisor (a piece of software) imitates a particular piece of computer hardware or the entire computer.
- Furthermore, a hypervisor is not the same as an emulator; both are computer programs that imitate hardware, but their domain of use in language differs.

Virtual Machine Technology Cloud data centre

Case studies :

Google

- The cloud is certainly one of Google's biggest business ventures, and they offer a couple of tools to help draw customers to their cloud. In this section, we'll talk about what Google offers.

Google App Engine

- Google App Engine enables developers to build their web apps on the same infrastructurethat powers Google's own applications.

Features

Leveraging Google App Engine, developers can accomplish the following tasks:

- **Write code once and deploy** Provisioning and configuring multiple machines forweb serving and data storage can be expensive and time-consuming. Google AppEngine makes it easier to deploy web applications by dynamically providingcomputing resources as they are needed. Developers write the code, and GoogleApp Engine takes care of the rest.
- **Absorb spikes in traffic** When a web app surges in popularity, the suddenincrease in traffic can be overwhelming for applications of all sizes, from startups tolarge companies that find themselves re-architecting their databases and entiresystems several times a year. With automatic replication and load balancing, GoogleApp Engine makes it easier to scale from one user to one million by takingadvantage of Bigtable and other components of Google's scalable infrastructure.
- **Easily integrate with other Google services** It's unnecessary and inefficient fordevelopers to write components like authentication and email from scratch for eachnew application. Developers using Google App Engine can make use of built-incomponents and Google's broader library of APIs that provide plug-and-playfunctionality for simple but important features.

Google Web Toolkit

- With Google Web Toolkit, developers can develop and debug web applications in thefamiliar Java programming language, and then deploy them as highly optimized JavaScript.
- In doing so, developers sidestep common AJAX headaches like browser compatibility and enjoy significant performance and productivity gains. Google Health is one recentlylaunched application to use Google Web Toolkit.
- Google Web Toolkit includes Java 5 language support so that developers can enjoyusing the full capabilities of the Java 5 syntax.
- These capabilities include Java generics,enumerated types, annotations, auto-boxing, variable parameter lists, and more.
- Thecompiler in Google Web Toolkit 1.5 produces faster code than ever, delivering performancegains big enough for end users to notice.
- Indeed, often the compiler produces fasterJavaScript than a person would write by hand in JavaScript.
- Google Web Toolkit 1.5accomplishes this by performing deep inlining, better dead-code elimination, and otherforms of enhanced static analysis.
- Google Web Toolkit also continues to provide a rich and growing set of libraries thathelp developers build world-class AJAX, including thoroughly tested, reusable librariesfor implementing user interfaces, data structures, client/server communication,internationalization, testing, and accessibility.

Microsoft

- Microsoft offers a number of cloud services for organizations of any size—from enterprisesall the way down to mom-and-pop shops or individuals. A good portion of Microsoft'scloud offerings are cloud variants of products that people already use, so cloud versionsaren't that difficult to use.

Azure Services Platform

- The cornerstone of Microsoft's offerings is the Azure Services Platform.
- The Azure ServicesPlatform is a cloud computing and services platform hosted in Microsoft datacenters.
- The Azure Services Platform supplies a broad range of functionality to build applicationsto serve individuals or large enterprises, and everyone in between.
- The platform offers acloud operating system and developer tools. Applications can be developed with industrystandard protocols like REST and SOAP.
- Azure services can be used individually or in conjunction with one another to buildnew applications or to enhance existing ones. Let's take a closer look at the Azure Services

Platform components.

Windows Azure

- Windows Azure is a cloud-based operating system that enables the development, hosting,and service management environment for the Azure Services Platform.
- To build applications and services, developers can use the Visual Studio skills they alreadyhave. Further, Azure supports existing standards like SOAP, REST, and XML.
- Windows Azure can be used to
    - Add web service capabilities to existing applications
    - Build and modify applications and then move them onto the Web
    - Make, test, debug, and distribute web services efficiently and inexpensively
    - Reduce the costs of IT management

SQL Services

- Microsoft SQL Services extends SQL Server capabilities to the cloud as web-based services.
- This allows the storage of structured, semistructured, and unstructured data. SQL Servicesdelivers a set of integrated services that allow relational queries, search, reporting, analytics,integration, and synchronization of data. This can be done by mobile users, remote offices, or business partners.
- .NET Services
- Microsoft .NET Services are a set of Microsoft-hosted, developer-oriented services thatprovide the components required by many cloud-based and cloud-aware applications.
- .NET Services are similar to the .NET Framework, providing high-level class librariesthat make development much more robust.

Windows Live

- Windows Live is an integrated set of online services that makes it easier and more fun forconsumers to communicate and share with others.
- The new generation of Windows Liveincludes updated experiences for photo sharing, email, and instant messaging, as well asintegration with multiple third-party sites.
- The release also includes Windows LiveEssentials, free downloadable software that enhances consumers' Windows experience byhelping them simplify and enjoy digital content scattered across their PC, phone, and onweb sites.

SharePoint Services

- Microsoft offers its SharePoint Services to aid collaboration efforts.
- SharePoint Servicesprovides communities for team collaboration and makes it easy for users to work togetheron documents, tasks, contacts, events, and other information.
- Additionally, team and sitemanagers can coordinate site contents and user activity.
- SharePoint sites are made up of Web Parts and Windows ASP.NET-based components.
- Web Parts are designed to be add-ons to web pages and configured by site administratorsand users to create complete page-based applications.

Microsoft Dynamics CRM

- Microsoft Dynamics CRM Online is an on-demand customer relationship managementservice hosted and managed by Microsoft. The Internet service delivers a full suite ofmarketing, sales, and service capabilities through a web browser or directly into MicrosoftOffice and Outlook. It provides "instant-on" access to businesses that want a full-featuredCRM solution with no IT infrastructure investment or setup required.

Amazon

- Amazon may be the most widely known cloud vendor.
- They offer services on many differentfronts, from storage to platform to databases. Amazon seems to have their finger in a numberof cloud technologies.

Amazon Elastic Compute Cloud (Amazon EC2)

- Amazon Elastic Compute Cloud (Amazon EC2) is a web service that offers resizablecompute capacity in the cloud and is designed to make web scaling easier for developers.

- Amazon EC2 provides a simple web interface that allows you to obtain and configurecapacity with little difficulty. It allows you control of your computing resources.
- AmazonEC2 cuts the time it takes to obtain and boot new server instances to a few minutes,allowing you to change scale as your needs change.

Amazon SimpleDB

- For database services, Amazon offers its Amazon SimpleDB.
- It provides core databasefunctions of data indexing and querying.
- This service works closely with Amazon SimpleStorage Service (Amazon S3) and Amazon EC2. This provides the ability to store, process,and query data sets in the cloud.
- Amazon offers the feature because traditional relational databases require a sizableupfront expense.
- They are also complex to design and often require the employment of adatabase administrator.
- Amazon SimpleDB is—as the name says—simpler.
- It requires noschema, automatically indexes data, and provides a simple API for storage and access.
- Thismakes the process easier to manage and eliminates the administrative burden of datamodeling, index maintenance, and performance tuning.

Amazon Simple Storage Service (Amazon S3)

- Amazon Simple Storage Service (Amazon S3) is Amazon's storage solution for the Internet.
- It is designed to make web-scale computing easier for developers.
- Amazon S3 utilizes a simple web services interface that can be used to store and retrieveany amount of data from anywhere on the Web.
- It gives developers access to the same datastorage infrastructure that Amazon uses to run its own retail empire.

Amazon CloudFront

- Amazon CloudFront is a web service for content delivery.
- It works in conjunction with otherAmazon Web Services to give developers and businesses an easy way to distribute content toclients.
- Amazon promises low latency, high data transfer speeds, and no commitments.
- The service delivers content using a global network of edge locations.
- Object requestsare automatically routed to the nearest edge location, so content is delivered with the bestperformance possible.

Amazon Simple Queue Service (Amazon SQS)

- Amazon Simple Queue Service (Amazon SQS) offers a scalable, hosted queue for storingmessages as they travel between computers.
- Developers can move data between distributedcomponents of their applications that perform different tasks, without losing messages orrequiring each component to be always available.
- Amazon SQS allows an automated workflow to be created and works closely withAmazon EC2 and other Amazon Web Services.