CSR21003

SUBJECT CODE : CS3551

Strictly as per Revised Syllabus of

ANNA UNIVERSITY

Choice Based Credit System (CBCS)

Semester - V (CSE / IT / AI&DS)

DISTRIBUTED COMPUTING

Iresh A. Dhotre

M.E. (Information Technology)

Ex-Faculty, Sinhgad College of Engineering

Pune

TECHNICAL PUBLICATIONS

SINCE 1993

An Up-Thrust for Knowledge

DISTRIBUTED COMPUTING

Subject Code : CS3551

Semester - V (CSE / IT / AI&DS)

Published by :

TECHNICAL PUBLICATIONS
SINCE 1993 An Up-Thrust for Knowledge

Amit Residency, Office No.1, 412, Shaniwar Peth,
Pune - 411030, M.S. INDIA, Ph.: +91-020-24495496/97
Email : info@technicalpublications.in Website : www.technicalpublications.in

Printer :
Yogiraj Printers & Binders
Sr.No. 10/1A,
Ghule Industrial Estate, Nanded Village Road,
Tal. - Haveli, Dist. - Pune - 411041.

ISBN 978-93-5585-381-3

9 789355 853813

AU 21

9789355853813 [1]

PREFACE

The importance of **Distributed Computing** is well known in various engineering fields. Overwhelming response to my books on various subjects inspired me to write this book. The book is structured to cover the key aspects of the subject **Distributed Computing**.

The book uses plain, lucid language to explain fundamentals of this subject. The book provides logical method of explaining various complicated concepts and stepwise methods to explain the important topics. Each chapter is well supported with necessary illustrations, practical examples and solved problems. All the chapters in the book are arranged in a proper sequence that permits each topic to build upon earlier studies. All care has been taken to make students comfortable in understanding the basic concepts of the subject.

Representative questions have been added at the end of section to help the students in picking important points from that section.

The book not only covers the entire scope of the subject but explains the philosophy of the subject. This makes the understanding of this subject more clear and makes it more interesting. The book will be very useful not only to the students but also to the subject teachers. The students have to omit nothing and possibly have to cover nothing more.

I wish to express my profound thanks to all those who helped in making this book a reality. Much needed moral support and encouragement is provided on numerous occasions by my whole family. I wish to thank the **Publisher** and the entire team of **Technical Publications** who have taken immense pain to get this book in time with quality printing.

Any suggestion for the improvement of the book will be acknowledged and well appreciated.

Author
D. A. Dhotre

Dedicated to God.

SYLLABUS

Distributed Computing - (CS3551)

UNIT I INTRODUCTION

Introduction : Definition-Relation to Computer System Components – Motivation – Message -Passing Systems versus Shared Memory Systems – Primitives for Distributed Communication – Synchronous versus Asynchronous Executions – Design Issues and Challenges; A Model of Distributed Computations : A Distributed Program – A Model of Distributed Executions – Models of Communication Networks – Global State of a Distributed System. **(Chapter - 1)**

UNIT II LOGICAL TIME AND GLOBAL STATE

Logical Time : Physical Clock Synchronization : NTP – A Framework for a System of Logical Clocks – Scalar Time – Vector Time; Message Ordering and Group Communication : Message Ordering Paradigms – Asynchronous Execution with Synchronous Communication – Synchronous Program Order on Asynchronous System – Group Communication – Causal Order – Total Order; Global State and Snapshot Recording Algorithms : Introduction – System Model and Definitions – Snapshot Algorithms for FIFO Channels. **(Chapter - 2)**

UNIT III DISTRIBUTED MUTEX AND DEADLOCK

Distributed Mutual exclusion Algorithms : Introduction – Preliminaries – Lamport's algorithm – Ricart-Agrawala's Algorithm — Token-Based Algorithms – Suzuki-Kasami's Broadcast Algorithm; Deadlock Detection in Distributed Systems : Introduction – System Model – Preliminaries – Models of Deadlocks – Chandy-Misra-Haas Algorithm for the AND model and OR Model. **(Chapter - 3)**

UNIT IV CONSENSUS AND RECOVERY

Consensus and Agreement Algorithms : Problem Definition – Overview of Results – Agreement in a Failure-Free System(Synchronous and Asynchronous) – Agreement in Synchronous Systems with Failures; Checkpointing and Rollback Recovery : Introduction – Background and Definitions – Issues in Failure Recovery – Checkpoint-based Recovery – Coordinated Checkpointing Algorithm - - Algorithm for Asynchronous Checkpointing and Recovery. **(Chapter - 4)**

UNIT V CLOUD COMPUTING

Definition of Cloud Computing – Characteristics of Cloud – Cloud Deployment Models – Cloud Service Models – Driving Factors and Challenges of Cloud – Virtualization – Load Balancing – Scalability and Elasticity – Replication – Monitoring – Cloud Services and Platforms : Compute Services – Storage Services – Application Services. **(Chapter - 5)**

TABLE OF CONTENTS

UNIT I

UNIT II

UNIT V

UNIT I

1 Introduction

Syllabus

Introduction : Definition-Relation to Computer System Components - Motivation - Message - Passing Systems versus Shared Memory Systems - Primitives for Distributed Communication - Synchronous versus Asynchronous Executions - Design Issues and Challenges; A Model of Distributed Computations : A Distributed Program - A Model of Distributed Executions - Models of Communication Networks - Global State of a Distributed System.

Contents

1.1 Definition

AU : May-22

Definition of distributed systems :

- A distributed system is one in which components located at networked computers communicate and co-ordinate their actions only by-passing messages.
- A distributed system is collection of independent entities that co-operate to solve a problem that cannot be individually solved.
- Tanenbaum's definition : A distributed system is a collection of independent computers that appears to its users a single coherent system.
- DS can be characterized as a collection of mostly autonomous processors communicating over a communication network. It having following features and consequences

1. **Concurrency :** The capacity of the system to handle shared resources can be increased by adding more resources to the network. The p and q are concurrent if either p can happen before q or q can happen before p, thus having interleaving semantics.
2. **No global clock :** The only communication is by sending messages through a network. Not possible to synchronize many computers on a network and guarantee synchronization over time, thus events are logically ordered. Not possible to have a process that can be aware of a single global state.
3. **Independent failures :** The programs may not be able to detect whether the network has failed or has become unusually slow. Running processes may be unaware of other failures within context. Failed processes may go undetected. Both are due to processes running in isolation.
4. **Autonomy and heterogeneity :** The processors are loosely coupled in that they have different speeds and each can be running a different OS.

1.1.1 Disadvantages of DS

1. **Software :** Difficult to develop software for distributed systems.
2. **Network :** Saturation, lossy transmissions.
3. **Security :** Easy access also applies to secret data.
4. Absence of global clock.

1.1.2 Difference between Parallel Computing and Distributed Computing

Sr. No.	Parallel computing	Distributed computing
1.	The goal of parallel computing has traditionally been to provide performance, either in terms of processor power or memory.	The goal of distributed computing is to provide convenience, where convenience includes high availability, reliability and physical distribution.
2.	In parallel computation the interaction between processors is frequent.	In distributed computation the interaction is infrequent.
3.	It is typically fine grained with low overhead.	It is heavier weight.
4.	Assumed to be reliable.	Assumed to be unreliable.
5.	Parallel computation values short execution time.	Distributed computation values long up time.

University Question

1. *Explain how a parallel system differs from a distributed system.* **AU : May-22, Marks 13**

1.2 Relation to Computer System Components

- Fig. 1.2.1 shows typical distributed system.

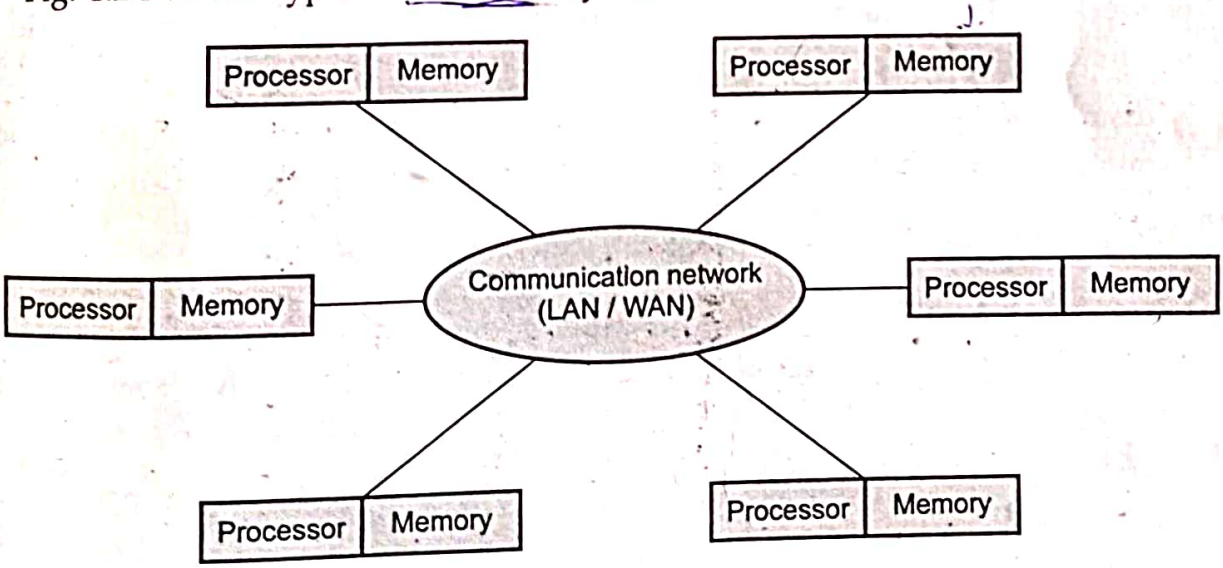


Fig. 1.2.1 Distributed system

- In distributed computing system, each node consists of a processor (CPU), local memory and interface. Communication between any two or more nodes is only by message passing because there is no common memory available.

- Distributed software is also termed as middleware. The distributed system uses a layered architecture to break down the complexity of system design.
- Each computer has memory processing unit and the computers are connected by a communication network. All the computers can communicate with each other through LAN and WAN. The distributed system uses a layered architecture to break down the complexity of system design.
- A distributed system is an information-processing system that contains a number of independent computers that cooperate with one another over a communications network in order to achieve a specific objective.
- Usually, distributed systems are asynchronous, i.e., they do not use a common clock and do not impose any bounds on relative processor speeds or message transfer times. Differences between the various computers and the ways in which they communicate are mostly hidden from users.
- Users and applications can interact with a distributed system in a consistent and uniform way, regardless of where and when interaction takes place. Each host executes components and operates a distribution middleware.
- Middleware enables the components to co-ordinate their activities. Users perceive the system as a single, integrated computing facility.
- A distributed computer system consists of multiple software components that are on multiple computers, but run as a single system. The computers that are in a distributed system can be physically close together and connected by a local network or they can be geographically distant and connected by a wide area network.
- A distributed system can consist of any number of possible configurations, such as mainframes, personal computers, workstations, minicomputers and so on.

1.3 Motivation

1. **Economics :** A collection of microprocessors offer a better price/performance than mainframes. Low price /performance ratio is the cost effective way to increase computing power.
2. **Speed :** A distributed system may have more total computing power than a mainframe.
3. Distributed systems can be extended through the addition of components, thereby providing better scalability compared to centralized systems.
4. **Inherent distribution :** Some applications are inherently distributed e.g. a supermarket chain.

5. **Reliability :** If one machine crashes, the system as a whole can still survive. It gives higher availability and improved reliability.
6. **Incremental growth :** Computing power can be added in small increments.

1.3.1 Need of Distributed System

- Resource sharing is main motivation of the distributed system. The term "resource" is a rather abstract one, but it best characterizes the range of things that can usefully be shared in a networked computer system.
- Resources may be the software resources or hardware resources. Printers, disks, CDROM and data are the example of software and hardware resources. Sharing of resource extends from hardware components such as disks and printers to software - defined entities such as files, databases and data objects of all kinds.
- It also includes the stream of video frames and audio connection that a mobile phone call represents. A resource manager is a software module that manages a set of resources of a particular type.
- Primary requirement of distributed system are as follows :
  1. Fault tolerance
  2. Consistency of replicated data
  3. Security
  4. Reliability
  5. Concurrent transactions

1.3.2 Focus on Resource Sharing

- The term resource is a rather abstract one, but it best characterizes the range of things that can usefully be shared in a networked computer system.
- Resources in a distributed system are encapsulated within one computer and can only be accessed from other computers by communication. For effective sharing each resource must be managed by a program called resource manager offering a communication interface enabling the resource being accessed, manipulated and updated consistently and reliably.
- Equipments are shared to reduce cost. Data shared in database or web pages are high-level resources which are more significant to users without regard for the server or servers that provide these.
- Types of resources :

1. Hardware resource : Hard disk, printer, camera, scanner
2. Data : File, database, web page.
3. Service : Search engine

- Patterns of resource sharing vary widely in their scope and in how closely users work together :
  1. Search Engine : Users need no contact between users.
  2. Computer Supported Co-operative Working (CSCW) : Users cooperate directly share resources. Mechanisms to coordinate user's action are determined by the pattern of sharing and the geographic distribution.
- For effective sharing, each resource must be managed by a program that offers a communication interface enabling the resource to be accessed and updated reliably and consistently.
- Service : Manages a collection of related resources and presents their functionalities to users and applications.
- **Server** is basically storage of resources and it provides services to the authenticated clients. It is running program on a networked computer. Server accepts requests from client and performs a service and responds to request. Example is Apache server and IIS server.
- The complete interaction between server machine and client machine, from the point when the client sends its request to when it receives the server's response, is called a **remote invocation.**
- Resources may be encapsulated as objects and accessed by client objects. In this case a client object invokes a method upon a server object.

Hardware and software resource sharing

- Examples of hardware resources that can be usefully be shared and examples of their sharing.

Hardware resources :

1. **CPU :**
   a. Computing server : It executes processor-intensive applications for clients.
   b. Remote object server : It executes methods on behalf of clients.
   c. Worm program : It shares CPU capacity of desktop machine with the local user.
2. **Memory :** Cache server holds recently-accessed web pages in its RAM, for faster access by other local computers.
3. **Disk :** File server, virtual disk server, video on demand server.
4. **Screen :** Network window systems, such as X-11, allow processes in remote computers to update the content of windows.

5. **Printer :** Networked printers accept print jobs from many computers and managing them with a queuing system.

Software resources :

1. **Web page :** Web servers enable multiple clients to share read-only page content
2. **File :** File servers enable multiple clients to share read-write files.
3. **Object :** Possibilities for software objects are limitless. Shared whiteboard, shared diary and room booking system are examples of this type.
4. **Database :** Databases are intended to record the definitive state of some related sets of data. They have been shared ever since multi-user computers appeared. They include techniques to manage concurrent updates.
5. **Newsgroup content :** The netnews system makes read-only copies of the recently-posted news items available to clients throughout the Internet.
6. **Video/audio stream :** Servers can store entire videos on disk and deliver them at playback speed to multiple clients simultaneously.

1.4 Message-Passing Systems Versus Shared Memory Systems

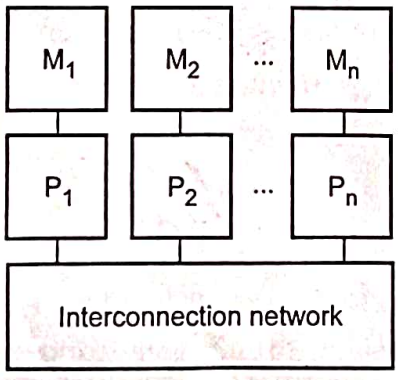
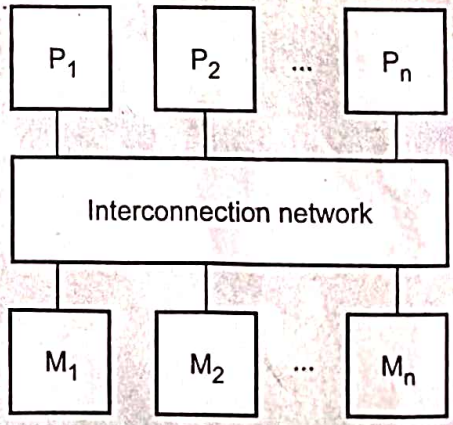
AU : Dec.-22

Message passing

- Two processes communicate with each other by passing messages. Message passing is direct and indirect communication. Indirect communication uses mailbox for sending receiving message from other process.
- Message passing system requires the synchronization and communication between the two processes. Message passing used as a method of communication in microkernels.
- Message passing systems come in many forms. Messages sent by a process can be either fixed or variable size. The actual function of message passing is normally provided in the form of a pair of primitives.
  a) Send (destination_name, message)
  b) Receive (source_name, message).
- Send primitive is used for sending a message to destination. Process sends information in the form of a message to another process designated by a destination. A process receives information by executing the receive primitive, which indicates the source of the sending process and the message.

Shared memory

- Shared memory systems are those in which there is common shared address space throughout the system. Communication among processors takes place via shared data variables and control variables for synchronization among the processor.
- A region of memory that is shared by co-operating processes is established. Processes can then exchange information by reading and writing data to the shared region.
- Shared memory allows maximum speed and convenience of communication, as it can be done at memory speeds when within a computer.
- Shared memory is faster than message passing, as message-passing systems are typically implemented using system calls and thus require the more time-consuming task of Kernel intervention.

Sr. No.	Message - passing systems	Shared memory systems
1.	Platforms that exchange messaging for sharing data are called message passing platforms.	Platforms that provide a shared memory for data sharing are called multiprocessors.
2.	Message passing is useful for sharing small amounts of data so that conflicts need not occur.	In shared memory make sure that the processes are not writing to the same location simultaneously.
3.	In message passing the communication is slower when compared to shared memory technique.	It follows a faster communication strategy when compared to message passing technique.
4.	In this message passing process model, the processes communicate with others by exchanging messages.	If the process wishes to initiate communication and has data to share, create a shared memory region in its address space.
5.	This technique can be used in heterogeneous computers. 	This cannot be used to heterogeneous computers. 

1.4.1 Emulating Message - Passing Systems on a Shared Memory Systems

- Shared address space is divided into two disjoint parts and assigned to each processor. Send and receive operations are implemented by using writing and reading the information from receiver and sender processor.
- Synchronization primitives like write and read operation are controlled by sender/receiver processor.
- A $P_i - P_j$ message passing can be emulated by a write by P_i to the mailbox and then a read by P_j from the mailbox.

University Question

1. *Illustrate the difference between message passing and shared memory process communication model.*
AU : Dec.-22, Marks 13

1.5 Primitives for Distributed Communication

1.5.1 Blocking / Non-blocking, Synchronous / Asynchronous Primitives

- Message send communication primitives is denoted by Send( ) and receive communication primitives denoted by Receive( ).
- Message passing primitive commands
  SEND (msg, dest)
  RECEIVE (src, buffer)
- Send primitives uses two options for sending data : Buffered and unbuffered.
- In buffered options, user data is copied in the kernel buffer. In unbuffered options, the data gets copied directly from the user buffer onto the network.
- The communication of a message between two processes implies some level of synchronization between the two processes. Sender and receiver can be blocking or nonblocking. Three combinations are possible using blocking and nonblocking.
  a. Blocking send, blocking receive.
  b. Nonblocking send, blocking receive.
  c. Nonblocking send, nonblocking receive.

1. **Blocking send, blocking receive :** Both the sender and receiver are blocked until the message is delivered. This is called Rendezvous. This combination allows for tight synchronization between processes.

2. **Nonblocking send, blocking receive :** Sender may continue on, the receiver is blocked until the requested message arrives. A process that must receive a message before it can do useful work needs to be blocked until such message arrives. An example is a server process that exists to provide a service or resource to other processes.
3. **Nonblocking send, nonblocking receive :** Sending process sends the message and resumes the operation. Receiver retrieves either a valid message or a null i.e. neither party is required to wait.

- **Processor synchrony :** Processor synchrony indicates that all the processors execute in lock step with their clocks synchronized.

1.6 Synchronous versus Asynchronous Executions

Synchronous execution :

Main features : A system in which the following bounds are defined :

1. Lower and upper bounds on execution time of processes can be set.
2. Transmitted messages are received within a known bounded time.
3. Drift rates between local clocks have a known bound.

Important consequences :

1. In a synchronous distributed system there is a notion of global physical time (with a known relative precision depending on the drift rate).
2. Only synchronous distributed systems have a predictable behaviour in terms of timing. Only such systems can be used for hard real-time applications.
3. In a synchronous distributed system it is possible and safe to use timeouts in order to detect failures of a process or communication link.

- It is difficult and costly to implement synchronous distributed systems.

Asynchronous executions :

- Many distributed systems (including those on the Internet) are asynchronous.
- No bound-on process execution time i.e. nothing can be assumed about speed, load, and reliability of computers.
- No bound-on message transmission delays i.e. nothing can be assumed about speed, load, reliability of interconnections.
- No bounds on drift rates between local clocks.

Important consequences :

1. In an asynchronous distributed system there is no global physical time. Reasoning can be only in terms of logical time.

2. Asynchronous distributed systems are unpredictable in terms of timing.
3. No timeouts can be used.

- Asynchronous systems are widely and successfully used in practice. In practice timeouts are used with asynchronous systems for failure detection. However, additional measures have to be applied in order to avoid duplicated messages, duplicated execution of operations, etc.

Sr. No.	Synchronous execution	Asynchronous execution
1.	Synchronous execution means the first task in a program must finish processing before moving on to executing the next task.	Asynchronous execution means a second task can begin executing in parallel, without waiting for an earlier task to finish.
2.	Lower and upper bounds on execution time of processes can be set.	No bound-on process execution time.
3.	Transmitted messages are received within a known bounded time.	No bound-on message transmission delay.
4.	Drift rates between local clocks have a known bound.	No bounds on drift rates between local clocks.

1.7 Design Issues and Challenges

AU : May-22

1.7.1 Challenges from System Perspective

- **Communication mechanisms :** This task involves designing appropriate mechanism for communication among the processes in the network. For example : Remote Procedure Call (RPC), Remote Object Invocation (ROI), message-oriented vs. stream-oriented communication.
- **Processes :** Issue involved are code migration, process/thread management at clients and servers, design of software and mobile agents.
- **Naming :** Easy to use identifiers needed to locate resources and processes transparently and scalable.
- **Synchronization :** Mechanisms for synchronization or coordination among the processes are essential. Mutual exclusion is the classical example of synchronization, but many other forms of synchronization, such as leader election are also needed.
- **Data storage and access :** Various schemes for data storage, searching and lookup should be fast and scalable across network. Revisit file system design.

- **Consistency and replication :** To avoid bottleneck, to provide fast accesses to da and provide replication for fast access, scalability. Require consisten management among replicas.
- **Distributed systems security :** Secure channels, access control, key managemen (key generation and key distribution), authorization, secure group management ar the various method used to provide security.

1.7.2 Challenges

- Designing the distributed systems does not come for free. Some challenges need to be overcome in order to get the ideal systems. Design issues and challenges of distributed systems are as follows :

1. Heterogeneity
2. Openness
3. Security
4. Scalability
5. Failure handling
6. Concurrency
7. Transparency

1.7.2.1 Heterogeneity

- Modern distributed systems are highly heterogeneous in many dimensions, including available bandwidth, processor speed, disk capacity, security, failure rate, and pattern of failures. It applies to all the following :
  1. Computer networks : LAN, wireless network, satellite link
  2. Computer hardware devices : Laptop, computer, mobiles phones, tablets
  3. Operating systems : Linux, UNIX, Windows
  4. Programming languages : C/C++, Java, PHP
  5. Different roles of software developers, designers, system managers
- There may be many different representations of data in the system. This might include different representations for integers, byte streams, floating point numbers, and character sets.
- Most of the data can be marshaled from one system to another without losing significance. Attempts to provide a universal canonical form of information is lagging.
- The integration of heterogeneous components implies the construction of distributed systems. In a distributed system, heterogeneity is almost unavoidable, as different components may require different implementation technologies.

Middleware

- Middleware is a software layer that provides a programming abstraction as well as masking the heterogeneity of the underlying platform. E.g., CORBA, DCOM, Java RMI, etc.

- Middleware serves to hide both these aspects by providing uniform, standard, high-level interfaces to the application developers and integrators, so that applications can be easily composed, reused, ported, and made to interoperate. Middleware services provide common services to perform various general purpose functions. Fig. 1.7.1 shows position of middleware. Middleware software resides above the network and below the application software.

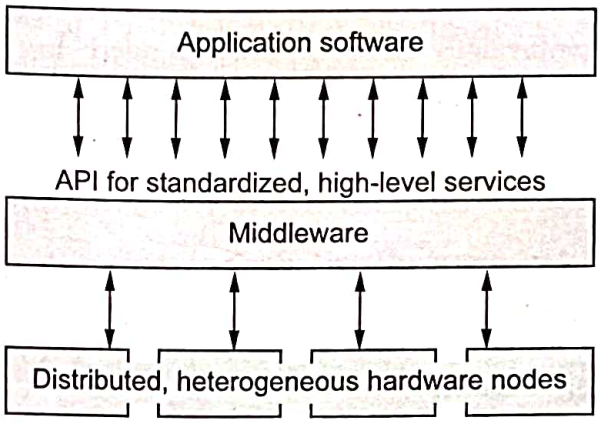


Fig. 1.7.1 Position of middleware

- Mobile code is used to refer to program code that can be transferred from one computer to another and run at the destination computer. Example of mobile code is Java applets.
- Code suitable for running on one computer is not necessarily suitable for running on another because executable programs are normally specific both to the instruction set and to the host operating system. Middleware should make the network transparent to the applications and end users.
- Users and applications should be able to perform the same operations across the network that they can perform locally. Middleware should hide the details of computing hardware, OS, software components across networks.

1.7.2.2 Openness

- Openness means that the system can be easily extended and modified. Openness refers to the ability to plug and play. You can, in theory, have two equivalent services that follow the same interface contract, and interchange one with the other.
- The integration of new components means that they have to be able to communicate with some of the components that already exist in the system. Openness and distribution are related. Distributed system components achieve openness by communicating using well-defined interfaces.
- If the well-defined interfaces for a system are published, it is easier for developers to add new features or replace sub-systems in the future.
- Open systems can easily be extended and modified. New components can be integrated with existing components.

- Differences in data representation or interface types on different processors have to be resolved. Openness and distribution are related to each other. System components need to have well-defined and well-documented interfaces.
- It can be constructed from heterogeneous hardware and software. Openness is concerned with extensions and improvements of distributed systems. Detailed interfaces of components need to be published. New components have to be integrated with existing components.
- The system needs to have a stable architecture so that new components can be easily integrated while preserving previous investments.
- An **open distributed system** offers services according to standard rules that describe the syntax and semantics of those services.

1.7.2.3 Security

- Security becomes even more important in a distributed system. Authentication, authorization, digital signatures, non-repudiation, encryption, and privacy become major issues in the distributed system.
- The four basic goals of a security system are to protect information, to detect an intrusion, to confine the security breach, and to repair the damage and return the system to a known stable and secure state.
- Security for information resources has three components :
  1. **Confidentiality :** Protection against disclosure to unauthorized individuals, e.g. ACL in unix file system.
  2. **Integrity :** Protection against alteration or corruption, e.g. checksum.
  3. **Availability :** Protection against interference with the means to access the resources, e.g. Denial of service.
- Encryption provides protection of shared resources, keeps sensitive information secret when transmitted. Security challenges that are not yet fully met :
  1. Denial of service attacks
  2. Security of mobile code.
- A denial-of-service attack is an attempt to make a computer or network resource unavailable to its intended users.
- Security of mobile code : Mobile code systems are conceived to operate in large scale settings where networks are composed of heterogeneous hosts, managed by different authorities with different levels of trust and connected by links with different bandwidths.

- Mobile code systems address a wide range of needs and requirements, such as service customization, dynamic extension of application functionality, autonomy, fault tolerance and support for disconnected operations.

1.7.2.4 Scalability

- A system is said to be scalable if it can handle the addition of users and resources without suffering a noticeable loss of performance or increase in administrative complexity.
- The ability to accommodate any growth in the future be it expected or not. Distributed system architectures achieve scalability through employing more than one host. Distributed systems can be scalable because additional computers can be added in order to host additional components.
  1. **In size :** Dealing with large numbers of machines, users, tasks.
  2. **In location :** Dealing with geometric distribution and mobility.
  3. **In administration :** Addressing data passing through different regions of ownership.
- The design of scalable distributed systems presents the following challenges :
  1. Controlling the cost of resources.
  2. Controlling the performance loss. 1408
  3. Preventing software resources from running out.
  4. Avoiding performance bottlenecks.
- Controlling the cost of physical resources i.e. servers and users.
- Controlling the performance loss : DNS hierarchic structures scale better than linear structures and save time for access structured data.
- Preventing software resources running out : Internet 32 bits addresses run out soon. 128 bits one gives extra space in messages.
- Avoiding performance bottlenecks : DNS name table was kept in a single master file partitioning between servers.
- **Example :** File system scalability is defined as the ability to support very large file systems, large files, large directories and large numbers of files while still providing I/O performance. Google file system aims at efficiently and reliably managing many extremely large files for many clients, using commodity hardware.
- Various techniques such as replication, caching and cache memory management and asynchronous processing help to achieve scalability.

Scaling techniques

1. **Hiding communication latencies :** Examples would be asynchronous communication as well as pushing code down to clients (e.g. Java applets and Javascript).
2. **Distribution :** Taking a component, splitting into smaller parts, and subsequently spreading them across the system.
3. **Replication :** Replicating components increases availability, helps balance the load leading to better performance, helps hide latencies for geographically distributed systems. Caching is a special form of replication.

1.7.2.5 Failure Handling

- When faults occur in hardware or software, programs may produce incorrect results or they may stop before they have completed the intended computation. Failure handling is difficult in distributed systems because failure is partial i.e. some components fail while other continue to function.
- Hardware, software and networks are not free of failures. Operations that continue even in the presence of faults are referred to as **fault-tolerant.**
- Distributed systems can maintain availability even at low levels of hardware /software /network reliability.
- Fault tolerance, that should not involve users or system administrators, is achieved by recovery and replication of components.
- Techniques for dealing with failures :
  1. Detecting failures
  2. Masking failures
  3. Tolerating failures
  4. Recovering from failures
  5. Redundancy.
- **Detecting failures :** Not all failures are detected but some of the failures can be detected. For example : Corrupted data from file is detected by using checksum. Detection of failure in remote crashed server in the internet is hard to detect.
- **Masking failures :** Failures are hidden or made less sever. Examples of hiding failures are :
  1. Messages could be retransmitted.
  2. Files can be duplicated on different place.

- **Tolerating failures :** In the internet, client can be designed to tolerate failures which generally involve the users tolerating them as well.
- **Recovery from failures :** Recovery involves the design of software so that the state of permanent data can be recovered or roll back after a server has crashed.
- **Redundancy :** Services can be made to tolerate failures. Let us consider following examples :

1. There should be at least 2 routes between any 2 routers in the internet.
2. In DNS, every name table is replicated in at least 2 servers.
3. A database can be replicated in several servers. Clients are redirected from the failure one to working one.

1.7.2.6 Concurrency

- Components in distributed systems are executed in concurrent processes. Fig. 1.7.2 shows concept of concurrency.
- There is a possibility that several clients will attempt to access a shared resource at the same time. Multiple users make requests on same resource for read, write and update operation. Each resource must be safe in a concurrent environment.
- Any object that represents a shared resource in a distributed system must be responsible for ensuring that operates correctly in a concurrent environment.

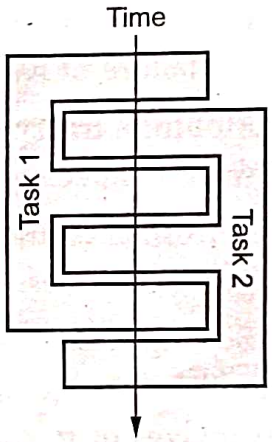


Fig. 1.7.2 Concurrency

1.7.2.7 Transparency

- Distributed systems should be perceived by users and application programmers as a whole rather than as a collection of cooperating components. Transparency is a key issue within distributed system.
- Concept : Hide different aspects of distribution from the client. It is the ultimate goal of many distributed systems.
- It can be achieved by providing lower-level services. The client uses these services instead of hard coding the information. The service layer provides a service with a certain Quality of Service.

- Transparency is an important goal, but has to be considered together with all other non-functional requirements and with respect to particular demands.
  1. **Location transparency :** User using resources need not know the location of the resource. Any request to retrieve or update data from any site is automatically forwarded by the system to the site or sites related to the processing request. For example : URL
  2. **Access transparency :** Using identical operations to access local and remote resources, e.g. Hyperlink in web page.
  3. **Concurrency transparency :** Several processed operate concurrently using shared resources without interference with between them.
  4. **Replication transparency :** Multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers, e.g. web cache.
  5. **Failure transparency :** Users and applications to complete their tasks despite the failure of hardware and software components, e.g., email.
  6. **Mobility transparency :** Movement of resources and clients within a system without affecting the operation of users and programs, e.g., mobile phone.
  7. **Performance transparency :** Allows the system to be reconfigured to improve performance as loads vary.
  8. **Scaling transparency :** Allows the system and applications to expand in scale without change to the system structure or the application algorithms.
- **Advantages of transparency :**
- **Easier for the user :**
  a. Doesn't have to bother with system topography.
  b. Doesn't have to know about changes.
  c. Easier to understand.
- **Easier for the programmer :**
  a. Doesn't have to bother with system topography.
  b. Doesn't have to know about changes.
  c. Easier to understand.

Disadvantages of transparency :

a. Optimization cannot be done by programmer or user.

b. Strange behavior when the underlying system fails.

c. Underlying system can be very complete.

1.7.3 Application of Distributed Computing and Challenges

1. Mobile system

- The portability of the devices, such as laptop computers, PDA, mobile phone, refrigerators, together with their ability to connect conveniently to networks in different places, makes mobile computing possible.
- Ubiquitous computing is the harnessing of many small cheap computational devices that are present in user's physical environments, including the home, office and elsewhere.
- Mobile devices are
  1. Laptop computers.
  2. Handheld devices, including PDAs, cell phones, pagers, video cameras and digital cameras.
  3. Wearable devices, such as smart watches.
  4. Devices embedded in appliances such as washing machines, hi-fi systems, cars.
- Mobile computing (nomadic computing)
  1. People can still access resources while he is on the move or visiting places other than their usual environment.
  2. Location-aware computing : Utilize resources that are conveniently nearby.
- Ubiquitous computing (pervasive computing)
  1. The harnessing of many small, cheap computational devices those are present in user's physical environments, including the home, office and elsewhere.
  2. It benefits users while they remain in a single environment such as home.
- Fig. 1.7.3 shows the portable and handheld devices in a distributed system.
(See Fig. 1.7.3 on next page)
- Mobile and ubiquitous computing raise significant system issues presents an architecture for mobile computing.
- User has access to three forms of wireless connection :
  1. A laptop is connected to host's wireless LAN.
  2. A mobile phone is connected to internet using WAP via a gateway.
  3. A digital camera is connected to a printer over an infra-red link.

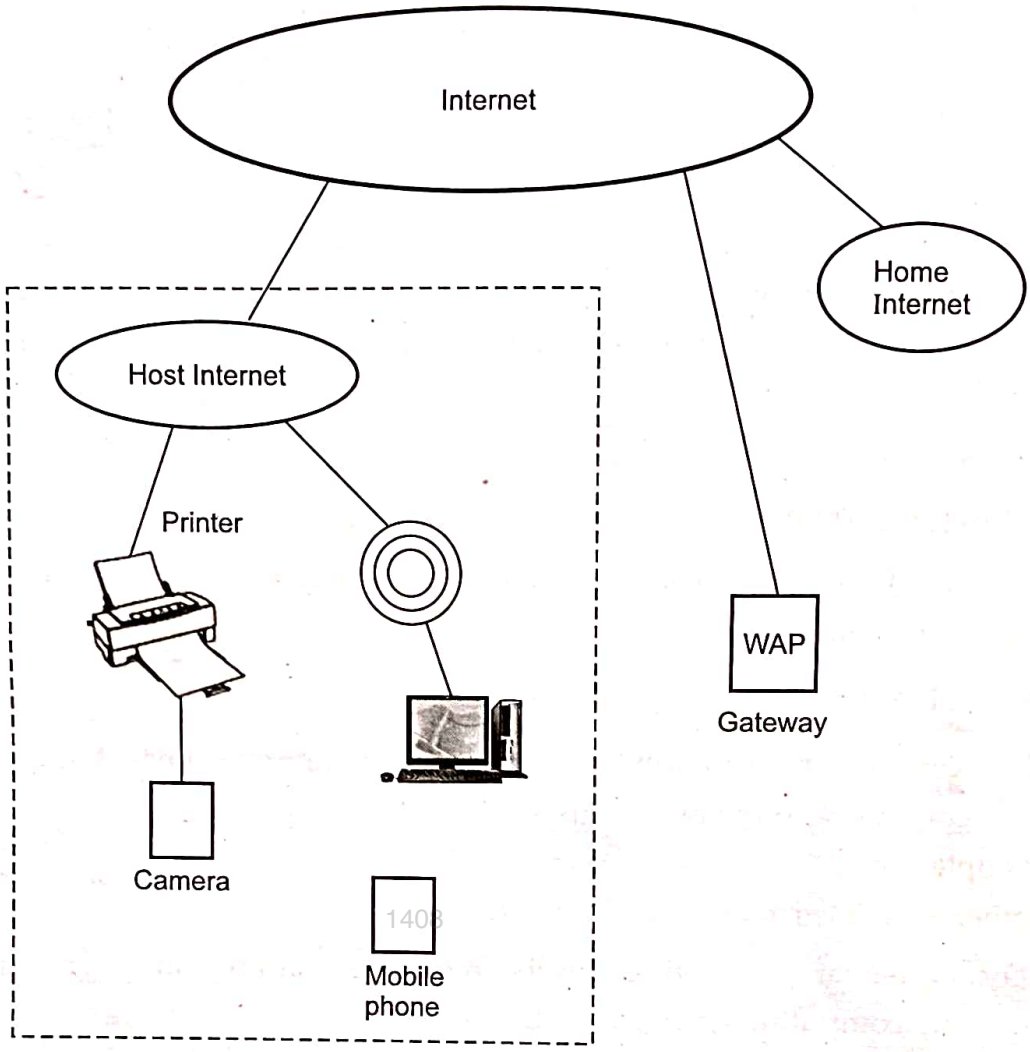


Fig. 1.7.3 Portable devices in DS

2. Pervasive computing

- Networking has become a pervasive resource and devices can be connected at any time and any place. The modern Internet is collection of various computer networks.
- Computer network are of different types. Example of network includes a wide range of wireless communication technologies such as WiFi, WiMAX, Bluetooth and third-generation mobile phone networks.
- Fig. 1.7.4 shows the typical portion of the internet.
- Programs running on the computers connected to it interact by passing messages, employing a common means of communication.

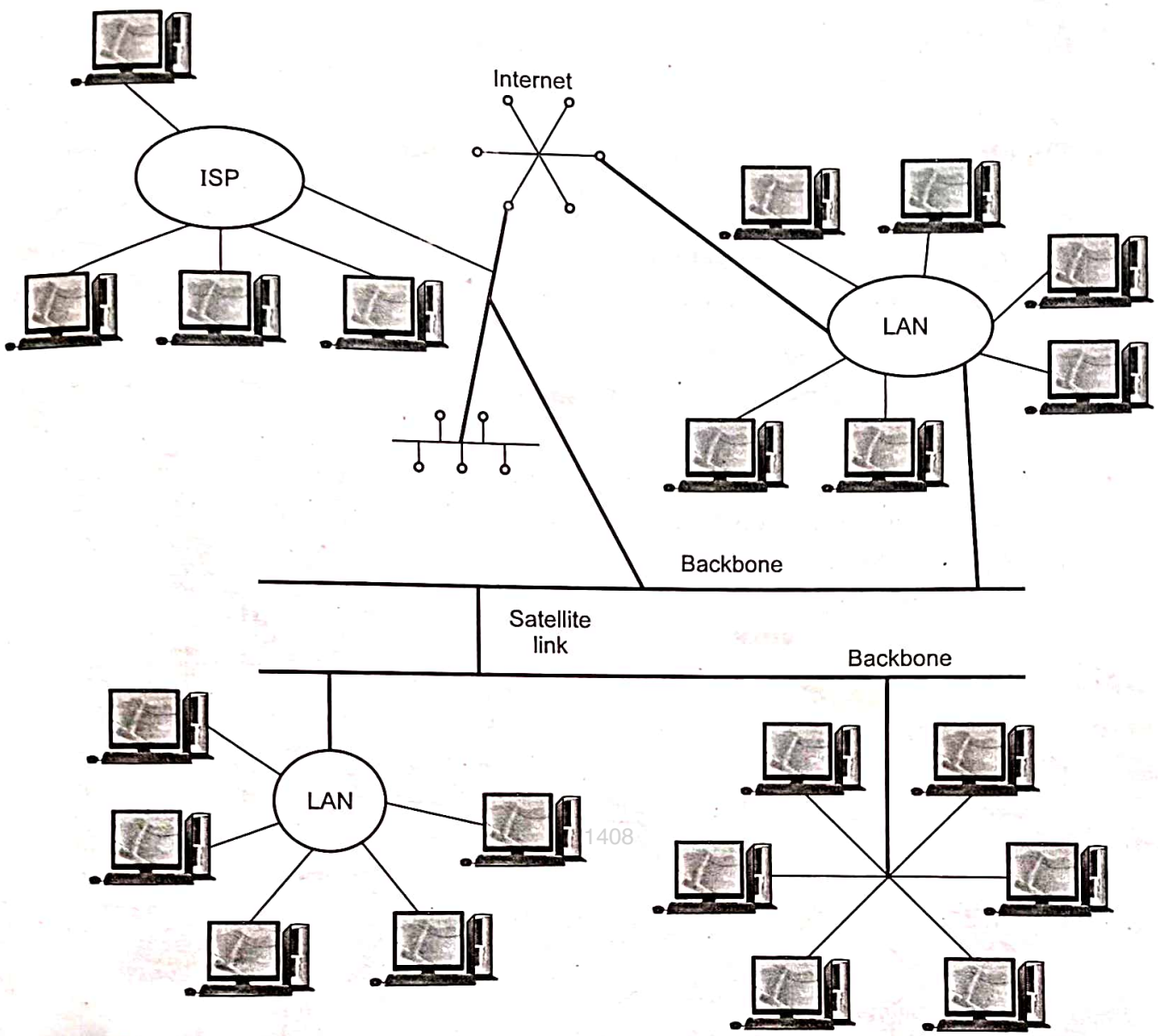


Fig. 1.7.4 Typical portion of Internet

- The internet is a collection of large number of computer networks of many different types. Internet communication mechanism is big technical achievement and it is possible by using passing of messages.
- The Internet is a very large distributed system. The web is not equal to the Internet. The implementation of the internet and services that it supports has entailed the development of practical solutions to many distributed system issues.
- Internet service providers are companies that provide modem links and other types of connection to individual users and small organizations, enabling them to access services anywhere. It also provides local services such as email and web hosting.

- A backbone in a network link with a high transmission capacity, employing satellite connections and fibre optic cable. The Internet also provides multimedia services. User can download audio and video files. Using Internet user can watch TV, play online games and do the video conference.

Intranet

- An intranet is a private network that is contained within an enterprise. It may consist of many interlinked local area networks and also use leased lines in the WAN.
- Intranet is composed of several local area networks linked by backbone connections. Routers are used to connect intranet to internet.
- An intranet is a portion of the internet that is separately administered and has a boundary that can be configured to enforce local security policies. Fig. 1.7.5 shows intranet.

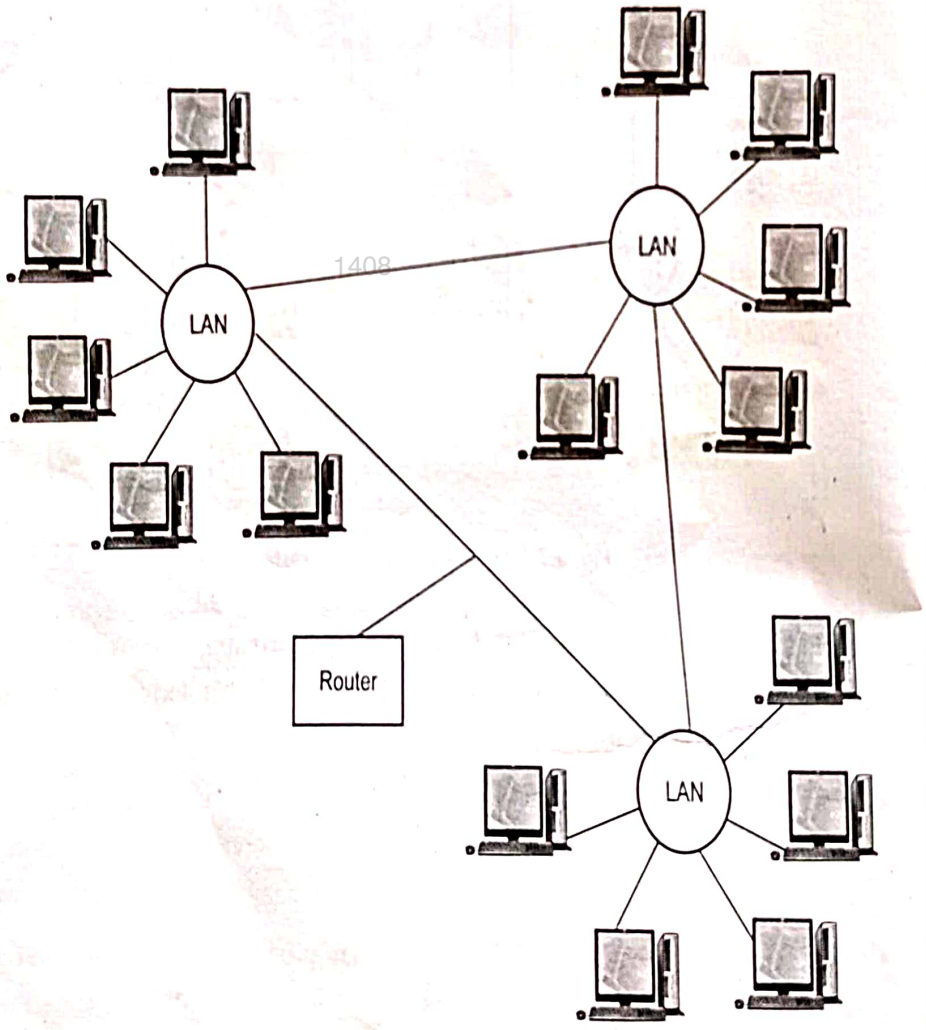


Fig. 1.7.5 Intranet

- The main issues arising in the design of components for use in intranets are : File services, firewalls and cost.

- Firewalls protect an intranet by preventing unauthorized messages leaving or entering; implementing by filtered messages. The cost of software installation and support is reduce by the use of system architectures such as network computers and thin clients.

3. Multimedia system

- Digital multimedia : Computer-controlled integration of text, graphics, still images, moving pictures, animation, sound, and any other medium. All these data types are represented, stored, transmitted, and processed digitally.
- A continuous media type has an implicit time dimension, while a discrete type does not. Referring to video and audio data as continuous and time based. Continuous refers to the user's view of data. Internally, continuous media are represented as sequences of discrete values that replace each other over time.
- Multimedia streams are said to be time-based because timed data elements in audio and video streams define the content of the stream. The systems that support multimedia applications need to preserve the timing when they handle continuous data.
- A distributed multimedia system should be able to perform the same functions for continuous media types such as audio and video. It should be able to store and locate audio or video files, to transmit them across the network, to support the presentation of the media types to the user and optionally also to share the media types across a group of users.
- Distributed multimedia applications typically handle two different types of communication : Request/Reply interaction for control information as well as real-time streaming data.

Web casting

- Web casting is an application of distributed multimedia technology. It broadcast continuous media over the internet. A webcast uses streaming media technology to take a single content source and distribute it to many simultaneous listeners/viewers.
- Web casting support following distributed multimedia application :
  1. It supports for encoding and encryption formats. For example : MP3 standard, MPEG-1, HDTV.
  2. It supports quality of service.
  3. It uses resources management strategies, including appropriate scheduling policies to support the desired quality of service.
- **Webcasting** is also called push technology. It is a method of obtaining information in which a server automatically downloads content to your computer at regular intervals or whenever updates are made to the site.

University Question

1. *Discuss the design issues and challenges in distributed system from a system perspective.*

AU : May-22, Marks 13

1.8 A Model of Distributed Computations : A Distributed Program

- Distributed program is composed of a set of "n" asynchronous processes like p_1, p_2, p_3, ..., p_i, p_n that communicate by message passing over the communication network.
- Here we assume that, each process is running on a different processor. The processes communicate only by using passing messages method.
- Let C_{ij} denote the channel from process p_i to process p_i and let m_{ij} denote a message sent by p_i to p_j. The communication delay is finite and unpredictable.
- Also, processes do not share a global clock. Process execution and message transfer are asynchronous. - a process may execute an action spontaneously and a process sending a message does not wait for the delivery of the message to be complete.

1.9 A Model of Distributed Executions

- Process execution means sequential execution of its action. Actions are atomic. Actions of process are of three types : Interval events, message send events and message receive events.
- Internal event : Affects only the process which is executing the event,
- Send event : A process passes messages to other processes.
- Receive event : A processes gets messages from other processes.
- For a message m, send(m) denotes its send events and rec(m) denote its receive events. A send event changes the state of the process that sends the message and the state of the channel on which the message is sent.
- A receive event changes the state of the process that receives the message and the state of the channel on which the message is received.
- Fig. 1.9.1 shows space-timing diagram for distributed execution.

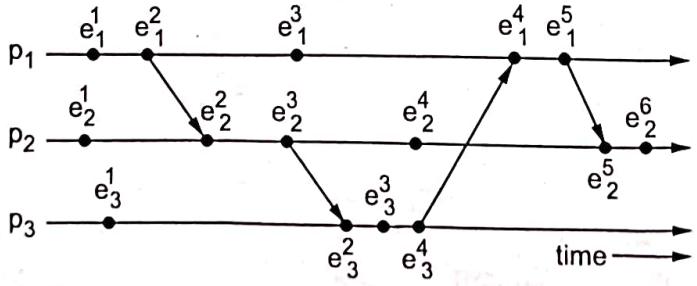


Fig. 1.9.1

- The send and the receive events signify the flow of information between processes and establish causal dependency from the sender process to the receiver process.
- In the figure, for process p_1, the second event is a message send event, the third event is an internal event and the fourth event is a message receive event.

Causal precedence relation

- Ordering of events for a single process is simple: they are ordered by their occurrence.
- Send and receive events signify the flow of information between processes and establish causal precedence between events at the sender and receiver.
- For any two events e_i and e_j, $e_i \rightarrow e_j$ denotes the fact that event e_j does not directly or transitively dependent on event e_i. That is, event e_i does not causally affect event e_j.

Logical vs. Physical concurrency

- In a distributed computation, two events are logically concurrent if and only if they do not causally affect each other.
- Physical concurrency, on the other hand, has a connotation that the events occur at the same instant in physical time.
- Two or more events may be logically concurrent even though they do not occur at the same instant in physical time.
- However, if processor speed and message delays would have been different, the execution of these events could have very well coincided in physical time.
- Whether a set of logically concurrent events coincide in the physical time or not, does not change the outcome of the computation.
- Therefore, even though a set of logically concurrent events may not have occurred at the same instant in physical time, we can assume that these events occurred at the same instant in physical time.

1.10 Models of Communication Networks

- There are several models of the service provided by communication networks, namely, FIFO, Non-FIFO and causal ordering.
- In the FIFO model, each channel acts as a first-in first-out message queue and thus, message ordering is preserved by a channel.
- In the non-FIFO model, a channel acts like a set in which the sender process adds messages and the receiver process removes messages from it in a random order.
- The "causal ordering" model is based on Lamport's "happens before" relation. A system that supports the causal ordering model satis?es the following property :

For any two messages m_{ij} and m_{kj}, if send $(m_{ij}) \rightarrow$ send (m_{kj}), then rec $(m_{ij}) \rightarrow$ rec (m_{kj}).

- This property ensures that causally related messages destined to the same destination are delivered in an order that is consistent with their causality relation. Causally ordered delivery of messages implies FIFO message delivery.
- Causal ordering model considerably simplifies the design of distributed algorithms because it provides a built-in synchronization.

1.11 Global State of Distributed System

Definition : *"The global state of a distributed computation is the set of local states of all individual processes involved in the computation plus the state of the communication channels."*

Requirements of global states

- Fig. 1.11.1 shows detecting global properties.

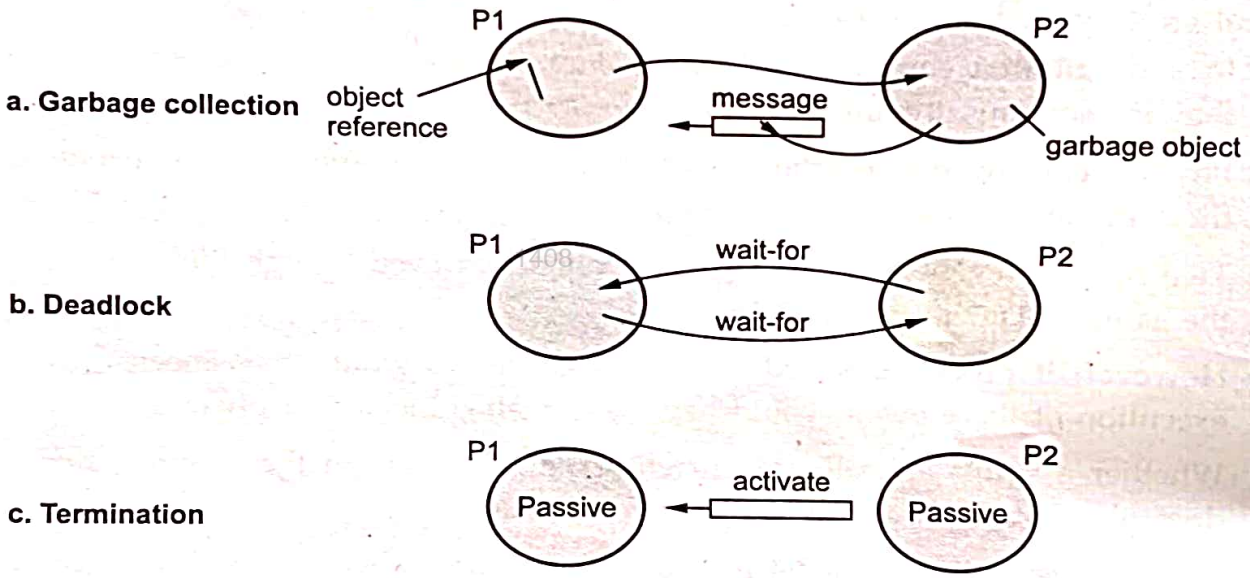


Fig. 1.11.1

1. **Distributed garbage collection :** An object is considered to be garbage if there are no longer any references to it anywhere in the distributed system. It is based on reference counting and should include the state of communication channels.
2. **Distributed deadlock detection :** It occurs when each of collection of processes waits for another process to send it a message and look for "waits-for" relationship.
3. **Distributed termination detection :** Look for state in which all processes are passive.
4. **Distributed debugging :** Need collect values of distributed variables at the same time.

1.12 Two Marks Questions with Answers

Q.1 What do you mean by message passing ? AU : Dec-22

Ans. : In this model, data is shared by sending and receiving messages between co-operating processes, using system calls. Message passing refers to services performing a simple, one-way transfer operation between two programs.

Q.2 Define distributed program. AU : Dec-22

Ans. : Distributed program is composed of a set of "n" asynchronous processes like $p_1, p_2, p_3, \cdots, p_i, p_n$ that communicate by message passing over the communication network.

Q.3 What do you mean by synchronous and asynchronous execution ? AU : Dec-22

Ans. : Synchronous execution means the first task in a program must finish processing before moving on to executing the next task. Asynchronous execution means a second task can begin executing in parallel, without waiting for an earlier task to finish.

Q.4 List out the features of distributed systems. AU : May-22

Ans. : Features of distributed systems are heterogeneity, openness, scalability, fault tolerance, transparency and resource sharing.

Q.5 Differentiate between synchronous and asynchronous execution. AU : May-22

Ans. :

Sr. No.	Synchronous execution	Asynchronous execution
1.	Synchronous execution means the first task in a program must finish processing before moving on to executing the next task.	Asynchronous execution means a second task can begin executing in parallel, without waiting for an earlier task to finish.
2.	Lower and upper bounds on execution time of processes can be set.	No bound-on process execution time.
3.	Transmitted messages are received within a known bounded time.	No bound-on message transmission delay.
4.	Drift rates between local clocks have a known bound.	No bounds on drift rates between local clocks.

Q.6 What is distributed system ? AU : May-19

Ans. : A distributed system is one in which components located at networked computers communicate and co-ordinate their actions only by passing messages. A distributed system is a collection of independent computers that appears to its users a single coherent system.

Q.7 Write down the principles of distributed systems. **AU : May-18**

Ans. : Distributed system consists of a collection of autonomous computers, connected through a network and distribution middleware, which enables computers to coordinate their activities and to share the resources of the system, so that users perceive the system as a single, integrated computing facility.

Q.8 State the objectives of resource sharing model. **AU : May-18**

Ans. : Ability to use any hardware, software or data anywhere in the system. Resource manager controls access, provides naming scheme and controls concurrency.

Q.9 What are the significant consequences of distributed systems ? **AU : May-16**

Ans. : **a. No global clock :** The only communication is by sending messages through a network.

b. Independent failures : The programs may not be able to detect whether the network has failed or has become unusually slow.

c. Concurrency : The capacity of the system to handle shared resources can be increased by adding more resources to the network.

Q.10 Define transparency. What are its types ? **AU : May-17**

Ans. : A distributed system needs to hide the fact that its processes and resources are physically distributed across multiple computers.

Q.11 What is the need of openness in distributed system ? **AU : May-17**

Ans. : Distributed system must be able to interact with services from other open systems, irrespective of the underlying environment. Systems should conform to well-defined interfaces and should support portability of applications.

Q.12 List any two resources of hardware and software, which can be shared in distributed systems with example. **AU : Dec.-17**

Ans. : Hardware resource : Memory cache server and CPU servers do some computation for their clients hence their CPU is a shared resource.

Software resource : File : File servers enable multiple clients to have read/write access to the same files. Database : The content of a database can be usefully shared. There are many techniques that control the concurrent access to a database.

Q.13 List an example of distributed system.

Ans. : Distributed system examples : Internet, an intranet which is a portion of the internet managed by an organization, mobile and ubiquitous computing.

Q.14 Enlist the design issues and challenges of distributed systems.

Ans. : Design issues and challenges of distributed systems are heterogeneity, openness, security, scalability, failure handling, concurrency and transparency.

Q.15 Define access transparency.

Ans. : Enables local and remote information objects to be accessed using identical operations.

Q.16 What is replication transparency ?

Ans. : It enables multiple instances of information objects to be used to increase reliability and performance without knowledge of the replicas by users or application programs. Example : Distributed DBMS.

Q.17 What is the goal of concurrency and failure transparency ?

Ans. : Enables several processes to operate concurrently using shared information objects without interference between them.

Failure transparency : Allows users and applications to complete their tasks despite the failure of other components.

Q.18 Differentiate between buffering and caching. **AU : May-15**

Ans. : Cache is made from static ram which is faster than the slower dynamic ram used for a buffer. A cache transparently stores data so that future requests for that data can be served faster. A buffer temporarily stores data while the data is the process of moving from one place to another, i.e. the input device to the output device. The buffer is mostly used for input/output processes while the cache is used during reading and writing processes from the disk.

Q.19 What is open distributed system ?

Ans. : Open distributed system is a system that offers services according to standard rules that describe the syntax and semantics of those services.

Q.20 Give the example of relocation transparency ?

Ans. : When mobile users can continue to use their wireless laptops while moving from place to place without ever being disconnected.

Q.21 Describe what is meant by a scalable system ?

Ans. : A system is scalable with respect to either its number of components, size or number and size of administrative domains, if it can grow in one or more of these dimensions without an unacceptable loss of performance.

Q.22 What is the role of middleware in a distributed system ? **AU : May-15**

Ans. : To enhance the distribution transparency that is missing in network operating systems. In other words, middleware aims at improving the single system view that a distributed system should have.

Q.23 What is scalability ?

Ans. : A system is said to be scalable if it can handle the addition of users and resources without suffering a noticeable loss of performance or increase in administrative complexity.

Q.24 Name some services and examples of middleware. **AU : May-16**

Ans. : Middleware services are sets of distributed software that exist between the application and the OS and network services on a system node in the network.

Example : Open Software Foundation's Distributed Computing Environment (DCE), CORBA and COM/DCOM.

UNIT II

2 Logical Time and Global State

Syllabus

Logical Time : Physical Clock Synchronization: NTP – A Framework for a System of Logical Clocks– Scalar Time – Vector Time; Message Ordering and Group Communication : Message Ordering Paradigms – Asynchronous Execution with Synchronous Communication – Synchronous Program Order on Asynchronous System – Group Communication – Causal Order – Total Order; Global State and Snapshot Recording Algorithms : Introduction – System Model and Definitions – Snapshot Algorithms for FIFO Channels.

Contents

2.1 Clock Events and Process State

- How to order the events that occur at a single processor ?
- A distributed system is defined as a collection P of N processes p_i, i = 1, 2,... N. Each process executes on a single processor and the processors do not share memory. Each process p_i has a state s_i consisting of its variables.
- Processes communicate only by messages (via a network). We can view each process pi as to execute a sequence of actions that fall in one of the following categories :
  1. Sending a message;
  2. Receiving a message;
  3. Performing a computation that alters its state s_i.
- We define an event to be the execution of a single action by p_i. Event is the occurrence of a single action that a process carries out as it executes. For example : Send, Receive, change state. The sequence of events within a single process pi can be totally ordered, which is generally denoted by $e \rightarrow_i e'$ if and only if the event e occurs before e' at p_i.
- We can then define the history of process pi to be the sequence of events that take place within : history $(p_i) = h_i = < e_i^0, e_i^1, e_i^2, ... >$

2.1.1 Physical Clock

- Computer physical clocks are electronic devices that count oscillations occurring in a crystal at a definite frequency. This count can be stored in a counter register. The clock output can be read by software and scaled into a suitable time unit. This value can be used to timestamp any event experienced.
- Most computers today keep track of the passage of time with a battery-backed-up CMOS clock circuit, driven by a quartz resonator. This allows the time keeping to take place even if the machine is powered off. When on, an operating system will generally program a timer circuit to generate an interrupt periodically. The interrupt service procedure simply adds one to a counter in memory.
- While the best quartz resonators can achieve an accuracy of one second in 10 years, they are sensitive to changes in temperature and acceleration and their resonating frequency can change as they age.
- The only problem with maintaining a concept of time is when multiple entities attempt to do it concurrently. Two watches hardly ever agree. Computers have the same problem : A quartz crystal on one computer will oscillate at a slightly

different frequency than on another computer, causing the clocks to tick at different rates.

- The phenomenon of clocks ticking at different rates, creating a ever widening gap in perceived time is known as clock drift. The difference between two clocks at any point in time is called clock skew and is due to both clock drift and the possibility that the clocks may have been set differently on different machines.
- Fig. 2.1.1 shows skew with two clocks.

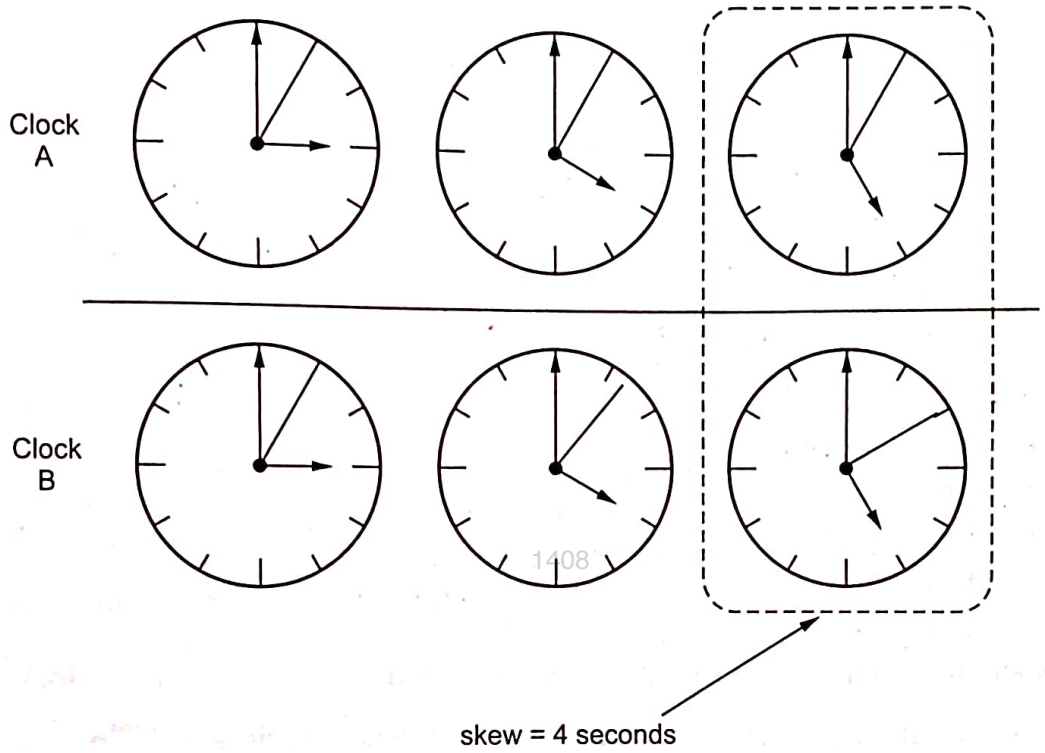


Fig. 2.1.1 Clock drift and clock skew

- Consider two clocks A and B, where clock B runs slightly faster than clock A by approximately two seconds per hour. This is the clock drift of B relative to A. At one point in time, the difference in time between the two clocks is approximately 4 seconds. This is the clock skew at that particular time.
- Successive events will correspond to different timestamps only if the clock resolution is smaller than the rate at which events can occur. The rate at which events occur depends on such factors as the length of the processor instruction cycle.
- Applications running at a given computer require only the value of the counter to timestamp events. The date and time-of-day can be calculated from the counter value. Clock drift may happen when computer clocks count time at different rates.

- Co-ordinated Universal Time (UTC) is an international standard that is based on atomic time. UTC signals are synchronized and broadcast regularly from land-based radio stations and satellites.
- If the computer clock is behind the time service's, it is OK to set the computer clock to be the time service's time. However, when the computer clock runs faster then it should be slowed down for a period instead of set back to the time service's time directly.
- The way to cause computer's clock run to slow for a period can be achieved in software without changing the rate of the hardware clock. Also called timer, usually a quartz crystal, oscillating at a well-defined frequency.

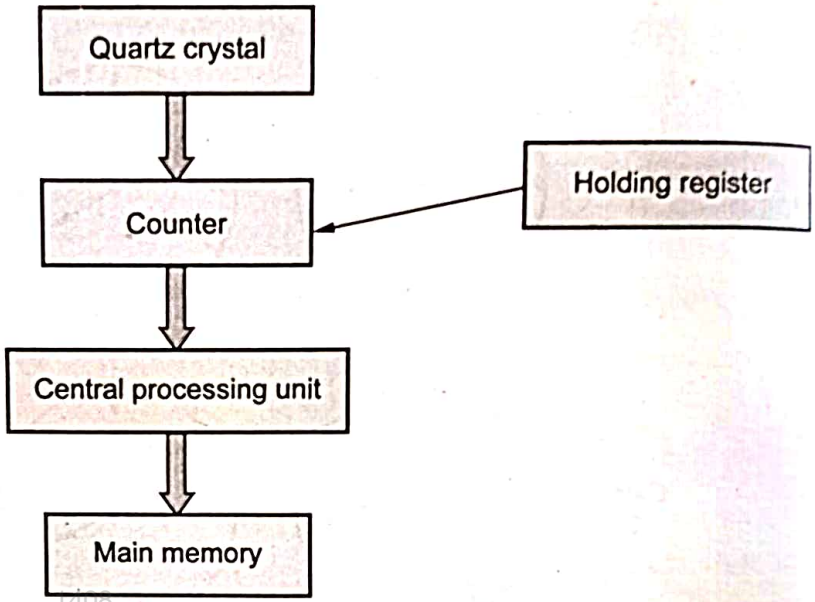


Fig. 2.1.2 Working of computer clock

- A timer is associated with two registers: a counter and a holding register, and counter decreasing one at each oscillation. When the counter gets to zero, an interruption is generated and is called **one clock tick.**
- Crystals run at slightly different rates, the difference in time value is called a clock skew. Clock skew causes time-related failures. Fig. 2.1.2 shows working of computer clock.

Working :

1. Oscillation at a well-defined frequency
2. Each crystal oscillation decrements the counter by 1
3. When counter gets 0, its value reloaded from the holding register
4. When counter is 0, an interrupt is generated, which is call a clock tick
5. At each clock tick, an interrupt service procedure add 1 to time stored in memory

- **Synchronization of physical clocks with real-world clock :**

1. TAI (International Atomic Time) : Cs133 atomic clock
2. UTC (Universal Co-ordinated Time) : Modern civil time, can be received from WWV (shortwave radio station), satellite, or network time server.
3. ITS (Internet Time Service) NTS (Network Time Protocol)

- **Some definitions :**

1. **Transit of the sun :** The event of the sun's reaching its highest apparent point in the sky.
2. **Solar day :** The interval between two consecutive transits of the sun is called the solar day.
3. **Coordinated Universal Time (UTC) :** The most accurate physical clocks known use atomic oscillators, whose accuracy is about one part in 10^{13} . The output of these atomic clocks is used as the standard for elapsed real time, known as International Atomic Time. Co-ordinated universal time is an international standard that is based on atomic time, but a so-called leap second is occasionally inserted or deleted to keep in step with astronomical time.

2.1.2 Clock Skew and Drift Compensating

- A quartz crystal on one computer will oscillate at a slightly different frequency than on another computer, causing the clocks to tick at different rates. The phenomenon of clocks ticking at different rates, creating ever widening gap in perceived time is known as clock drift.
- The difference between two clocks at any point in time is called clock skew and is due to both clock drift and the possibility that the clocks may have been set differently on different machines.
- Fig. 2.1.3 shows the drift rate of clocks

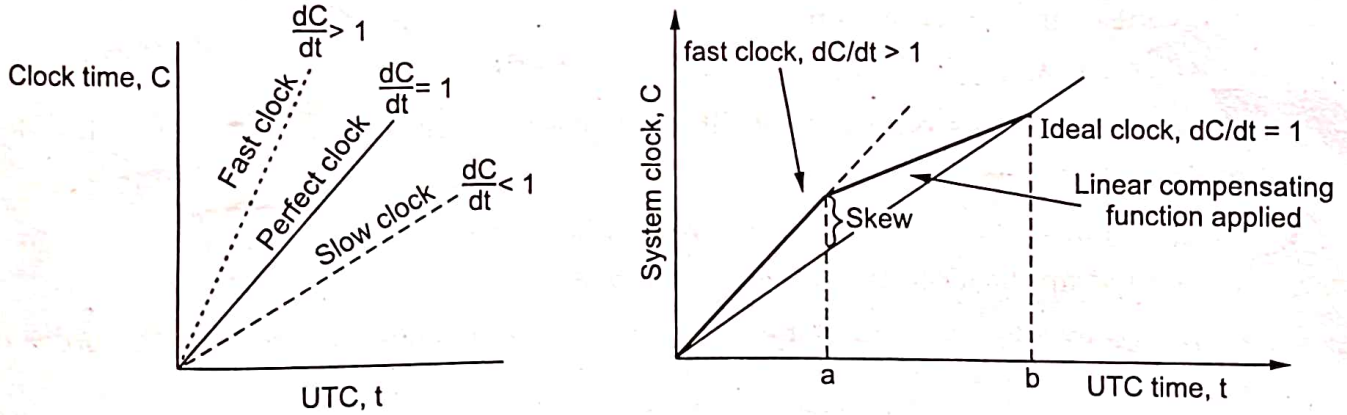


Fig. 2.1.3 Drift rate of clocks

- If a clock is fast, it simply has to be made to run slower until it synchronizes. If a clock is slow, the same method can be applied and the clock can be made to run faster until it synchronizes.
- The operating system can do this by changing the rate at which it requests interrupts. For example, suppose the system requests an interrupt every 17 milliseconds and the clock run a bit too slowly. The system can request interrupts at a faster rate, say every 16 or 15 milliseconds, until the clock catches up. This adjustment changes the slope of the system time and is known as a linear compensating function.

2.2 Logical Time

- For a certain class of algorithms, it is the internal consistency of the clocks that matters. The convention in these algorithms is to speak of logical clocks.
- Lamport showed clock synchronization need not be absolute. What is important is that all processes agree on the order in which events occur.
- A logical clock Cp of a process p is a software counter that is used to timestamp events executed by p so that the happened-before relation is respected by the timestamps.

2.2.1 Event Ordering

- A person with one watch knows what time it is. But a person with two or more watches is never sure.
- Lamport defined a relation called **happens before,** represented by $\rightarrow$.
- The relation $\rightarrow$ on a set of events of a system is the relation satisfying three conditions :

Conditions of happens before

1. If **a** and **b** are events in the same process, and **a** comes before **b**, then a $\rightarrow$ b.
2. If **a** is the sending event of a message **msg** by one process and **b** is the receipt event of **msg**, then a $\rightarrow$ b.
3. If **a** $\rightarrow$ **b, b**$\rightarrow$ **c,** then **a** $\rightarrow$ **c.**

- The order of events occurring at different processes can be critical in a distributed application.
- To determine the order of the events that occur at different processes in a distributed application, two obvious points are :
  - If two events occurred at the same process, then they occurred in the order in which it observes them.
  - Whenever a message is sent between processes, the event of sending the message occurred before the event of receiving the message.
- The ordering obtained by generalizing these two relationships is called **happen-before** relation.
- Lamport developed a "happens before" notation to express the relationship between events : a < b means that a happens before b.
- If a represents the timestamp of a message sent and b is the timestamp of that message being received, then a < b must be true; a message cannot be received before it is sent. This relationship is transitive.

- If a → b and b → c then a → c. If a and b are events that take place in the same process the a < b is true if a occurs before b.
- The importance of measuring logical time is in assigning a time value to each event such that everyone will agree on the final order of events. That is, if a→b then clock(a) < clock(b) since the clock must never run backwards.
- If a and b occur on different processes that do not exchange messages then a→b is not true. These events are said to be concurrent : there is no way that a could have influenced b.
- We write x → p y if two events x and y occurred at a single process p and x occurred before y. Using this restricted order we can define the happened-before relation, denoted by → , as follows :
  1. HB1 : If ∃ process p : x → py, then x → y.
  2. HB2 : For any message m, send(m) → rcv(m),

  where send(m) is the event of sending the message, and rcv(m) is the event of receiving it.
  3. HB3 : If x, y and z are events such that x → y and y → z, then x→ z.
- If x → y, then we can find a series of events occurring at one or more processes such that either HB1 or HB2 applies between them. The sequence of events need not be unique.
- If two events are not related by the → relation (i.e., neither a→ b nor b→a), then they are concurrent (a || b).
  a → b → c → d f ; e → f but a || e.
- **Example : Event ordering**

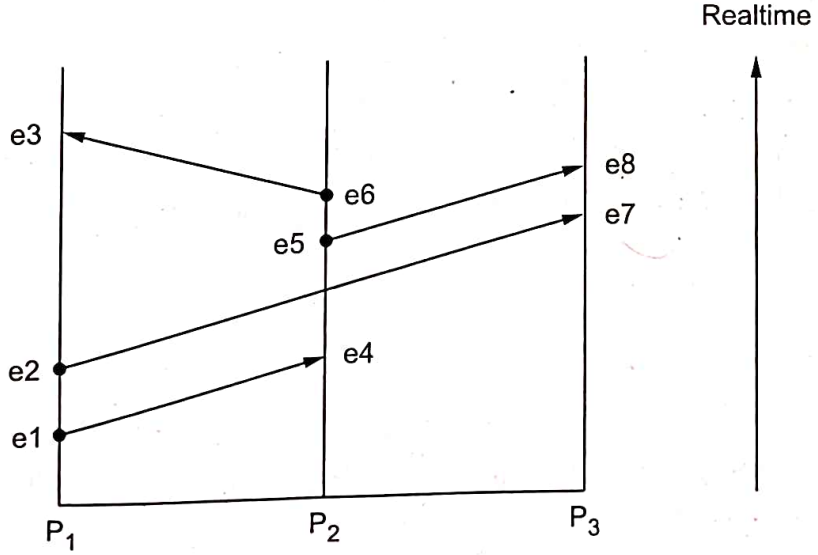


Fig. 2.2.1 Event ordering

- Possible event ordering :
  1. e1 → e2 → e3
  2. e1 → e4 → e5 → e6 → e3
  3. e2 → e7 → e8
- Event ordering is not possible in this condition : e2 → e4 and e5 → e7

Logical clock condition :

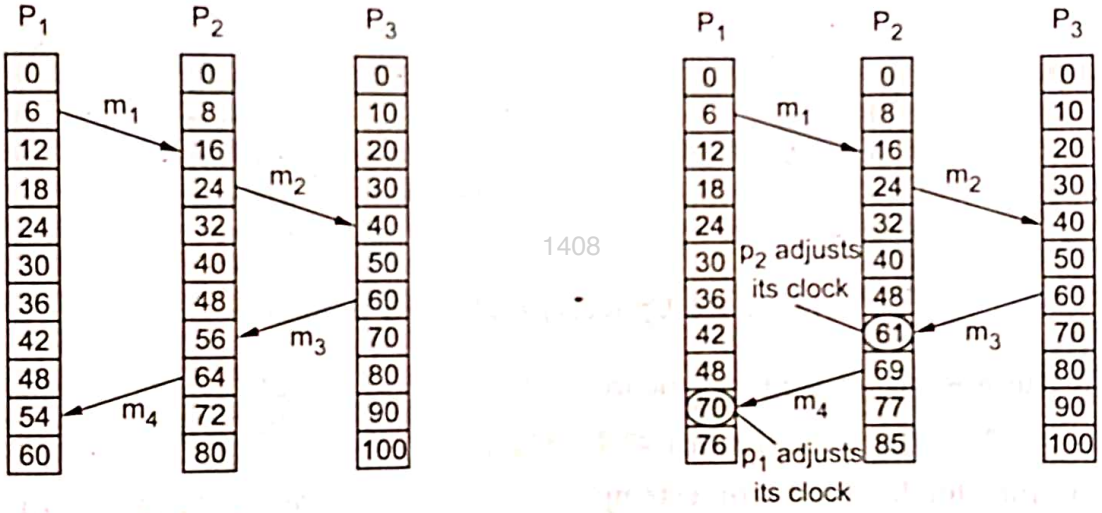
1. For any events a and b, if a → b then C(a) < C(b)
2. From the definition, the clock condition is satisfied if the following two conditions hold :
   i. Condition 1 : If a and b are events in Pi, and a comes before b, then Ci(a) < Ci(b).
   ii. Condition 2 : If a is the sending of a msg by Pi and b is the receipt of the msg by Pj, then Ci(a) < Cj(b).

2.2.2 Lamport Timestamp

- Lamport algorithm is used to synchronize the logical clock. It uses happens-before relationship to provide a partial ordering of events :
  1. All processes use a counter (clock) with initial value of zero.
  2. The counter is incremented by and assigned to each event, as its timestamp.
  3. A send (message) event carries its timestamp.
  4. For a receive (message) event the counter is updated by

     Max (receiver-counter, message-timestamp) + 1
- Goal of the lamport algorithm is to assign totally ordered timestamps to events.
- **Requirements**
  a. if a → b then T(a) < T(b)
  b. T(a) ≠ T(b) for any two different events a and b
- Happens-before relation is a transitive, so if a → b and b → c then a → c. if two events , x and y happen in different processes that do not exchange messages, then x → y is not true.
- **Algorithm**
  a. if C_i (a) ≠ C_j (b), then T(a) = Ci(a) and T(b) = Ci(b)
  b. if C_i (a) = C_j (b) and i < j, then T(a) < T(b)
- Consider three processes where each process runs on different machines with its own clock and speed. It is shown in Fig. 2.2.2.

- The processes run on different machines, each with its own clock and running with own speed.
- When the clock has ticked 6 times in process P_1, it has ticked 8 times in process P_2 and 10 times in process P_3. Each clock runs at a constant rate, but rate varies according to the crystals.
- At time 6, process P_1 sends message m_1 to process P_2. The clock in process 2 reads 16 when it arrives. Process 2 will conclude that it took 10 ticks to reach from process 1 to process 2.
- According to this reasoning, message m_2 from process 2 to process 3 takes 16 ticks.
- In Fig. 2.2.2 (a) , message m_3 from process 3 to process 2 leaves at 60 and arrives at 56. Similarly, message m4 from process 2 to process 2 leave at 64 and arrive at 54. These values are not possible.



(a) Process with its own clock runs at different rate

(b) Lamport's algorithm

Fig. 2.2.2 Clock timestamp

- Lamport solution is given in Fig. 2.2.2 (b) which uses happen-before relation. Since message m3 left at 60, it must arrive at 61 or later. So each message carry its sending time according to the sender's clock.
- When message arrives and the receiver's clock shows a value prior to the time the message was sent, the receiver fast forwards its clock to be one more than sending time.

Totally ordered multicasting :

- We can use the logical clocks satisfying the clock condition to place a total ordering on the set of all system events. Simply order the events by the times at which occur.

- To break the ties, lamport proposed the use of any arbitrary total ordering of the processes, i.e. process id
- Using this method, we can assign a unique timestamp to each event in a distributed system to provide a total ordering of all events. Very useful in distributed system for solving the mutual exclusion problem
- We sometimes need to guarantee that concurrent updates on a replicated database are seen in the same order everywhere :
- P_1 adds ₹ 100 to an account (initial value: ₹ 1000)
- P_2 increments account by 1 %
- There are two replicas. Fig. 2.2.3 shows the updating a replicated database and leaving it in an inconsistent state.

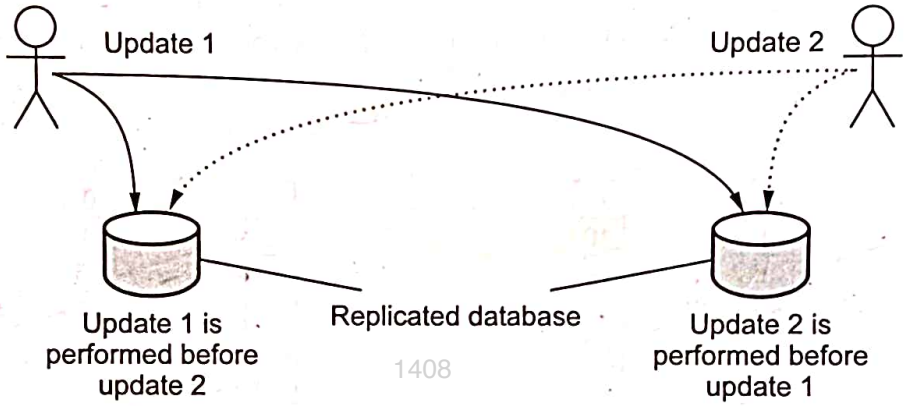


Fig. 2.2.3 Updating replicated database

Result : In absence of proper synchronization :

replica #1 ₹ : 1111, while replica #2 ₹ : 1110.

- Examples for Lamport's timestamp

Initial condition : C(P) = 0, C(Q) = 2, C(R) = 0

Processor ids : pid(P) = 0, pid(Q) = 1, pid(R) = 2

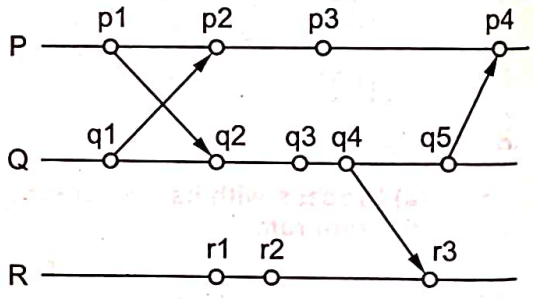


Fig. 2.2.4

Partially ordered Lamport's timestamps :

1. p1 = 1, p2 = 4, p3 = 5, p4 = 8
2. q1 = 3, q2 = 4, q3 = 5, q4 = 6, q5 = 7
3. r1 = 1, r2 = 2, r3 = 7

Totally ordered Lamport's timestamps :

1. p1 = 10, p2 = 40, p3 = 50, p4 = 80
2. q1 = 31, q2 = 41, q3 = 51, q4 = 61, q5 = 71
3. r1 = 12, r2 = 22, r3 = 72

Limitation of Lamport's logical clock

- If $a \rightarrow b$ then $C(a) < C(b)$, but if $C(a) < C(b)$ then $a \rightarrow b$ is not necessary true, i.e. concurrency information is lost

2.2.3 Vector Timestamp

- With Lamport's clocks, one cannot directly compare the timestamps of two events to determine their precedence relationship.
- The main problem is that a simple integer clock cannot order both events within a process and events in different processes. Collin Fidge developed an algorithm that overcomes this problem.
- A vector clock system is a mechanism that associates timestamps with events (local states) such that comparing two events' timestamps indicates whether those events (local states) are causally related.
- Fidge's clock is represented as a **vector** [C1 , C2 , ..., Cn] with an integer clock value for each process (Ci contains the clock value of process i) .
- Each process Pi has an array VCi [1..n], where VCi [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 VCi [i] and sends VCi along with m as vector timestamp vt(m). Result : upon arrival, recipient knows Pi 's timestamp.
- When a process Pj delivers a message m that it received from Pi with vector timestamp ts(m), it updates each VCj [k] to max{VCj [k], ts(m)[k]} and increments VCj [j] by 1.
- We can now ensure that a message is delivered only if all causally preceding messages have already been delivered.

Adjustment

- Pi increments VCi [i] only when sending a message, and Pj "adjusts" VCj when receiving a message (i.e., effectively does not change VCj [j]).
- In the time-stamping system, each process Pi has a vector of integers VCi[1..n] (initialized to [,0,...,0]) that is maintained as follows :
- Each time process Pi produces an event (send, receive, or internal), it increments its vector clock entry VCi[i] (VCi[i] : = VCi[i]+1) to indicate that it has progressed.
- When a process Pi sends a message m, it attaches to it the current value of VCi. Let m.VC denote this value.
- When Pi receives a message m, it updates its vector clock as
- Note that VCi[i] counts the number of events that Pi has so far produced.

$$\forall x: VC_i[x] := \max(VC_i[x], m.VC[x]) \text{ (abbreviated as } VC_i := \max(VC_i, m.VC).$$

- VCi[j] represents the number of events Pj produced that belong to the current causal past of Pi. When a process Pi produces an event e, it can associate with that event a vector timestamp whose value equals the current value of VCi.

Example : Assign the Lamport's clock values for all the events in the above timing diagram. Assume that each process's logical clock is set to 0 initially.

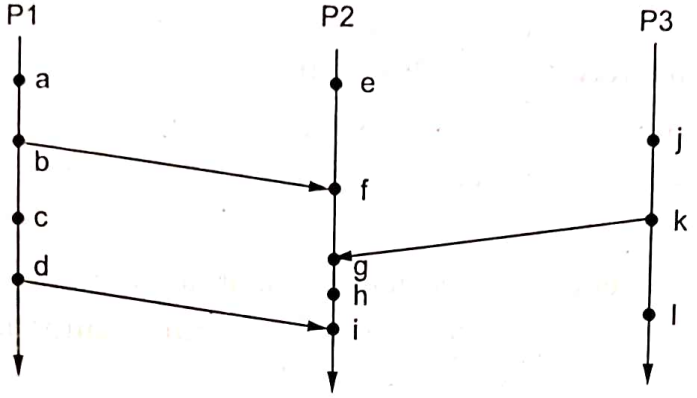


Fig. 2.2.5

Solution : Lamport clocks :

$2 < 5$ since $b \rightarrow h$,

$3 < 4$ but $c \rightarrow g$.

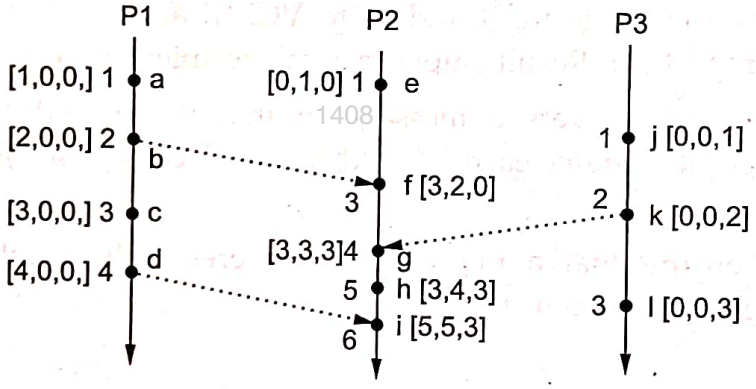


Fig. 2.2.6

2.3 Physical Clock Synchronization : NTP

- Clock synchronization is done by two methods :
  1. Internal synchronization
  2. External synchronization

External synchronization

- External synchronization refers to synchronization of process' clocks C_i with an authoritative external source S. Let $D > 0$ be the synchronization bound and S be the source of UTC. Then $|S(t) - C_i(t)| < D$ for i = 1, 2, ..., N and for all real times t.
- We say that clocks C_i are accurate within the bound of D.

Internal synchronization

- Internal synchronization refers to synchronization of process' clocks C_i with each other. Let $D > 0$ be the synchronization bound and C_i and C_j are clocks at processes pi and pj, respectively.
- Then $| C_i(t) - C_j(t) | < D$ for i, j = 1, 2, ..., N and for all real times t. We say that clocks C_i, C_j agree within the bound of D.
- Note that clocks that are internally synchronized are not necessarily externally synchronized. i.e., eventhough they agree with each other, the drift collectively from the external source of time. if the set of processes P is synchronized externally within a bound D, it is also internally synchronized within bound 2D.

Clock correctness

- A hardware clock, H is said to be correct if its drift rate is within a bound $\rho > 0$. (e.g. 10-6 secs/ sec).
- This means that the error in measuring the interval between real times t and t' is bounded :
  $(1-\rho)(t'-t) = H(t) - H(t) = (1+\rho)(t'-t)$ (where $t' > t$)
- Weaker condition of monotonicity may suffice : $t' > t$? $C(t) > C(t)$ that is, a clock C only ever advances and it can achieve monotonicity with a hardware clock that runs fast by adjusting the values of α and β in $C_i(t) = \alpha H_i(t) + \beta$.
- A **faulty clock** is one that does not obey its correctness condition. A **clock's crash failure** is said to occur when the clock stops ticking altogether. Arbitrary **failure** is any other failure e.g. jumps in time.

2.3.1 Synchronization in a Synchronous System

- A synchronous distributed system is one in which the following bounds are defined :
  1. The time to execute each step of a process has known lower and upper bounds.
  2. Each message transmitted over a channel is received within a known bounded time.
  3. Each process has a local clock whose drift rate from real time has a known bound.
- One process p_1 sends its local time t to process p_2 in a message m. The receiving process p_2 could set its clock to $t + t_{trans}$ where T_{trans} is the time to transmit m. T_{trans} is unknown but $min = T_{trans} = max$.
- So, set $T_{trans} = min + max/2$ and receiver's clock = t + (min + max) / 2

2.3.2 Cristian's Method for Synchronizing Clocks

- The clock requests time and synchronizes only if the round-trip is within a certain bound.

2.3.2.1 Christian's Algorithm

- **Christian's algorithm is centralized passive time server type algorithm.**
- The simplest algorithm for setting the time would be to simply issue a remote procedure call to a time server and obtain the time.
- **Assumptions :** There is a machine with WWV receiver, which receives precise UTC . It is called the **time server.**

Algorithm :

1. A machine sends a request to the time server at least every d/2r seconds, where d is the maximum difference allowed between a clock and the UTC.
2. The time server sends a reply message with the current UTC when receives the request.
3. The machine measures the time delay between time serve's sending the message and the machine's receiving the message. Then, it uses the measure to adjust the clock.

- Fig. 2.3.1 shows getting the current time from a time server.

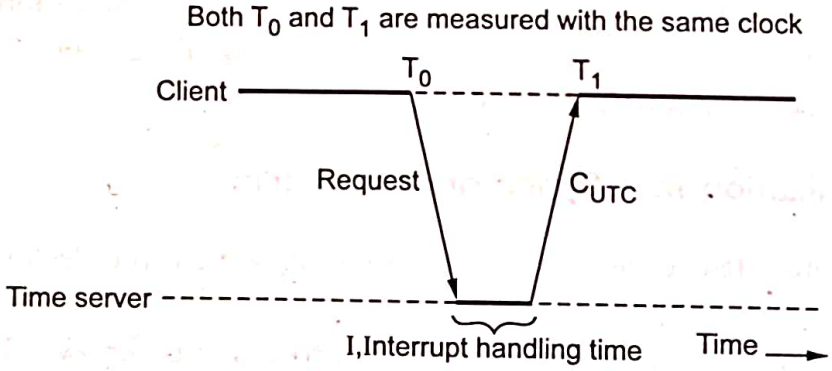


Fig. 2.3.1

- Both the starting time T_0 and the ending time T_1 are measured using the same clock, so the arrival will be relatively accurate even if the sender's clock is off from UTC by a substantial amount.
- The best estimate of the message propagation time = $(T_0 + T_1)/2$
- The new time can be set to the time returned by the server plus the time that elapsed since the server generated the timestamp :

$$T_{new} = T_{server} + (T_0 + T_1) / 2$$

- **Adjust the clock :**
  1. If the local clock is faster than the UTC, add less to the time memory for each clock tick.
  2. If the local clock is slower than the UTC, add more to the time memory for each clock tick.
- Cristian's algorithm suffers from the problem that afflicts all single-server algorithms : the server might fail and clock synchronization will be unavailable. It is also subject to malicious interference.

2.3.3 Berkeley Algorithm

- Time server is a active machine.
- The server polls each machine periodically, asking it for the time. The time at each machine may be estimated by using Cristian's method to account for network delays.
- When all the results are in, the master computes the average time (including its own time in the calculation).
- Instead of sending the updated time back to the slaves, which would introduce further uncertainty due to network delays, it sends each machine the offset by which its clock needs adjustment. The operation of this algorithm is shown in Fig. 2.3.2.

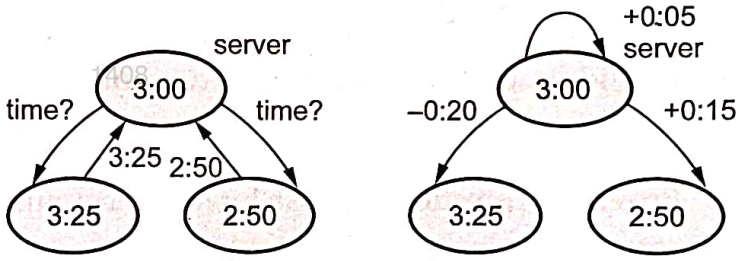


Fig. 2.3.2 Berkeley algorithm

- Three machines have times of 3:00, 3:25, and 2:50. The machine with the time of 3:00 is the server. It sends out a synchronization query to the other machines in the group.
- Each of these machines sends a timestamp as a response to the query. The server now averages the three timestamps : the two it received and its own, computing

  = (3:00+3:25+2:50)/3 = 3:05
- Now it sends an offset to each machine so that the machine's time will be synchronized to the average once the offset is applied. The machine with a time of 3:25 gets sent an offset of - 0:20 and the machine with a time of 2:50 gets an offset of +0:15. The server has to adjust its own time by +0:05.
- The algorithm also has provisions to ignore readings from clocks whose skew is too great. The master may compute a fault-tolerant average i.e. averaging values

from machines whose clocks have not drifted by more than a certain amount. If the master machine fails, any other slave could be elected to take over.

2.3.4 Network Time Protocol

- Cristian's method and the Berkeley algorithm are intended for intranets.
- The Network Time Protocol (NTP) is yet another method for synchronizing clocks that uses a hierarchical architecture where the top level of the hierarchy are servers connected to a UTC time source such as a GPS unit.

2.3.4.1 Localized Averaging Distributed Algorithms

- Each node exchanges its clock time with its neighbors.
- Then sets its clock time to the average of its own clock and the clock times of its neighbors

Network time protocol

Goals :

1. Enable clients across the Internet to be accurately synchronized to UTC despite message delays.
2. Provide a reliable service that can survive lengthy losses of connectivity.
3. Enable clients to synchronize frequently and offset the effects of clock drift.
4. Provide protection against interference; authenticate that the data is from a trusted source.

- The NTP servers are connected in a logical hierarchy, where servers in level n are synchronized directly to those in level n–1. The logical hierarchy can be reconfigured as servers become unreachable or failed.
- NTP servers synchronize with one another in one of three modes in the order of increasing accuracy :
  1. Multicast on high speed local LANs
  2. Procedure call mode
  3. Symmetric mode

Features of NTP :

1. To provide a service enabling clients across the Internet to be synchronized accurately to UTC; despite the large and variable message delays encountered in Internet communication.
2. To provide a reliable service that can survive lengthy losses of connectivity.

3. To enable clients to resynchronize sufficiently; to offset the rate of drift found in most computers.
4. To provide protection against interference with the time service; whether malicious or accidental. The time service uses authentication techniques to check that timing data originate from the claimed trusted sources. It also validates the return addresses of messages sent to it.

- NTP servers synchronize modes are as follows :
  1. Multicast
  2. Procedure-call
  3. Symmetric mode.

1. Multicast mode :

- Multicast mode is intended for use on a high-speed LAN.
- One or more servers periodically multicasts the time to the servers running in other computers connected by the LAN, which set their clocks assuming a small delay.
- This mode can only achieve relatively low accurate, but ones which nonetheless are considered sufficient for many purposes.

2. Procedure-call mode :

- Procedure call mode is similar to the operation of Cristian's algorithm.
- In this mode, one server accepts requests from other computers, which it processes by replying with its timestamp.
- This mode is suitable where higher accuracies are required that can be achieved with multicast or where multicast is not supported in hardware.
- For example, file servers on the same or a neighboring LAN, which need to keep accurate timing information for file accesses, could contact a local master server in procedure-call mode.

3. Symmetric mode :

- Symmetric mode is intended for use by the master servers that supply time information in LANs and by the higher levels of the synchronization subnet, where the highest accuracies are to be achieved.
- A pair of servers operating in symmetric mode exchange messages bearing timing information.
- Timing data are retained as part of an association between the servers that is maintained in order to improve the accuracy of their synchronization over time.
- In all modes, messages are delivered unreliably, using the standard UDP Internet transport protocol. In procedure-call mode and symmetric mode, messages are exchanged in pairs.

2.4 A Framework for a System of Logical Clocks

- The logical time in distributed systems is used to maintain the consistent ordering of events. The concept of causality, i.e. the causal precedence relationship, is fundamental for distributed systems. Usually, it is tracked using physical time, but physical clocks are hard to maintain in a distributed system, so logical clocks are used instead.
- A system of logical clocks consists of a time domain T and a logical clock C. Elements of T form a partially ordered set over a relation <.
- Relation < is called the happened before or causal precedence.
- The logical clock C is a function that maps an event e in a distributed system to an element in the time domain T, denoted as C(e) and called the timestamp of e, and is defined as follows :

$C : H \rightarrow T$

such that the following property is satisfied :

for two events e_i and e_j, $e_i \rightarrow e_j \Rightarrow C(e_i) < C(e_j)$

- This monotonicity property is called the clock consistency condition. When T and C satisfy the condition, for two events e_i and e_j, $e_i \rightarrow e_j \Leftrightarrow C(e_i) < C(e_j)$ the system of clocks is said to be strongly consistent.

Implementing logical clocks :

- Implementation of logical clocks requires addressing following issues :
  a) Data structures local to every process to represent logical time.
  b) Protocol to update the data structures to ensure the consistency condition.
- Each process p_i maintains data structures that allow it the following two capabilities :
  a) Local logical clock (lc_i) helps process p_i measure its own progress.
  b) Logical global clock (gc_i) which is a representation of process p_i's local view of the logical global time.
- Logical time has many applications in distributed systems.
  a) It is used to resolve conflicts, track individual events, and take concurrency measures.
  b) Logical time also has its application in debugging distributed systems. With all the information about the causal relationship of events, it is easy to spot which event caused the error.
- Systems of logical clocks differ in their representation of logical time and also in the protocol to update the logical clocks. Three kinds of logical clocks are Scalar, Vector and Matrix.

2.5 Scalar Time

- Scalar time is designed by Lamport to synchronize all the events in distributed systems. Time domain is the set of non-negative integers.
- The logical local clock of a process p_i and its local view of the global time are squashed into one integer variable c_i.
- Each process keeps its own logical clock used to timestamp events. Fig 2.5.1 shows the space-time diagram of a distributed execution.

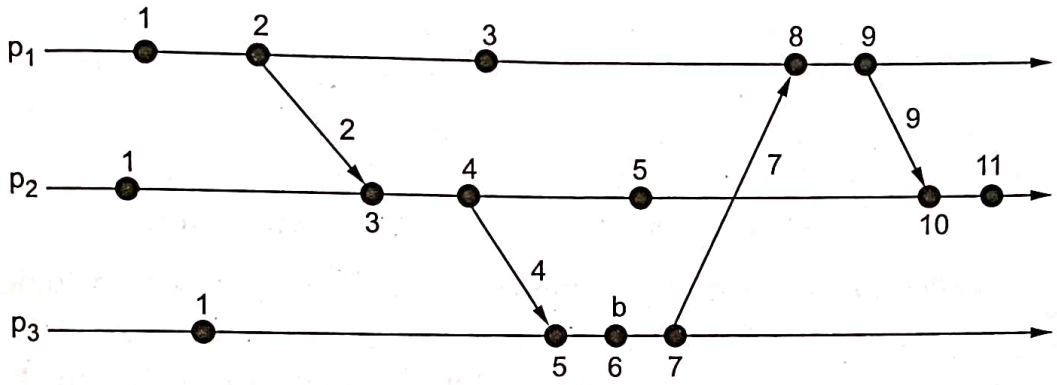


Fig. 2.5.1 The space-time diagram of a distributed execution

Basic Properties :

1. **Consistency property :** Scalar clocks satisfy the monotonicity and hence the consistency property for two events e_i and e_j, $e_i \rightarrow e_j \Rightarrow C(e_i) < C(e_j)$.
2. **Total ordering :** Scalar clocks can be used to totally order events in a distributed system. The main problem in totally ordering events is that two or more events at different processes may have identical timestamp. For example in Fig. 2.5.1, the third event of process p_1 and the second event of process p_2 have identical scalar timestamp. A tie-breaking mechanism is needed to order such events.
3. **Event counting :** If the increment value d is always 1, the scalar time has the following property : if event e has a timestamp h, then h^{-1} represents the minimum logical duration, counted in units of events, required before producing the event e; We call it the height of the event e. In other words, h^{-1} events have been produced sequentially before the event e regardless of the processes that produced these events. For example, in Fig. 2.5.1, five events precede event b on the longest causal path ending at b.
4. System of scalar clocks is not strongly consistent.

2.6 Vector Time

- Vector clocks are used in a distributed systems to determine whether pairs of events are causally related. Using vector clocks, timestamps are generated for each event in the system, and their causal relationship is determined by comparing those timestamps.
- A vector clock of a system of N processes is an array/vector of N logical clocks, one clock per process; a local "smallest possible values" copy of the global clock-array is kept in each process, with the following rules for clock updates :
  1. Initially all clocks are zero.
  2. Each time a process experiences an internal event, it increments its own logical clock in the vector by one.
  3. Each time for a process to send a message, it must increment its own clock (as in the bullet above) and then send a copy of its own vector.
  4. Each time a process receives a message, it increments its own logical clock in the vector by one and updates each element in its vector by taking the maximum of the value in its own vector clock and the value in the vector in the received message (for every element).

2.7 Message Ordering Paradigms

AU : May-22, Dec.-22

- Message delivery in orderly manner is important because it determines the messaging behaviour that can be expected by the distributed program.
- Message orders are non-FIFO, FIFO, causal order, synchronous order etc.
- An **asynchronous execution** is an execution for which the causality relation is a partial order.
- There are no causality cycles. On any logical link, not necessarily FIFO delivery. For example : Network layer in IPv4 is connectionless service.
- All physical links obey FIFO rule.
- **FIFO execution** is an A-execution in which for all (s, r) and (s′, r′) ∈ T, (s ~ s′ and r ~ r' and s < s′) ⇒ r < r′.
- Logical link is inherently non-FIFO. It can assume connection-oriented service at transport layer. For example: TCP
- To implement FIFO over non-FIFO link, use sequence number and connection-id for each message. Receiver uses buffer to order messages.
- **Causally order** execution is an A-execution in which for all (s, r) and (s′, r′) ∈ T, (r ~ r' and s < s′) ⇒ r < r′.

- If two send events are related by causality ordering, then a causally ordered execution requires that their corresponding receive events occur in the same order at all common destinations.

University Questions

1. *Discuss in detail about message ordering paradigms.* **AU : May-22, Marks 13**
2. *What are the four different types of ordering the messages ? Explain.* **AU : Dec.-22, Marks 13**

2.8 Asynchronous Execution with Synchronous Communication

- **Synchronous order :** All the communication between pairs of processes is by using synchronous send and receive primitives.
- Both send and receive message event appears instantaneous. Algorithm that runs on an asynchronous system may deadlock on a synchronous system.
- Communication program for an asynchronous system deadlocks when using synchronous primitives.

Process A	Process B
.....	.....
Send(B)	Send(A)
Receive(B)	Receive(A)
.....	.....

- An algorithm that runs on an asynchronous system may deadlock on a synchronous system.

2.8.1 Execution Realizable with Synchronous Communication

- An execution is realizable with synchronous communication if each send event is immediately followed by its corresponding receive event.
- If the couple (send event, corresponding receive event) is viewed atomically, this corresponds to a synchronous communication execution.
- Non-separated linear extension of (E,<) : A linear extension of (E,<) such that for each pair (s, r) $\in$ T, the interval $\{ x \in E \mid s < x < r \}$ is empty.
- An A-execution (E,<) is an RSC execution if there exists a non-separated linear extension of the partial order (E,<).

Crown definition :

- Let E be an execution. A crown of size k in E is a sequence $< (s^i, r^i), i \in \{ 0, ..., k - 1 \} >$ of pairs of corresponding send and receive events such that : $S^0 < r^1, S^1 < r^2, \ldots \ldots S^{k-1} < r^{k-1}, s^{k-1} < r^0$.
- In crown, send and receive event may lie on the same process.

2.8.2 Hierarchy of Message Ordering Paradigms

- Fig. 2.8.1 shows hierarchy of the Communication Models.
- The executions which are valid for a given model may also be valid for another model. This means that, for these executions, we can switch whatever of these models is considered. Knowing which model can be replaced by which is of great interest to study the potential compatibility of peers.

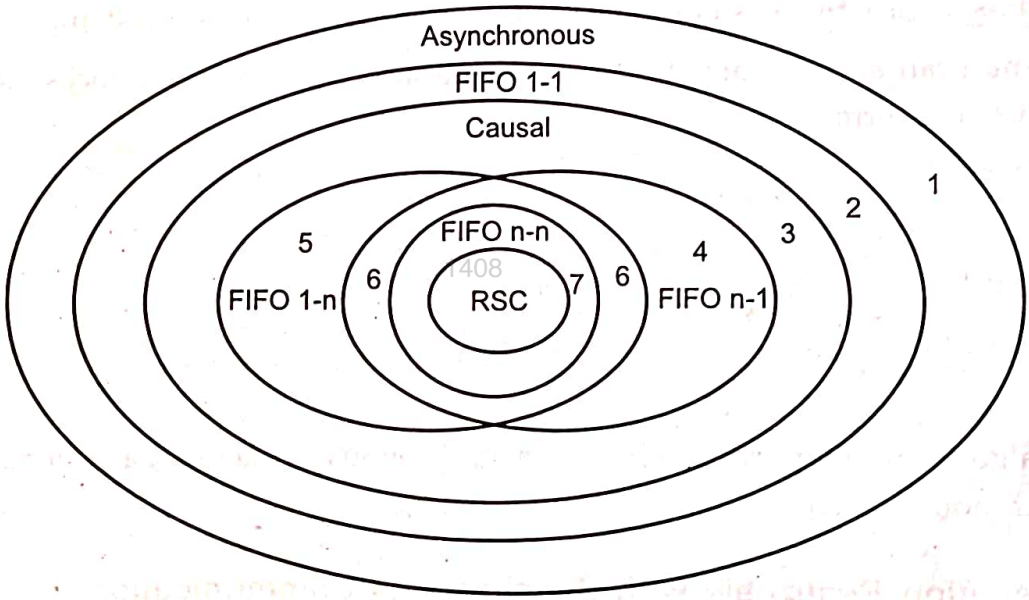
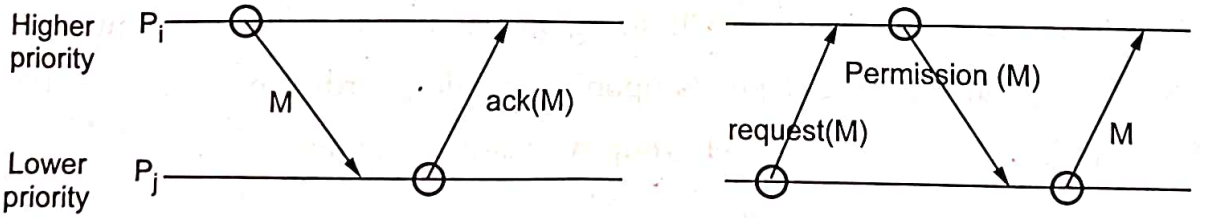


Fig. 2.8.1 Hierarchy of the Communication Models

- An A-execution is realizable with synchronous communication, if and only if A is an S-execution.
- More restrictions on the possible message orderings in the smaller classes. The degree of concurrency is most in A, least in SYNC.
- A program using synchronous communication easiest to develop and verify. A program using non-FIFO communication, resulting in an A-execution, hardest to design and verify.

2.9 Synchronous Program Order on Asynchronous System

- Rendezvous is a synchronization mechanism based on procedural decomposition. Rendezvous is similar to a procedure call with the difference that the caller and the callee belong to different tasks.
- The called procedure is usually called an entry point of the corresponding task. A call to an entry point is synchronous, i.e. the caller is blocked until completion
- Rendezvous is an architecture for creating multi-user applications. It provides support for managing a multi-user session, for performing fundamental input and output activities, and for controlling the degree to which the multiple users either share or do not share both information and control.
- **Binary rendezvous** implementation using tokens. Token for each enabled interaction schedule online, atomically, in a distributed manner crown-free scheduling; also progress to be guaranteed fairness and efficiency in scheduling.
- Bagrodia's Algorithm for Binary rendezvous :
  Assumptions :
  1. Receives are always enabled
  2. Send, once enabled, remains enabled
  3. To break deadlock, PIDs used to introduce asymmetry
  4. Each process schedules one send at a time
- Message types: M, ack(M), request(M), permission(M)
- Process blocks when it knows it can successfully synchronize the current message.
- Fig. 2.9.1 shows high and lower priority process blocks.



(a) High priority process block (b) Low priority process does not block

Fig. 2.9.1

2.10 Group Communication

AU : Dec.-22

- Groups are dynamic. They may be created and destroyed. Processes may join or leave groups and processes may belong to multiple groups.
- Groups allow processes to deal with collections of processes as one abstraction. Ideally, a process should only send a message to a group and need not know or care who its members are.
- Group communication can be implemented in several ways.
  1. One to many 2. Many to one 3. Many to many

2.10.1 One to Many Communication

- Message is sent by one sender to multiple receivers.
- One-to-many scheme is also known as multicast communication. A special case of multicast communication is broadcast communication.

Group management

- Here receiver processes forms a group. Groups are of two types : **Closed** and **open.**
- **Closed group :** Only the members of the group can send a message to the group. An outside process cannot send a message to the group as a whole, although it may send a message to an individual member of the group.
- **Open group :** Any process in the system can send a message to the group as a whole.
- Use of open or closed group is depends upon application.

Group addressing

- A two-level naming scheme is normally used for group addressing. These are high level and low level group name.
- High level group naming is ASCII string and it is location independent.
- Low level group naming depends upon underlying hardware.
- User applications use high level group names in programs.

Buffered and unbuffered multicast

- Multicasting is an asynchronous communication mechanism.
- This is because multicast send cannot be synchronous due to the following reasons :
  a. It is unrealistic to expect a sending process to wait until all the receiving processes that belong to the multicast group are ready to receive the multicast message.

b. The sending process may not be aware of all the receiving processes that belong to the multicast group.

- **Un-buffered multicast :** Message is not buffered for the receiving process and is lost if the receiving process is not in a state ready to receive it. Therefore, the message is received only by those processes of the multicast group that are ready to receive it.
- **Buffered multicast :** Message is buffered for the receiving process, so each process of the multicast group will eventually receive the message.

Send-to-all and bulletin-board semantics

Following are two types of semantics for one-to-many communications :

a. **Send-to-all semantics :** A copy of the message is sent to each process of the multicast group and message is buffered until it is accepted by the process.

b. **Bulletin-board semantics :** A message to be multicast is addressed to a channel instead of being sent to every individual process of the multicast group.

Flexible reliability in multicast communication

- Different applications require different degrees of reliability.
- In one-to-many communication, the degree of reliability is normally expressed in the following forms :
  a. **0-reliability :** No response is expected by the sender from any of the receivers.
  b. **1-reliability :** The sender expects a response from any of the receivers.
  c. **m-out-of-n-reliable :** The multicast group consists of n receivers and the sender expects a response from m ($1 < m < n$) of the receivers.
  d. **All-reliable :** The sender expects a response message from all the receivers of the multicast group.

Atomic multicast

- Atomic multicast has an all-or-nothing property.
- When a message is sent to a group by atomic multicast, it is either received by all the correct processes that are members of the group or else it is not received by any of them.

2.10.2 Many-to-One Communication

- Multiple senders send messages to a single receiver. Single receiver may be selective or nonselective.
- **Selective receiver :** Specifies a unique sender and message exchange takes place only if that sender sends a message.

- **Nonselective receiver :** Specifies a set of sender and if any one sender in the set sends a message to this receiver a message exchange takes place.
- Many-to-one scheme is non-determinism.

2.10.3 Many-to-Many Communication

- Multiple sender send messages to multiple receivers.
- Ordered message delivery ensures that all messages are delivered to all receivers in an order acceptable to the application.
- **For example :** Suppose two senders send messages to update the same record of a database to two server processes having a replica of the database. If the message of the two senders is received by the two servers in different orders, then the final values of the updated record of the database may be different in its two replicas.

2.10.3.1 Message Ordering

- R1 and R2 receive m1 and m2 in a different order. Fig. 2.10.1 shows no ordering constraint for message delivery.
- Some message ordering required
  1. Absolute ordering
  2. Consistent/Total ordering
  3. Causal ordering
  4. FIFO ordering

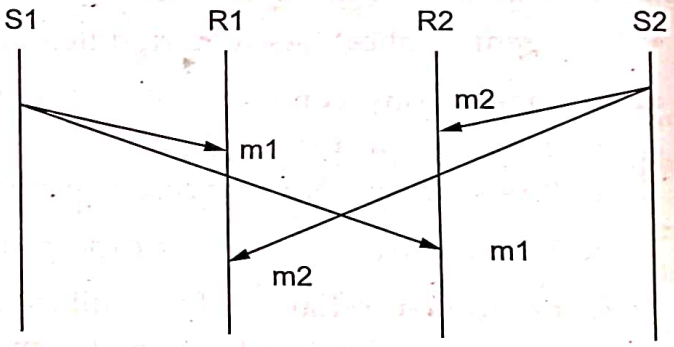


Fig. 2.10.1 No ordering

Absolute ordering

- Fig. 2.10.2 shows absolute ordering. In this ordering, all messages are delivered to all receiver processes in the exact order in which they were sent.
- **Rule :** mi must be delivered before mj if Ti < Tj
- **Implementation :** A clock synchronized among machines is required. A sliding time window used to commit message delivery whose timestamp is in this window. Window size is properly chosen taking into consideration the maximum possible time that may be required by a message to go from one machine to other machine in the network.

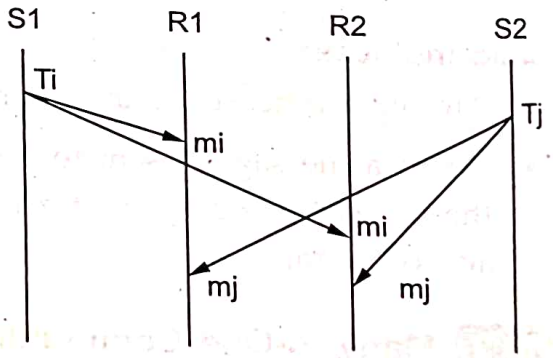


Fig. 2.10.2 Absolute ordering

- **Example :** Distributed simulation
- **Drawbacks :**
  1. Too strict constraint
  2. No absolute synchronized clock
  3. No guarantee to catch all tardy messages

Consistent / Total ordering

- It ensures that all messages are delivered to all receiver processes in the same order.
- Fig. 2.10.3 shows consistent ordering.
- **Rule :** Messages received in the same order, regardless of their timestamp.
- **Implementation :** A message sent to a sequencer, assigned a sequence number, and finally multicast to receivers. A message retrieved in incremental order at a receiver.
- **Example :** Replicated database updates
- **Drawback :** A centralized algorithm

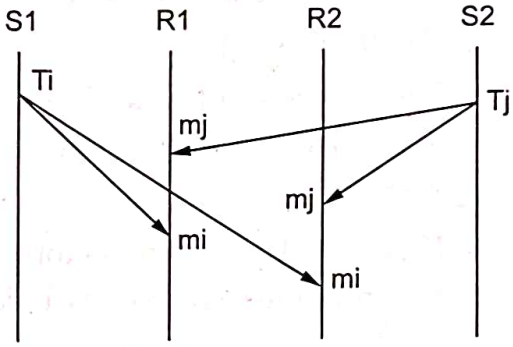


Fig. 2.10.3 Consistent ordering

Causal ordering

- If two message sending events are not causally related, the two messages may be delivered to the receivers in any order.
- Two message sending events are said to be causally related if they are correlated by the happened before relation.
- **Rule : Happened-before** relation
  - If e^k_i, $e^l_i \in h$ and $k < l$, then $e^k_i \in e^l_i$,
  - If e_i = send(m) and e_j = receive(m), then $e_i \in e_j$,
  - If $e \rightarrow e'$ and $e' \rightarrow e''$, then $e \rightarrow e''$
- **Implementation :** Use of a vector message.
- **Example :** Distributed file system
- **Drawbacks :**
  1. Vector as an overhead 2. Broadcast assumed

University Question

1. *Explain the types of group communications used in distributed system.*

AU : Dec.-22, Marks 13

2.11 Causal Order

- Causal ordering of messages was proposed by Birman and Joseph.
- Multiple sender send messages to multiple receivers. Ordered message delivery ensures that all messages are delivered to all receivers in an order acceptable to the application.
- The purpose of causal ordering of messages is to insure that the same causal relationship for the "message send" events correspond with "message receive" events.
- If send(m1) happens before send (m2), then every recipient of both messages m1 and m2 must receive m1 before m2.
- If two message sending events are not causally related, the two messages may be delivered to the receivers in any order.
- Two message sending events are said to be causally related if they are correlated by the happened before relation. Fig. 2.11.1 shows the causal ordering

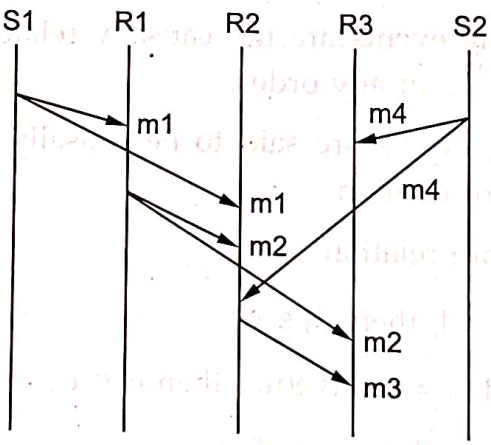


Fig. 2.11.1 Causal ordering

- Rule : Happened-before relation
  - If $e^k_i, e^l_i \in h$ and $k < l$, then $e^k_i \rightarrow e^l_i$,
  - If e_i = send(m) and e_j = receive(m), then $e_i \rightarrow e_j$,
  - If $e \rightarrow e'$ and $e' \rightarrow e''$, then $e \rightarrow e''$
- Given a system with FIFO channels, causal order needs to be explicitly enforced by a protocol.

2.11.1 Raynal-Schiper-Toueg Algorithm

- In this algorithm, each process P_i maintains an n × n matrix, $SENT_i$, where n is the total number of processes, to record the number of messages, as it has known, sent from each process to each other's.
- Every message transmitted by P_i is tagged with the contents of $SENT_i$. Each process P_i also maintains an n-entry vector, $DELIV_i$, to record the number of messages delivered to P_i from all others.
- On receiving a message, say m, process P_i can determine whether m can be delivered by comparing $DELIV_i$ with the i^{th} column of the SENT matrix tagging m. If m can be delivered, $SENT_i$ as well as $DELIV_i$ are updated, and the SENT matrix tagging m can be discarded

2.12 Total Order

AU : Dec.-22

- It ensures that all messages are delivered to all receiver processes in the same order.
- If a correct process delivers message *m* before it delivers *m'*, then any other correct process that delivers *m'* will deliver *m* before *m'*.
- Fig. 2.12.1 shows consistent ordering.

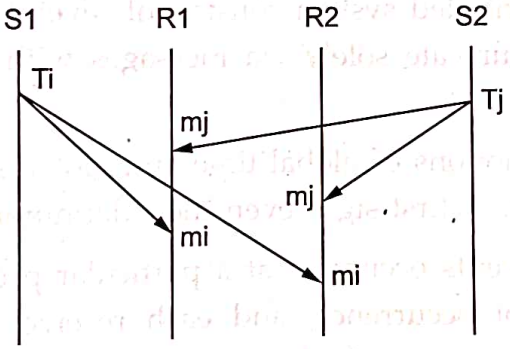


Fig. 2.12.1 Consistent ordering

- For each pair of processes P_i and P_j and for each pair of messages M_x and M_y that are delivered to both the processes, P_i is delivered M_x before M_y if and only if P_j is delivered M_x before M_y
  - **Rule :** Messages received in the same order, regardless of their timestamp.
  - **Implementation :** A message sent to a sequencer, assigned a sequence number, and finally multicast to receivers. A message retrieved in incremental order at a receiver.
  - **Example :** Replicated database updates
- **Drawback :** A centralized algorithm

2.12.1 Three Phase Distributed Algorithm

Sender :

- **Phase 1 :** Process multicast the message M with a locally unique tag and the local time stamp to the group members.
- **Phase 2 :** The sender process awaits a reply from all the group members who respond with a tentative proposal for a revised timestamp for that message.
- **Phase 3 :** The process multicasts the final timestamp to the group in line.

Receivers :

- **Phase 1 :** Receiver receives the message with a tentative timestamp. It updates the variable priority.
- **Phase 2 :** The receiver sends the revised timestamp back to the sender.
- Final timestamp is received from the multi-caster.

University Question

1. *Elucidate on the total and casual order in distributed system with a neat diagram.*

AU : Dec.-22, Marks 13

2.13 Global State and Snapshot Recording Algorithms

- An asynchronous distributed system consists of several processes without common memory which communicate solely via messages with unpredictable transmission delays.
- In such a system the notions of global time and global state play an important role but are hard to realize, at first sight even their definition is not all clear.
- Events are related : Events occurring at a particular process are totally ordered by their local sequence of occurrence, and each receive event has a corresponding send event. This relationship is the heart of any notion of virtual time.
- However, the central concept seems to be the causality relation which determines the primary characteristic of time, namely that the future cannot influence the past.
- Formally, an event structure is a pair (E; <), where E is a set of events, and `<' is an irreflexive partial order on E called the causality relation.
- Event structures represent distributed computations in an abstract way. For a given computation, e < e' holds if one of the following conditions holds:
  1. e and e' are events in the same process and e precedes e'
  2. e is the sending event of a message and e' the corresponding receive event,

2.13.1 System Model

- A distributed computing system consists of spatially separated processes that do not share a common memory and communicate asynchronously with each other by passing messages over communication channels
- There is no physical global clock in the system. Message send and receive is asynchronous. Messages are delivered reliably with finite but arbitrary time delay.
- The system can be described as a directed graph in which vertices represent the processes and edges represent unidirectional communication channels.
- Let C_{ij} denote the channel from process i to process j.
- Processes and channels have states associated with them. The state of a process at any time is defined by the contents of processor registers, stacks, local memory, etc and may be highly dependent on the local context of the distributed application.
- The state of channel C_{ij}, denoted by SC_{ij}, is given by the set of messages in transit in the channel.
- Process performed as action, it is called as event and events are of three types: internal events, message send events, and message receive events. A channel is a distributed entity and its state depends on the local states of the processes on which it is incident.

2.13.2 Consistent Global State

- The global state of a distributed system is a collection of the local states of the processes and the channels.
- Global state GS is defined as

$$GS = \{\bigcup_{i} LS_i, \bigcup_{i,j} SC_{ij}\}$$

- A global state GS is a consistent global state if and only if, it satisfies the following two conditions

C1 : send $(m_{ij}) \in LS_i \Rightarrow m_{ij} \in SC_{ij} \oplus$ rec $(m_{ij}) \in LS_j$ ($\oplus$ is Ex-OR operator)

C2 : send $(m_{ij}) \notin LSi \Rightarrow m_{ij} \notin SC_{ij} \wedge$ rec $(m_{ij}) \notin LS_j$.

- In a consistent global state, every message that is recorded as received is also recorded as sent and such a state captures the notion of causality that a message cannot be received if it was not sent.
- Consistent global states are meaningful global states and inconsistent global states are not meaningful in the sense that a distributed system can never be in an inconsistent state.

2.14 Snapshot Algorithms for FIFO Channels

AU : May-22

2.14.1 Chandy-Lamport Algorithm

- Chandy-Lamport algorithm records a set of process and channel states such that the combination is a consistent global state. Communication channels assumed to be FIFO
- **Assumptions :**
  a. No failure, all messages arrive intact, exactly once
  b. Communication channels are unidirectional and FIFO - ordered
  c. There is a communication channel between each pair of processes
  d. Any process may initiate the snapshot (send "Marker")
  e. Snapshot does not interfere with normal execution
- The Chandy-Lamport algorithm uses a control message, called a **marker** whose role in a FIFO system is to separate messages in the channels.
- After a site has recorded its snapshot, it sends a marker, along all of its outgoing channels before sending out any more messages.
- A marker separates the messages in the channel into those to be included in the snapshot from those not to be recorded in the snapshot. A process must record its snapshot no later than when it receives a marker on any of its incoming channels.

Algorithm :

1. *Initiator process P_0 records its state locally*
2. *Marker sending rule for process P_i :*

   After P_i has recorded its state, for each outgoing channel Ch_{ij}, P_i sends one marker message over Ch_{ij} (before P_i sends any other message over Ch_{ij})
3. *Marker receiving rule for process P_i :*

   Process P_i on receipt of a marker over channel Ch_{ji}

   If (P_i has not yet recorded its state) it

   Records its process state now;

   Records the state of Ch_{ji} *as empty set;*

   Starts recording messages arriving over other incoming channels;

   else (P_i *has already recorded its state)*

   Pi records the state of Ch_{ji} as the set of all messages it has received over Ch_{ji} since it saved its state.

2.14.2 Property of the Recorded Global State

- The recorded global state may not correspond to any of the global states that occurred during the computation
- Each process records its own state, and the two processes that a channel is incident on cooperate in recording the channel state.
- We cannot ensure that the states of all processes and channels will be recorded at the same instant because there is no global clock; however, we require that the recorded process and channel states form a "meaningful" global system state.
- The algorithm, may send messages and require processes to carry out computations; however, the messages and computation required to record the global state must not interfere with the underlying computation.
- Assume that the state of p is recorded in global state S_o, so the state recorded for p is A. After recording its state, p sends a marker along channel c.
- Now assume that the system goes to global state S_1, then S_2, and then S_3 while the marker is still in transit, and the marker is received by q when the system is in global state SB.
- On receiving the marker, q records its state, which is D, and records the state of c to be the empty sequence. After recording its state, q sends a marker along channel c′.
- On receiving the marker, p records the state of c' as the sequence consisting of the single message M′. The recorded global state S* is shown in Fig. 2.14.1.

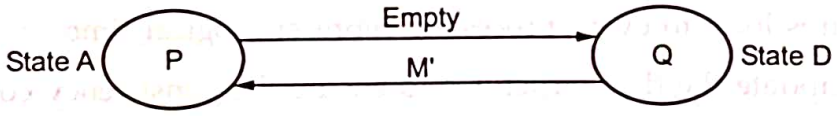


Fig. 2.14.1

- The recording algorithm was initiated in global state S_0 and terminated in global state S_3 .
- Observe that the global state S* recorded by the algorithm is not identical to any of the global states S_o, S_1, S_2, S_3 that occurred in the computation.

University Question

1. *Explain the Chandy-Lamport snapshot algorithm.* **AU : May-22, Marks 13**

2.15 Two Marks Questions with Answers

Q.1 What is meant by asynchronous programming ? AU : Dec-22

Ans. : Asynchronous programming provides opportunities for a program to continue running other code while waiting for a long-running task to complete.

Q.2 What is meant by group communication in distributed system ? AU : -May-22

Ans. : Group communication offers a service whereby a message is sent to a group and then this message is delivered to all members of the group. The sender is not aware of the identities of the receivers.

Q.3 Write application of casual order. AU : May-22

Ans. : Causal ordering is used for implementing distributed shared memory, fair resource allocation. Other applications are updating replicated data, synchronizing multimedia streams and allocating requests in a fair manner.

Q.4 What is synchronous order ?

Ans. : When all the communication between pairs of processes is by using synchronous send and receive primitives , the resulting order is synchronous order.

Q.5 Define scalar time.

Ans. : Scalar time is designed by Lamport to synchronize all the events in distributed systems. Time domain is the set of non-negative integers

Q.6 List the issue related with implementation of logical clocks.

Ans. : Addressing following issues :

- Data structures local to every process to represent logical time.
- Protocol to update the data structures to ensure the consistency condition.

Q.7 List the properties of scalar time.

Ans. : Properties of scalar time are consistency, total ordering, event counting and system of scalar clocks is not strongly consistent.

Q.8 What is Rendezvous ?

Ans. : Rendezvous is an architecture for creating multi-user applications. It provides support for managing a multi-user session, for performing fundamental input and output activities and for controlling the degree to which the multiple users either share or do not share both information and control.

Q.9 What is clock tick ?

Ans. : When the counter gets to zero, an interruption is generated and is called one clock tick.

Q.10 Define solar day.

Ans. : The interval between two consecutive transits of the sun is called the solar day.

Q.11 What is clock skew ?

AU : May-18

Ans. : With n computers, all n crystals will run at slightly different rates, causing the software clocks to gradually get out of sync. This difference in time values is called clock skew.

Q.12 What is clock drift rate.

AU : Dec.-16, May-18

Ans. : A clock drift rate is the change in the offset between the clock and a nominal perfect reference clock per unit of time measured by the reference clock.

Q.13 List the design aims and features of NTP ?

Ans. : Design aims are as follows :

- To provide a service enabling client across the internet to be synchronized accurately to UTC.
- To provide a reliable service that can be survive lengthy losses of connectivity.
- To enable clients to resynchronize sufficiently frequently to offset the rates of drift found in most computers.
- To provide protection against interference with the time service whether malicious or accidental.

Q.14 List the name of modes the NTP servers synchronize.

Ans. : NTP servers synchronize with one another in one of three modes: multicast, procedure-call and symmetric mode.

Q.15 What are the two modes of synchronization ?

Ans. : The two modes are :

1. External synchronization : For a synchronization bound $D > 0$, and for a source S of UTC time, $|S(t) - C_i(t)| < T$, for $i = 1,2,...N$ and for all real times t in I.
2. Internal synchronization: For a synchronization bound $D > 0, |C_i(t) - C_j(t)| < D$, for $i, j = 1,2,...N$. and for all real times t in I.

Q.16 What is logical clock ?

Ans. : Logical clock is a monotonically increasing software counter, whose value need bear no particular relationship to any physical clock. Each process P_i keeps its own logical clock L_i which it uses to apply so called Lamport timestamps to events.

Q.17 What is global state of the distributed system ?

Ans. : The global state of the distributed system consists of the local state of each process, together with the messages which are in transit.

Q.18 Write the happens-before relation ? **AU : May-16**

Ans. : The happens-before relation can be observed directly in two situations:

1. If a and b are events in the same process, and a occurs before b then a 'b is true.
2. If a is the event of a message being sent by one process, and b is the event of the message being received by another process, then a 'b is also true.

Q.19 What is need of physical clock ?

Ans. : Lamport's algorithm for logical clock synchronization gives an unambiguous event ordering, the time values assigned to events are not necessarily close to the actual times at which they occur.

In some systems like real-time systems, the actual clock time is important. For these systems external physical clocks are required.

Q.20 Explain the difference between logical and physical clocks. **AU : May-16**

Ans. : Physical clocks measure the time of day. Logical clocks are used to mark relationships among events in a distributed system.

Q.21 What problem with Lamport's clocks to vector clocks solve ?

Ans. : With Lamport's clocks, you cannot tell whether two events are causally related or concurrent by looking at the timestamps. Just because L(a) < L(b) does not mean that a->b. Vector clocks allow you to compare two vector timestamps to determine whether the events are concurrent or not.

Q.22 What is vector clock ?

Ans. : Vector clocks are used in a distributed system to determine whether pairs of events are causally related. Using vector clocks, timestamps are generated for each event in the system, and their causal relationship is determined by comparing those timestamps.

Q.23 How vector clock timestamps are assigned ?

Ans. : Vector clock timestamps are assigned as follows :

a. **Events :** Every time an event is generated, a process increments its clock and assigns a timestamp to the event based on its knowledge of all the clocks in the system.

b. **Sending messages :** When a message is sent the timestamp of the sending event is given to the message.

c. **Receiving messages :** When a message is received, the process updates its knowledge of the system clock states by taking the maximum of each component of the message timestamp and its current knowledge of the system clock states.

Q.24 What is global state ?

Ans. : A global state of a distributed system is a set of component process and channel states : The initial global state is one in which the state of each process is its initial state and the state of each channel is the empty sequence

Q.25 What is FIFO and Non-FIFO model ?

Ans. : In FIFO model, each channel acts as a frst-in-frst-out message queue and thus, message ordering is preserved by a channel. In non-FIFO model, a channel acts like a set in which the sender process adds messages and the receiver process removes messages from it in a random order.

Q.26 What do you mean interpretation in terms of cuts ?

Ans. : A cut in a space-time diagram is a line joining an arbitrary point on each process line that slices the space-time diagram into a PAST and a FUTURE

Q.27 How to distinguish between the messages to be recorded in the snapshot from those not to be recorded ?

Ans. : Any message that is sent by a process before recording its snapshot, must be recorded in the global snapshot. Any message that is sent by a process after recording its snapshot, must not be recorded in the global snapshot

Q.28 What is Chandy-Lamport algorithm ?

Ans. : The Chandy-Lamport algorithm is a snapshot algorithm that is used in distributed systems for recording a consistent global state of an asynchronous system.

UNIT III

3 Distributed Mutex and Deadlock

Syllabus

Distributed Mutual exclusion Algorithms : Introduction - Preliminaries - Lamport's algorithm - Ricart-Agrawala's Algorithm - Token-Based Algorithms - Suzuki-Kasami's Broadcast Algorithm ; Deadlock Detection in Distributed Systems : Introduction - System Model - Preliminaries - Models of Deadlocks - Chandy-Misra-Haas Algorithm for the AND model and OR Model.

Contents

3.1 Distributed Mutual Exclusion Algorithms : Introduction

- Mutual exclusion ensures that concurrent processes make a serialized access to shared resources or data. It requires that the actions performed by a user on a shared resource must be atomic.
- In a distributed system neither shared variables nor a local kernel can be used in order to implement mutual exclusion. Thus, mutual exclusion has to be based exclusively on message passing, in the context of unpredictable message delays and no complete knowledge of the state of the system.
- **Mutual exclusion :** Makes sure that concurrent process access shared resources or data in a serialized way. If a process, say Pi, is executing in its critical section, then no other processes can be executing in their critical sections.
- **Example :** Updating a DB or sending control signals to an I/O device
- Problem of mutual exclusion frequently arises in distributed systems whenever concurrent access to shared resources by several sites is involved.
- Mutual exclusion is the fundamental issue in the design of distributed systems.
- **Entry section :** The code executed in preparation for entering the critical section
- **Critical section :** The code to be protected from concurrent execution
- **Exit section :** The code executed upon leaving the critical section
- **Remainder section :** The rest of the code
- Each process cycles through these sections in the order : *remainder, entry, critical, exit.*

3.2 Preliminaries

3.2.1 System Model

- Distributed mutual exclusion algorithms must deal with unpredictable message delays and incomplete knowledge of the system state.

System model

- The system consists of N sites, $S_1, S_2, \ldots, S_N$. We assume that a single process is running on each site. The process at site S_i is denoted by pi .
- At any instant, a site may have several requests for critical section. A site queues up these requests and serves them one at a time.
- Site may in one of the three states :
  1. Requesting CS
  2. Executing CS
  3. Neither requesting nor executing requests for CS

Classification of Mutual Exclusion

- Different types of algorithm are used to solve problem of mutual exclusion in distributed system. But these algorithms differ in their communication topology. Topology may be ring, bus, star etc. They also maintain different types of information.
- These algorithms are divided into two classes :
  1. Non-token based : Require multiple rounds of message exchanges for local states to stabilize
  2. Token based : Permission passes around from one site to another. Site is allowed to enter its critical section if it possesses the token and it continues to hold the token until the execution of the critical section is over.

3.2.2 Requirement of Mutual Exclusion

1. **Freedom from deadlocks :** Two or more sites should not endlessly wait for message that will never arrive.
2. **Freedom from starvation :** A site should not wait indefinitely while other sites repeatedly access the CS.
3. **Strict fairness :** Requests are served in the (logical) order in which they arrive.
4. **Fault tolerance :** An algorithm should be able to detect and recover from failures.

3.2.3 Performance Metrics

1. **Message complexity :** The number of messages required per CS execution by a site.
2. **Synchronization delay :** After a site leaves the CS, it is the time required before the next site enters the CS.
3. **Response time :** The time interval a request waits for its CS execution to be over after its request messages have been sent out.
4. **System throughput :** The rate at which the system executes requests for the CS.

$$\text{System throughput} = 1/(SD+E)$$

where SD is the synchronization delay and E is the average critical section execution time.

- Fig. 3.2.1 shows the synchronization delay.
- Synchronization delay (*sd*) : A leaves → B enters
- Response time : A requests → A leaves
- Throughput : CS requests handled per time unit = $1 / (SD + E)$ where E is average execution time of CS.

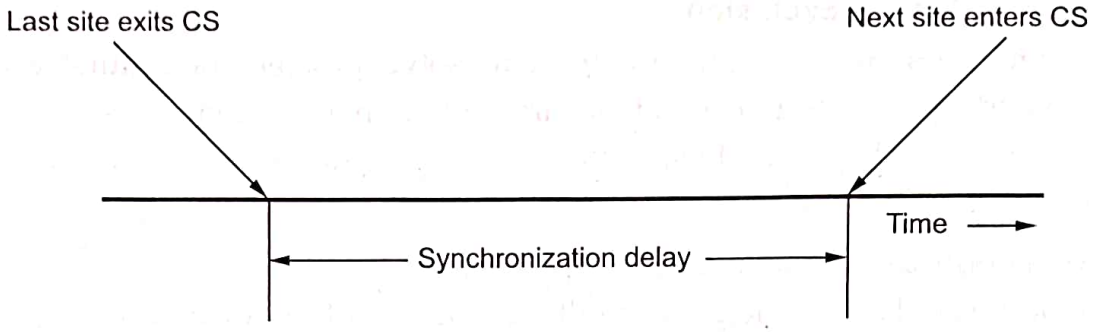


Fig. 3.2.1 Synchronization delay

- Performance of mutual exclusion algorithm depends upon the loading condition of the system.
- Performance may depend on whether load is low or high. Best case, worst case, and average cases are all of interest.
- If the load is high then there is always pending requests for mutual exclusion.
- Fig. 3.2.2 shows the response time.

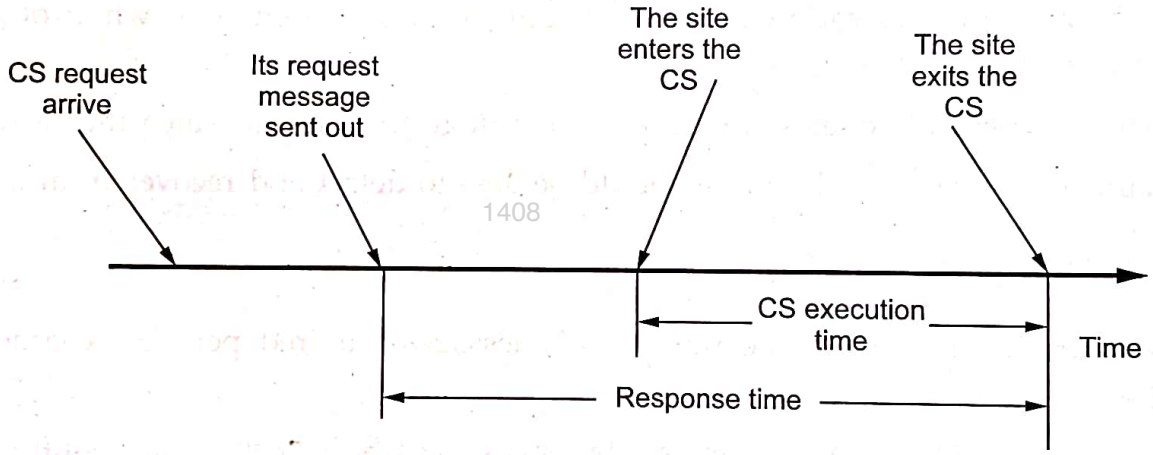


Fig. 3.2.2 Response time

3.3 Lamport's Algorithm

- Each process freely and equally competes for the right to use the shared resource; requests are arbitrated by a central control site or by distributed agreement.
- Permission from all processes.
- Example of non-token based algorithms are : Lamport, Ricart- Agarwala, Raicourol-Carvalho etc.
- Let Ri be the request set of site Si, i.e. the set of sites from which Si needs permission when it wants to enter CS.
- Each site Si keeps a request queue of requests ordered by logical timestamp

Requesting the critical section

- Send a REQUEST(Ci, i) to each other site, and place in the request queue.
- When Sj receives the REQUEST(Ci, i) it returns a timestamped REPLY to Si and places the REQUEST in the request queue.

Conditions for entering CS

- L1 : Si has received a message with a timestamp larger than (Ci, i) from all other sites.
- L2 : Si's request is at the top of the request queue.

Releasing the CS

- Remove (Ci, i) from the request queue, and send a RELEASE to all sites in the request set.
- When Sj receives RELEASE, it removes the request from the request queue.

Correctness

- Assume two processes S1, S2 in CS simultaneously. [L1, L2] hold in both processes i.e. both have received messages with larger timestamps from all others, their requests are at the top of their queues.
- WLOG, assume S1's request is earlier So, S1's request is in S2's queue, but S2's request is at the top of the queue contradiction.

Optimization

- In Lamport's algorithm, REPLY messages can be omitted in certain situations.
- For example, if site Sj receives a REQUEST message from site Si after it has sent its own REQUEST message with timestamp higher than the timestamp of site Si 's request, then site Sj need not send a REPLY message to site Si .
- This is because when site Si receives site Sj 's request with timestamp higher than its own, it can conclude that site Sj does not have any smaller timestamp request which is still pending.
- With this optimization, Lamport's algorithm requires between 3(N – 1) and 2(N – 1) messages per CS execution.
- Fig. 3.3.1 shows the optimization of Lamport algorithm.

Step 1 : Sites S1 and S2 are making request for critical section.

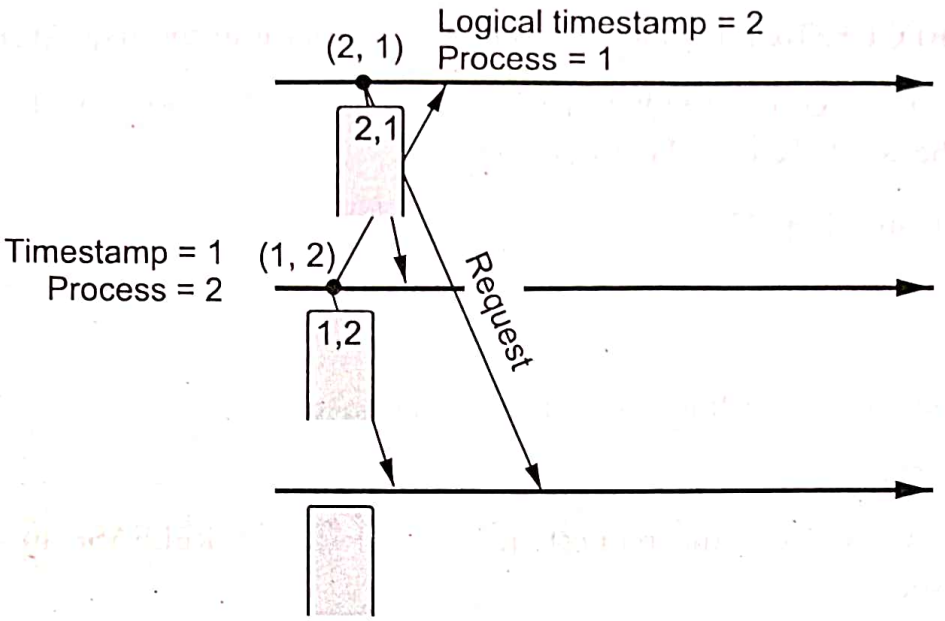


Fig. 3.3.1 (a)

Step 2 : Site S2 enters the critical section.

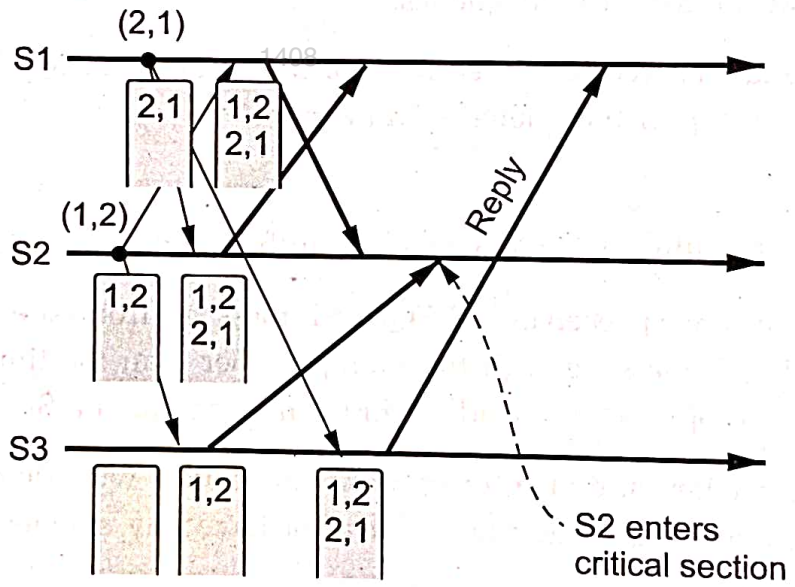


Fig. 3.3.1 (b)

Step 3 : Site S2 exits critical section and sends RELEASE messages.

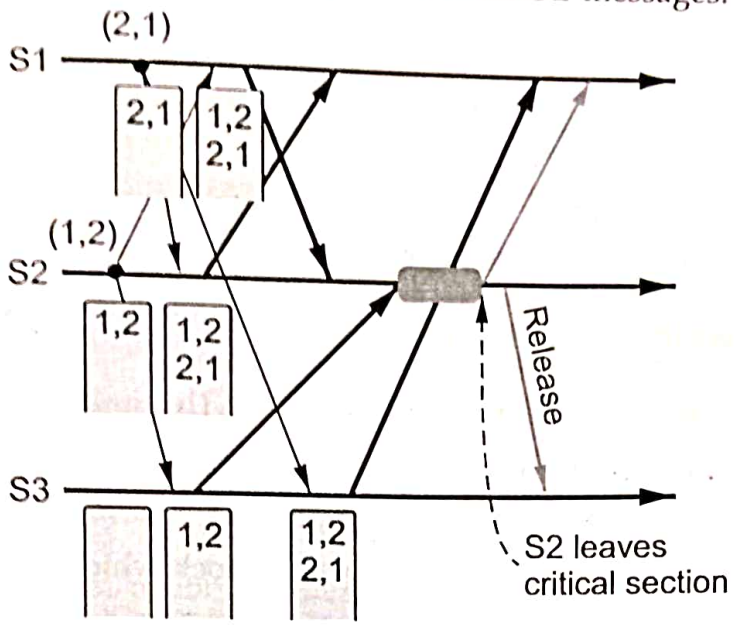


Fig. 3.3.1 (c)

Step 4 : Site S1 enters critical section

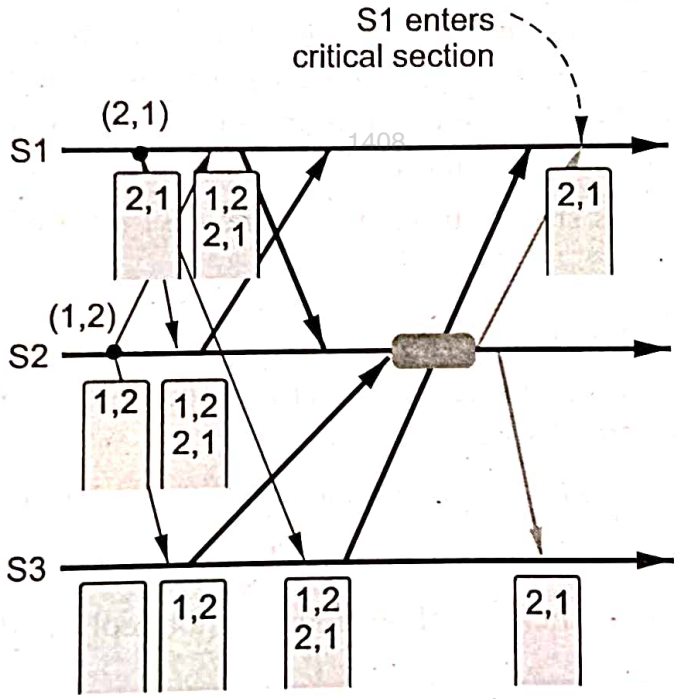


Fig. 3.3.1 (d)

Lamport evaluation

- Deadlock : processes do not wait forever
- Fairness : CS requests are granted in order of logical clock, which is fair
- Fault tolerance :
  a. If a process fails before a request, fine
  b. If a process fails after a few requests...

c. One method use a probabilistic failure detector or if a process times out, remove it from all queues.

d. If a process fails in critical section...

- Performance
  a. 3(N – 1) messages per CS execution.
  b. Sync delay is T (average message latency).

3.4 Ricart-Agrawala's Algorithm

AU : May-22, Dec.-22

- Ricart-Agrawala algorithm is an optimization on Lamport algorithm.
- Ricart-Agrawala algorithm uses only two types of messages : *REQUEST* and *REPLY*.
- It is assumed that all processes keep a logical clock which is updated according to the clock rules.
- The algorithm requires a total ordering of requests. Requests are ordered according to their global logical timestamps; if timestamps are equal, process identifiers are compared to order them.
- The process that requires entry to a CS multicasts the request message to all other processes competing for the same resource. Process is allowed to enter the CS when all processes have replied to this message. The request message consists of the requesting process' timestamp (logical clock) and its identifier.
- Each process keeps its state with respect to the CS : released, requested, or held.

Algorithm :

Requesting the critical section :

1. Request when Si wants to enter the critical section, it broadcast timestamped REQUEST message to all site.
2. When a process receives a REQUEST message, it may be in one of three states :

Case 1 : The receiver is not interested in the critical section, send reply (OK) to sender.

Case 2 : The receiver is in the critical section; do not reply and add the request to a local queue of requests.

Case 3 : The receiver also wants to enter the critical section and has sent its request. In this case, the receiver compares the timestamp in the received message with the one that it has sent out. The earliest timestamp wins. If the receiver is the loser, it sends a reply (OK) to sender. If the receiver has the earlier timestamp, then it is the winner and does not reply. Instead, it adds the request to its queue.

Executing the critical section :

3. When site Si enters critical section after it has received REPLY message from all the site in its request set.

Releasing the critical section :

4. When site Si exits the CS, it sends RELEASE(i) messages to all sites in its request set Ri.
5. When a site Sj receives the RELEASE(i) message from site, it sends a REPLY(j) message to the next site waiting in the queue and deletes that entry from the queue. If the queue is empty, then the site updates its state to reflect that the site has not sent out any REPLY message.

Optimization :

- Once site Si has received a REPLY message from a site Sj, the authorization implicit in this message remains valid until Si sends a REPLY message to Sj.
- Fig. 3.4.1 shows operation of Ricart-Agrawala algorithm.

Step 1 : Site S1 and S2 are making request for critical section.

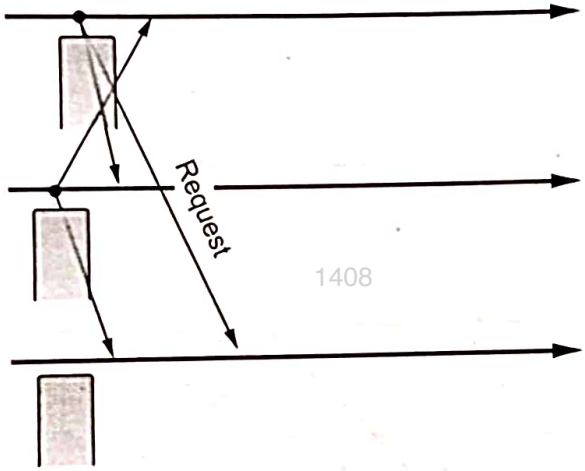


Fig. 3.4.1 (a)

Step 2 : Site S2 enters the critical section.

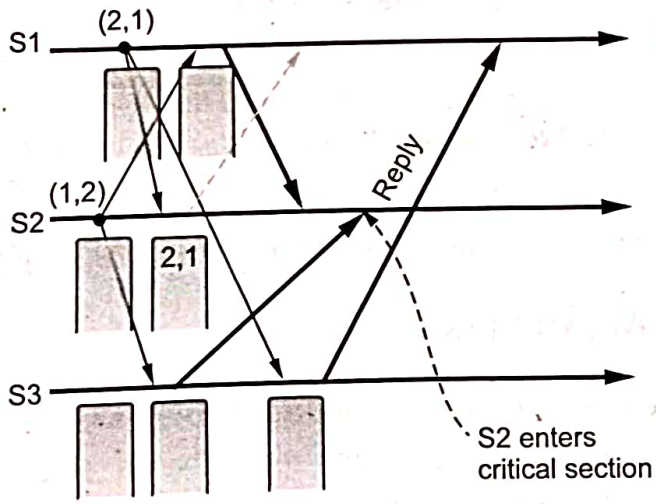


Fig. 3.4.1 (b)

Step 3 : Site S2 exits the CS and sends a REPLY messages to S1.

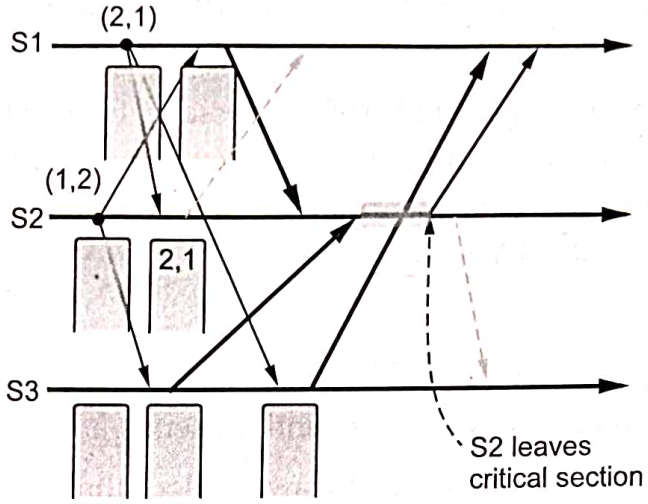


Fig. 3.4.1 (c)

Step 4 : Site S1 enters the critical section

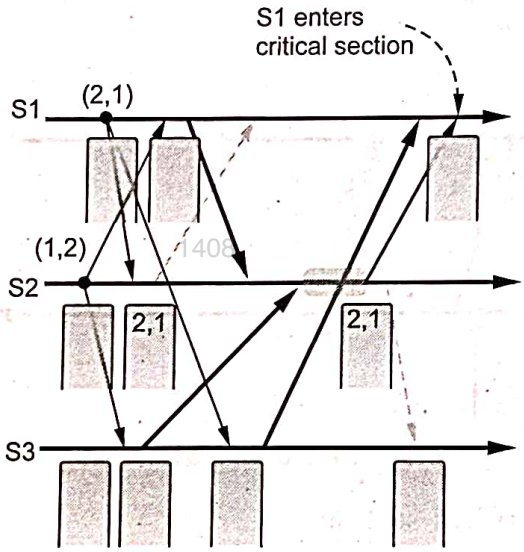


Fig. 3.4.1 (d)

University Questions

1. *Show that in the Ricart-Agrawala algorithm the critical section is accessed in increasing order of timestamp.* **AU : May-22, Marks 15**
2. *Explain Ricart Agrawala Algorithm with an example.* **AU : Dec.-22, Marks 13**

3.5 Token-Based Algorithms

AU ; Dec.-22

- In the Token-based algorithm, a unique token is shared among all the sites in distributed computing systems. In non-token based algorithm, there is no token even not any concept of sharing token for access.

- Token based algorithms are Suzuki-Kasami algorithm and Raymond's tree algorithm.
- A site can access the lock (critical section) if it has a token. Every process maintains a sequence number (request id).

3.5.1 Suzuki-Kasami's Broadcast Algorithm

- Logical token representing the access right to the shared resource is passed in a regulated fashion among the processes; whoever holds the token is allowed to enter the critical section.
- A unique token is shared among all sites. This means a sequence number is used instead of a timestamp.
- Sites increment its sequence number every time it requests the token. This means, issues of liveness (deadlock-freedom, starvation) are more interesting.
- To enter CS, a site broadcasts its REQUEST message to all other sites.
- Upon receiving a REQUEST message, a site that has the token sends the token to the requesting site only if this site is not in CS. If it is in CS, it sends the token only after it has exited the CS.
- A site can repeatedly enter CS as long as it holds the token and there are no pending requests from other sites.

Major Design Issues

- No RELEASE/REPLY messages as in assertion based protocols
- Each site needs to be able to distinguish outdated REQUEST messages from current REQUEST messages (outstanding requests)
- The site with the token needs to know which site to send the token next.

Important Data Structures

- The token consists of Q, a queue of requesting sites, Array of integers, LN[1..N], where LN[j] is the sequence number of the request that site Sj executed most recently and N is the number of sites and LN keeps track of the latest request that each site has executed.
- A site Si keeps an array of integers RNi[1..N] where RNi[j] is the largest sequence number received so far from Sj. A REQUEST(j,n) indicates that site Sj is requesting for CS with n as its sequence number.

Algorithm :

Requesting CS :

1. Site Si increments its sequence number, RNi[i] and sends a REQUEST(i , sn) to all other sites with sequence number sn.
2. When site Sj receives this message, it sets RNj[i]:=max(RNj[i],sn). If Sj has an idle token, it sends the token to Si if RNj[i]=LN[i]+1.

Executing CS :

3. Site Si executes CS when it has received the token

Releasing CS :

4. LN[i] <- Rn[i][i]
5. For each Sj not in the token queue, append ID to queue if RN[i][j]=LN[j]+1
6. If token queue is nonempty after update, delete top ID and send to site indicated by ID

Theorem : A requesting site enters CS in finite time

Proof : Token request reaches other sites in finite time and this request will be placed in the token queue in finite time. Since there can be at most N – 1 requests in front of this request, the requesting site can be in CS in finite time

Performance :

1. The algorithm is simple and efficient. It requires 0 or N messages per CS invocations.
2. Synchronization delay : 0 or T. No message is needed or the synchronization delay is zero if a site holds the idle token at the time of its request.

University Question

1. *Analyse Suzuki -Kasami's broadcast algorithm for mutual exclusion in distributed system.*

AU : Dec.-22, Marks 15

3.6 Deadlock Detection in Distributed Systems : Introduction

AU : May-22

- A distributed system consists of a number of sites connected by a network. Each site maintains some of the resources of the system.
- Processes with a globally unique identifier run on the distributed system. They make resource requests to a controller. There is one controller per site.
- If the resource is local, the process makes a request of the local controller.

- If the desired resource is at a remote site, the process sends a message. After a process makes a request, but before it is granted, it is blocked and said to be dependent on the process that holds the desired resource.
- The controller at each site could maintain a WFG on the process requests that it knows about. This is the local WFG.
- However, each site's WFG could be cycle free and yet the distributed system could be deadlocked. This is called **global deadlock**.
- This would occur in the following situation :
  1. Process A at site 1 holds a lock on resource X.
  2. Process A has requested, but has not been granted, resource Y at site 2.
  3. Process B at site 2 holds a lock on resource Y.
  4. Process B has requested, but has not been granted, resource X at site 1.
- Both processes are blocked by the other one. There is a global deadlock.
- However, the deadlock will not be discovered at either site unless they exchange information via some detection scheme.

3.6.1 Necessary Condition

- A process can be in two states : **Running or blocked**
- In the running state (also called the active state), a process has all the needed resources and is either executing or is ready for execution.
- In the blocked state, a process is waiting to acquire some resources.
- Deadlock is a situation in which a set of processes is blocked waiting for other process in the set to release the resource.
- Following conditions should hold simultaneously for deadlock to occur :
  1. Mutual exclusion
  2. No pre-emption
  3. Hold and wait
  4. Circular wait

1. **Mutual exclusion :** Only one process may use a resource at a time. Once a process has been allocated a particular resource, it has exclusive use of the resource. No other process can use a resource while it is allocated to a process.
2. **Hold and wait :** A process may hold a resource at the same time it requests another one.

3. **Circular waiting :** A situation can arise in which process, holds resource while it requests resource and process holds while it requests resource. Each process holds at least one resource needed by the next process in the chain. There may be more than two processes involved in a circular wait (e.g. Fig. 3.6.1).

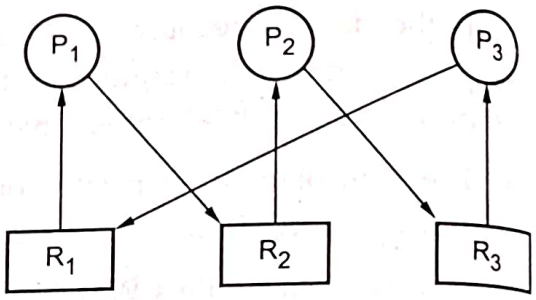


Fig. 3.6.1 Three deadlocked processes

4. **No pre-emption :** No resource can be forcibly removed from a process holding it. Resources can be released only by the explicit action of the process, rather than by the action of an external authority.

A deadlock is possible only if all four of these conditions simultaneously hold in the community of processes. These conditions are necessary for a deadlock to exist.

University Question

1. *Discuss with suitable example to show that a deadlock cannot occur if any one of the four conditions is absent.* **AU : May-22, Marks 13**

3.7 System Model

- Resource types *R*1, *R*2, . . ., *R*m (Resources : *CPU cycles, memory space, I/O devices)*
- Each resource type *R*i has *W*i instances.
- Each process utilizes a resource as follows :
  1. Request 2. Use 3. Release

3.7.1 Wait for Graph

- The state of process-resource interaction in distributed systems can be modeled by a bi-partite directed graph called a resource allocation graph. The nodes of this graph are processes and resources of a system, and the edges of the graph depict assignments or pending requests.
- A pending request is represented by a request edge directed from the node of a requesting process to the node of the requested resource. A resource assignment is represented by an assignment edge directed from the node of an assigned resource to the node of the assigned process.
- A system is deadlocked if its resource allocation graph contains a directed cycle or a knot. Fig. 3.7.1(a) shows resource allocation graph.

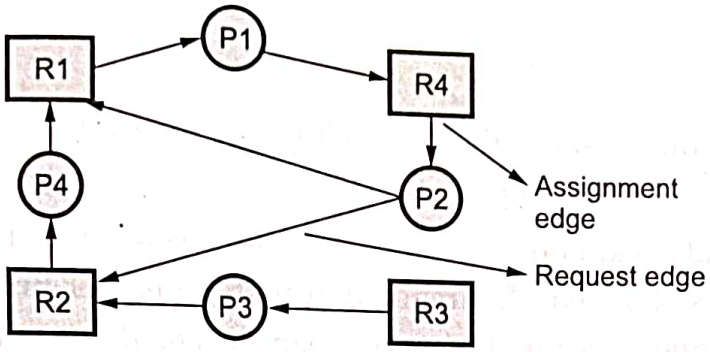


Fig. 3.7.1 (a) Resource allocation graph

- In distributed systems, the system state can be modeled or represented by a directed graph, called a Wait-For Graph (WFG). In a WFG, nodes are processes and there is a directed edge from node P1 to node P2 if P1 is blocked and is waiting for P2 to release some resource.
- A system is deadlocked if and only if there is a directed cycle or not (depending upon the underlying model) in the WFG. Resulting wait for graph of resource allocation graph is shown in Fig. 3.7.1 (b).

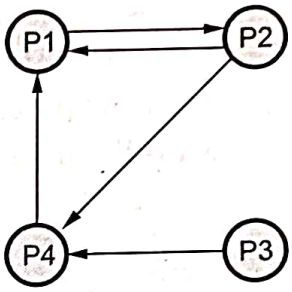


Fig. 3.7.1 (b) Wait-for graph

3.8 Preliminaries : Deadlock Handling Strategies

- There are three strategies for handling deadlocks :
  1. Deadlock prevention 2. Deadlock avoidance 3. Deadlock detection.

3.8.1 Deadlock Prevention

First method :

- Prevent the circular-wait condition by defining a linear ordering of resource types. A process can be assigned resources only according to the linear ordering.
- **Disadvantages :** Resources cannot be requested in the orders that are needed. Resources will be longer than necessary.

Second method :

- Prevent the hold-and-wait condition by requiring the process to acquire all needed resources before starting execution.
- **Disadvantages :**
  1. Inefficient use of resources.
  2. Reduced concurrency.
  3. Process can become deadlocked during the initial resource acquisition.
  4. Future needs of a process cannot be always predicted.

Third method :

Use of time-stamps

- **Example :** Use time-stamps for transactions to a database - Each transaction has the time-stamp of its creation.
- The circular wait condition is avoided by comparing time-stamps : Strict ordering of transactions is obtained, the transaction with an earlier time-stamp always wins.

1. **"Wait-die" method :** A non-preemptive approach. If a younger process is using the resource, then the older process (that wants the resource) waits. If an older process is holding the resource, the younger process (that wants the resource) kills itself. This forces the resource utilization graph to be directed from older to younger processes, making cycles impossible. This algorithm is known as the **wait-die** algorithm. Fig. 3.8.1 shows wait-die method.

```
if [ e (T2) < e (T1) ]
halt_T2 ('wait');
else
kill_T2 ('die');
```

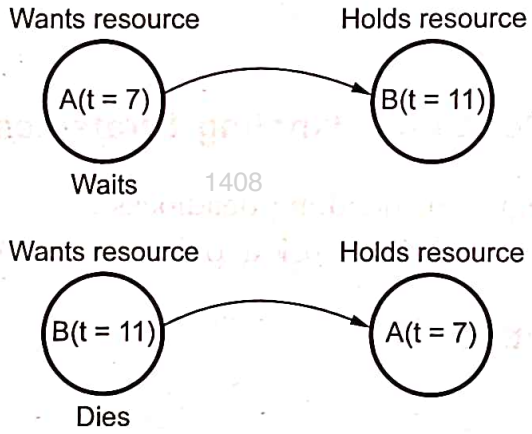


Fig. 3.8.1 Wait-die method

2. **"Wound-wait" method :** A pre-emptive approach. An alternative method by which resource request cycles may be avoided is to have an old process preempt (kill) the younger process that holds a resource. If a younger process wants a resource that an older one is using, then it waits until the old process is done. In this case, the graph flows from young to old and cycles are again impossible. This variant is called the wound-wait algorithm. Fig. 3.8.2 shows wound-wait method.

```
if [ e (T2) < e (T1) ]
kill_T1 ('wound');
else
halt_T2 ('wait');
```

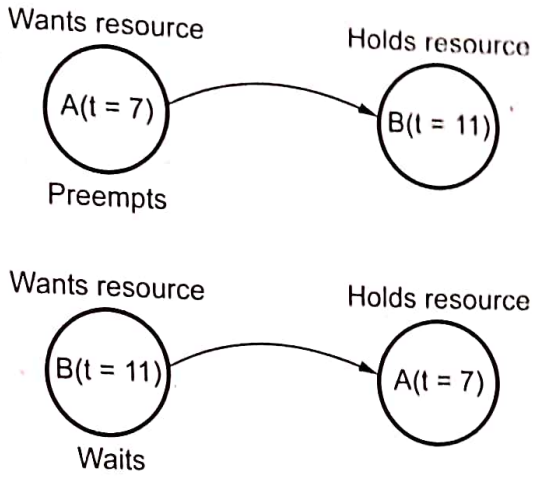


Fig. 3.8.2 Wound - wait method

3.8.2 Dead Avoidance

- Decision made dynamically, before allocating a resource, the resulting global system state is checked, if it is safe state then allow for allocation.
- Because of the following drawback, deadlock avoidance can be impractical in distributed system.
- **Disadvantages**
  1. Every site has to maintain global state of system (extensive overhead in storage and communication).
  2. Different sites may determine (concurrently) that state is safe, but global state may be unsafe : Verification for safe global state by different sites must be mutually exclusive.
  3. Large overhead to check for every allocation (distributed system may have large number of processes and resources).

3.8.3 Deadlock Detection

- **Principle of operation :** Detection of a cycle in WFG proceeds concurrently with normal operation.
- Requirements for the deadlock detection and resolution algorithms :
- Detection
  1. The algorithm must detect all existing deadlock in finite time
  2. The algorithm should not report non-existent (phantom) deadlock.
- **Resolution (recovery) :** All existing wait-for dependencies in WFG must be removed, i.e. roll-back one or more processes that are deadlocked and give their resources to other blocked processes.
- **Observation :** Deadlock detection is the most popular strategy for handling deadlocks in distributed systems.

3.9 Models of Deadlocks

AU : Dec.-22

- Models of deadlocks distributed system allows different kind of resource requests so; that means, they are represented by different model a process might require a single resource or a combination of resources for its execution.

3.9.1 The Single Resource Model

- In the single resource model, a process can have at most one outstanding request for only one unit of a resource.
- Since the maximum out-degree of a node in a WFG for the single resource model can be 1, the presence of a cycle in the WFG shall indicate that there is a deadlock.

3.9.2 The AND Model

- Set of deadlocked processes, where each process waits for a resource held by another process (e.g. data object in a database, I/O resource on a server).
- Uses AND condition.
- **AND condition :** A process that requires resources for execution can proceed when it has acquired all those resources.
- The condition for deadlock in a system using the AND condition is the existence of a *cycle*.
- Since in the single-resource model, a process can have at most one outstanding request, the AND model is more general than the single-resource model

3.9.3 The OR Model

- Set of deadlocked processes, where each process waits to receive messages (communication) from other processes in the set.
- Uses OR condition.
- **OR condition :** A process that requires resources for execution can proceed when it has acquired at least one of those resources.
- The condition for deadlock in a system using the OR condition is the existence of a *knot*. A knot (K) consists of a set of nodes such that for every node a in K, all nodes in K and only the nodes in K are reachable from node a.
- In the OR model, the presence of a knot indicates a deadlock.

3.9.4 The AND-OR Model

- A generalization of the previous two models (OR model and AND model) is the AND-OR model.

- In the AND-OR model, a request may specify any combination of and and or in the resource request. For example, in the AND-OR model, a request for multiple resources can be of the form x and (y or z).
- To detect the presence of deadlocks in such a model, there is no familiar construct of graph theory using WFG.
- Since a deadlock is a stable property, a deadlock in the AND-OR model can be detected by repeated application of the test for OR-model deadlock.

University Question

1. *Name and explain the different types of deaklock models in distributed system with the commonly used strategies to handle deadlocks with a neat diagram.*

AU : Dec.-22, Marks 13

3.10 Chandy-Misra-Haas Algorithm for the AND Model

- This is considered an edge-chasing, probe-based algorithm. It is also considered one of the best deadlock detection algorithms for distributed systems.
- If a process makes a request for a resource which fails or times out, the process generates a probe message and sends it to each of the processes holding one or more of its requested resources.
- Each probe message contains the following information :
  1. The id of the process that is blocked (the one that initiates the probe message);
  2. The id of the process is sending this particular version of the probe message; and
  3. The id of the process that should receive this probe message.
- When a process receives a probe message, it checks to see if it is also waiting for resources. If not, it is currently using the needed resource and will eventually finish and release the resource.
- If it is waiting for resources, it passes on the probe message to all processes it knows to be holding resources it has itself requested.
- The process first modifies the probe message, changing the sender and receiver ids. If a process receives a probe message that it recognizes as having initiated, it knows there is a cycle in the system and thus, deadlock.
- Fig. 3.10.1 shows deadlock example.
- In this case P_1 initiates the probe message, so that all the messages shown have P_1 as the initiator. When the probe message is received by process P_3, it modifies it and sends it to two more processes. Eventually, the probe message returns to process P_1. Deadlock!

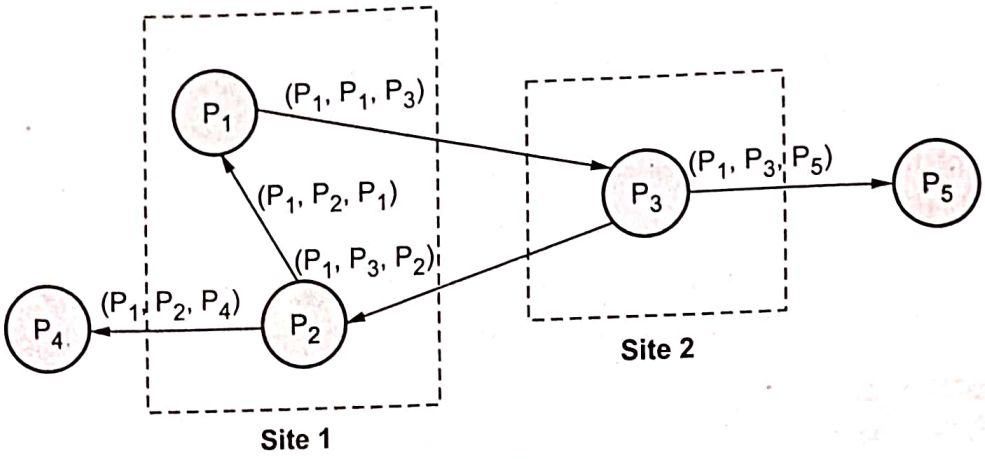


Fig. 3.10.1 Deadlock example

- **Advantages :**
  a) It is easy to implement.
  b) Each probe message is of fixed length.
  c) There is very little computation.
  d) There is very little overhead.
  e) There is no need to construct a graph, nor to pass graph information to other sites.
  f) This algorithm does not find false (phantom) deadlock.
  g) There is no need for special data structures.

3.11 Chandy-Misra-Haas Algorithm for the OR Model

- Chandy-Misra-Haas distributed deadlock detection algorithm for OR model is based on the approach of diffusion-computation. A blocked process determines if it is deadlocked by initiating a diffusion computation.
- Two types of messages are used in a diffusion computation : query(i, j, k) and reply(i, j, k), denoting that they belong to a diffusion computation initiated by a process P_i and are being sent from process P_j to process P_k.
- A blocked process initiates deadlock detection by sending query messages to all processes in its dependent set. If an active process receives a query or reply message, it discards it.
- When a blocked process P_k receives a query(i, j, k) message, it takes the following actions :
  1) If this is the first query message received by P_k for the deadlock detection initiated by P_i (called the engaging query), then it propagates the query to all the processes in its dependent set and sets a local variable numk (i) to the number of query messages sent.

2) If this is not the engaging query, then P_k returns a reply message to it immediately provided P_k has been continuously blocked since it received the corresponding engaging query. Otherwise, it discards the query.

- Process P_k maintains a boolean variable waitk (i) that denotes the fact that it has been continuously blocked since it received the last engaging query from process P_i. When a blocked process P_k receives a reply(i, j, k) message, it decrements num_k (i) only if $wait_k$ (i) holds.
- A process sends a reply message in response to an engaging query only after it has received a reply to every query message it had sent out for this engaging query. The initiator process detects a deadlock when it receives reply messages to all the query messages it had sent out.

3.12 Two Marks Questions with Answers

Q.1 Explain the term mutual exclusion. **AU : Dec.-22**

Ans. : Asynchronous programming provides opportunities for a program to continue running other code while waiting for a long-running task to complete.

Q.2 What is deadlock ? **AU : Dec.-22**

Ans. : Deadlock is the problem of multiprogramming system. Deadlock can be defined as the permanent blocking of a set of processes that either complete for system resources.

Q.3 Name the two types of messages used in Ricart-Agrawala's algorithm. **AU : May-22**

Ans. : Two type of messages used by Ricart-Agrawala are REQUEST and REPLY and communication channels are assumed to follow FIFO order. Site send a REQUEST message to all other site to get their permission to enter critical section. A site send a REPLY message to other site to give its permission to enter the critical section.

Q.4 What are the conditions for deadlock ? **AU : May-22**

Ans Conditions should hold simultaneously for deadlock to occur are :

a) Mutual exclusion b) No preemption

c) Hold and wait d) Circular wait.

Q.5 What is mutual exclusion ?

Ans. : Mutual exclusion in a distributed system states that only one process is allowed to execute the critical section (CS) at any given time. In a distributed system, shared variables or a local kernel cannot be used to implement mutual exclusion.

Q.6 Which are the three basic approaches for implementing distributed mutual exclusion ?

Ans. : There are three basic approaches for implementing distributed mutual exclusion :

1. Token based approach
2. Non-token based approach
3. Quorum based approach

Q.7 What are the requirements of mutual exclusion algorithms ?

Ans. : Requirements of mutual exclusion algorithms are

a. Freedom from deadlocks b. Freedom from starvation

c. Strict fairness d. Fault tolerance

Q.8 What are the performance metric of mutual exclusion algorithm ?

Ans. : Performances metric are message complexity, synchronization delay, response time and system throughput.

Q.9 What is response time ?

Ans. : The time interval a request waits for its CS execution to be over after its request messages have been sent out.

Q.10 Which are the criteria for evaluating performance of algorithms for mutual exclusion ?

Ans. : Criteria for evaluating performance of algorithms for mutual exclusion are :

a. Bandwidth consumed which is proportional to the number of messages sent in each entry and exit operation.

b. Client delay incurred by a process at each entry and exit operation.

c. Throughput of the system.

Q.11 What is the advantage if your server side processing uses threads instead of a single process ?

Ans. : An important property of threads is that they can provide a convenient means of allowing blocking system calls without blocking the entire process in which the thread is running. This property makes threads particularly attractive to use in distributed systems as it makes it much easier to express communication in the form of maintaining multiple logical connections at the same time.

Q.12 What is a phantom deadlock ?

Ans. : A deadlock that is 'detected' but is not really a deadlock is called a phantom deadlock.

Q.13 What is wait for graph ?

Ans. : The state of process-resource interaction in distributed systems can be modeled by a bi-partite directed graph called a resource allocation graph. The nodes of this

graph are processes and resources of a system, and the edges of the graph depict assignments or pending requests. A pending request is represented by a request edge directed from the node of a requesting process to the node of the requested resource.

Q.14 Explain Wait-die" method.

Ans. : A non-preemptive approach. If a younger process is using the resource, then the older process waits. If an older process is holding the resource, the younger process kills itself. This forces the resource utilization graph to be directed from older to younger processes, making cycles impossible. This algorithm is known as the wait-die algorithm.

Q.15 List the deadlock handling strategies in distributed system.

Ans. : There are three strategies for handling deadlocks, viz., deadlock prevention, deadlock avoidance, and deadlock detection.

Q.16 What do you mean by deadlock avoidance ?

Ans. : Deadlock avoidance depends on additional information about the long term resource needs of each process. The system must be able to decide whether granting a resource is safe or not and only make the allocation when it is safe. When a process is created, it must declare its maximum claim, i.e. the maximum number of unit resource. The resource manager can grant the request if the resources are available.

Q.17 Define deadlock detection in distributed systems.

Ans. : Deadlock detection requires examination of process-resource interaction for the presence of cyclic wait.

UNIT IV

4 Consensus and Recovery

Syllabus

Consensus and Agreement Algorithms : Problem Definition - Overview of Results - Agreement in a Failure-Free System (Synchronous and Asynchronous) - Agreement in Synchronous Systems with Failures; Check-pointing and Rollback Recovery : Introduction - Background and Definitions - Issues in Failure Recovery - Checkpoint-based Recovery - Coordinated Checkpointing Algorithm - Algorithm for Asynchronous Checkpointing and Recovery.

Contents

4.1 Consensus and Agreement Algorithms : Problem Definition

- Processes/Sites in distributed systems often compete as well as cooperate to achieve a common goal. Mutual Trust/agreement is very much required.
- In distributed data bases, there may be a situation where data managers have to decide "Whether to commit or Abort the Transaction". When there is no failure, reaching an agreement is easy.
- However, in case of failures, processes must exchange their values with other processes and relay the values received from others several times to isolate the effect of faulty processor.
- Agreement protocols helps to reach an agreement in presence of failures.
- Examples : Agreeing whether to commit or to abort a transaction in a distributed database management system. Agreeing on a common clock value in a distributed system.
- In the absence of failures or faulty processors, values (that is to be decided) can be exchanged. A vote can be taken and decision/agreement can be made based on : majority, minimum vote, mean, etc.
- Presence of failure : Processors can fail or misbehave intentionally. Several rounds of message exchanges might be needed before agreement can be reached.

4.2 Byzantine Agreement Problem

- **The Problem :** "Several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general. After observing the enemy, they must decide upon a common plan of action. Some of the generals may be traitors, trying to prevent the loyal generals from reaching agreement."
- Three or more generals are agree to attack or to retreat. Once the commander is issues the order, lieutenants to the commander are to decide to attack or retreat.
- But the one or more of the generals may be treacherous, i.e. faulty.
- If the commander is treacherous, he proposes attacking to one general and retreating to another.
- If a lieutenant is treacherous, he tells one of his peers that the commander told him to attack and another that they are to retreat.
- Source processor broadcasts its values to others. Solution must meet following objectives :

Agreement : All non-faulty processors agree on the same value.

Validity : If source is nonfaulty, then the common agreed value must be the value supplied by the source processor.

- "If source is faulty then all non - faulty processors can agree on any common value". "Value agreed upon by faulty processors is irrelevant"
- Fig 4.2.1 shows Byzantine agreement.

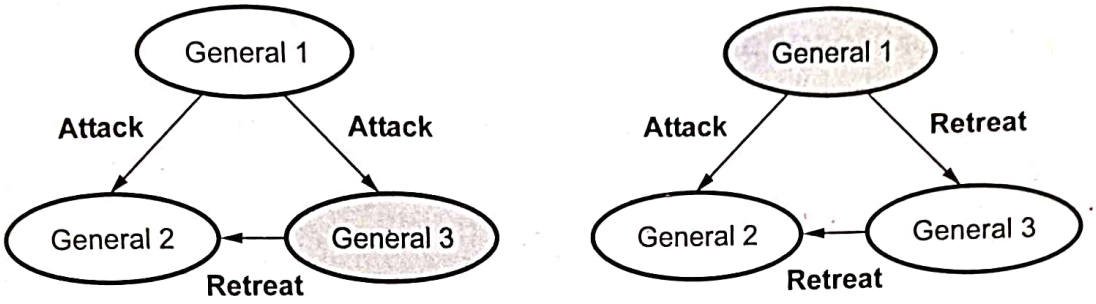


Fig. 4.2.1

- No solution for three processes can handle a single traitor. In a system with m faulty processes agreement can be achieved only if there are $2m+1$ (more than 2/3) functioning correctly.

4.2.1 Consensus Problem

i. Every processor broadcasts its initial value to all other processors.

ii. Initial values may be different for different processors.

iii. Every processor has its own initial value.

iv. All non faulty processors must agree on a single common value.

- If initial value of non-faulty processors is different then all non - faulty processors can agree on any common value.
- Value agreed upon by faulty processors is irrelevant

1. **Agreement :** All non-faulty processors agree on the same single value.
2. **Validity :** If initial value of every non-faulty processor is v , then the common agreed value by all non-faulty processors must be v.

- "If initial value of non-faulty processors is different then all non - faulty processors can agree on any common value". "Value agreed upon by faulty processors is irrelevant".

4.2.2 Interactive Consistency Problem

i. Every processor has its own initial value.

ii. All non faulty processors must agree on a set of common values.

- In all the previous mentioned problems, all non faulty processors must reach an agreement.
- In Byzantine and consensus problems, agreement is on a single value.
- In interactive consistency problem, agreement is on a set of common values.
- In Byzantine agreement problem, only one processor initializes the value where as in other two cases, every processor has its own initial value.

4.3 Overview of Results

- Consensus is not solvable in asynchronous system even if one process can fail by crashing.

Sr. No.	Failure mode	Synchronous system	Asynchronous system
1.	No failure	Agreement attainable	Agreement attainable
2.	Crash failure	Agreement attainable [f < n process]	Agreement not attainable
3.	Byzantine failure	Agreement attainable [f <= (n – 1) /3] byzantine process	Agreement not attainable

4.4 Solution to Byzantine Agreement Problem

- First defined and solved by Lamport.
- An arbitrary source processor broadcasts its initial value to all others.
- If the source processor is faulty, other non-faulty processor can agree on any common value.
- Faulty processors' values and agreements do not matter.
- If faulty processors are in majority, then non-faulty processors cannot reach an agreement.
- Number of faulty processors, m, cannot exceed : trunc[(n–1)/3].
- This bound can be relaxed for systems using authenticated messages.
- Solution must meet following objectives :

i. Agreement : All non-faulty processors agree on the same value.

ii. Validity : If source is nonfaulty, then the common agreed value must be the value supplied by the source processor.

4.4.1 Impossible Scenario

- Consider a system with 3 processors : p0, p1, p2.
- Two possibilities :
  - Case 1 : p0 (source) is not faulty. p2 is faulty. p1 should agree upon 1 as the value. Not possible.
  - Case 2 : p0 is faulty. p1 may agree on 1 and p2 on 0
  - Fig. 4.4.1 processor p0 non-faulty and processor p0 faulty

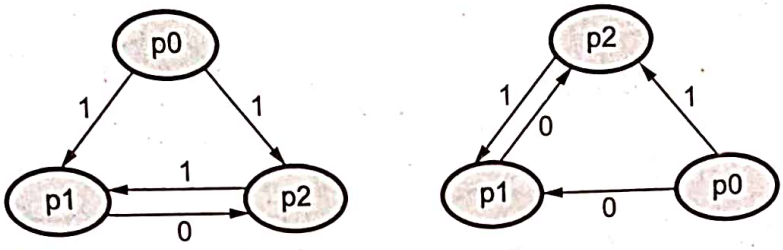


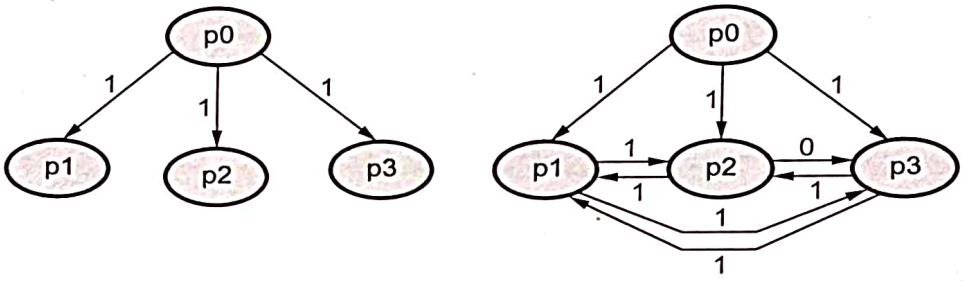
Fig. 4.4.1

4.4.2 Lamport-Shostak-Pease Algorithm

- This algorithm also known as Oral Message Algorithm OM(m) where m is the number of faulty processors
- 'n' = Number of processors and n >= 3m+1
- Algorithm is recursively defined as follows :
- Algorithm OM(0)
  1. Source processor sends its values to every processor
  2. Each processor uses the value it receives from source. [If no value is received default value 0 is used]
- Algorithm OM(m), m>0
  1. Source x broadcasts value to all processes
  2. Let vi = value received by process i from source (0 if no value received). Process i acts as a new source and initiates OM(m – 1), sending vi to remaining (n – 2) processes
  3. For each i, j, i ? j, let vj = value received by process i from process j in step 2 using O(m–1). Process i uses the value majority(v1, v2, ..., vn – 1)
- **Time complexity = m+1 rounds**
- **Message complexity = $O(n^m)$**
- You can reduce message complexity to polynomial by increasing time

Lamport's Algorithm : Example 1

- System with 4 processors : p0, p1, p2, p3. p0 is source, p2 is faulty.
- Assumption : possible values are only 1 and 0.
- Step 1 : p0 initiates the initial value to be 1. (Algorithm OM(1), m =1).
- Step 2 : OM(0). p1 sends 1 to {p2, p3}. p3 sends 1 to {p1,p2}
- p2 (the faulty one) sends 1 to p1 and 0 to p3.
- Step 3 : Majority function at p1 and p3 is 1, which is the desired result. (Not bothered about p2, the faulty one).



(a) Processor p0 executes the algorithm OM(1)

(b) Processor p1, p2, p3 executes the algorithm OM(0)

Fig. 4.4.2

1408

Example 2 :

- System with 4 processors : p0, p1, p2, p3. p0 is source, and is faulty.
- Assumption : Possible values are only 1 and 0.
- Step 1 : p0 initiates the initial value to be 1 for p1 and p3. For p2, it sends a 0(Algorithm OM(1), m =1).
- Step 2 : OM(0). p1 sends 1 to {p2, p3}. p3 sends 1 to {p1,p2}
- p2 sends 0 to p1 and p3.
- Step 3 : Majority function at p1, p2, p3 is still the same (1), which is the desired result.

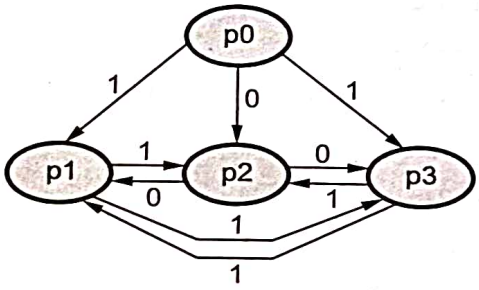


Fig. 4.4.3

4.5 Agreement in a Failure-Free System (Synchronous and Asynchronous)

- In a failure-free system, consensus can be reached by collecting information from the different processes, arriving at a "decision," and distributing this decision in the system.
- A distributed mechanism would have each process broadcast its values to others, and each process computes the same function on the values received.
- Synchronous system : This is implemented in a constant number of rounds. Common knowledge of the decision value can be obtained using an additional round.
- Asynchronous system : Consensus can similarly be reached in a constant number of message hops.

4.6 Agreement in Synchronous Systems with Failures AU : May-22

- Algorithm for consensus with up to f fail-stop processes in a system of n processes are as follows

```
int f ; // global constant : maximum number of crash failures tolerated
integer: x← local value;
Process P_i executes the consensus algorithm for up to f crash failures
for round from 1 to f +1 do
if the current value of x has not been broadcast then
broadcast(x);
 y_j ← value (if any) received from process j in this round;
 x ← min ∀_j (x, y_j)
output x as the consensus value
```

- The agreement condition is satisfied because in the f + 1 rounds, there must be at least one round in which no process failed.
- The validity condition is satisfied because processes do not send fictitious values in this failure model
- The termination condition is seen to be satisfied.
- Complexity : There are f + 1 rounds, where f < n. The number of messages is at most $O(n^2)$ in each round, and each message has one integer. Hence the total number of messages is $O((f+1) \cdot n^2)$.

University Question

1. *List the agreement statements that should be followed in synchronous systems with failure.*

AU : May-22, Marks 13

4.7 Introduction of Check-pointing and Rollback Recovery

AU : Dec.-22

- Checkpointing and rollback recovery are well-known techniques that all processes to make progress in spite of failures. The failures under consideration are transient problems such as hardware errors and transaction aborts.
- When failure occurs, the process rolls back to its most recent checkpoint, assume the state saved in that checkpoint and resumes execution.
- Rollback recovery treats a distributed system application as a collection of processes that communicate over a network. The saved state is called a checkpoint and the procedure of restarting from a previously checkpointed state is called rollback recovery.
- In distributed system, rollback recovery is complicated because message induce inter-process dependencies during failure free operation.
- Checkpoint is a designated place in a program at which normal process is interrupted specifically to preserve the status information necessary to allow resumption of processing at a later time.
- Coordinated checkpointing: here processes coordinate their checkpoints to form a system wide consistent state.

University Question

1. *Illustrate briefly the two kinds of checkpoints for checkpoint algorithm.*

AU : Dec.-22, Marks 13

4.8 Background and Definitions

4.8.1 System Model

- We consider the distributed system with fixed number of processes ($P_1, P_2, \ldots, P_n$). They are communicating with each other by using messages. The processes do not share a common memory or a common clock.
- Fig. 4.8.1 shows system consisting of three processes and their interactions.
- Rollback recovery protocol generally make assumptions about the reliability of the inter-process communication.

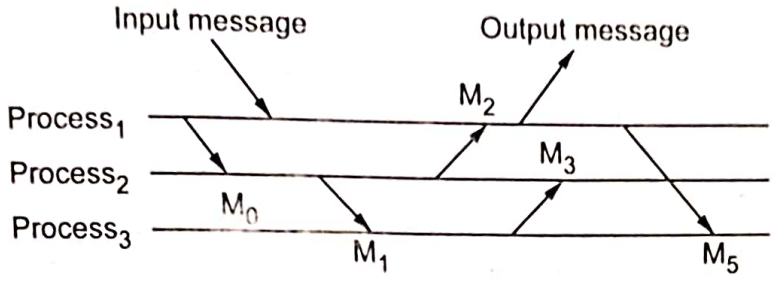


Fig. 4.8.1 Three processes and their interactions

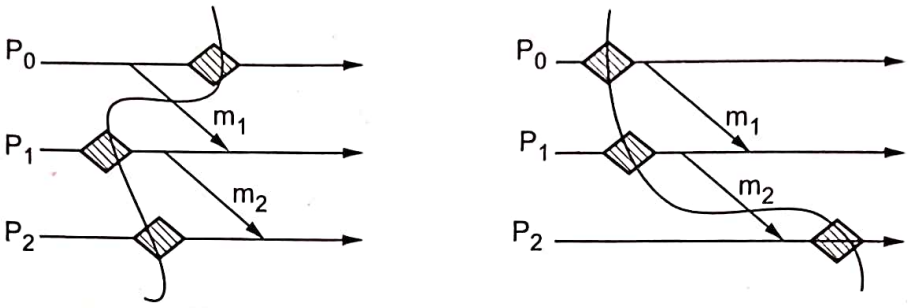
- The computation is asynchronous, i.e., each process progresses at its own speed and messages are exchanged through reliable channels, whose transmission delays are finite but arbitrary.
- The messages generated by the underlying distributed application are referred to as computation messages.
- The messages generated by processes to advance checkpoints are referred to as system messages.

4.8.2 Local Checkpoint

- Local checkpoint is a snapshot of the state of the process at a given instance and the event of recording the state of a process is called local checkpointing.
- Each checkpoint taken by a process is assigned a unique sequence number. The i^{th} ($i > 0$) checkpoint of process P_p is assigned a sequence number i and is denoted by $C_{p,i}$.
- We also assume that each process P_p takes an initial checkpoint $C_{p,0}$ immediately before execution begins and ends with a virtual checkpoint that represents the last state attained before termination.
- The i^{th} checkpoint interval of process P_p denotes all the computation performed between its i^{th} and $(i+1)^{th}$ checkpoint, including the i^{th} checkpoint but not the $(i+1)^{th}$ checkpoint.

4.9 Consistent Set of Checkpoints

- Fig. 4.9.1 shows consistent and inconsistent set.
- Checkpointing in distributed systems requires that all processes (sites) that interact with one another establish periodic checkpoints.
- All the sites save their local states : ***local checkpoints***. All the local checkpoints, one from each site, collectively form a ***global checkpoint***.



(a) Consistent state (b) Inconsistent state

Fig. 4.9.1

- The domino effect is caused by orphan messages, which in turn are caused by rollbacks.

Strongly consistent set of checkpoints

- The most recent distributed snapshot in a system is also called the *recovery line.*
- Fig. 4.9.2 shows recovery line and checkpoint.

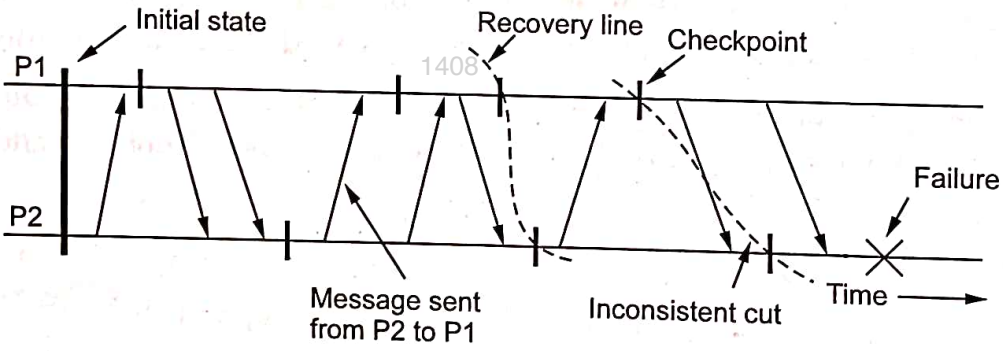


Fig. 4.9.2 Recovery line and checkpoint

- Establish a set of local checkpoints (one for each process in the set) such that no information flow takes place (i.e., no orphan messages) during the interval spanned by the checkpoints.
- A strongly consistent set of checkpoints (recovery line) corresponds to a strongly consistent global state.
- There is one recovery point for each process in the set during the interval spanned by checkpoints; there is no information flow between any pair of processes in the set and process in the set and any process outside the set.
- A consistent set of checkpoints corresponds to a consistent global state.

- Fig. 4.9.3 shows consistent set of checkpoint.

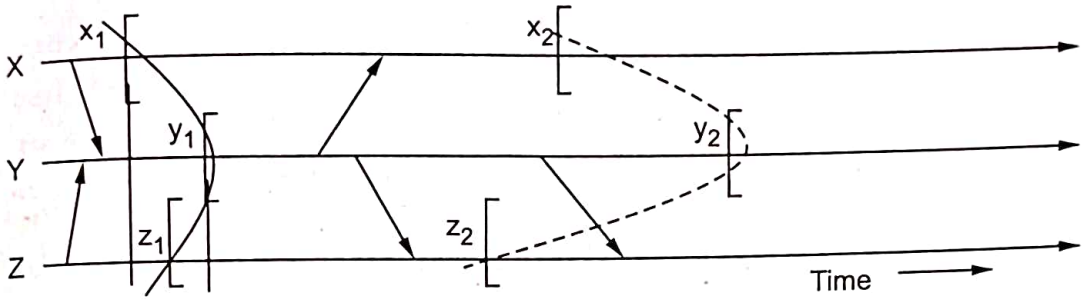


Fig. 4.9.3 Consistent set of checkpoint

- Set $\{x_1, y_1, z_1\}$ is a strongly consistent set of checkpoints.
- Set $\{x_2, y_2, z_2\}$ is a consistent set of checkpoints (need to handle lost messages).
- No local checkpoint includes an effect whose would be undone due to the rollback of another process.

Consistent set of checkpoints

- Similar to the consistent global state.
- Each message that is received in a checkpoint (state) should also be recorded as sent in another checkpoint (state).
- Suppose that Y fails after receiving message 'm'. If Y restarts from checkpoint, message 'm' is lost due to rollback.

Checkpoint notation

- Each node maintains :
  1. Monotonically increasing counter with which each message from that node is labelled.
  2. Records of the last message from and the first message to all other nodes.

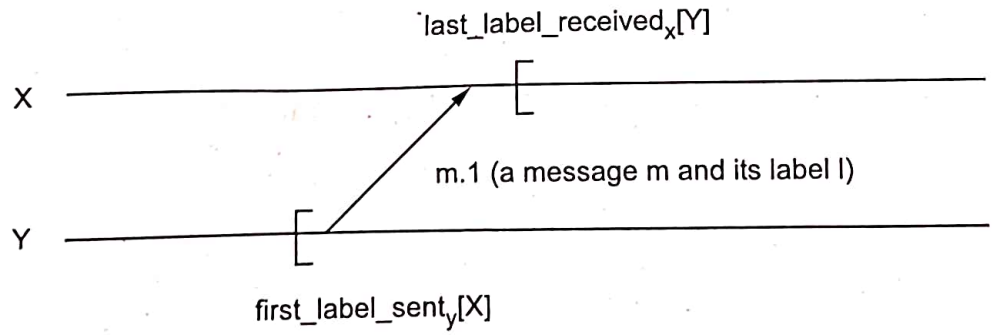


Fig. 4.9.4

Note : "sl" denotes a "smallest label" that is < any other label and "ll" denotes a "largest label" that is > any other label.

Simple method for taking consistent set of checkpoint

- Every process takes a checkpoint after sending every message.
- The set of the most recent checkpoints is always consistent.

4.9.1 Synchronous Checkpointing and Recovery

- Livelock problem during recovery is avoided by taking a consistent set of checkpoints.
- Algorithm is said to be synchronous when the processes involved coordinate their local checkpointing actions such that the set of all recent checkpoints in the system guaranteed to be consistent.

4.9.1.1 Checkpointing Algorithm

- Make some simplifying assumptions
  1. Processes communicate by exchanging messages through channels.
  2. Channels are FIFO, end-to-end protocols cope with message loss due to rollback recovery.
  3. Communication failures do not partition the network.
- A single process invokes the algorithm. The checkpoint and the rollback recovery algorithms are not invoked concurrently.

Two types of checkpoints

1. **Tentative :** A temporary checkpoint that is made a permanent checkpoint on the successful termination of the checkpoint algorithm
2. **Permanent :** A local checkpoint at a process.

Phase One

- Initiating process P_i takes a tentative checkpoint and requests that all the processes take tentative checkpoints.
- Each process informs P_i whether it succeeded in taking a tentative checkpoint.
- If P_i learns that all processes have taken tentative checkpoints, P_i decides that all tentative checkpoints should be made permanent.
- Otherwise, P_i decides that all tentative checkpoints should be discarded.

Phase Two

1. P_i propagates its decision to all processes.
2. On receiving the message from P_i, all processes act accordingly.

- Between tentative checkpoint and commit/abort of checkpoint process must hold back messages.
- Does this guarantee we have a strongly consistent state ? Can you construct an example that shows we can still have lost messages ?

Synchronous Checkpointing : Properties

- All or none of the processes take permanent checkpoints.
- There is no record of a message being received but not sent.

Optimization of the Checkpoint Algorithm

- A minimal number of processes take checkpoints.
- All processes from which P_i has received messages after it has taken its last checkpoint take a checkpoint to record the sending of those messages.
- Fig. 4.9.5 shows the checkpoints taken unnecessarily.

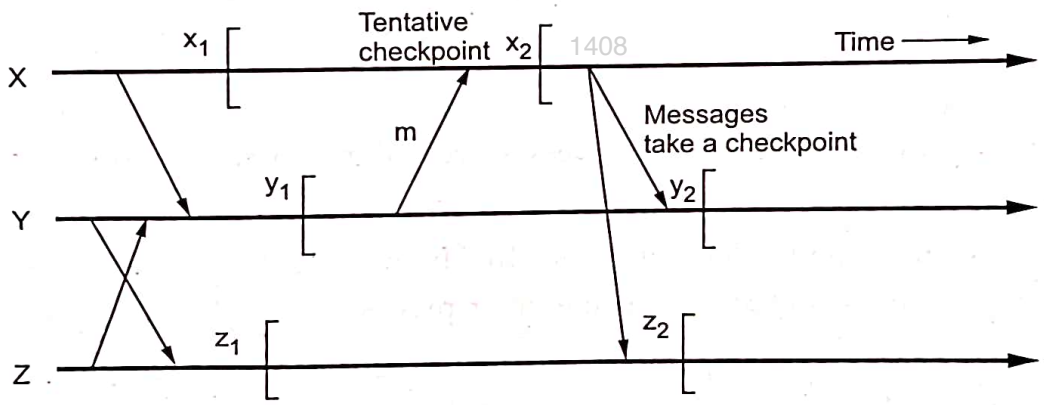


Fig. 4.9.5 Checkpoints taken unnecessarily

1. Process X decides to initiate checkpoint algorithm after receiving message 'm'.
2. It take a tentative checkpoints x_2 and sends take tentative checkpoint messages to processes Y and Z , causing Y and Z to take checkpoints y_2 and z_2 respectively.
3. Now { x_2, y_2, z_2} forms a consistent set of checkpoints.
4. { x_2, y_2, z_1} also forms a consistent set of checkpoints.

5. Y takes tentative checkpoint only if the last message received by X from Y was sent after Y sent the first message after the last checkpoint (last_recv(x, y) >= first_send(y, x)).

- When a process takes a checkpoint, it will ask all other processes that sent messages to the process to take checkpoints.

Synchronous Checkpointing Disadvantages

1. Additional messages must be exchanged to coordinate checkpointing.
2. Synchronization delays are introduced during normal operations.
3. No computational messages can be sent while the checkpointing algorithm is in progress.
4. If failure rarely occurs between successive checkpoints, then the checkpoint algorithm places an unnecessary extra load on the system, which can significantly affect performance.

4.9.2 The Rollback Recovery Algorithm

- Restore the system state to a consistent state after a failure with assumptions: single initiator, checkpoint and rollback recovery algorithms are not invoked concurrently.

Phase One :

- Process P_i checks whether all processes are willing to restart from their previous checkpoints.
- A process may reply "no" if it is already participating in a checkpointing or recovering process initiated by some other process.
- If all processes are willing to restart from their previous checkpoints, P_i decides that they should restart.
- Otherwise, P_i decides that all the processes continue with their normal activities.

Phase Two :

- P_i propagates its decision to all processes.
- On receiving P_i's decision, the processes act accordingly.

Optimization

- A minimum number of processes roll back.
- Y will restart from its permanent checkpoint only if X is rolling back to a state where the sending of one or more messages from X to Y is being undone.

- Fig. 4.9.6 shows the unnecessary rollback.

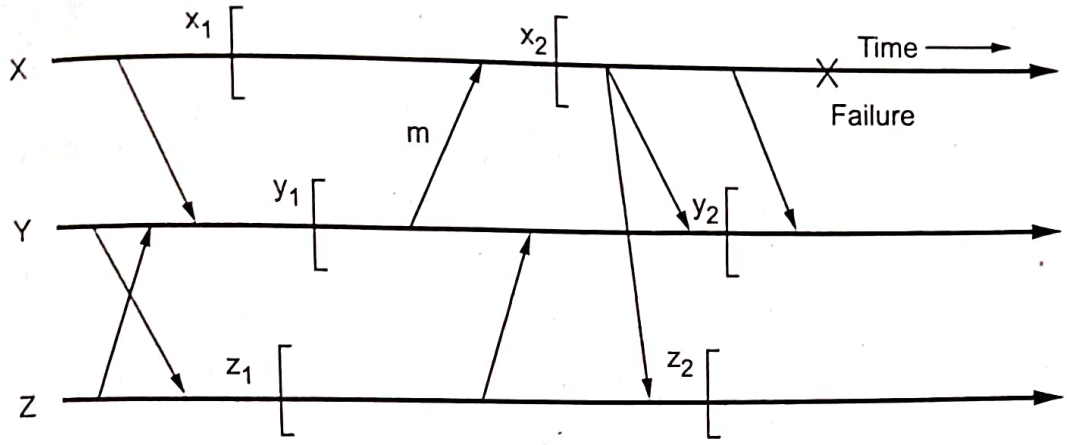


Fig. 4.9.6 Unnecessary rollback

4.9.3 Message Types

- In-transit message : Messages that have been sent but not yet received
- Lost messages : Messages whose "send" is done but "receive" is undone due to rollback
- Delayed messages : Messages whose "receive" is not recorded because the receiving process was either down or the message arrived after rollback
- Orphan messages : Messages with "receive" recorded but message "send" not recorded - do not arise if processes roll back to a consistent global state
- Duplicate messages : Arise due to message logging and replaying during process recovery.

4.10 Issues in Failure Recovery

AU : May-22, Dec.-22

- Recovery refers to restoring a system to its normal operational state. Once a failure has occurred, it is essential that the process where the failure happened can recover to a correct state.
- Fundamental to fault tolerance is the recovery from an error.
- Resources are allocated to executing processes in a computer. For example : a process has memory allocated to it and a process may have locked shared resources, such as files and memory.
- Following are some solution on process recovery :
  1. Reclaim resources allocated to process
  2. Undo modification made to databases and
  3. Restart the process
  4. Or restart process from point of failure and resume execution

- In distributed process recovery, undo effect of interactions of failed process with other cooperating processes.

4.10.1 Basic Concept

- System is combination of hardware and software components. These components provide a specified service.
- Failure of a system occurs when the system does not perform its service in the manner specified.
- An erroneous state of the system is a state which could lead to a system failure by a sequence of valid state transitions.
- A system is said to "fail" when it cannot meet its promises. A failure is brought about by the existence of "errors" in the system.
- A system is said to have a failure if the service it delivers to the user deviates from compliance with the system specification for a specified period of time.
- Fig. 4.10.1 shows concept of fault and recovery.

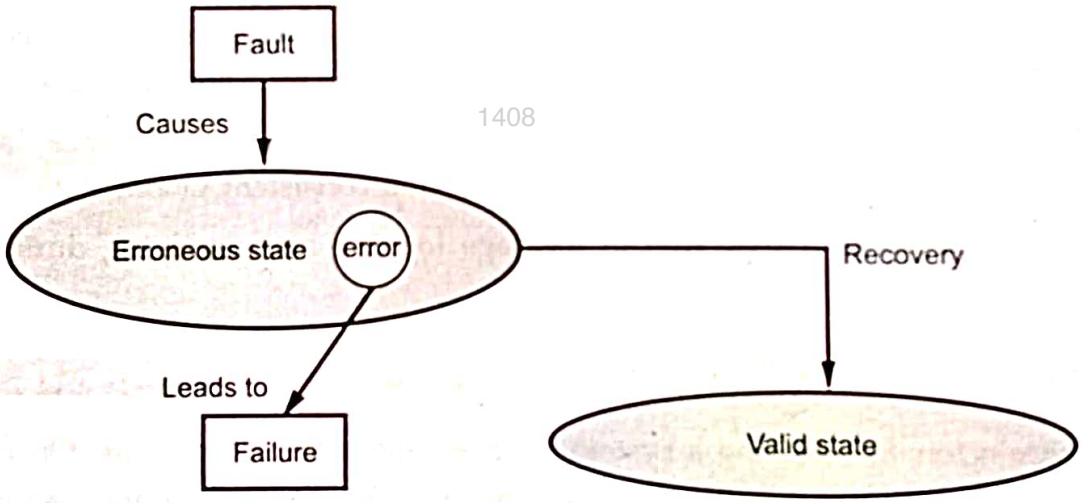


Fig. 4.10.1 Concept of recovery

- System failure : System does not meet requirements, i.e. does not perform its services as specified.
- Erroneous system state : State which could lead to a system failure by a sequence of valid state transitions.
- Error : the part of the system state which differs from its intended value.
- Fault : Anomalous physical condition, e.g. design errors, manufacturing problems, damage, external disturbances.

University Questions

1. *Discuss the issues in failure recovery with an example.* **AU : May-22, Marks 13**
2. *Illustrate the different types of failures in distributed systems and explain how to prevent them.* **AU : Dec.-22, Marks 13**

4.11 Checkpoint-based Recovery

- The basic idea behind checkpoint-recover is the saving and restoration of system state. By saving the current state of the system periodically or before critical code sections, it provides the baseline information needed for the restoration of lost state in the event of a system failure.
- While the cost of checkpoint-recovery can be high, by using techniques like memory exclusion, and by designing a system to have as small a critical state as possible may minimize the cost of checkpointing enough to be useful in even cost sensitive embedded applications.
- When a system is checkpointed, the state of the entire system is saved to non-volatile storage.
- The checkpointing mechanism takes a snapshot of the system state and stores the data on some non-volatile storage medium.
- Clearly, the cost of a checkpoint will vary with the amount of state required to be saved and the bandwidth available to the storage mechanism being used to save the state.
- In the event of a system failure, the internal state of the system can be restored, and it can continue service from the point at which its state was last saved.
- Typically this involves restarting the failed task or system, and providing some parameter indicating that there is state to be recovered.
- Depending on the task complexity, the amount of state, and the bandwidth to the storage device this process could take from a fraction of a second to many seconds.
- This technique provides protection against the transient fault model. Typically upon state restoration the system will continue processing in an identical manner as it did previously.
- This will tolerate any transient fault, however if the fault was caused by a design error, then the system will continue to fail and recover endlessly. In some cases, this may be the most important type of fault to guard against, but not in every case.

1. Uncoordinated Checkpointing

- Each process has autonomy in deciding when to take checkpoints
- Advantages : The lower runtime overhead during normal execution
- Disadvantages :
  a. Domino effect during a recovery
  b. Recovery from a failure is slow because processes need to iterate to find a consistent set of checkpoints.
  c. Each process maintains multiple checkpoints and periodically invoke a garbage collection algorithm
  d. Not suitable for application with frequent output commits
- The processes record the dependencies among their checkpoints caused by message exchange during failure-free operation.
- In order to determine a consistent global checkpoint during recovery, the processes record the dependencies among their checkpoints caused by message exchange during failure free operation.

Direct dependency tracking technique

- Assume each process P_i starts its execution with an initial checkpoint $C_{i,0}$.
- $I_{i,x}$ is a checkpoint interval and it is an interval between $C_{i,x-1}$ and $C_{i,x}$.
- When P_j receives a message m during $I_{j,y}$ it records the dependency from $I_{i,x}$ to $I_{j,y}$ which is later saved onto stable storage when P_j takes $C_{j,y}$.
- When failure occurs, the recovering process initiates rollback by broad casting a dependency request message to collect all the dependency information maintained by each process.

2. Coordinated Checkpointing

- Coordinated checkpointing simplifies failure recovery and eliminates domino effects in case of failures by preserving a consistent global checkpoint on stable storage.
- However, the approach suffers from high overhead associated with the checkpointing process.
- Two approaches are used to reduce the overhead: first is to minimize the number of synchronization messages and the number of checkpoints, the other is to make the checkpointing process nonblocking.

Blocking Checkpointing

- After a process takes a local checkpoint, to prevent orphan messages, it remains blocked until the entire checkpointing activity is complete.
- Fig. 4.11.1 shows blocking checkpointing.

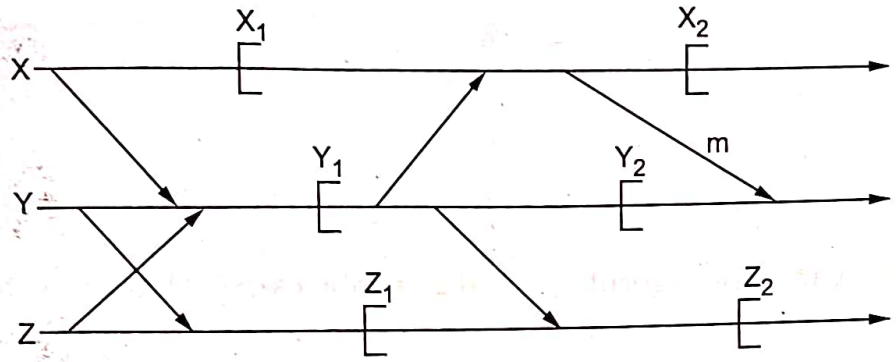


Fig. 4.11.1 Blocking checkpointing

- When a process takes a checkpoint, it engages a protocol to coordinate with other processes to take checkpoint
  a) Coordinator takes a checkpoint; broadcasts a message to all processes.
  b) Process receives this message and halts execution; takes tentative checkpoint.
  c) Coordinator receives acknowledgement from all processes; broadcasts commit message to end protocol.
  d) Process receives commit message, removes old permanent checkpoint and makes tentative checkpoint permanent.
  e) Processes resume execution.
- Disadvantages : The computation is blocked during the checkpointing.

Non-blocking Checkpointing

- The processes need not stop their execution while taking checkpoints.
- Key issue with coordinated checkpointing: Being able to prevent a process from receiving application messages that could make the checkpoint inconsistent.
- Problem can be avoided by preceding the first post-checkpoint message on each channel by a checkpoint request, forcing each process to take a checkpoint upon receiving the first checkpoint-request message.
- A fundamental problem in coordinated checkpointing is to prevent a process from receiving application messages that could make the checkpoint inconsistent.

- Fig. 4.11.2 shows non-blocking checkpoint.

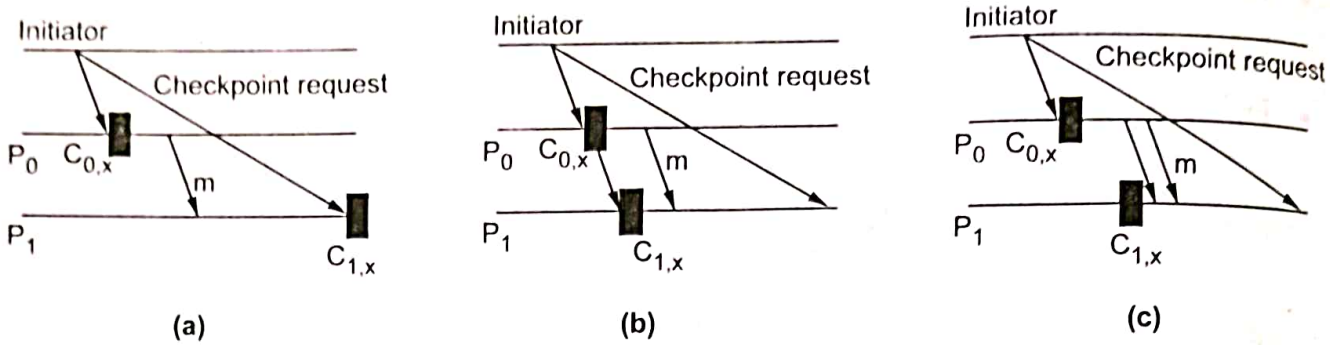


Fig. 4.11.2 Non-blocking checkpoint

- The Chandy-Lamport algorithm is the nonblocking algorithm for coordinated checkpointing.

Example of Coordinated Checkpointing

a) Checkpoint inconsistency : The process P_0 sent message "m" after receiving a checkpoint request from the checkpoint coordinator. Assume message "m" reaches process P_1 before the checkpoint request. This situation results in an inconsistent checkpoint since checkpoint $C_{1,x}$ shows the receipt of message "m" from P_0, while checkpoint $C_{0,x}$ does not show m being sent from P_0.

b) Solution with FIFO channels : If channels are FIFO, this problem can be avoided by preceding the first post-checkpoint message on each channel by a checkpoint request, forcing each process to take a checkpoint before receiving the first post-checkpoint message.

3. Communication-induced Checkpointing

- The Communication-Induced Checkpointing (CIC) protocols are popular, because they help in bounding rollback propagation during failure recovery, by ensuring that each checkpoint taken is part of a consistent global checkpoint of the distributed computation, while at the same time allowing each process to take checkpoints independently.
- It avoids domino effect, while allowing processes to take some of their checkpoints independently.
- Communication-induced checkpointing forces each process to take checkpoints based on information piggybacked on the application messages it receives from other processes.
- Checkpoints are taken such that a system-wide consistent state always exists on stable storage, thereby avoiding the domino effect.

- Communication-induced checkpointing piggybacks protocol and related information on each application message.
- The receiver of each application message uses the piggybacked information to determine if it has to take a forced checkpoint to advance the global recovery line.
- The forced checkpoint must be taken before the application may process the contents of the message. In contrast with coordinated checkpointing, no special coordination messages are exchanged.
- Two types of communication-induced checkpointing are model-based checkpointing and index-based checkpointing.
- **Model-based checkpointing** relies on preventing patterns of communications and checkpoints that could result in inconsistent states among the existing checkpoints. A model is set up to detect the possibility that such patterns could be forming within the system, according to some heuristic.
- **Index-based communication**-induced checkpointing works by assigning monotonically increasing indexes to checkpoints, such that the checkpoints having the same index at different processes form a consistent state
- Communication-Induced Checkpointing protocols do not take useless checkpoints.

4.11.1 Difference between Uncoordinated, Coordinated and Communication Induced Check Pointing

Parameters	Uncoordinated, check pointing	Coordinated check pointing	Communication induced check pointing
Number of check point	Many	One	Many
Domino effect	Possible	No	No
Orphan process	Possible	No	No
Rollback extent	Unbounded	Last global checkpoint	Possible several checkpoints
Output commit	Not possible	Global coordination required	Global coordination required

4.12 Coordinated Checkpointing Algorithm

- Uncoordinated checkpointing may lead to domino effect or to live-lock.
- Two basic approaches to checkpoint coordination :

1. The Koo-Toueg algorithm, which has a process to initiate the system-wide checkpointing process.
2. An algorithm which staggers checkpoints in time; Staggering checkpoints can help avoid near-simultaneous heavy loading of the disk system.

Koo-Toueg Algorithm :

- Koo-Toueg in 1987 proposed a coordinated checkpointing and recovery technique that takes a consistent set of checkpointing and avoids domino effect and Livelock problems during the recovery.
- Suppose P wants to establish a checkpoint at P_3. This will record that q_1 was received from Q, to prevent q_1 from being orphaned, Q must checkpoint as well.
- Fig. 4.12.1 shows Koo-Toueg algorithm.

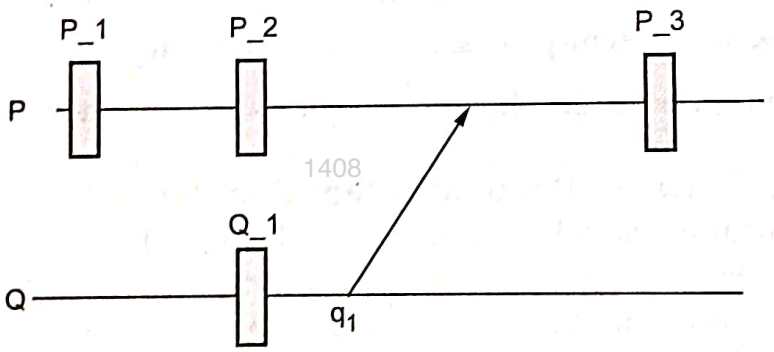


Fig. 4.12.1 Koo-Toueg algorithm

- Thus, establishing a checkpoint at P_3 by P forces Q to take a checkpoint to record that q_1 was sent
- An algorithm for such coordinated checkpointing has two types of checkpoints - tentative and permanent
- P first records its current state in a tentative checkpoint, then sends a message to all other processes from whom it has received a message since taking its last checkpoint
- Call the set of such processes Π .
- The message tells each process in Π (e.g., Q), the last message, m_qp, that P has received from it before the tentative checkpoint was taken.
- If m_qp was not recorded in a checkpoint by Q : to prevent m_qp from being orphaned, Q is asked to take a tentative checkpoint to record sending m_qp.

- If all processes in Π, that need to, confirm taking a checkpoint as requested, then all tentative checkpoints can be converted to permanent.
- If some members of Π, are unable to checkpoint as requested, P and all members of Π abandon the tentative checkpoints, and none are made permanent.
- This may set off a chain reaction of checkpoints.
- Each member of Π can potentially spawn a set of checkpoints among processes in its corresponding set.

4.13 Algorithm for Asynchronous Checkpointing and Recovery

- Here we discuss, Juang-Venkatesan algorithm for asynchronous checkpointing and recovery.
- Assumptions : communication channels are reliable, delivery messages in FIFO order, infinite buffers, message transmission delay is arbitrary but finite.
- They gave an algorithm that is based on asynchronous checkpointing. During the recovery, we need to find a consistent set of checkpoints to which the system can be restored.
- In this recovery algorithm each process keeps track of both the number of messages it has send to and received from other processes.
- Several iterations of rollback by processes are also involved in this recovery. This algorithm avoids the existence of Orphan messages.
- Whenever a process rollbacks, it is necessary for all other processes to find if any message send by the rolled back process has become an orphan message.
- Orphan messages are discovered, if the number of messages received by processor Pi from process Pj is greater than number of messages sent by process Pj to process Pi, according to the current state of processes, then one or more message at process Pj are orphan messages.
- Then process Pj must rollback to a state where number of messages received are equal to the number of messages sent by the process.
- Two type of log storage are maintained :
  a. Volatile log : short time to access but lost if processor crash. Move to stable log periodically.
  b. Stable log : longer time to access but remained if crashed
- Asynchronous checkpointing : After executing an event, the triplet is recorded without any synchronization with other processes. Local checkpoint consists of set of records, first are stored in volatile log, then moved to stable log.

Recovery algorithm :

- Number of messages received by p_j from p_j, from the beginning of computation to checkpoint
- Number of messages sent by p_j to p_j, from the beginning of computation to checkpoint
- Idea : From the set of checkpoints, find a set of consistent checkpoints. Doing that based on the number of messages sent and received.

4.14 Two Marks Questions with Answers

Q.1 State the use of Rollback recovery. **AU : Dec.-22**

Ans. :
- Restore the system back to a consistent state after a failure.
- Achieve fault tolerance by periodically saving the state of a process during the failure-free execution.
- Treats a distributed system application as a collection of processes that communicate over a network.

Q.2 What is consensus in distributed system ? **AU : Dec-22**

Ans. : Each process has an initial value and all the correct processes must agree on a single value.

1408

Q.3 Write the purpose of using checkpoints. **AU : May-22**

Ans. : Checkpointing is most typically used to provide fault tolerance to applications. Checkpointing techniques are useful not only for availability, but also for program debugging, process migration, and load balancing.

Q.4 What do you mean by agreement problem in distributed system ? **AU : May-22**

Ans. : In the agreement problem, to achieve overall system reliability in the presence of a number of faulty processes and single process has the initial value.

Q.5 What is the difference between agreement and consensus problem ?

Ans. : The difference between the agreement problem and the consensus problem is that, in the agreement problem, a single process has the initial value, whereas in the consensus problem, all processes have an initial value.

Q.6 Define recovery.

Ans. : Recovery refers to restoring a system to its normal operational state. Once a failure has occurred, it is essential that the process where the failure happened can recover to a correct state. Fundamental to fault tolerance is the recovery from an error.

Q.7 List classification of failures.

Ans. : Failures in a computer system can be classified as follows :

1. Process failure
2. System failure
3. Secondary storage failure
4. Communication medium failure

Q.8 Define domino effect.

Ans. : The process of a cascaded rollback may lead to what is called the domino effect.

Q.9 What is orphan process ?

Ans. : An orphan process is a process that survives the crash of another process, but whose state is inconsistent with the crashed process after its recovery.

Q.10 Explain two types of checkpoints.

Ans. : **1. Tentative :** A temporary checkpoint that is made a permanent checkpoint on the successful termination of the checkpoint algorithm.

2. Permanent : A local checkpoint at a process.

Q.11 List drawback of synchronous check pointing.

Ans. : 1. Additional messages must be exchanged to coordinate check pointing.

2. Synchronization delays are introduced during normal operations.
3. No computational messages can be sent while the check pointing algorithm is in progress.
4. If failure rarely occurs between successive checkpoints, then the checkpoint algorithm places an unnecessary extra load on the system, which can significantly affect performance.

Q.12 How shadow versions are helpful in recovery ?

Ans. : Shadow version uses a map to locate versions of the server's objects in a file called a version store. The map associates the identifiers of the server's objects with the positions of their current versions in the version store. The versions written by each transaction are 'shadows' of the previous committed versions. The transaction status entries and intentions lists are stored separately. When a transaction commits, a new map is made by copying the old map and entering the positions of the shadow versions. To complete the commit process, the new map replaces the old map.

Q.13 Define fault and failure. What are different approaches to fault-tolerance ?

Ans. : Fault : Anomalous physical condition, e.g. design errors, manufacturing problems, damage, external disturbances

Failure of a system occurs when the system does not perform its service in the manner specified.

Q.14 List the requirements of consensus algorithm to hold for execution.

Ans. : The requirements of consensus algorithm to hold for execution are : Termination, Agreement and Integrity.

Q.15 What are the performance aspects of agreement protocols ?

Ans. : Following metrics are used :

1. Time : No of rounds needed to reach an agreement.
2. Message traffic : Number of messages exchanged to reach an agreement.
3. Storage overhead : Amount of information that needs to stored at processors during execution of the protocol.

Q.16 What are the application of agreement algorithm ?

Ans. : Applications of agreement algorithms

- Fault-tolerant clock synchronization.
- Distributed systems require physical clocks to synchronized.
- Physical clocks have drift problem.
- Agreement protocols may help to reach a common clock value.
- Synchronizing distributed clocks :
- At any time, values of clocks of all non-faulty processes must be approximately equal.
- There is a small bound on amount by which the clock of a non-faulty process is changed during re-synchronization.

Q.17 State Byzantine agreement problem.

Ans. : • In the Byzantine agreement problem, n processors communicate with each other in order to reach an agreement on a binary value b. There are bad processors that may collaborate with each other in order to prevent an admissible agreement. Each processor has an initial binary value. The agreement must reflect to a certain extent the majority among the initial value.

Q.18 What is local checkpoints ?

Ans. : A process may take a local check point anytime during the execution. The local checkpoints of different processes are not coordinated to form a global consistent checkpoint.

Q.19 What is forced checkpoints ?

Ans. : To guard against the domino effect, a communication induced checkpoint protocol piggybacks protocol-speci?c information to application messages that processes exchange. Each process examines the information and occasionally is forced to take a checkpoint according to the protocol.

Q.20 Explain useless checkpoints.

Ans. : A useless checkpoint of a process is one that will never be part of a global consistent state. Useless checkpoints are not desirable because they do not contribute to the recovery of the system from failures, but they consume resources and cause performance overhead

Q.21 What is checkpoint intervals ?

Ans. : A checkpoint interval is the sequence of events between two consecutive checkpoints in the execution of a process.

Q.22 Define orphan messages.

Ans. : Messages with receive recorded but message send not recorded are called the orphan messages.

Q.23 Write down the goals to achieve an optimal assignment. **AU : Dec.-16**

Ans. : Goal to achieve an optimal assignment is finding a minimum weight cutest. The weight of a cutset is the sum of the weights of the edges in the cutset. This sums up the execution and communication costs for that assignment.

Q.24 Define consistent cut. **AU : May-17**

Ans. : A cut C = $\{c_1, c_2, c_3, \ldots\}$ is consistent if for all sites there are no events e_i and e_j such that (e_i --> e_j) and (e_j --> c_j) and (e_i -/-> c_i)

Q.25 What is the basic idea behind task assignment approach ? **AU : May-17**

Ans. : Basic idea :

a. A process has already been split up into pieces called tasks.

b. The amount of computation required by each task and the speed of each CPU are known.

c. The cost of processing each task on every node is known.

d. The IPC costs between every pair of tasks is known.

e. Precedence relationships among the taks are known.

f. Reassignment of tasks is not possible.

Q.26 Mention some motivations for replication. **AU : May-18**

Ans. : The motivations for replication include :

Performance enhancement : The caching of data at clients and servers is by now familiar as a means of performance enhancement.

Increased availability : Users require services to be highly available.

Fault tolerance : Highly available data is not necessarily strictly correct data. It may be out of date.

UNIT V

5 Cloud Computing

Syllabus

Definition of Cloud Computing – Characteristics of Cloud – Cloud Deployment Models – Cloud Service Models – Driving Factors and Challenges of Cloud – Virtualization – Load Balancing – Scalability and Elasticity – Replication – Monitoring – Cloud Services and Platforms: Compute Services – Storage Services – Application Services

Contents

5.1 Definition of Cloud Computing

- Idea of cloud computing was introduced by computer scientist John McCarthy publicly in 1961.
- Then in 1969, Leonard Kleinrock, a chief scientist of the ARPANET project comments about Internet.
- The general public has been leveraging forms of Internet-based computer utilities since the mid-1990s through various incarnations of search engines, e-mail services, open publishing platforms and other types of social media.
- Though consumer-centric, these services popularized and validated core concepts that form the basis of modern-day cloud computing.
- The Salesforce.com provides remote service from 1990 to organization. Amazon launched its web services in 2002 and it provides services to organization for storage and remote computing.
- Cloud computing definition as per Gartner "a style of computing in which scalable and elastic IT-enabled capabilities are delivered as a service to external customers using Internet technologies".
- In 2008, Gartner's original definition of cloud was changed. In the definition, "massively scalable" was used instead of "scalable and elastic."
- **NIST definition of cloud :** Cloud computing is a pay-per-use model for enabling available, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, services) that can be rapidly provisioned and released with minimal management effort or service-provider interaction.
- The above cloud definition was published by NIST in 2009, followed by a revised version after further review and industry input that was published in September of 2011.
- Cloud computing refer to a variety of services available over the Internet that deliver compute functionality on the service provider's infrastructure.
  - Its environment (infrastructure) may actually be hosted on either a grid or utility computing environment, but that doesn't matter to a service user.
  - Cloud computing refer to a variety of services available over the Internet that deliver compute functionality on the service provider's infrastructure. Its environment (infrastructure) may actually be hosted on either a grid or utility computing environment, but that doesn't matter to a service user.

Cloud Computing = Software as a service + Platform as a service
+ Infrastructure as a service + Data as a service

- Cloud computing is a general term used to describe a new class of network based computing that takes place over the Internet, basically a step on from utility computing.
- In other words, this is a collection/group of integrated and networked hardware, software and Internet infrastructure (called a platform).
- Fig. 5.1.1 shows cloud symbol. It denotes cloud boundary.
- Using the Internet for communication and transport provides hardware, software and networking services to clients.
- These platforms hide the complexity and details of the underlying infrastructure from users and applications by providing very simple graphical interface or API.

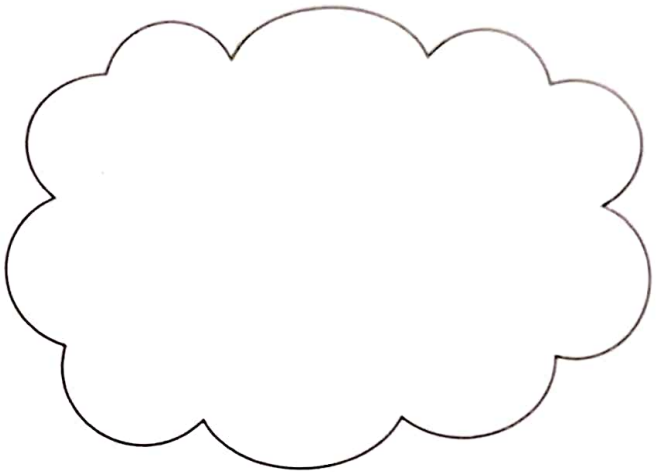


Fig. 5.1.1 Cloud symbol

- In addition, the platform provides on demand services that are always on anywhere, anytime and anyplace. Pay for use and as needed, elastic.
- The hardware and software services are available to the general public, enterprises, corporations and business markets.

5.1.1 Cloud Components

- Cloud computing solutions are made up of several elements. Fig. 5.1.2 shows cloud components.
  1. **Clients :** Mobile, terminals or regular computers.
  2. **Benefits :** Lower hardware costs, lower IT costs, security, data security, less power consumption, ease of repair or replacement, less noise.
  3. **Data centers :** Collection of servers where the application to subscribe is housed. It could be a large room in the basement of your building or a room full of servers on the other side of the world.
  4. **Virtualizing servers :** Software can be installed allowing multiple instances of virtual servers to be used and a dozen virtual servers can run on one physical server.
  5. **Distributed servers :** Servers don't all have to be housed in the same location. It can be in geographically disparate locations. If something were to happen at one site, causing a failure, the service would still be accessed through another site. If the cloud needs more hardware, they can add them at another site.

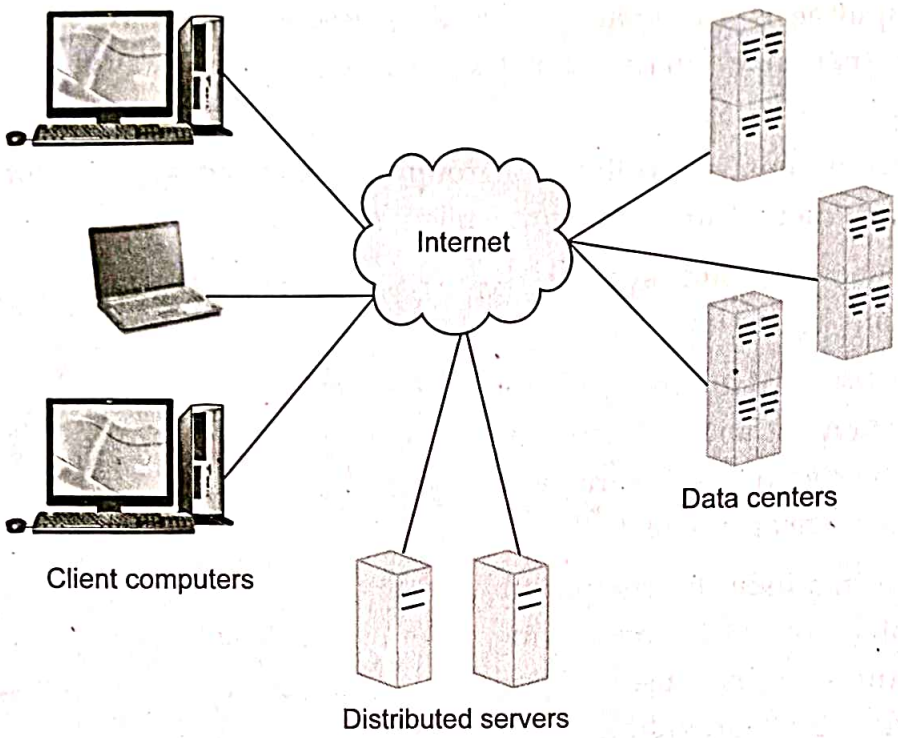


Fig. 5.1.2 Cloud components

5.1.2 Pros and Cons of Cloud Computing

Pros of cloud computing :

1. **Lower computer costs :** Since applications run in the cloud, not on the desktop PC, your desktop PC does not need the processing power or hard disk space demanded by traditional desktop software.
2. **Improved performance :** Computers in a cloud computing system boot and run faster because they have fewer programs and processes loaded into memory.
3. **Reduced software costs :** Instead of purchasing expensive software applications, you can get most of what you need for free.
4. **Instant software updates :** When you access a web-based application, you get the latest version - without needing to pay for or download an upgrade.
5. **Improved document format compatibility :** You do not have to worry about the documents you create on your machine being compatible with other user's applications or operating systems.
6. **Unlimited storage capacity :** Cloud computing offers virtually limitless storage.
7. **Increased data reliability :** Unlike desktop computing, in which if a hard disk crashes and destroy all your valuable data, a computer crashing in the cloud should not affect the storage of your data.

8. **Universal document access :** All your documents are instantly available from wherever you are.
9. **Latest version availability :** The cloud always hosts the latest version of your documents; as long as you are connected, you are not in danger of having an outdated version.
10. **Easier group collaboration :** Sharing documents leads directly to better collaboration.
11. **Device independence :** Move to a portable device and your applications and documents are still available.

Cons of cloud computing :

1. It requires a constant Internet connection : Cloud computing is impossible if you cannot connect to the Internet.
2. Features might be limited.
3. Stored data might not be secure : With cloud computing, all your data is stored on the cloud.
4. Does not work well with low-speed connections.

5.1.3 Application of Cloud Computing

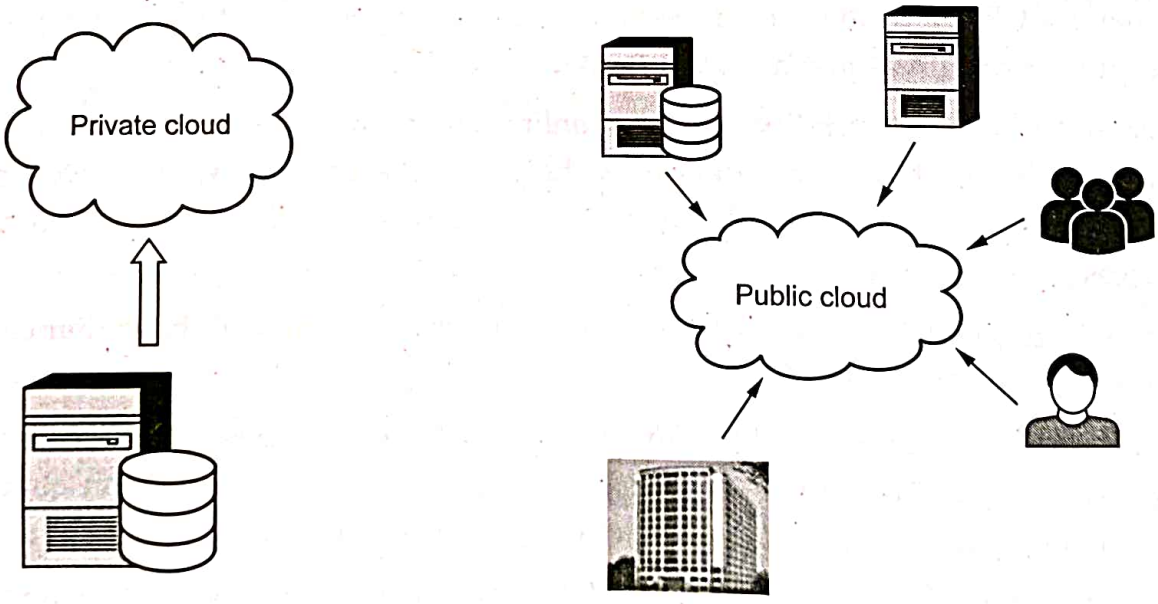
1. Through cloud cost flexibility, online marketplace gains access to more powerful analytics online. Cloud takes away the need to fund the building of hardware, installing software or paying dedicated software license fees.
2. Greater business scalability enables online video retailer to meet spikes in demand : Cloud enables businesses not just IT operations to add or provision computing resources just at the time they're needed.
3. Greater market adaptability provides online entertainment platform the ability to reach any type of customer device. A third of the executives we surveyed believe cloud can help them adapt to diverse user groups with a diverse assortment of devices.
4. Masked complexity enables access to services, no matter how intricate the technology they're built on.
5. With context-driven variability, "intelligent assistants" are possible. "Because of its expanded computing power and capacity, cloud can store information about user preferences, which can enable product or service customization," the report states.
6. Ecosystem connectivity enables information exchange across business partners.

5.2 Characteristics of Cloud

1. **On-demand self-service :** A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed without requiring human interaction with each service's provider.
2. **Ubiquitous network access :** Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms.
3. **Location-independent resource pooling :** The provider's computing resources are pooled to serve all consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand.
4. **Rapid elasticity :** Capabilities can be rapidly and elastically provisioned to quickly scale up, and rapidly released to quickly scale down.
5. **Pay per use :** Capabilities are charged using a metered, fee-for-service, or advertising-based billing model to promote optimization of resource use.

5.3 Cloud Deployment Models

- Cloud deployment models are refers to the location and management of the cloud's infrastructure.
- Deployment models are defined by the ownership and control of architectural design and the degree of available customization. Cloud deployment models are private public and community clouds.
- Fig. 5.3.1 shows cloud deployment model.



(a) Private cloud (b) Public cloud

Fig. 5.3.1

1 Public cloud :

- The cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services.
- Public cloud is a huge data centre that offers the same services to all its users. The services are accessible for everyone and much used for the consumer segment.
- Examples of public services are Facebook, Google and LinkedIn.
- **Public cloud benefits :**
  a) Low investment hurdle : Pay for what user use.
  b) Good test/development environment for applications that scale to many servers.
- **Public cloud risks :**
  a) Security concerns : Multi-tenancy and transfers over the Internet.
  b) IT organization may react negatively to loss of control over data center function.

2. **Private cloud :**

- The cloud infrastructure is operated solely for a single organization. It may be managed by the organization or a third party and may exist on-premises or off-premises.
- **Private cloud benefits :**
  a) Fewer security concerns as existing data center security stays in place.
  b) IT organization retains control over data center.
- **Private cloud risks :**
  a) High investment hurdle in private cloud implementation, along with purchases of new hardware and software.
  b) New operational processes are required; old processes not all suitable for private cloud.

3. **Community cloud :**

- The cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g. mission, security requirements, policy, or compliance considerations). It may be managed by the organizations or a third party and may exist on-premises or off-premises.

4. **Hybrid cloud :**

- The cloud infrastructure is a composition of two or more clouds (private, community or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load-balancing between clouds).

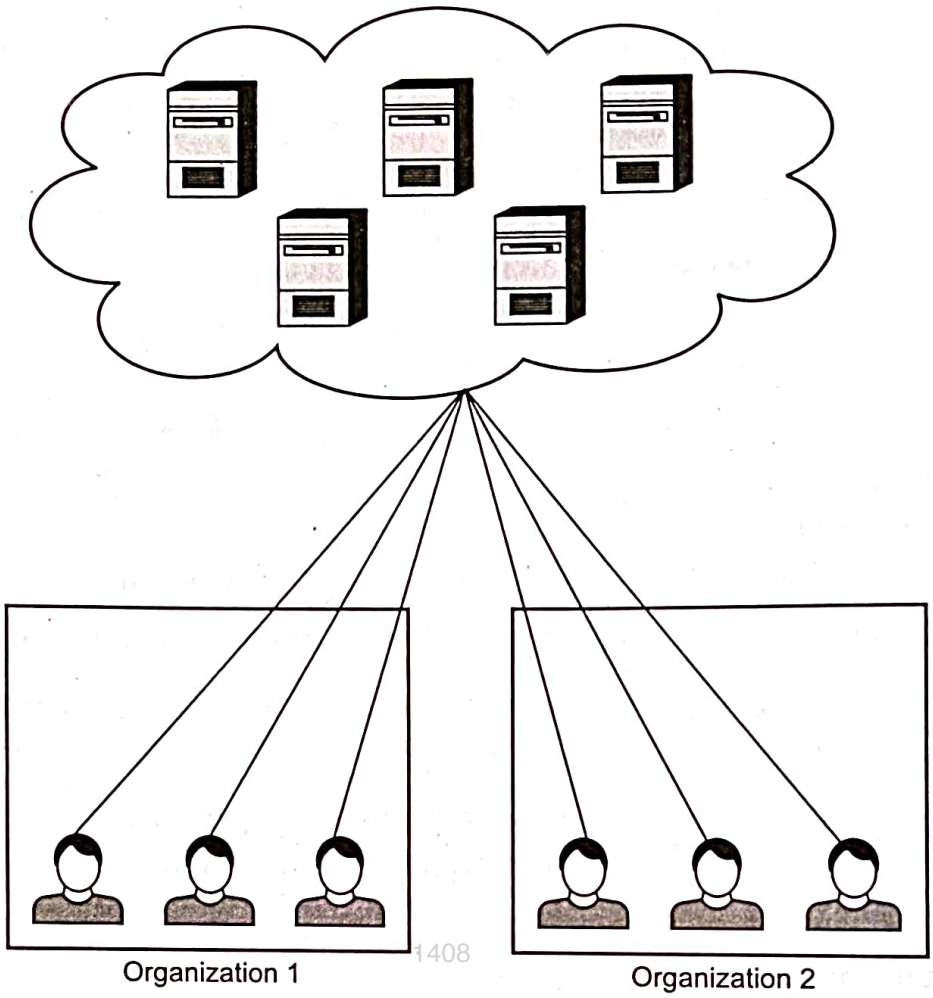


Fig. 5.3.2 Community cloud

- **Hybrid cloud benefits :**
  a) Operational flexibility : Run mission critical on private cloud, dev/test on public cloud
  b) Scalability : Run peak and bursty workloads on the public cloud
- **Hybrid cloud risks :**
  a) Hybrid clouds are still being developed; not many in real use
  b) Control of security between private and public clouds, some of same concerns as in public cloud

5.3.1 Difference between Public and Private Cloud

Public cloud	Private cloud
Public cloud infrastructure is offered via web applications and also as web services over Internet to the public.	Private cloud infrastructure is dedicated to a single organization.
Support multiple customer.	Support dedicated customer.

Full utilized of infrastructure.	Does not utilize shared infrastructure.
Security is low as compared to private cloud.	High level of security.
Low cost.	High cost.
Azure, Amazon Web Services, Google App Engine and Force.com are a few examples of public clouds.	An example of the private cloud is NIRIX's one server with dedicated servers.

5.4 Cloud Service Models

- Service models describe the type of service that the service provider is offering. The best-known service models are Software as a Service, Platform as a Service, and Infrastructure as a Service.
- The service models build on one another and define what a vendor must manage and what the client's responsibility is.
- Service models : This consists of the particular types of services that you can access on a cloud computing platform.
- Cloud service is any service made available to users on demand via the Internet from a cloud computing provider's servers as opposed to being provided from a company's own on-premises servers.
- Cloud services are designed to provide easy, scalable access to applications, resources and services, and are fully managed by a cloud services provider.
- A cloud service can exist as a simple web-based software program with a technical interface invoked via the use of a messaging protocol, or as a remote access point for administrative tools or larger environments and other IT resources.
- The organization that provides cloud-based IT resources is the cloud provider. Cloud providers normally own the IT resources for lease by cloud consumers, and could also resell IT resources leased from other providers.

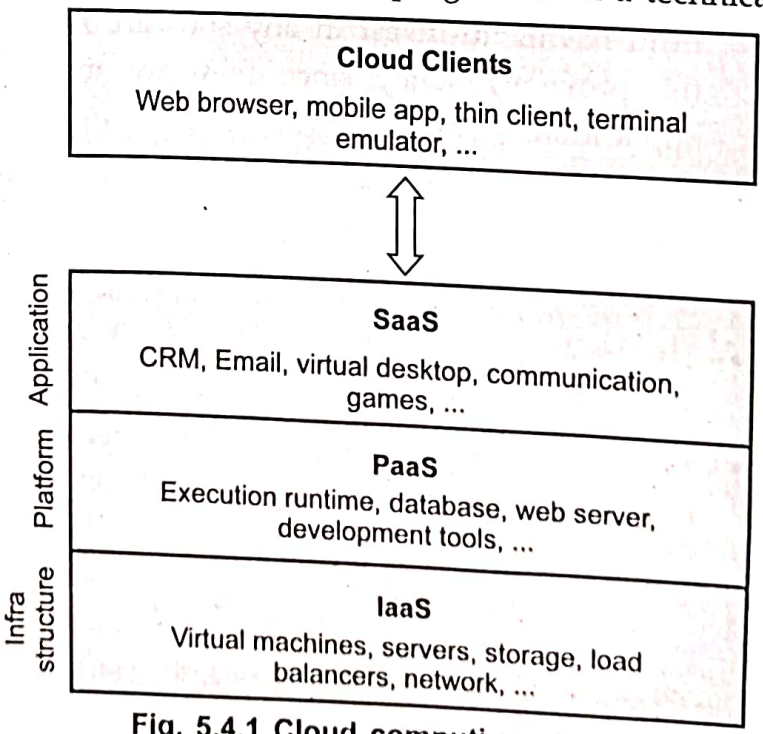


Fig. 5.4.1 Cloud computing stack

- Cloud computing, often described as a stack, has a broad range of services built on top of one another under the name cloud.
- Fig. 5.4.1 shows cloud computing stack.
- Flavors of cloud computing is as follows;
  1. SaaS applications are designed for end-users, delivered over the web
  2. PaaS is the set of tools and services designed to make coding and deploying those applications quick and efficient
  3. IaaS is the hardware and software that powers it all - servers, storage, networks, operating systems.

5.4.1 Software as a Service (SaaS)

- Model in which an application is hosted as a service to customers who access it via the Internet.
- The provider does all the patching and upgrades as well as keeping the infrastructure running.
- The traditional model of software distribution, in which software is purchased for and installed on personal computers, is referred to as product.
- In this model, the user, client or consumer runs an application from a cloud infrastructure. Through an interface such as a web browser, the client or user may access this application from a variety of devices.
- The complete application is offered as on demand service. This saves the client from having to invest in any software licenses or servers up front, and can save the provider money since they are maintaining and providing only a single application.
- In this model, the client does not manage cloud infrastructure, networks or servers, storage, or operating systems. Even, Microsoft, Google, and Zoho offer SaaS.
- The SaaS concept can be defined as providing robust "web-based, on-demand software, storage and various applications" to organizations.
- The SaaS model has emerged as an alternative to traditional one-time licensing for providing and maintaining the software needed by knowledge workers within organizations.
- Fig. 5.4.2 shows SaaS.

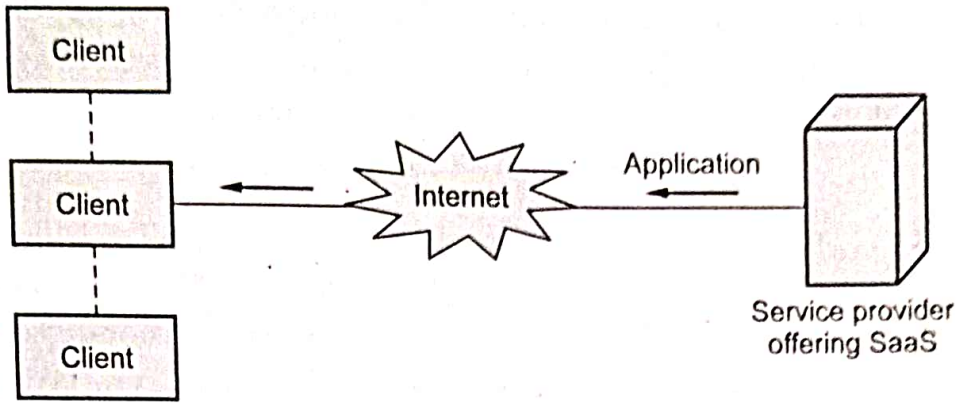


Fig 5.4.2 SaaS

Characteristics of SaaS :

1. Software applications or services are stored remotely.
2. A user can then access these services or software applications via the Internet.
3. In most cases, a user does not have to install anything onto their host machine, all they require is a web browser to access these services and in some cases, a browser may require additional plug-in/add-on for certain services.
4. 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 Internet.
5. Application delivery from a one-to-many model, as opposed to a traditional one-to- one model.

Benefits of SaaS :

1. You only pay for what you use
2. Easier administration and invoicing
3. Automatic updates and patch management
4. Compatibility : All users have access to the same version of software
5. Easier collaboration
6. It support automated update and patch management services.

5.4.2 Platform as a Service (PaaS)

- Platform as a service is another application delivery model and also known as **cloud-ware**. Supplies all the resources required to build applications and services completely from the Internet, without having to download or install software.

- Services include : Application design, development, testing, deployment, and hosting, team collaboration, web service integration, database integration, security, scalability, storage, state management, and versioning.
- PaaS is closely related to SaaS but delivers a platform from which to work rather than an application to work with.
- This model involves software encapsulated and offered as a service, from which higher levels of service may then be built. The user, customer, or client in this model is the one building applications which then run on the provider's infrastructure.
- This in turn provides customers and clients with the capability to deploy applications onto the cloud infrastructure using programming tools and languages, which the provider supports.
- The customer still does not manage the framework, network, servers or operating system, but has control over deployed applications and sometimes over the hosting environment itself.
- Some examples of Platform as a Service include Google's App Engine or Force.com
- PaaS consists of following components :
  1. Browser based development studio
  2. Pay contrary to billing
  3. Management and supervising tools
  4. Seamless deployment to host run time environment.
- **Characteristics of PaaS :**
  1. It support multi-tenant architecture.
  2. It support for development of group collaboration.
  3. PaaS systems can be deployed as public cloud services or as private cloud services.
  4. Provision of runtime environments. Typically each runtime environment supports either one or a small set of programming languages and frameworks
  5. Support for custom applications. Support for the development, deployment and operation of custom applications.
  6. Preconfigured capabilities. Many PaaS systems are characterized by capabilities that are preconfigured by the provider, with a minimum of configuration available to developers and customer operations staff.
  7. Support for porting existing applications. While many PaaS systems are primarily designed to support "born on the cloud" applications.

8. Security is an important characteristic in PaaS. It needs to provide authentication and authorization to differentiate the access rights of different users

Benefits of Paas :

1. Scalability including rapid allocation and deallocation of resources with a pay-as-you-use model
2. Reduced capital expenditure
3. Reduced lead times with on-demand availability of resources
4. Self-service with reduced administration costs
5. Reduced skill requirements
6. Support of team collaboration
7. Ability to add new users quickly.

5.4.3 Infrastructure as a Service (IaaS)

- IaaS gives the storage room likeness to the in-house datacenter stood out from various organizations sorts.
- Center datacenter framework segments are capacity, servers (registering units), the system itself, and administration apparatuses for foundation upkeep and checking.
- Each of these parts has made a different market specialty. While some little organizations have practical experience in just a single of these IaaS cloud specialties, vast cloud suppliers like Amazon or Right Scale have offerings over all IaaS territories.
- Fig. 5.4.3 shows IaaS.

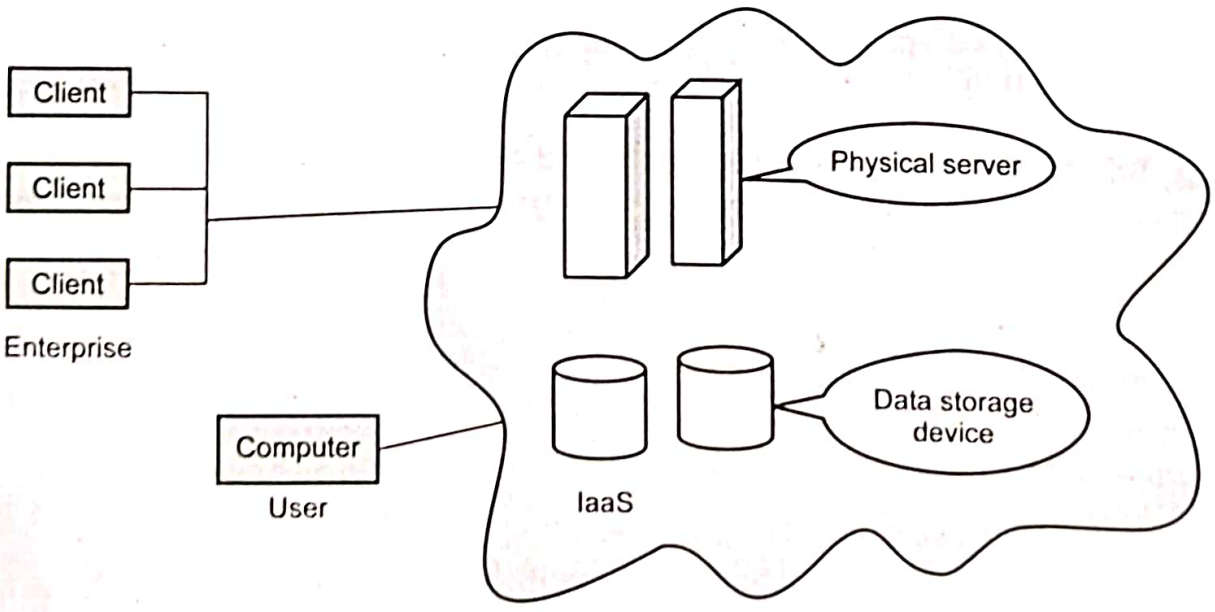


Fig. 5.4.3 IaaS

- It offers the hardware so that your organization can put whatever they want onto it. Rather than purchase servers, software, racks, and having to pay for the datacenter space for them, the service provider rents those resources :
  1. Server space 2. Network equipment
  3. Memory 4. CPU cycles 5. Storage space
- Again, the customer is not managing cloud infrastructure, but in this case, the customer does control operating systems, deployed applications, storage, and sometimes-certain networking components
- Examples : Amazon EC2, Rackspace Mosso, GoGrid
- IaaS server types :
  1. **Physical server :** Actual hardware is allocated for the customer's dedicated use.
  2. **Dedicated virtual server :** The customer is allocated a virtual server, which runs on a physical server that may or may not have other virtual servers.
  3. **Shared virtual server :** The customer can access a virtual server on a device that may be shared with other customers.

Advantages of IaaS :

1. Elimination of an expensive and staff-intensive data center
2. Ease of hardware scalability
3. Reduced hardware cost
4. On-demand, pay as you go scalability
5. Reduction of IT staff
6. Suitability for ad hoc test environments
7. Allows complete system administration and management
8. Support multiple tenants

5.4.4 Difference between IaaS, PaaS and SaaS

IaaS	PaaS	SaaS
IaaS gives users automated and scalable environments	PaaS provides a framework for quickly developing and deploying applications	SaaS makes applications available through the internet.
Amazon Web Services, for example, offers IaaS through the Elastic Compute Cloud, or EC2	Google Cloud Platform provides another PaaS option in App Engine	SaaS applications such as Gmail, Dropbox, Salesforce, or Netflix

In IaaS, infrastructure as a service.	In PaaS, platform as a service	In SaaS, software as a service
Virtual platform on which required operating environment and application deployed	Operating environment was included.	Operating environment largely irrelevant, fully functional application provided
IaaS is a cloud service that provides basic computing infrastructure: servers, storage, and networking resources. In other words, IaaS is a virtual data center	PaaS refers to cloud platforms that provide runtime environments for developing, testing, and managing applications	SaaS allows people to use cloud-based web applications.
Major IaaS providers include Amazon Web Services, Microsoft Azure, and Google Compute Engine.	Examples of PaaS services are Heroku and Google App Engine.	email services such as Gmail and Hotmail are examples of cloud-based SaaS services.
IaaS services are available on a pay-for-what-you-use model	PaaS solutions are available with a pay-as-you-go pricing model.	SaaS services are usually available with a pay-as-you-go pricing model
Used by IT administrator	Used by software developers	Used by end user

5.5 Driving Factors and Challenges of Cloud

1. Increased security vulnerabilities.
2. Reduced operational governance control.
3. Limited portability between cloud providers.
4. Multi-regional compliance and legal issues.

- Use of cloud for business purpose means that the responsibility over data security becomes shared with the cloud provider. Organization extends their trust boundary to cloud consumer to external cloud.
- It is clear that the security issue has played the most important role in hindering cloud computing acceptance.
- Without doubt, putting your data, running your software on someone else's hard disk using someone else's CPU appears daunting to many.
- Well-down security issues such as data loss, phishing, pose serious threats to organization's data and software.

5.6 Virtualization

- Virtualization is a broad term that refers to the abstraction of resources across many aspects of computing. For our purposes : One physical machine to support multiple virtual machines that run in parallel. Virtualization is a frame work or methodology of dividing the resources of computer into multiple execution environments.
- Virtualization is an abstraction layer that decouples the physical hardware from the operating system to deliver greater IT resource utilization and flexibility. It allows multiple virtual machines, with heterogeneous operating systems to run in isolation, side-by-side on the same physical machine.
- Virtualization means running multiple machines on a single hardware. The "Real" hardware invisible to operating system. OS only sees an abstracted-out picture. Only Virtual Machine Monitor (VMM) talks to hardware.
- It is "a technique for hiding the physical characteristics of computing resources from the way in which other systems, applications, or end users interact with those resources. This includes making a single physical resource appear to function as multiple logical resources; or it can include making multiple physical resources appear as a single logical resource."
- Fig. 5.6.1 shows concept of virtualization.

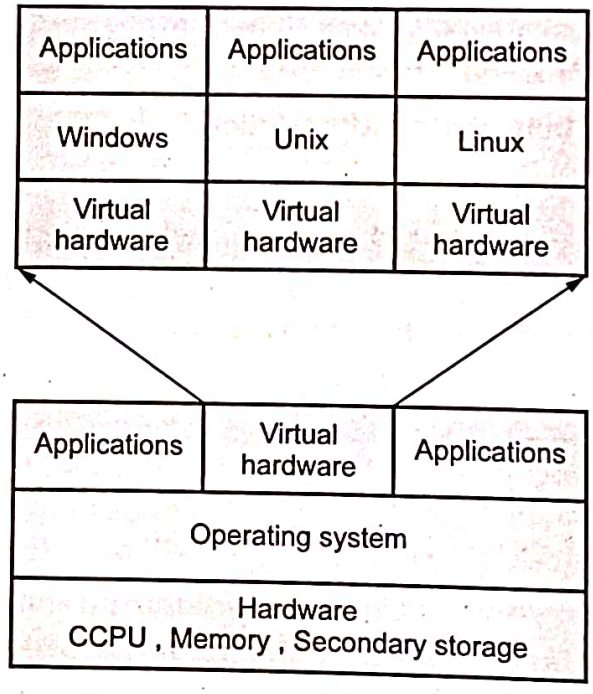


Fig. 5.6.1 Virtual machine

- It is divided into two main categories :

1. Platform virtualization involves the simulation of virtual machines.

2. Resource virtualization involves the simulation of combined, fragmented, or simplified resources.

- Following are the reasons for using virtualizations :
  a) Virtual machines offer software developers isolated, constrained, test environments.
  b) The most important function of virtualization is the capability of running multiple operating systems and applications on a single computer or server.
  c) Virtualization can usually improve overall application performance due to technology that can balance resources, and provide only what the user needs.
  d) It provides fault and error containment.
  e) It helps in building secured computing platform.
  f) Server virtualization provides a way to implement redundancy without purchasing additional hardware.

5.6.1 Hypervisor

- In computing, a hypervisor is a virtualization platform that allows multiple operating systems to run on a host computer at the same time. The term usually refers to an implementation using full virtualization.
- A hypervisor is a software layer installed on the physical hardware, which allows splitting the physical machine into many virtual machines. This allows multiple operating systems to be run simultaneously on the same physical hardware.
- The operating system installed on the virtual machine is called a guest OS and is sometimes also called an instance. The hardware the hypervisor runs on is called the host machine.
- A hypervisor management console, which is also called a Virtual Machine Manager (VMM), is computer software that enables easy management of virtual machines.
- Hypervisors are currently classified in two types : Type 1 and type 2.
- Type 1 hypervisor is software that runs directly on a given hardware platform. A "guest" operating system thus runs at the second level above the hardware.
- Fig. 5.6.2 shows Type 1 hypervisor.
- Type 1 VMs have no host operating system because they are installed on a bare system. An operating system running on a Type 1 VM is a full virtualization because it is a complete simulation of the hardware that it is running on.

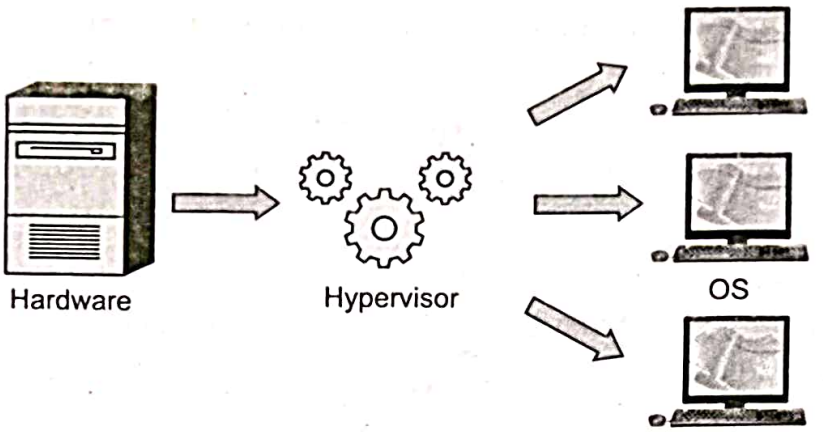


Fig. 5.6.2 Type 1 hypervisor

- Type 1 hypervisor is also called a native or bare-metal hypervisor that is installed directly on the hardware, which splits the hardware into several virtual machines where we can install guest operating systems.
- Type 2 Hypervisor is also known as hosted hypervisor. In this case, the hypervisor is installed on an operating system and then supports other operating systems above it.

5.6.2 Para-Virtualization

- Paravirtualization is a type of virtualization in which a guest operating system (OS) is recompiled, installed inside a virtual machine (VM), and operated on top of a hypervisor program running on the host OS.
- Para-virtualization refers to communication between the guest OS and the hypervisor to improve performance and efficiency.
- Para-virtualization involves modifying the OS kernel to replace non-virtualizable instructions with hyper-calls that communicate directly with the virtualization layer hypervisor.
- The hypervisor also provides hyper-call interfaces for other critical kernel operations such as memory management, interrupt handling and time keeping.
- Fig. 5.6.3 shows para-virtualization architecture.

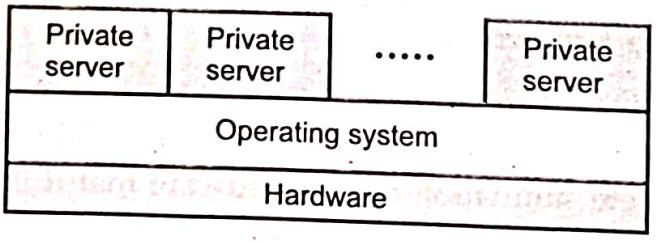


Fig. 5.6.3 Para-virtualization architecture

- In Para-virtualization, the virtual machine does not necessarily simulate hardware, but instead offers a special API that can only be used by modifying the "guest" OS. This system call to the hypervisor is called a "hypercall" in Xen.
- Xen is an open source para-virtualization solution that requires modifications to the guest operating systems but achieves near native performance by collaborating with the hypervisor.
- Microsoft Virtual PC is a para-virtualization virtual machine approach. User-mode Linux (UML) is another para-virtualization solution that is open source.
- Each guest operating system executes as a process of the host operating system. Cooperative Linux, is a virtualization solution that allows two operating systems to cooperatively share the underlying hardware.
- Linux-V server is an operating system-level virtualization solution for GNU/Linux systems with secure isolation of independent guest servers.
- The Linux KVM is virtualization technology that has been integrated into the mainline Linux kernel . Runs as a single kernel loadable module, a Linux kernel running on virtualization-capable hardware is able to act as a hypervisor and support unmodified Linux and Windows guest operating systems.
- Para-virtualization shares the process with the guest operating system.

Problems with para-virtualization

1. Para-virtualized systems won't run on native hardware.
2. There are many different para-virtualization systems that use different commands, etc.

- The main difference between full virtualization and paravirtualization in Cloud is that full virtualization allows multiple guest operating systems to execute on a host operating system independently while paravirtualization allows multiple guest operating systems to run on host operating systems while communicating .

5.6.3 Full-Virtualization

- Full Virtualization doesn't need to modify the host OS; it relies upon binary translation to trap and to virtualize certain sensitive instructions.
- Fig. 5.6.4 shows full virtualization.
- VMware Workstation applies full virtualization, which uses binary translation to automatically modify x86 software on-the-fly to replace critical instructions.
- Normal instructions can run directly on the host OS. This is done to increase the performance overhead - normal instructions are carried out in the normal manner,

but the difficult and precise executions are first discovered using a trap and executed in a virtual manner.

- This is done to improve the security of the system and also to increase the performance.

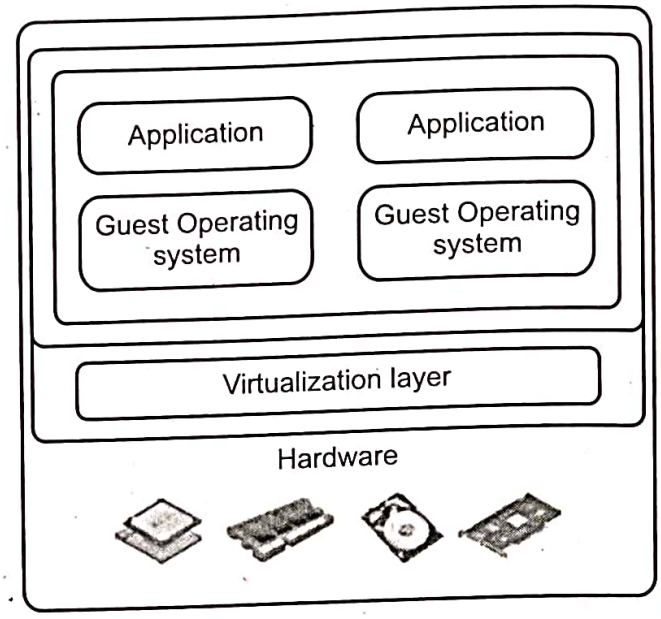


Fig. 5.6.4 Full virtualization

Host based virtualization :

- Virtualization implemented in a host computer rather than in a storage subsystem or storage appliance.
- Virtualization can be implemented either in host computers, in storage subsystems or storage appliances, or in specific virtualization appliances in the storage interconnect fabric.
- The guest OS are installed and run on top of the virtualization layer. Dedicated applications may run on the VMs. Certainly, some other applications can also run with the host OS directly.
- **Advantages** of host-based architecture :
  1. The user can install this VM architecture without modifying the host OS.
  2. The host-based approach appeals to many host machine configurations.

5.6.4 Difference between Full-Virtualization and Para-Virtualization

Sr. No.	Full Virtualization	Para Virtualization
1.	Full Virtualization relies upon binary translation to trap and to virtualize certain sensitive instructions. Example : VMware	Para-Virtualization refers to communication between the guest OS and the hypervisor to improve performance and efficiency. Example : Xen architecture
2.	Full Virtualization doesn't need to modify the host OS.	Para-Virtualization involves modification of OS kernel.
3.	Normal instructions can run directly on the host OS.	Para-virtualized systems won't run on native hardware.
4.	Full Virtualization uses binary translation and direct execution.	Para-Virtualization uses hyper - calls.
5.	Performance is good.	Performance is better in certain cases.
6.	Guest software does not require any modification since the undelying hardware is fully simulated.	Hardware is not simulated and the guest software run their own isolated domains.

5.6.5 Difference between Cloud and Virtualization

Sr. No.	Virtualization	Cloud Computing
1.	Virtualization is the process of creating a virtual environment on an existing server to run your desired program, without interfering with any of the other services provided by the server or host platform to other users.	Cloud computing means storing and accessing data and programs over the Internet instead of your computer's hard drive.
2.	Location of virtual machine is on a specific host.	Location of virtual machine is on any host.
3.	Instance storage is persistent.	Instance storage is shortly lived.
4.	Virtualization uses customizable VM resource like CPU and RAM.	Cloud computing uses standard VM resource like CPU and RAM
5.	Recovery from failures: attempt to recover failed VM.	Recovery from failures : Discard instance spin up new one.

5.6.6 Pros and Cons of Virtualization

a) Pros

1. Data center and energy-efficiency savings : As companies reduce the size of their hardware and server footprint, they lower their energy consumption.
2. Operational expenditure savings : Once servers are virtualized, your IT staff can greatly reduce the ongoing administration and management of manual work.
3. Reduced costs : It reduced cost of IT infrastructure.
4. Data does not leak across virtual machine.
5. Virtual machine is completely isolated from host machine and other virtual machine.
6. Simplifies resource management by pooling and sharing resources.
7. Significantly reduce downtime.
8. Improved performance of IT resources.

b) Cons

1. Not all hardware or software can be virtualized.
2. Not all servers are applications are specifically designed to be virtualization-friendly.

5.7 Load Balancing

- Load balancing can be defined as the process of task distribution among multiple computers, processes, disk, or other resources in order to get optimal resource utilization and to reduce the computation time.
- Load balancing is an important means to achieve effective resource sharing and utilization.
- Google, Yahoo!, Amazon, and Microsoft experience millions of user hits per day. Across the web, sites experience a wide range of network traffic requirements.
- To handle such web requests, the sites use a technique known as load balancing, to share the requests across multiple servers.
- Load balancing uses a server to route traffic to multiple servers which, in turn, share the workload.
- In general, load balancing algorithms can be divided into following three types :
  1. **Centralized approach :** In this approach, a single node is responsible for managing the distribution within the whole system.
  2. **Distributed approach :** In this approach, each node independently builds its own load vector by collecting the load information of other nodes. Decisions are made locally using local load vectors. This approach is more suitable for widely distributed systems such as cloud computing.
  3. **Mixed approach :** A combination between the two approaches to take advantage of each approach.

Metrics for load balancing in clouds :

- Various metrics considered in existing load balancing techniques in cloud computing are discussed below :
  1. Throughput is used to calculate the no. of tasks whose execution has been completed. It should be high to improve the performance of the system.
  2. Overhead associated determines the amount of overhead involved while implementing a load-balancing algorithm. It is composed of overhead due to movement of tasks, interprocessor and inter-process communication. This should be minimized so that a load balancing technique can work efficiently.
  3. Fault tolerance is the ability of an algorithm to perform uniform load balancing in spite of arbitrary node or link failure. The load balancing should be a good fault-tolerant technique.
  4. Migration time is the time to migrate the jobs or resources from one node to other. It should be minimized in order to enhance the performance of the system.

5. Response time is the amount of time taken to respond by a particular load balancing algorithm in a distributed system. This parameter should be minimized.
6. Resource utilization is used to check the utilization of resources. It should be optimized for an efficient load balancing.
7. Scalability is the ability of an algorithm to perform load balancing for a system with any finite number of nodes. This metric should be improved.
8. Performance is used to check the efficiency of the system. This has to be improved at a reasonable cost, e.g., reduce task response time while keeping acceptable delays

- In cloud computing, load balancing is required to distribute the dynamic local workload evenly across all the nodes. It helps to achieve a high user satisfaction and resource utilization ratio by ensuring an efficient and fair allocation of every computing resource.

5.8 Scalability and Elasticity

- Scalability is the ability of a system or network to handle increased load or usage. At the same time, elasticity is the ability to automatically expand and contract resources to meet demand.
- Cloud elasticity is a system's ability to manage available resources according to the current workload requirements dynamically. This is a vital feature of a system infrastructure. It comes in handy when the system is expected to experience sudden spikes of user activity and, as a result, a drastic increase in workload demand.
- System scalability is the system's infrastructure to scale for handling growing workload requirements while retaining a consistent performance adequately. Unlike elasticity, which is more of makeshift resource allocation - cloud scalability is a part of infrastructure design.
- Scalability is one of the prominent features of cloud computing. In the past, a system's scalability relied on the company's hardware, and thus, was severely limited in resources. With the adoption of cloud computing, scalability has become much more available and more effective.
- Scalability can either be vertical (scale-up with in a system) or horizontal (scale-out multiple systems-sometimes linearly). This means applications have the room to scale up or scale out to prevent a lack of resources from hindering performance.

5.9 Replication

- Cloud replication refers to the process of replicating data from on-premises storage to the cloud, or from one cloud instance to another. Traditional data replication involves replicating data across different physical servers on the company's local network.
- Replication is used to create and maintain multiple copies of the data in the cloud. Cloud enables rapid implementation of replication solutions for disaster recovery of organizations.
- With cloud-based data replication organizations can plan for disaster recovery without making any capital expenditures on purchasing, configuring and managing secondary site locations.
- Types of replications are **array-based , network-based** and **host-based replication.**
- An array-based data replication strategy uses built-in software to automatically replicate data. With this type of data replication, the software is used in compatible storage arrays to copy data between each storage array.
- Host-based data replication uses the servers to copy data from one site to another site. Host-based replication software usually includes options like compression, encryption and, throttling, as well as failover.
- Network-based data replication uses a device or appliance that sits on the network in the path of the data to manage replication. The data is then copied to a second device. These devices usually have proprietary replication technology but can be used with any host server and storage hardware.

5.10 Monitoring

- Monitoring services allow cloud users to collect and analyze the data on various monitoring metrics. A monitoring service collects data on various system and application metrics from the cloud computing instances.
- Monitoring of cloud resources is important because it allows the users to keep track of the health of applications and services deployed in the cloud.
- Examples of monitoring metrics are as follows :

Device types	Metric
CPU	CPU-Usage, CPU-Idle
Memory	Memory-used, Memory-free, Page-cache
Disk	Disk-usage, Bytes/sec (read/write), Operations/sec
Interface	Packets/sec (incoming/outgoing), Octets/sec (incoming/outgoing)

5.11 Cloud Services and Platforms : Compute Services

- Compute services contains the fundamental element of cloud computing systems. Example of compute service is Amazon EC2.
- Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers and system administrators.
- Amazon EC2 reduces the time required to obtain and boot new server instances (called Amazon EC2 instances) to minutes, allowing user to quickly scale capacity, both up and down, as your computing requirements change.
- EC2 allows creating Virtual Machines (VM) on-demand. Pre-configured template Amazon Machine Image (AMI) can be used get running immediately. Creating and sharing your own AMI is also possible via the AWS Marketplace.
- Features :
  a. Scalable : Auto scaling policies can be defined for compute services that are triggered when the monitor metrics go above pre-defined thresholds.
  b. Secure : It provides various security features.
  c. Cost effective.
  d. Flexible.

1408

5.11.1 Amazon Elastic Compute Cloud

- Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers and system administrators.
- The Amazon EC2 simple web service interface allows you to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon's proven computing environment.
- Amazon EC2 reduces the time required to obtain and boot new server instances (called Amazon EC2 instances) to minutes, allowing you to quickly scale capacity, both up and down, as your computing requirements change.
- Amazon EC2 changes the economics of computing by allowing you to pay only for capacity that you actually use. Amazon EC2 provides developers and system administrators the tools to build failure resilient applications and isolate themselves from common failure scenarios.

- EC2 allows creating Virtual Machines (VM) on-demand. Pre-configured template Amazon Machine Image (AMI) can be used get running immediately. Creating and sharing your own AMI is also possible via the AWS marketplace.
- Amazon Machine Image (AMI) is a template for software configuration (Operating System, Application Server, and Applications). Fig. 5.11.1 shows AMI and instance.
- Instance is a AMI running on virtual servers in the cloud. Each instance type offers different compute and memory facilities. Create an Amazon Machine Image (AMI) containing your applications, libraries, data and associated configuration settings. Or use pre-configured, templated images to get up and running immediately.

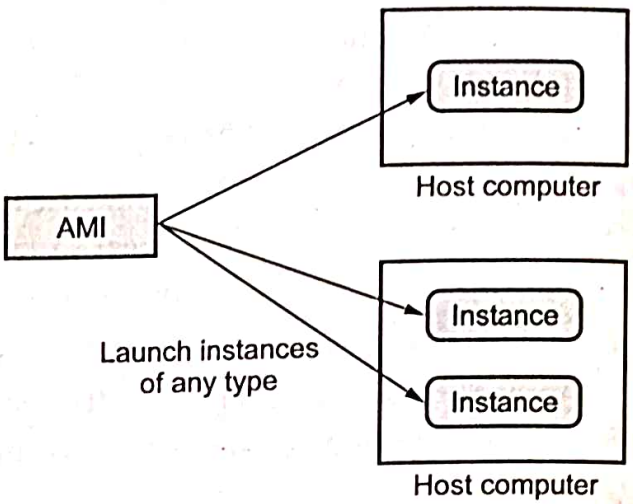


Fig. 5.11.1 AMI and instance

- Auto scaling allows automatically scale of the capacity up seamlessly during demand spikes to maintain performance and scales down during demand lulls to minimize costs.
- Elastic load balancing automatically distributes incoming application traffic across multiple Amazon EC2 instances. It provide tools to build failure resilient applications by launching application instances in separate availability zones.
- Pay only for resources actually consume, instance-hours. VM Import/Export enables you to easily import virtual machine images from your existing environment to Amazon EC2 instances and export them back at any time.
- Boto is a Python package that provides programmatic connectivity to Amazon Web Services.

launching an EC2 instance :

```
#/usr/bin/python
import boto.ec2
conn = boto.ec2.connect_to_region("us-west-2")
conn.run_instances(
   'ami-6ac2a85a',
   key_name='nitheesh_oregon',
   instance_type='t1.micro',
   security_groups=['nitheesh_oregon']
)
```

Stop instances :

```
#/usr/bin/python
import boto.ec2
conn = boto.ec2.connect_to_region("us-west-2")
conn.stop_instances(instance_ids=['instance-id-1',
'instance-id-2'])
```

- Boto supports more than fifty Amazon services, running the whole range from compute, database, application and payments and billing.
- EC2 functions :
  1. Load variety of operating system.
  2. Install custom applications.
  3. Manage network access permission.
  4. Run image using as many/few systems as you desire.
- EC2 advantages :
  1. Amazon EC2 enables you to increase or decrease capacity within minutes.
  2. User have complete control of your Amazon EC2 instances.
  3. Support flexible cloud hosting services
  4. Secure : Amazon EC2 works in conjunction with Amazon VPC to provide security and robust networking functionality.
  5. Reliable : Amazon EC2 offers a highly reliable environment where replacement instances can be rapidly and predictably commissioned.

5.11.2 Windows Azure

- Windows Azure is a cloud computing platform and infrastructure, created by Microsoft, for building, deploying and managing applications and services through a global network of Microsoft - managed data centers.
- Azure queue storage is a service for storing large numbers of messages that can be accessed from anywhere in the world via authenticated calls using HTTP or HTTPS. A single queue message can be up to 64 KB in size, and a queue can contain millions of messages, up to the total capacity limit of a storage account.
- Azure is a virtualized infrastructure to which a set of additional enterprise services has been layered on top, including, a virtualization service called Azure AppFabric that creates an application hosting environment. AppFabric is a cloud-enabled version of the .NET framework.
- Windows Azure is Microsoft's application platform for the public Cloud. Applications can be deployed on to Azure in various models.

- Windows Azure is used to :
  1. Build a web application that runs and stores its data in Microsoft data centers.
  2. Store data while the applications that consume this data run on premise (outside the public Cloud).
  3. Create virtual machines to develop and test, or run SharePoint and other out-of-the-box applications.
  4. Develop massively scalable applications with many users.
  5. Offer a wide range of services.
- Azure has three components : compute, storage and fabric
  1. **Compute :** Windows Azure provides a hosting environment for managed code. It provides a computation service through roles. Windows Azure supports three types of roles :
     a) Web roles used for web application programming and supported by IIS7.
     b) Worker roles are also used for background processing of web roles.
     c) Virtual Machine (VM) roles are generally used for migrating windows server applications to Windows Azure in an easy way.
  2. **Storage :** Windows Azure provides storage in the cloud. It provides four different types of storage services :
     a) Queues for messaging between web roles and worker roles.
     b) Tables for storing structural data.
     c) BLOBs (Binary Large Objects) to store text, files or large data.
     d) Windows Azure Drives (VHD) to mount a page blob. They can easily be downloaded and uploaded via blobs.
  3. AppFabric provides infrastructure services for developing, deploying and managing Windows Azure application. It provides five services: Service bus, Access, Caching, Integration and Composite.
- Fig. 5.11.2 shows Windows Azure platform architecture.
- Microsoft Azure is a cloud computing service created by Microsoft for building, testing, deploying and managing applications and services through a global network of Microsoft-managed data centers.
- It provides software as a service (SaaS), platform as a service and infrastructure as a service and supports many different programming languages, tools and frameworks, including both Microsoft-specific and third-party software and systems.

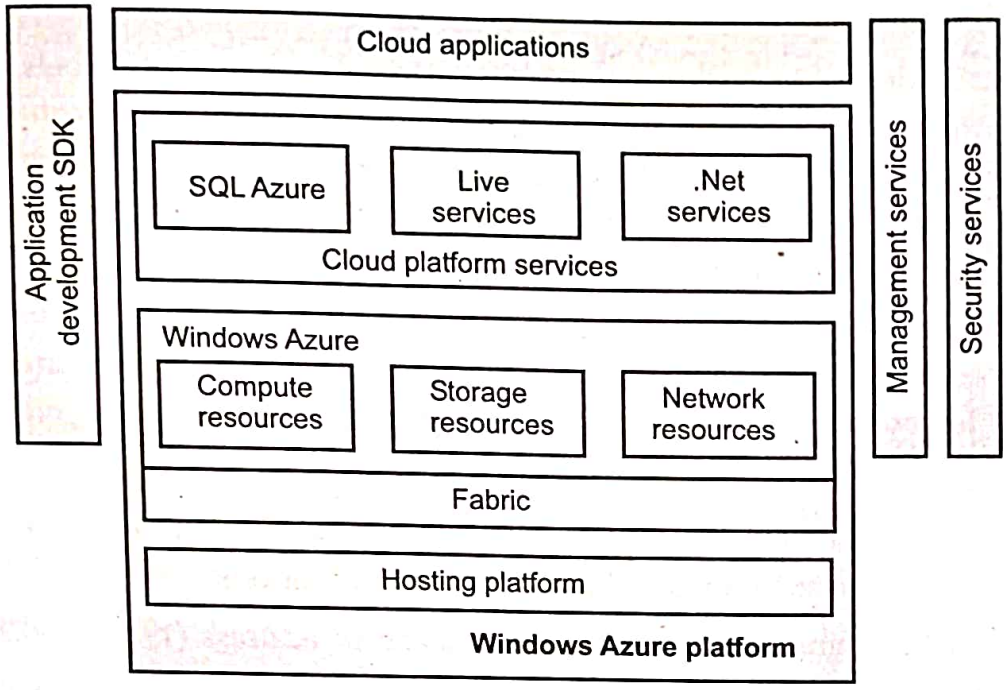


Fig. 5.11.2 Windows Azure platform architecture

- Windows Azure provides resources and services for consumers. For example, hardware is abstracted and exposed as compute resources.
- Physical storage is abstracted as storage resources and exposed through very well-defined interfaces.
- A common windows fabric abstracts the hardware and the software and exposes virtual compute and storage resources.
- Each instance of an application is automatically managed and monitored for availability and scalability.
- If an application goes down, the Fabric is notified and a new instance of the application is created. Because virtualization is a key element in cloud computing, no assumption must be made on the state of the underlying hardware hosting the application.
- Advantages of Microsoft Azure
  1. Microsoft Azure offers high availability.
  2. It offers you a strong security profile.
  3. It is a cost-effective solution for an IT budget.
  4. Azure allows you to use any framework, language, or tool.
  5. Azure allows businesses to build a hybrid infrastructure.

5.12 Storage Service

- Amazon S3 defines a bucket name as a series of one or more labels, separated by periods, that adhere to the following rules : The bucket name can be between 3 and 63 characters long, and can contain only lower-case characters, numbers, periods, and dashes
- Amazon S3 defines a bucket name as a series of one or more labels, separated by periods, that adhere to the following rules :
  1. The bucket name can be between 3 and 63 characters long, and can contain only lower-case characters, numbers, periods, and dashes.
  2. Each label in the bucket name must start with a lowercase letter or number.
  3. The bucket name cannot contain underscores, end with a dash, have consecutive periods, or use dashes adjacent to periods.
  4. The bucket name cannot be formatted as an IP address (198.51.100.24).
- A bucket is owned by the AWS account that created it. By default, you can create up to 100 buckets in each of your AWS accounts. If you need additional buckets, you can increase your bucket limit by submitting a service limit increase.
- The following are the rules for naming S3 buckets in all AWS Regions :
  1. Bucket names must be unique across all existing bucket names in Amazon S3.
  2. Bucket names must comply with DNS naming conventions.
  3. Bucket names must be at least 3 and no more than 63 characters long.
  4. Bucket names must not contain uppercase characters or underscores.
  5. Bucket names must start with a lowercase letter or number.
  6. Bucket names must be a series of one or more labels. Adjacent labels are separated by a single period (.). Bucket names can contain lowercase letters, numbers, and hyphens. Each label must start and end with a lowercase letter or a number.
  7. Bucket names must not be formatted as an IP address (for example, 192.168.5.4).
  8. When you use virtual hosted-style buckets with Secure Sockets Layer (SSL), the SSL wildcard certificate only matches buckets that don't contain periods. To work around this, use HTTP or write your own certificate verification logic. We recommend that you do not use periods (".") in bucket names when using virtual hosted-style buckets.

5.12.1 Google Cloud Storage

- Google cloud storage allows world-wide storage and retrieval of any amount of data at any time. It can be used for a range of scenarios including serving website content, storing data for archival and disaster recovery, or distributing large data objects to users via direct download.
- Cloud storage is a service for storing your objects in Google Cloud. An object is an immutable piece of data consisting of a file of any format. We store objects in containers called buckets. All buckets are associated with a project, and we can group the projects under an organization.
- Access control list are used to control access to objects and buckets. There are 4 ways to interact with the Google cloud storage service :
  a. **Cloud console :** It provides a visual interface to manage the data in a browser.
  b. **Client libraries :** The client libraries allow users to manage their data using one of their preferred languages, which includes C++, C#, Go, Java, Node.js, PHP, Python, and Ruby.
  c. **gsutil :** It is a command-line tool that allows users to interact with Cloud Storage through a terminal.
  d. **REST APIs :** It manages the data using the JSON or XML API.
- Google cloud storage offers 4 types of storage classes for any workloads which can be used as per the requirement :
  a. **Standard storage :** It is appropriate for "hot" data that is accessed frequently, including websites, streaming videos, and mobile apps.
  b. **Nearline storage :** It is a low-cost option suited for data that can be stored for at least 30 days, including data backup and long-tail multimedia content.
  c. **Coldline storage :** Very low cost suited for data that can be stored for at least 90 days, including disaster recovery.
  d. **Archive storage :** It offers the lowest cost and suited for data that can be stored for at least 365 days, including regulatory archives.

5.13 Application Services

- Here we discuss various cloud application services like application runtime and framework, queuing service, email services and media services.

5.13.1 Application Framework and Runtime : Google App Engine

- Google App Engine (GAE) is a Platform as a Service cloud computing platform for developing and hosting web applications in Google-managed data centers.

- Google App Engine is a way to write your own web applications and have them hosted on Google servers. It enables developers to build their web applications on the same scalable system that power Google applications.
- An app is a piece of software which can run on the computer, internet, phone or any other electronic device. Google refers to their online services as Apps. They also sell a specific suite of services known as Google Apps.
- Google's providing both SaaS and PaaS solutions in cloud computing. Some of the examples for SaaS solutions including Google Apps which including Gmail, Doc, etc. and PaaS includes Google App engine.
- Services provided by App engine includes :
  a) Platform as a Service (PaaS) to build and deploy scalable applications
  b) Hosting facility in fully-managed data centers
  c) A fully-managed, flexible environment platform for managing application server and infrastructure.
  d) Support in the form of popular development languages and developer tools.
- **Major feature of Google App Engine :**
  1. Automatic scaling and load balancing.
  2. Authentication using Google Accounts API.
  3. Provides dynamic web services based on common standards.
  4. Integration with other Google Cloud Services and API.
  5. Support persistent storage, with query access sorting and transaction management features.
- Google App engine offers users the ability to build and host web applications on Google's infrastructure.

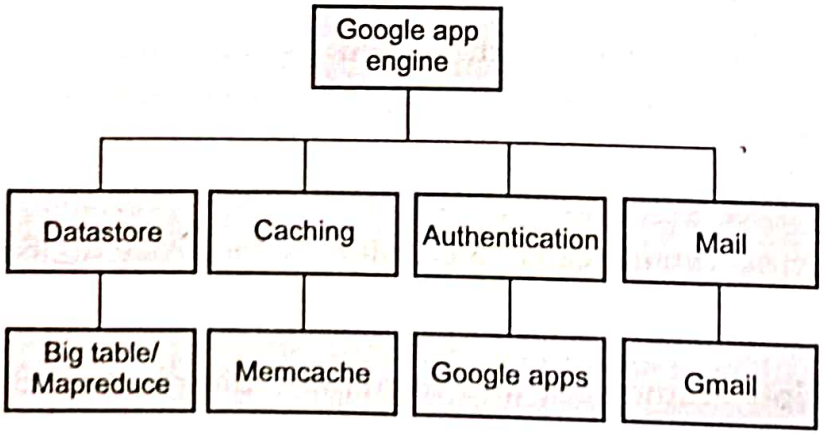


Fig. 5.13.1

- The App Engine offers a number of services that enable you to perform several common operations when managing your application. The following APIs are available to access these services :
  1. **Mail :** Using the mail API, the developers can send email messages.
  2. **Memcache :** The Memcache service gives the users the benefit of working efficiently by providing high retrieval speed, even when multiple users access the same application at the same instance of time.
  3. **Image manipulation :** The Image service allows you to manipulate images of your application. With the use of this API, you can resize, crop, rotate and flip images in JPEG and PNG formats.
- In the PaaS space Google is a key player. App Engine is a platform to create, store and run applications on Google's servers using development languages as java and python.
- App Engine includes tools for managing the data store, monitoring the site and its resource consumption and debugging and logging. A user can serve the app from his own domain name using Google Apps.

Key features of GAE programming mode using java and python

- The Google App engine Software Development Kit (SDK) provides Java and Python programming languages.
- The languages have their own web server application that contains all Google App Engine services on a local computer. The web server also simulates a secure sandbox environment.
- The Google App engine SDK has APIs and libraries including the tools to upload applications. The architecture defines the structure of applications that run on the Google App engine.

1. Python :

- The Google App engine allows implementation of applications using python programming language and running them on its interpreter.
- The Google App engine provides rich APIs and tools for designing web applications, data modeling, managing, accessing apps data, support for mature libraries and frameworks like Django.
- The main characteristics of Google App engine are its DataStore, configuration file app.yaml and how it serves an application.

2. Java :

- The Google App engine provides tools and APIs required for the development of web applications that run on the Google App engine Java run time.

- The application interacts with the environment using servlets and web technologies like Java Server Pages (JSPs) which can be developed using Java6.
- The GAE environment uses Java SE Runtime JRE platform 6 and libraries which the applications can access using APIs.
- Java SDK has implementations for Java Data Objects (JDO) and Java Persistence (JPA) interface.
- To exchange email messages with Google App engine, it provides the Google App Engine mail service through the Java Mail API.
- Support for other languages like JavaScript, Ruby or Scalar is also provided by Google App engine with the use of JVM compatible compilers and interpreters.
- When Google App engine gets a web request that corresponds to the URL mentioned in the applications deployment descriptor it invokes a servlet corresponding to the request and uses Java Servlets API to provide requested data and accepts response data.
- Google App engine makes it easy to build an applications that runs reliably, even under heavy load and with large amounts of data.
- App engine includes the below features :
  a) Dynamic web serving, with full support for common web technologies.
  b) Persistent storage with queries, sorting and transactions.
  c) Automatic scaling and load balancing.
  d) APIs for authenticating users and sending email using Google accounts.
  e) Scheduled tasks for triggering events at specified times and regular intervals.

5.13.2 Queuing Service : Amazon Simple Queue Service

- Amazon Simple Queue Service (SQS) is a fully managed message queuing service that enables user to decouple and scale microservices, distributed systems, and serverless applications. Amazon SQS is a message-queuing system that allows user to write distributed applications by exposing a message pipeline that can be processed in the background by workers.
- Using SQS, user can send, store, and receive messages between software components at any volume, without losing messages or requiring other services to be available. SQS offers two types of message queues. Standard queues offer maximum throughput, best-effort ordering, and at-least-once delivery.
- SQS FIFO queues are designed to guarantee that messages are processed exactly once, in the exact order that they are sent.

5.14 Two Marks Questions with Answers

Q.1 Explain NIST definition of cloud computing.

Ans. : NIST definition of cloud : Cloud computing is a pay-per-use model for enabling available, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, services) that can be rapidly provisioned and released with minimal management effort or service-provider interaction.

Q.2 What is cloud service ?

Ans. : Cloud service is any service made available to users on demand via the Internet from a cloud computing provider's servers as opposed to being provided from a company's own on-premises servers.

Q.3 What is public cloud ?

Ans. : Public cloud is built over the Internet and can be accessed by any user who has paid for the service. Public clouds are owned by service providers and are accessible through a subscription.

Q.4 What is private clouds ?

Ans. : A private cloud is built within the domain of an intranet owned by a single organization. Therefore, it is client owned and managed, and its access is limited to the owning clients and their partners.

Q.5 Explain about virtual machines.

Ans. : A Virtual Machine (VM) is a software construct that mimics the characteristics of a physical server. VM is a software program or operating system that not only exhibits the behavior of a separate computer, but is also capable of performing tasks such as running applications and programs like a separate computer.

Q.6 What is NIST definition of IaaS ?

Ans. : The ability given to the infrastructure architects to deploy or run any software on the computing resources provided by the service provider. The end users are responsible for managing applications that are running on top of the service provider cloud infrastructure.

Q.7 Explain characteristics of IaaS.

Ans. : Characteristics of IaaS

1. Resources are provided as a service.
2. Allows for dynamic scaling and elasticity.
3. It has a variable cost, usage based pricing model (pay per go and pay per use).
4. It has multi-tenet architecture, includes multiple users on a single piece of hardware.
5. IaaS typically has enterprise grade infrastructure.

Q.8 List the situations where PaaS may not be the best option.

Ans. :

- Integration with on-premise application
- Flexibility at the platform level
- Customization at the infrastructure level
- Frequent application migration

Q.9 What is Amazon EC2 ?

Ans. : Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers and system administrators.

Q.10 List the function of EC2.

Ans. : EC2 functions :

1. Load variety of operating system.
2. Install custom applications.
3. Manage network access permission.
4. Run image using as many/few systems as we desire.

Q.11 What is Azure ?

Ans. : Windows Azure is a cloud computing platform and infrastructure, created by Microsoft, for building, deploying and managing applications and services through a global network of Microsoft - managed data centers.

Q.12 What is Azure queues ?

Ans. : Azure queue storage is a service for storing large numbers of messages that can be accessed from anywhere in the world via authenticated calls using HTTP or HTTPS. A single queue message can be up to 64 KB in size, and a queue can contain millions of messages, up to the total capacity limit of a storage account.

Q.13 How virtualization employed in Azure ?

Ans. : Azure is a virtualized infrastructure to which a set of additional enterprise services has been layered on top, including, a virtualization service called Azure AppFabric that creates an application hosting environment. AppFabric is a cloud-enabled version of the .NET framework.

Q.14 What is service cloud ?

Ans. : Service cloud refers to the service module in Salesforce.com. It includes accounts, contacts, cases, and solutions. It also encompasses features such as the public knowledge base, web-to-case, call center, and self - service portal, as well as customer service automation.

SOLVED MODEL QUESTION PAPER

[As per New Syllabus]

Distributed Computing

Semester - V (CSE / IT / AI&DS)

Time : Three Hours] [Maximum Marks : 100

Answer ALL Questions

PART A - (10 × 2 = 20 Marks)

Q.1 *What is the role of middleware in a distributed system ?* **(Refer Two Marks Q.22 of Chapter - 1)**

Q.2 *What is distributed system ?* **(Refer Two Marks Q.6 of Chapter - 1)**

Q.3 *Write the happens-before relation ?* **(Refer Two Marks Q.18 of Chapter - 2)**

Q.4 *What is clock skew ?* **(Refer Two Marks Q.11 of Chapter - 2)**

Q.5 *What are the requirements of mutual exclusion algorithms ?* **(Refer Two Marks Q.7 of Chapter - 3)**

Q.6 *What do you mean by deadlock avoidance ?* **(Refer Two Marks Q.16 of Chapter - 3)**

Q.7 *Mention some motivations for replication.* **(Refer Two Marks Q.26 of Chapter - 4)**

Q.8 *Define recovery.* **(Refer Two Marks Q.6 of Chapter - 4)**

Q.9 *What is Azure ?* **(Refer Two Marks Q.11 of Chapter - 5)**

Q.10 *What is cloud service ?* **(Refer Two Marks Q.2 of Chapter - 5)**

PART B - (5 × 13 = 65 Marks)

Q.11 a) *i) Explain difference between message passing and shared memory.* **(Refer section 1.4)** **[6]**

ii) Discuss primitives for distributed communication. **(Refer section 1.5)** **[7]**

OR

b) *i) What is distributed system ? What is need of distribbuted system ?* **(Refer sections 1.1 and 1.3)** **[6]**

ii) Explain global state of distributed system. **(Refer section 1.11)** **[7]**

Q.12 a) *i) Explain happen-before relation with example.* **(Refer section 2.2.1)** **[7]**

ii) Discuss Berkeley algorithm. **(Refer section 2.3.3)** **[6]**

OR

b) *i) What is scalar time ? Explain properties of scalar time.* **(Refer section 2.5)** [7]

ii) Explain the types of group communications used in distributed system. **(Refer section 2.10)** [6]

Q.13 a) *Explain Ricart Agrawala algorithm with an example.* **(Refer section 3.4)** [13]

OR

b) *i) Explain requirement and performance metric of mutual exclusion.* **(Refer sections 3.2.2 and 3.2.3)** [6]

ii) Discuss AND model and OR model of deadlock. **(Refer sections 3.9.2 and 3.9.3)** [7]

Q.14 a) *i) Explain strongly consistent set of checkpoints.* **(Refer section 4.9)** [6]

ii) Discuss coordinated checkpointing algorithm. **(Refer section 4.12)** [7]

OR

b) *i) What is local checkpoint ? Explain.* **(Refer section 4.8.2)** [6]

ii) Discuss Byzantine agreement problem. **(Refer section 4.2)** [7]

Q.15 a) *i) What is cloud computing ? Explain advantages and disadvantages of cloud computing.* **(Refer section 5.1)** [6]

ii) Define virtualization. Explain full-virtualization and para-virtualization. Discuss difference between full-virtualization and para-virtualization. **(Refer section 5.6)** [7]

OR

b) *i) What is queuing service ? Discuss Amazon simple queue service.* **(Refer section 5.13.2)** [6]

ii) Explain any two cloud service models. **(Refer section 5.4)** [7]

PART C - (1 × 15 = 15 Marks)

Q.16 a) *Show that in the Ricart-Agrawala algorithm the critical section is accessed in increasing order of timestamp.* **(Refer section 3.4)** [15]

OR

b) *Discuss the design issues and challenges in distributed system from a system perspective.* **(Refer section 1.7)** [15]

❑❑❑