MAILAM Engineering College

Approved by AICTE, New Delhi, affiliated to Anna University, Chennai, Accredited by NBA & TCS

CS3551 DISTRIBUTED COMPUTING

[REGULATION-2021]

STUDY MATERIAL

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

NAME OF THE STUDENT:……

REGISTER NUMBER:……

YEAR / SEM:……

ACADEMIC YEAR:……

PREPARED BY

MAILAM Engineering College

(Approved by AICTE, New Delhi, Affiliated to Anna University Chennai, Accredited by NBA, TCS & NAAC - 'A' Grade)

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

CS3551 DISTRIBUTED COMPUTING

III Yr. / V SEM

SYLLABUS

COURSE OBJECTIVES:

- To introduce the computation and communication models of distributed systems
- To illustrate the issues of synchronization and collection of information in distributed systems
- To describe distributed mutual exclusion and distributed deadlock detection techniques
- To elucidate agreement protocols and fault tolerance mechanisms in distributed systems
- To explain the cloud computing models and the underlying concepts

UNIT I INTRODUCTION **08**

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.

UNIT II LOGICAL TIME AND GLOBAL STATE **10**

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.

UNIT III DISTRIBUTED MUTEX AND DEADLOCK **10**

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.

UNIT IV CONSENSUS AND RECOVERY **10**

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

UNIT V CLOUD COMPUTING **07**

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

TOTAL: 45 PERIODS

COURSE OUTCOMES:

Upon the completion of this course, the student will be able to

CO1: Explain the foundations of distributed systems (K2)

CO2: Solve synchronization and state consistency problems (K3)

CO3: Use resource sharing techniques in distributed systems (K3)

CO4: Apply working model of consensus and reliability of distributed systems(K3)

CO5: Explain the fundamentals of cloud computing (K2)

TEXT BOOKS

1. Kshemkalyani Ajay D, Mukesh Singhal, "Distributed Computing: Principles, Algorithms and Systems", Cambridge Press, 2011.
2. Mukesh Singhal, Niranjan G Shivaratri, "Advanced Concepts in Operating systems", Mc-Graw Hill Publishers, 1994.

REFERENCES

1. George Coulouris, Jean Dollimore, Time Kindberg, "Distributed Systems Concepts and Design", Fifth Edition, Pearson Education, 2012.
2. Pradeep L Sinha, "Distributed Operating Systems: Concepts and Design", Prentice Hall of India, 2007.

3. Tanenbaum A S, Van Steen M, "Distributed Systems: Principles and Paradigms", Pearson Education, 2007.
4. Liu M L, "Distributed Computing: Principles and Applications", Pearson Education, 2004.
5. Nancy A Lynch, "Distributed Algorithms", Morgan Kaufman Publishers, 2003.
6. Arshdeep Bagga, Vijay Madisetti, " Cloud Computing: A Hands-On Approach", Unive

Prepared by –

BC	**HOD**	**PRINCIPAL**
Ms.M.NITHYA,	Dr. S. Artheeswari,	
AP/AI&DS	AP/AI&DS	Prof. & Head

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.

PART A

1. What is meant distributed system?

Distributed System is a collection of autonomous computer systems that are physically separated but are connected by a centralized computer network that is equipped with distributed system software. The autonomous computers will communicate among each system by sharing resources and files and performing the tasks assigned to them.

2. What are the significance of distributed system?

- ✓ Concurrency of computers.
- ✓ No global clock.
- ✓ Independent failures.

3. Why do we need distributed system?

- ✓ Functional distribution
- ✓ Load distribution/balancing:
- ✓ Replication of processing power:
- ✓ Physical separation:
- ✓ Economics:

4. List the distributed systems challenges.

1. Heterogeneity: standards and protocols; middleware; virtual machine;
2. Openness: publication of services; notification of interfaces;
3. Security: firewalls; encryption; d. Scalability: replication; caching; multiple servers;
4. Failure Handling. failure tolerance; recover/roll-back; redundancy;
5. Concurrency control to ensure data consistency.
6. Transparency. Middleware; location transparent naming; anonymity

5. How can we characterize a distributed system?

A distributed system can be characterized as a collection of mostly autonomous processors communicating over a communication network and having the following features:

1. No common physical clock
2. No shared memory

3. Geographical separation
4. Autonomy and heterogeneity

6. What are the several standards used in distributed system?

There are several standards such as

- ✓ Object Management Group's (OMG)
- ✓ Common object request broker architecture (CORBA)
- ✓ Remote procedure call (RPC) mechanism

7. List some of the requirements of distributed system.

Distributed system have some of these following requirements

1. Modularity and incremental expandability
2. Scalability
3. Increased performance/cost ratio
4. Enhanced reliability
5. Access to geographically remote data and resources
6. Resource sharing
7. Inherently distributed computations

8. What is a shared memory systems?

DSM is a mechanism of allowing user processes to access shared data without using inter-process communications. In DSM every node has its own memory and provides memory read and write services and it provides consistency protocols. The distributed shared memory (DSM) implements the shared memory model in distributed systems but it doesn't have physical shared memory.

9. What are the two ways of sending data when the *Send* primitive is invoked ?

There are two ways of sending data when the *Send* primitive is invoked –the **buffered option** and the **unbuffered option**.

- ✓ The ***buffered option*** which is the standard option copies the data from the user buffer to the kernel buffer. The data later gets copied from the kernel buffer onto the network.
- ✓ The ***unbuffered option***, the data gets copied directly from the user buffer onto the network.

10. Definitions of Synchronous primitives.

A *Send* or a *Receive* primitive is *synchronous* if both the *Send()* and *Receive()* handshake with each other. The processing for the *Receive* primitive completes when thedata to be received is copied into the receiver's user buffer.

11. Definitions of Asynchronous primitives.

A *Send* primitive is said to be *asynchronous* if control returns back to the invoking process after the data item to be sent has been copied out of the user-specified buffer. It does not make sense to define asynchronous *Receive* primitives.

12. Definitions of Blocking primitives .

A primitive is *blocking* if control returns to the invoking process after the processing for the primitive (whether in synchronous or asynchronous mode) completes.

13. Definitions of Non-blocking primitives .

A primitive is *non-blocking* if control returns back to the invoking process immediately after invocation, even though the operation has not completed. For a non-blocking *Send*, control returns to the process even before the data is copied out of the user buffer.

14. What is asynchronous executions?

An *asynchronous execution* is an execution in which

- ✓ There is no processor synchrony and there is no bound on the drift rate of processor clocks,
- ✓ Message delays (transmission + propagation times) are finite but unbounded
- ✓ There is no upper bound on the time taken by a process to execute a step.

15.* What is *synchronous execution?

A *synchronous execution* is an execution in which

(i) processors are synchronized and the clock drift rate between any two processors is bounded,

(ii) message delivery (transmission + delivery) times are such that they occur in one logical step or round, and

(iii) there is a known upper bound on the time taken by a process to execute a step. An example of a synchronous execution with four processes P0 to P3

16. What are the Distributed systems challenges from a system perspective?

The following functions must be addressed when designing and building a distributed system

- Communication
- Processes
- Naming
- Synchronization
- Data storage and access
- Consistency and replication
- Fault tolerance

17. Define Transparency and list its types.

Transparency deals with hiding the implementation policies from the user, and can be classified as follows.

- ✓ *Access transparency*.
- ✓ *Location transparency*.
- ✓ *Migration transparency*.
- ✓ *Relocation transparency*
- ✓ *Replication transparency*.
- ✓ *Concurrency transparency*
- ✓ *Failure transparency*:

18. What is a Leader election in distinguished process?

All the processes need to agree on which process will play the role of a distinguished process – called a leader process. A leader is necessary even for many distributed algorithms because there is often some asymmetry – as in initiating some action like a broadcast or collecting the state of the system, or in "regenerating" a token that gets "lost" in the system.

19. Define Mutual exclusion.

This is clearly a synchronization problem because access to the critical resource(s) has to be coordinated.

20. Define Deadlock detection and resolution.

Deadlock detection should be coordinated to avoid duplicate work, and deadlock resolution should be coordinated to avoid unnecessary aborts of processes.

21. Define Termination detection.

This requires cooperation among the processes to detect the specific global state of quiescence.

22. What is Garbage collection?

Garbage refers to objects that are no longer in use and that are not pointed to by any other process. Detecting garbage requires coordination among the processes.

23. Define Reliable and fault-tolerant distributed system and its strategies.

A reliable and fault-tolerant environment has multiple requirements and aspects, and these can be addressed using various strategies:

- ✓ **Consensus algorithms**
- ✓ **Replication and replica management**

- ✓ **Voting and quorum systems**
- ✓ **Distributed databases and distributed commit**
- ✓ **Self-stabilizing systems**
- ✓ **Check pointing and recovery algorithms**

24. Define and forms of Load balancing.

The goal of load balancing is to gain higher throughput, and reduce the user perceived latency. Load balancing may be necessary because of a variety of factors such as high network traffic or high request rate causing the network connection to be a bottleneck, or high computational load.

The following are some forms of load balancing

- ✓ Data migration
- ✓ Computation migration
- ✓ Distributed scheduling

25. What is a Sensor networks?

A sensor is a processor with an electro-mechanical interface that is capable of sensing physical parameters, such as temperature, velocity, pressure, humidity, and chemicals. Recent developments in cost-effective hardware technology have made it possible to deploy very large (of the order of 106 or higher) low-cost sensors.

26. Define Ubiquitous or pervasive computing.

Ubiquitous systems represent a class of computing where the processors embedded in and seamlessly pervading through the environment perform application functions in the background, much like in sci-fi movies. The intelligent home, and the smart workplace are some example of ubiquitous environments currently under intense research and development.

27. What is Peer-to-peer computing?

Peer-to-peer (P2P) architecture is a distributed system in which each node acts as both a client and a server. This allows the nodes to share the workload or tasks among themselves. In a P2P system: Each node, or peer, has the same functional capability and holds part of the resources in the system.

28. What is distributed agents?

Agents are software processes or robots that can move around the system to do specific tasks for which they are specially programmed. The name “agent” derives from the fact that the agents do work on behalf of some broader objective. Agents collect and process information and can exchange such information with other agents

29. What is Distributed data mining?

Data mining algorithms examine large amounts of data to detect patterns and trends in the data, to *mine* or extract useful information. A traditional example is: examining the purchasing patterns of customers in order to profile the customers and enhance the efficacy of directed marketing schemes.

30. Define Grid computing.

Grid Computing can be defined as a network of computers working together to perform a task that would rather be difficult for a single machine. All machines on that network work under the same protocol to act as a virtual supercomputer.

31. What is Security in distributed systems?

The traditional challenges of security in a distributed setting include:

- ✓ confidentiality (ensuring that only authorized processes can access certain information)
- ✓ authentication (ensuring the source of received information and the identity of the sending process)
- ✓ Availability (maintaining allowed access to services despite malicious actions).

32. What are a model of distributed executions?

The execution of a process consists of a sequential execution of its actions. The actions are atomic and the actions of a process are modeled as three types of events, namely

- ✓ internal events,
- ✓ message send events,
- ✓ message receive events

33. What is the Causal precedence relation?

The execution of a distributed application results in a set of distributed events produced by the processes. Let $H = U_i h_i$ denote the set of events executed in a distributed computation. Next, we define a binary relation on the set H, denoted as $\rightarrow$, that expresses causal dependencies between events in the distributed execution

34. What are the Models of communication networks?

There are several models of the service provided by communication networks, namely, FIFO (first-in, first-out), 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.

35. What is causal ordering model?

- The "causal ordering model is based on Lamport's "happens before" relation.
- A system that supports the causal ordering model satisfies the following property:

  CO : For any two messages m_{ij} and m_{kj},

  if send(m_{ij}) $\longrightarrow$ send(m_{kj}) , then rec(m_{ij}) $\longrightarrow$ rec(m_{kj}).

36.State the difference from synchronous and asynchronous communication along with buffering strategy adapted in both. NOV/DEC 2023

Synchronous Communication:

- Real-time communication where sender and receiver are actively engaged.
- Sender waits for a response from the receiver before proceeding.
- Examples: Phone calls, video conferencing, instant messaging.

Buffering Strategy:

- No buffering or minimal buffering is required since data is processed immediately.

Asynchronous Communication:

- Communication where sender and receiver are not actively engaged at the same time.
- Sender sends data without waiting for a response, and receiver processes data at a later time.
- Examples: Email, text messaging, online forums.

Buffering Strategy:

- Buffering is essential to store data temporarily until the receiver is ready to process it.
- Buffering strategies like message queuing, data batching, and caching are used to handle data until it's processed.

37.List the role of shared memory systems. NOV/DEC 2023

1. Inter process Communication (IPC): Facilitating communication between processes.
2. Resource Sharing: Allowing multiple processes to access shared resources.
3. Data Sharing: Enabling processes to share data without copying or transferring it.
4. Synchronization: Coordinating access to shared resources to prevent conflicts.
5. Consistency: Ensuring data consistency across processes.
6. Efficient Data Transfer: Reducing data transfer overhead between processes.
7. Improved Performance: Enhancing system performance by reducing communication overhead.

38.Distinguish between message passing systems and shared memory systems. APR/MAY 2024

In a shared memory model, multiple workers all operate on the same data. This opens up a lot of the concurrency issues that are common in parallel programming. Message passing systems make workers communicate through a messaging system. Messages keep everyone separated, so that workers cannot modify each other's data.

39.What are the motivations for distributed systems? APR/MAY 2024

1. Scalability:
   Distributed systems can handle increased workload and demand by adding more nodes or machines, making them ideal for large-scale applications.
2. Fault tolerance:
   By replicating data and services across multiple nodes, distributed systems can continue functioning even if one or more nodes fail.
3. Improved performance:
   Distributed systems can process tasks concurrently across multiple nodes, reducing processing time and improving overall performance.
4. Resource sharing:
   Distributed systems enable sharing of resources, such as storage, memory, and processing power, across multiple nodes.

PART –B

1. **Explain how distributed system can be characterized.**
   - Autonomous processors communicating over a communication network
   - Some characteristics are
     - No common physical clock
     - No shared memory
     - Geographical separation
     - Autonomy and heterogeneity

Characteristics

- **No common physical clock**

  This is an important assumption because it introduces the element of "distribution "in the system and give rise to the inherent asynchrony amongst the processors.

- **No shared memory**

  This is a key feature that requires message-passing for communication. This feature implies the absence of the common physical clock.

  It may be noted that a distributed system may still provide the abstraction of a common address space via the distributed shared memory abstraction. Several aspects of shared memory multiprocessor systems have also been studied in the distributed computing literature.

- **Geographical separation**

  The geographically wider apart that the processors are, the more representative is the system of a distributed system. However, it is not necessary for the processors to be on a wide-area net- work (WAN). Recently, the network/cluster of workstations (NOW/COW) configuration connecting processors on a LAN is also being increasingly regarded as a small distributed system. This NOW configuration is becoming popular because of the low-cost high-speed off-the-shelf processors now available. The Google search engine is based on the NOW architecture.

- **Autonomy and heterogeneity**

  The processors are "loosely coupled" in that they have different speeds and each can be running a different operating system. They are usually not part of a dedicated system, but cooperate with one another by offering services or solving a problem jointly.

2.Explain a typical distributed system.

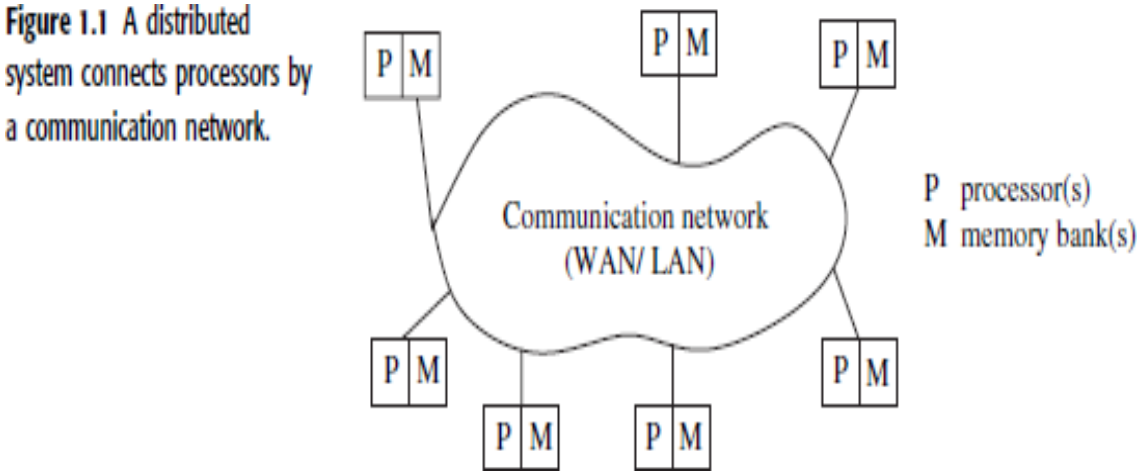


Figure 1.1 A distributed system connects processors by a communication network.

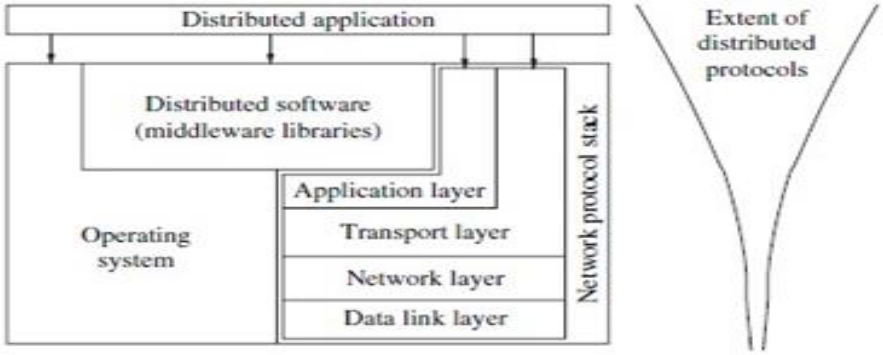


Fig 1.1 b Interaction of the software components of each processor

● The distributed software is also termed as middleware.

● A distributed execution is the execution of processes across the distributed system to collaboratively achieve a common goal. An execution is also sometimes termed a computation or a run in above Fig 1.1a &1.1 b

● The middleware is the distributed software that drives the distributed system ,while providing transparency of heterogeneity at the platform level.

● Middleware layer does not contain the traditional application layer functions of the network protocol stack, such as http, mail, ftp, and telnet.

● Various primitives and calls to functions defined in various libraries of the middleware layer are embedded in the user program code.

- There exist several libraries to choose from to invoke primitives for the more common functions such as reliable and ordered multicasting of the middleware layer.
- There are several standards such as Object Management Group's (OMG) common object request broker architecture (CORBA) and the remote procedure call (RPC) mechanism.

3. Explain the requirements for using a distributed system.

1. Inherently distributed computations

a. In many applications such as money transfer in banking, or reaching consensus among parties that are geographically distant, the computation is inherently distributed.

2. Resource sharing

a. Resources such as peripherals, complete data sets in databases, special libraries, as well as data (variable/files) cannot be fully replicated at all the sites because it is often neither practical nor cost-effective.

b. Further, they cannot be placed at a single site because access to that site might prove to be a bottleneck.

c. Therefore, such resources are typically distributed across the system. For example, distributed databases such as DB2 partition the data sets across several servers, in addition to replicating them at a few sites for rapid access as well as reliability.

3. Access to geographically remote data and resources

a. In many scenarios, the data cannot be replicated at every site participating in the distributed execution because it may be too large or too sensitive to be replicated. For example, payroll data within a multinational corporation is both too large and too sensitive to be replicated at every branch office/site. It is therefore stored at a central server which can be queried by branch offices. Similarly, special resources such as supercomputers exist only in certain locations, and to access such supercomputers, users need to log in remotely.

4. Enhanced reliability

a. A distributed system has the inherent potential to provide increased reliability because of the possibility of replicating resources and executions, as well as the reality that geographically distributed resources are not likely to crash/malfunction at the same time under normal circumstances.

b. Reliability entails several aspects:

i. availability, i.e., the resource should be accessible at all times;

ii. integrity, i.e., the value/state of the resource should be correct, in the face of concurrent access from multiple processors, as per the semantics expected by the application;

iii. fault-tolerance, i.e., the ability to recover from system failures, where such failures may be defined to occur in one of many failure models

5. Increased performance/cost ratio

a. By resource sharing and accessing geographically remote data and resources, the performance/cost ratio is increased.

b. Although higher throughput has not necessarily been the main objective behind using a distributed system, nevertheless, any task can be partitioned across the various computers in the distributed system.

c. Such a configuration provides a better performance/cost ratio than using special parallel machines.

6. Scalability

a. As the processors are usually connected by a wide-area network, adding more processors does not pose a direct bottleneck for the communication network.

7. Modularity and incremental expandability

a. Heterogeneous processors may be easily added into the system without affecting the performance, as long as those processors are running the same middleware algorithms.

b. Similarly, existing processors may be easily replaced by other processors.

4. Explain in detail Message-passing systems versus shared memory systems in distributes system.

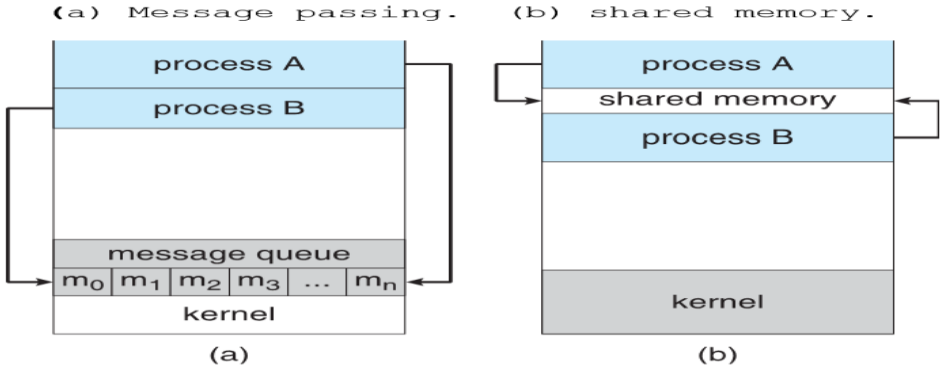


Fig 1.2 a Message Passing **Fig 1.2 b Shared memory**

- Shared memory systems are those in which there is a (common) shared address space throughout the system in above Fig 1.2 a
- Communication among processors takes place via shared data variables, and control variables for synchronization among the processors.
- Semaphores and monitors that were originally designed for shared memory uniprocessors and multiprocessors are examples of how synchronization can be achieved in shared memory systems.
- All multicomputer (NUMA as well as message-passing) systems that do not have a shared address space provided by the underlying architecture and hardware necessarily communicate by message passing.
- For a distributed system, this abstraction is called distributed shared memory. Implementing this abstraction has a certain cost but it simplifies the task of the application programmer.

Emulating MP in SM

1. The shared address space can be partitioned into disjoint parts, one part being assigned to each processor.
2. "Send" and "receive" operations can be implemented by writing to and reading from the destination/sender processor's address space, respectively.

Specifically, a separate location can be reserved as the mailbox for each ordered pair of processes.

3. A Pi–Pj message-passing can be emulated by a write by Pi to the mailbox and then a read by Pj from the mailbox.
4. The write and read operations need to be controlled using synchronization primitives to inform the receiver/sender after the data has been sent/received.

Emulating SM in MP

1. This involves the use of "send" and "receive" operations for "write" and "read" operations.
2. Each shared location can be modeled as a separate process;

a. "write" to a shared location is emulated by sending an update message to the corresponding owner process;

b. a "read" to a shared location is emulated by sending a query message to the owner process.

3. the latencies involved in read and write operations may be high even when using shared memory emulation

4. An application can of course use a combination of shared memory and message-passing.

5. In a MIMD message-passing multicomputer system, each "processor" may be a tightly coupled multiprocessor system with shared memory. Within the multiprocessor system, the processors communicate via shared memory. Between two computers, the communication is by message passing in above Fig 1.2 b

5. Explain in detail Blocking/non-blocking, synchronous/asynchronous primitives for distributed communication.

Synchronous

- A Send or a Receive primitive is synchronous if both the Send() and Receive() handshake with each other.
- The processing for the Send primitive completes only after the invoking processor learns that the other corresponding Receive primitive has also been invoked and that the receive operation has been completed.
- The processing for the Receive primitive completes when the data to be received is copied into the receiver's user buffer.

Asynchronous

- A Send primitive is said to be asynchronous, if control returns back to the invoking process after the data item to be sent has been copied out of the user-specified buffer.
- It does not make sense to define asynchronous Receive primitives.

Blocking primitives

- A primitive is blocking if control returns to the invoking process after the processing for the primitive (whether in synchronous or asynchronous) completes.

Non Blocking primitives

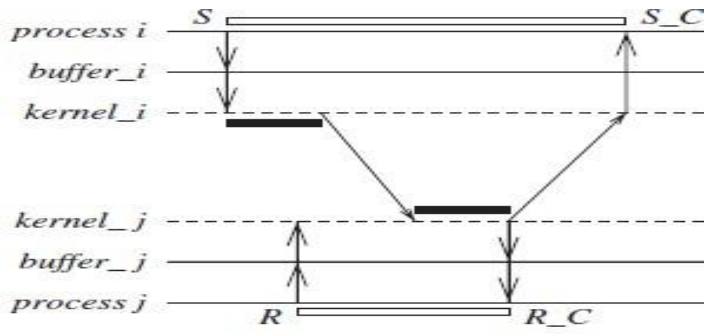
- A primitive is non-blocking if control returns back to the invoking process immediately after invocation. even though the operation has not completed. For a non –blocking send ,control returns to the process even before the data is copied out of the user buffer .For a Non –blocking receive ,control returns to the process even before the data may have arrived from the sender.

6.Explain four versions of the *Send* primitive in Blocking/non-blocking, synchronous/ asynchronous primitives for distributed communication. APR/MAY 2024

The send and receive primitives can be implemented in four modes:

- ❖ Blocking synchronous
- ❖ Non- blocking synchronous
- ❖ Blocking asynchronous
- ❖ Non- blocking asynchronous

❖ **Blocking synchronous**



(a) Blocking sync. *Send*, blocking *Receive*

	Duration to copy data from or to user buffer		
	Duration in which the process issuing send or receive primitive is blocked		
S	*Send* primitive issued	*S_C*	processing for *Send* completes
R	*Receive* primitive issued	*R_C*	processing for *Receive* completes
P	The completion of the previously initiated nonblocking operation		
W	Process may issue *Wait* to check completion of nonblocking operation		

Fig 1.3 a Blocking sync, Send, blocking Receive

Send

- The data gets copied from the user buffer to the kernel buffer and is then sent over the network.

● After the data is copied to the receiver's system buffer and a Receive call has been issued, an acknowledgement is sent to sender in above Fig 1.3 a

● The process that invoked the Send operation and completes the Send.

Receive

● The Receive call blocks until the data expected arrives and is written in the specified user buffer.

● Then control is returned to the user process

❖ **Non- blocking synchronous**

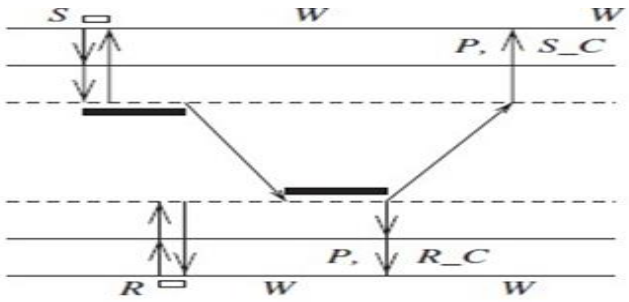


Fig 1.3 b Non Blocking sync, Send, non blocking Receive

Send

● Control returns back to the invoking process as soon as the copy of data from the user buffer to the kernel buffer is initiated in above Fig 1.3 b

● Handle is returned as parameter which can be used to check the process completion.

Receive

● When you make a Receive call, the system gives you a special ID (handle).

● Handle can be used to check if the non-blocking Receive operation is finished.

● The system sets this handle as "done" when the expected data arrives and is put in your specified place (buffer).

❖ **Blocking asynchronous**

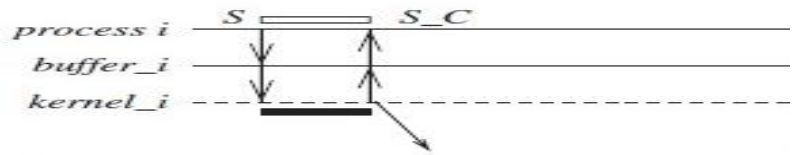


Fig 1.3 c Blocking async, Send

Send

● The user process that invokes the Send is blocked until the data is copied from the user's buffer to the kernel buffer in above Fig 1.3 c

❖ **Non- blocking asynchronous**

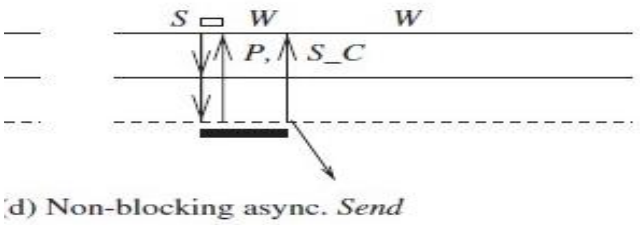


Fig 1.3 d Non Blocking async, Send

Send

● The user process that invokes the Send is blocked until the transfer of the data from the user's buffer to the kernel buffer is initiated in above Fig 1.3 d

● Control returns to the user process as soon as this transfer is initiated, and handle is given back.

● The asynchronous Send completes when the data has been copied out of the user's buffer

7. Write short notes on Processor synchrony and Libraries and standards? APR/MAY 2024

1. **Processor synchrony**

   This means that all the computers or processors in a system work together perfectly, like synchronized dancers following the same rhythm.

   Their internal clocks are all perfectly in sync

2. **In Distributed systems**

   In reality achieving perfect synchrony among all the processors in a distributed system is very difficult or impossible ,so what we do instead is synchronize them in a different way.

3. Synchronization at a Higher level

Instead of making every little action perfectly synchronized ,we group many actions into larger chunks called “steps” think of these steps like dance routines.

4.Barrier Synchronization

To make sure these steps are performed in sync ,we use a mechanism called “barrier synchronization”.

Its like a checkpoint in a dance routine. No dancer can move to the next step until everyone has completed the current one. Similarly in a distributed system ,no processor can move on to the next step of their work until all processors have finished their current step.

Libraries and Standards

● In computer systems, there are many ways for programs to talk to each other, like sending messages or making remote calls. Different software products and scientific tools use their own special ways to do this.

● For example, some big companies use their custom methods, like IBM's CICS software. Scientists often use libraries called MPI or PVM. Commercial software often uses a method called RPC, which lets you call functions on a different computer like you would on your own computer.

● All these methods use something like a hidden network phone line (called "sockets") to make these remote calls work.

● There are many types of RPC, like Sun RPC and DCE RPC. There are also other ways to communicate, like "messaging" and "streaming."

● As software evolves, there are new methods like RMI and ROI for object-based programs, and big standardized systems like CORBA and DCOM.

1.MPI-Message Passing Interface

2.PVM – Parallel Virtual Machine

3.RPC-Remote Procedure call

4.DCE-Distributed computing Environment

5.RMI-Remote Method Invocation

6.ROI-Remote Object Invocation

7.CORBA-Common Object Request Broker Architecture

8.DCOM- Distributed component Object Model

8.Explain the Synchronous versus asynchronous executions with a diagram.

Compare Synchronous versus asynchronous execution.

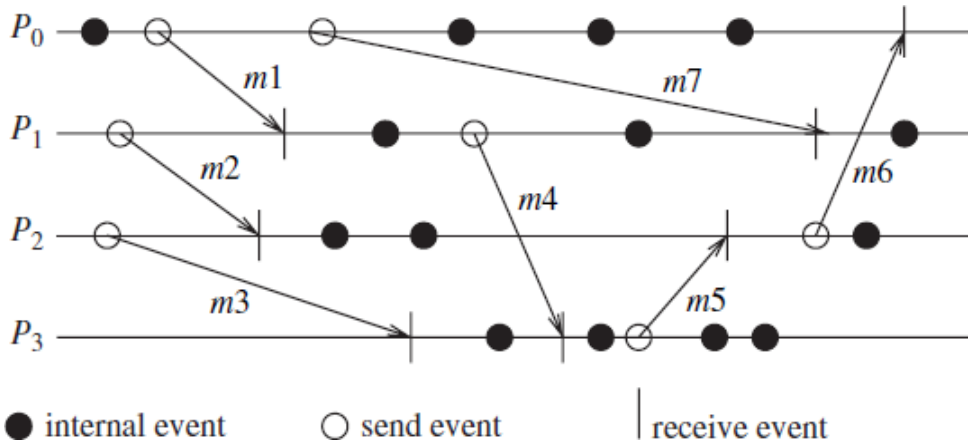


Fig 1.4 (a) asynchronous execution in a message passing system

Asynchronous

1. there is no processor synchrony and there is no bound on the drift rate of processor clocks,
2. message delays (transmission + propagation times) are finite but unbounded
3. there is no upper bound on the time taken by a process to execute a step in above Fig 1.4 a

Synchronous

1. processors are synchronized and the clock drift rate between any two processors is bounded
2. message delivery (transmission + delivery) times are such that they occur in one logical step or round

3. there is a known upper bound on the time taken by a process to execute a step in below Fig 1.4 b

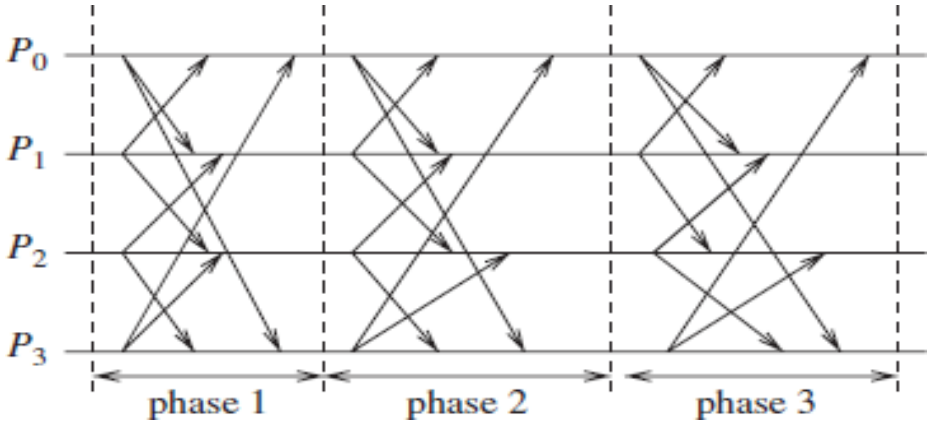


Fig 1.4(b) synchronous execution in a message passing system

9.Explain in details the Design issues and challenges Distributed computing systems. What are the functions must be addressed while designing and building a distributed system? (or) Analyze the concepts of heterogeneity, openness, security, scalability impact of the distributed system. How is the standardization of them make them effective? NOV/DEC 2023/APR/MAY 2024

Challenges from System Perspective

- **Processes :** Management of processes and threads at client/server ;code migration ;and the design of software and mobile agents.
- **Naming :** Easy to use and robust naming for identifiers ,and addresses is essential for locating resources and processes in a transparent and local manner.
- **Synchronization :** Mechanisms for synchronization or coordination among the processesare essential.
- **Consistency and replication :** To avoid bottleneck ,is to provide fast access to data and provide replication for fast access, scalability. Require consistency management among replicas

Design issues of distributed system

Heterogeneity

Heterogeneity refers to the differences that arise in networks, programming languages, hardware, operating systems and differences in software implementation. For example, there are different hardware devices, tablets, mobile phones, computers, and etc.

Some challenges may present themselves due to heterogeneity. When programs are written in different languages or developers utilize different implementations problems will arise when the computers try to communicate with each other.

Scalability:

Scalability of the system should remain efficient even with a significant increase in the number of users and resources connected. It shouldn't matter if a programme has 10 or 100 nodes; performance shouldn't vary. A distributed system's scaling requires consideration of a number of elements, including size, geography, and management.

Openness:

The openness of the distributed system is determined primarily by the degree to which new resource-sharing services can be made available to the users. Open systems are characterized by the fact that their key interfaces are published. It is based on a uniform communication mechanism and published interface for access to shared resources. It can be constructed from heterogeneous hardware and software.

Security:

Security of information system has three components Confidentially, integrity and availability. Encryption protects shared resources, keeps sensitive information secrets when transmitted.

Failure Handling:

When some faults occur in hardware and the software program, it may produce incorrect results or they may stop before they have completed the intended computation so corrective measures should to implemented to handle this case. Failure handling is difficult in distributed systems because the failure is partial i, e, some components fail while others continue to

function.

Concurrency:

There is a possibility that several clients will attempt to access a shared resource at the same time. Multiple users make requests on the same resources, i.e read, write, and update. Each resource must be safe in a concurrent environment. Any object that represents a shared resource in a distributed system must ensure that it operates correctly in a concurrent environment.

Transparency :

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

10. Explain in detail the Applications of distributed computing and newer challengesMobile systems.

1. **Mobile system**

In mobile system, wireless communication is crucial ,posing challenges like transmission range, power conservation ,and interfacing with the weird internet. Computer science problems include routing, location management, channel allocation and managing mobility.

There are two popular architecture.

- Base station approach (cellular):Mobile processes in a cell communicate through a static base station, which presents graph theoretical and engineering challenges.
- Ad-Hoc Network Approach :No central base station ;communication responsibilities are distributed among mobile nodes, requiring complex routing and presenting graph theoretical and engineering challenges.

2.Sensor Networks:

- Sensors, which can measure physical properties like temperature and humidity, have become affordable and are deployed in large numbers (over a million)

- They report external events, not internal computer processes. These networks have various applications, including mobile or static sensors that communicate wirelessly or through wires. Self-configuring ad-hoc networks introduce challenges like position and time estimation.

3.Ubiquitous Computing:

- Ubiquitous systems involve processors integrated into the environment, working in the background, like in sci-fi scenarios. Examples include smart homes and workplaces.
- These systems are essentially distributed, use wireless tech, sensors, and actuators, and can self-organize. They often consist of many small processors in a dynamic network, connecting to more powerful resources for data processing.

4.Peer-to-Peer (P2P) Computing:

- In P2P computing, all processors interact as equals without any hierarchy, unlike client-server systems. P2P networks are often self-organizing and may lack a regular structure.
- They don't use central directories for name resolution. Challenges include efficient object storage and lookup, dynamic reconfiguration, replication strategies, and addressing issues like privacy and security.

5.Publish-Subscribe, Content Distribution, and Multimedia: (Netflix)

- As information grows, we need efficient ways to distribute and filter it. Publish-Subscribe involves distributing information, letting users subscribe to what interests them, and then filtering it based on user preferences.
- Content distribution is about sending data with specific characteristics to interested users, often used in web and P2P settings. When dealing with multimedia, we face challenges like large data, compression, and synchronization during storage and playback.

6.Data Mining Algorithms:

- They analyze large data sets to find patterns and useful information. For example, studying customer buying habits for targeted marketing.
- This involves applying database and AI techniques to data. When data is distributed, as in private banking or large-scale weather prediction, efficient distributed data mining algorithms are needed.

7.Security Challenges in Distributed Systems:

❖ Traditional challenges include ensuring confidentiality, authentication, and availability.

❖ The goal is efficient and scalable solutions.

❖ In newer distributed architectures like wireless, peer-to-peer, grid, and pervasive computing, these challenges become more complex due to resource constraints, broadcast mediums, lack of structure, and network trust issues.

11.Explain in detail distributed program and model of distributed executions.

Distributed Program

- A distributed program is composed of a set of n asynchronous processes p1, p2, ..., pi , ,pn that communicate by message passing over the communication network.
- We assume that each process is running on a different processor.
- The processes do not share a global memory and communicate solely by passing messages.
- Cij - Channel from pi to process pj
- mij- a message sent by pi to pj.
- Don't share a global clock.
- Process execution and message transfer are asynchronous
- The global state of a distributed computation is composed of the states of the processes and the communication channels

Model for Distributed Execution

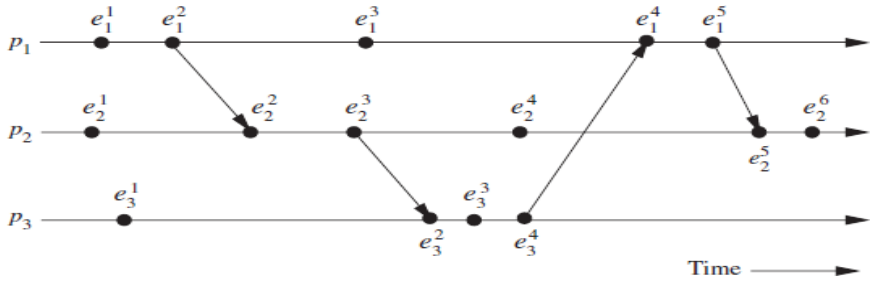


Fig 1.5 a Distributed Execution

Model for Distributed Execution

1. Execution of a process consists of a sequential execution of its actions.

2. The actions are atomic and the actions of a process are modeled as three types of events, internal events, message send events (send (m)), and message receive events(rec(m)).

3. The occurrence of events changes the states of respective processes and channels, thus causing transitions in the global system state in above Fig 1.5 a

4. An internal event changes the state of the process at which it occurs.

5. A send event (or a receive event) changes the state of the process that sends (or receives) the message and the state of the channel on which the message is sent (or received)

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.
- Note that two or more events may be logically concurrent even though they do not occur at the same instant in physical time.

12. Write short notes on Models of communication networks.

Models of Communication Networks (FIFO, Non-FIFO, Causal Ordering)

1. FIFO - each channel acts as a first-in first-out message queue and thus, message ordering is preserved by a channel.

2. 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.

3. Causal Ordering (built-in synch)- based on Lamport's "happens before" relation. A system that supports the causal ordering model satisfies the following property:

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

That is, 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. Furthermore, note that CO $\subset$ FIFO $\subset$ Non-FIFO.

13.Explain in detail the global state of a distributed system.

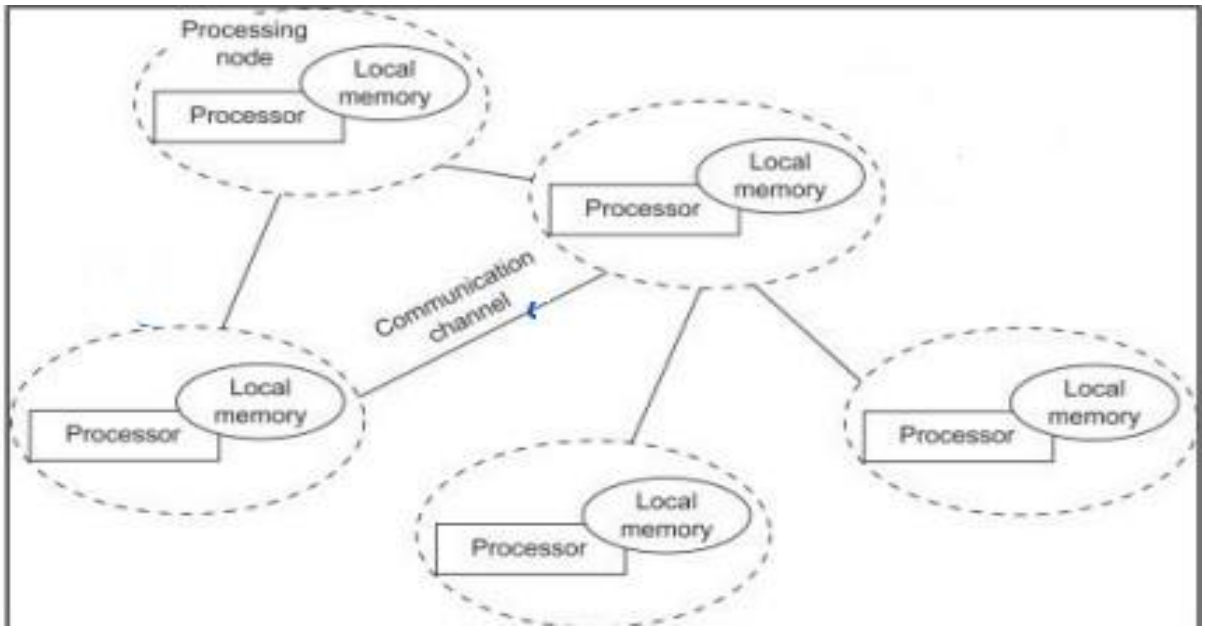


Fig 1.6 a Global state

1. The state of a process at any time is defined by the contents of processor registers, stacks, local memory, etc. and depends on the local context of the distributed application.

2. The state of a channel is given by the set of messages in transit in the channel in above Fig 1.6 a

3. The occurrence of events changes the states of respective processes and channels, thus causing transitions in global system state.

a. For example, an internal event changes the state of the process at which it occurs. A send event (or a receive event) changes the state of the process that sends (or receives) the message and the state of the channel on which the message is sent (or received).

State of Process

Let LS_i^x denote the state of process p_i after the occurrence of event e_i^x and before the event e_i^{x+1}. LS_i^0 denotes the initial state of process p_i.

$send(m) \leq LS_i^x$ denote the fact that $\exists y: 1 \leq y \leq x : e_i^y = send(m)$

$rec(m) \not\leq LS_i^x$ denote the fact that $\forall y: 1 \leq y \leq x :: e_i^y \neq rec(m)$.

State of Channel

Let $SC_{ij}^{x,y}$ denote the state of a channel C_{ij} defined as follows.

$$SC_{ij}^{x,y} = \{m_{ij} \mid send(m_{ij}) \leq LS_i^x \wedge rec(m_{ij}) \not\leq LS_j^y\}.$$

Thus, channel state $SC_{ij}^{x,y}$ denotes all messages that p_i sent up to event e_i^x and which process p_j had not received until event e_j^y.

How to find global state?

❖ For a global snapshot to be meaningful, the states of all the components of the distributed system must be recorded at the same instant.

❖ This will be possible if the local clocks at processes were perfectly synchronized or there was a global system clock that could be instantaneously read by the processes. However, both are impossible.

❖ Solution:

❖ Recording at different time will be meaningful provided every message that is recorded as received is also recorded as sent.

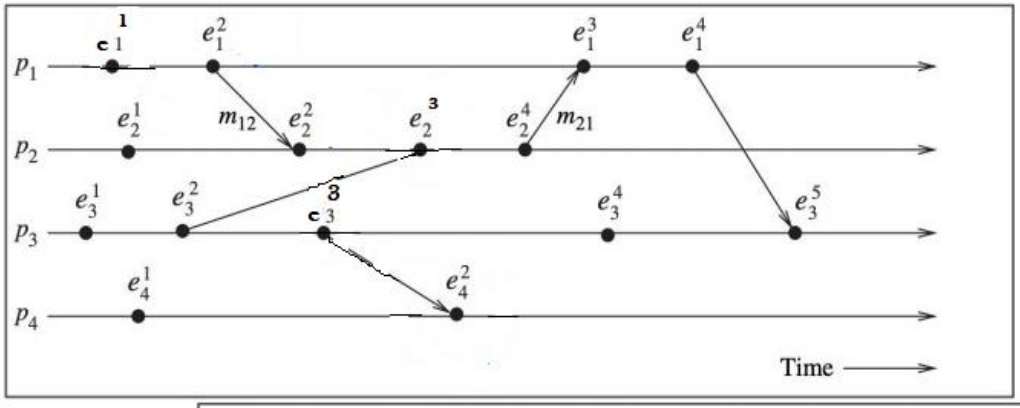
❖ Basic idea is that an effect should not be present without its cause.

❖ States that don't violate causality are called consistent global states and are meaningful global states in below Fig 1.6 b & Fig 1.6 c

A global state $GS = \{\bigcup_i LS_i^{x_i}, \bigcup_{j,k} SC_{jk}^{y_j,z_k}\}$ is a *consistent global state* iff it satisfies the following condition:

$$\forall m_{ij} : send(m_{ij}) \not\in LS_i^{x_i} \Rightarrow m_{ij} \not\in SC_{ij}^{x_i,y_j} \wedge rec(m_{ij}) \not\in LS_j^{y_j})$$

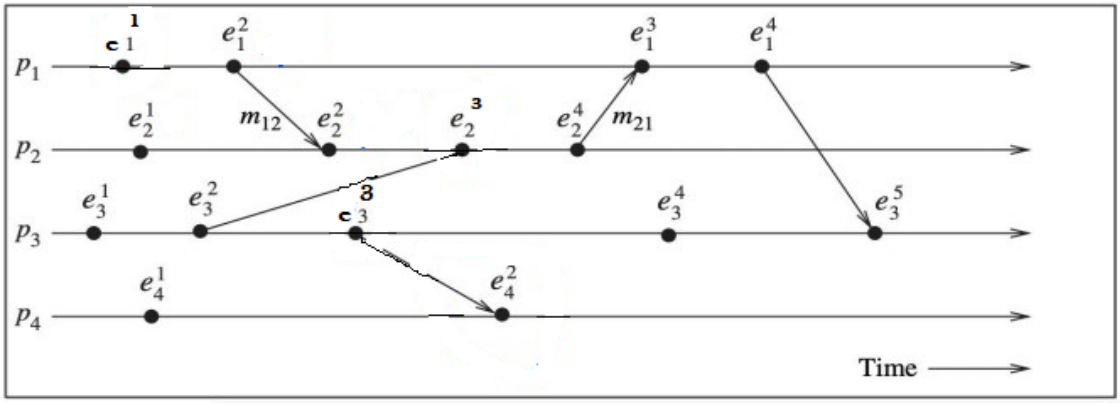
That is, channel state $SC_{ik}^{y_i,z_k}$ and process state $LS_k^{z_k}$ must not include any message that process p_i sent after executing event $e_i^{x_i}$.



Inconsistent state

of local states $\{LS_1^1, LS_2^3, LS_3^3, LS_4^2\}$ is inconsistent because the state of p_2 has recorded the receipt of message m_{12}, however, the state of p_1 has not recorded its send.

Fig 1.6 b Inconsistent state



Consistent state

states $\{LS_1^2, LS_2^4, LS_3^4, LS_4^2\}$ is consistent; all the channels are empty except C_{21} that contains message m_{21}.

Fig 1.6 c Inconsistent state

Strongly Consistent

A global state $GS = \{\bigcup_i LS_i^{x_i}, \bigcup_{j,k} SC_{jk}^{y_j,z_k}\}$ is *transitless* iff

$$\forall i, \forall j : 1 \leq i, j \leq n :: SC_{ij}^{y_i,z_j} = \phi.$$

Thus, all channels are recorded as empty in a transitless global state. A global state is *strongly consistent* iff it is transitless as well as consistent. Note that in Figure 2.2, the global state consisting of local states $\{LS_1^2, LS_2^3, LS_3^4, LS_4^2\}$ is strongly consistent.

14.Explain message passing system and discuss on the message oriented middleware and its types. Also explain their functionality in distributed computing. Nov/Dec 2023

Message Passing System

- Message passing in distributed systems refers to the communication medium used by nodes (computers or processes) to commute information and coordinate their actions. It involves transferring and entering messages between nodes to achieve various goals such as coordination, synchronization, and data sharing.
- Message passing is a flexible and scalable method for inter-node communication in distributed systems. It enables nodes to exchange information, coordinate activities, and share data without relying on shared memory or direct method invocations.
- Models like synchronous and asynchronous message passing offer different synchronization and communication semantics to suit system requirements. Synchronous message passing ensures sender and receiver synchronization, while asynchronous message passing allows concurrent execution and non-blocking communication in below Fig 1.7 a

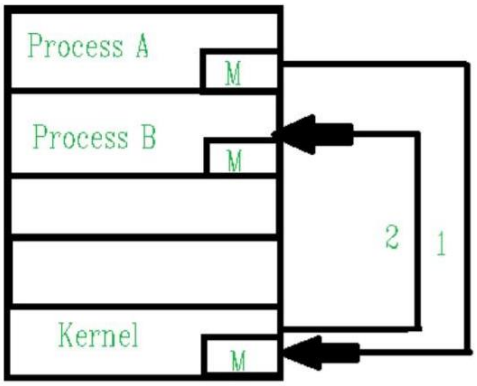


Fig 1.7 a Message passing system

Message Oriented Middleware

MOM provides asynchronous communication, and it just sends the message and performs its asynchronous operations. It consists of inter-application communication software that relies on asynchronous message passing which would oppose request-response architecture. So asynchronous system consists of a message queue that provides a temporary stage so that the destination program becomes busy or might not be connected. Message Queue helps in storing the message on a MOM platform. MOM clients can send and receive the message through the queue.

Queues act as a central component for implementing asynchronous interaction within MOM.

- Middleware is software that acts as a link between two or more objects
- Middleware simplifies complex distributed applications,
- It consists of web servers, application servers, and more, it is integrals to modern information technology based on XML, SOAP, service-oriented architecture.

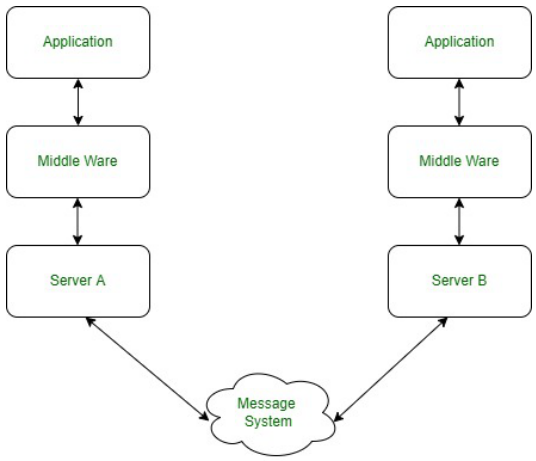


Fig 1.7 b **Block Representation of** Middleware

Functionality of Message Passing System:

Efficiency: Efficiency is a critical task in the distributed message-passing system. If the event that the passing system is inter-process communication. The whole message passing system is collapsed.

Reliability: The message passing system ought to be reliable. This feature can be accomplished when the message passing system is following the right convention. Generally, the communication link failures may interrupt the communication cycle. A reliable IPC protocol can manage failure issues and guarantee of delivery of a message.

Correctness: A message-passing system ought to confirm the clients regardless of whether they are the right clients. If the right client isn't found delivery of the message will become celled and again send the status to the sender in above Fig 1.7 b

Functionality of Message Oriented Middleware

1. Unified messaging
2. Provisioning and monitoring
3. Dynamic scaling
4. Management and control tools
5. Dynamic scaling
6. Flexible service quality
7. Secure communication
8. Integration with other tools

UNIT II - MESSAGE ORDERING & SNAPSHOTS

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 (CO) - Total order. **Global state and snapshot recording algorithms:** Introduction –System model and definitions –Snapshot algorithms for FIFO channels.

PART A

1. **Define Physical clock synchronization.**

   In physical clock synchronization, the goal is to minimize the time differences or clock drift between different clocks to ensure that events, processes, and data transmissions occur in a coordinated and orderly manner.

2. **Definition of Scalar time NOV/DEC 2023**

   Scalar Time is a simple implementation that uses local clocks and two rules to ensure correct order. Other methods, such as using timestamps, can also be used but must obey causality to work properly. By using logical clocks, processes in a distributed system can work together in an organized and efficient way.

3. **Define Vector time with notation. NOV/DEC 2023**

   Vector time, in the context of distributed systems and parallel computing, refers to a concept used to order and compare events that occur in different processes or nodes within a distributed system.

4.What is Strong consistency?

Strong consistency is a property in distributed computing and database systems that ensures that all nodes or replicas in a distributed system have a synchronized and identical view of data at all times.

5.Define Event counting.

Event counting, also known as event counting systems or event counters, refers to a mechanism used to track and count occurrences of specific events or incidents within a system, process, or application. It involves maintaining a numerical count of occurrences as events take place, allowing for analysis, monitoring, and control of various aspects of the system's behavior.

6. **Define and terminology for Time, Frequency, Offset, Skew and Drift (rate).**

   Time: Time refers to the continuous and irreversible progression of events from the past through the present to the future.

Frequency: Frequency is the number of occurrences of a repeating event (such as a waveform oscillation or a signal pulse) in a unit of time.

Offset: Offset, in the context of time synchronization, refers to the difference between the local time of a clock and a reference time, such as a standard time source.

Skew: Skew is the difference in time between two related events or clocks that are supposed to be synchronized.

Drift (Rate): Drift, often referred to as clock drift or frequency drift, represents the change in the rate of a clock's timekeeping over time. It indicates how quickly or slowly a clock gains or loses time compared to a more accurate reference clock.

7.What are Clock inaccuracies?

Clock inaccuracies refer to the discrepancies between the time displayed by a clock and the true, accurate time. In other words, it's the difference between what a clock shows and what the actual time is. Clock inaccuracies can be caused by various factors and sources, which can lead to the clock running faster or slower than the reference time.

8.What are message ordering paradigms used in distributed systems?

- ✓ Message ordering paradigms are approaches used in distributed systems to determine how messages are ordered and delivered among different processes or nodes. Several orderings on messages have been defined:
  - ✓ Unordered Delivery
  - ✓ FIFO (First-In-First-Out) Delivery
  - ✓ Causal Delivery
  - ✓ Total Order Delivery

9. Define Isomorphism?

Isomorphism refers to a similarity or equivalence of the structure or behavior of different components, nodes, or processes within the system. It implies that despite being distinct entities, these components exhibit a comparable pattern of interactions, functionality, or communication. Isomorphism in distributed systems helps in understanding and modeling relationships, behaviors, and interactions among system elements.

10. What are the two basic models of process communications?

There are two basic models of process communications

- ✓ synchronous
- ✓ asynchronous

11. What is *synchronous* communication model?

The *synchronous* communication model is a blocking type where on a message send, the sender process blocks until the message has been received by the receiver process. The sender process resumes execution only after it learns that the receiver process has accepted the message.

12. What is *asynchronous* communication model?

The *asynchronous* communication model is a non-blocking type where the sender and the receiver do not synchronize to exchange a message. After having sent a message, the sender process does not wait for the message to be delivered to the receiver process.

13. Define Group Communication in distributed System.

Group communication systems commonly provide specific guarantees about the total ordering of messages, such as, that if the sender of a message receives it back from the GCS, then it is certain that it has been delivered to all other nodes in the system. This property is useful when constructing data replication systems.

14. **What is causally ordered (CO) executions?**

Causally ordered executions ensure that if event A causally precedes event B (meaning that event B depends on the outcome of event A), then event A should be executed before event B in a distributed system. This ensures that the effects of one event are visible to events that are causally dependent on it.

15. What is Total Order?

In total order, all events or processes in a distributed system are globally ordered in a way that every process agrees on the same order for all events.

16.What is the Complexity of Three-phase distributed algorithm?

In 3PC, the commit process is divided into three phases: the "Prepare" phase, the "Pre-Commit" phase, and the "Commit" phase. Each phase involves communication and coordination between the coordinator and the participants. The goal is to ensure that all participants agree on whether to commit or abort the transaction.

17.Define a Consistent global state APR/MAY 2024

A "consistent" global state of a distributed system is one where the local state of each process does not depend on the receipt of a message that is yet to be sent.

18.Define global state. APR/MAY 2024

A global state is a set of local states which are all concurrent with each other. By concurrent, we mean that no two states have a cause and effect relationship with each other.

19.What are the Issues in recording a global state?

The following two issues need to be addressed:

Issue 1: How to distinguish between the messages to be recorded in the snapshot from those not to be recorded.

-Any message that is sent by a process before recording its snapshot, must be recorded in the global snapshot (from C1).

-Any message that is sent by a process after recording its snapshot, must not be recorded in the global snapshot (from C2).

Issue 2: How to determine the instant when a process takes its snapshot.

-A process pj must record its snapshot before processing a message m_{ij} that was sent by process p_i after recording its snapshot.

20. What is Chandy-Lamport algorithm?

The Chandy-Lamport algorithm, also known as the "Distributed Snapshot Algorithm," is a well-known algorithm used in distributed systems to capture a consistent global snapshot of the state of a distributed system. The Chandy-Lamport algorithm is particularly useful in scenarios where processes communicate asynchronously and concurrently, making it challenging to capture a consistent snapshot of the system's state.

21.What is the Complexity of Chandy-Lamport algorithm?

The complexity of the algorithm can be analyzed in terms of message complexity, time complexity, and space complexity**.**

22.Define the Properties of the recorded global state.

The properties of a recorded global state include:

Causality Preservation: The global state preserves the causal relationships between events and actions.

Concurrency Handling: The global state accounts for the concurrent execution of processes or threads.

Complete and Accurate: The global state provides a complete and accurate snapshot of the system.

23.What are the Models of communication?

The three main types of communication models in distributed systems are:

FIFO (first-in, first-out): each channel acts as a FIFO message queue.

Non-FIFO (N-FIFO): a channel acts like a set in which a sender process adds messages and receiver removes messages in random order.

Causal Ordering (CO): It follows Lamport's law. o The relation between the three models is given by CO $\subset$ FIFO $\subset$ N-FIFO.

24.Define binary rendezvous.

If multiple interactions are enabled, a process chooses one of them and tries to synchronize with the partner process. The problem reduces to one of scheduling messages satisfying the following constraints:

- Schedule on-line, atomically, and in a distributed manner.
- Schedule in a deadlock-free manner (i.e., crown-free).
- Schedule to satisfy the progress property in addition to the safety property.

25.What is a simple algorithm by Bagrodia that makes the following assumptions?

A simple algorithm by Bagrodia, makes the following assumptions:

1. Receive commands are forever enabled from all processes.
2. A send command, once enabled, remains enabled until it completes.
3. To prevent deadlock, process identifiers are used to break the crowns.
4. Each process attempts to schedule only one send event at any time.

26.Define Correctness of Snapshot algorithms.

To prove the correctness of the algorithm, it is shown that a recorded snapshot satisfies conditions C1 and C2.

• Since a process records its snapshot when it receives the first marker on any incoming channel, no messages that follow markers on the channels incoming to it are recorded in the process's snapshot.

27.Define FIFO executions.

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.

28. what are the limitations of logical clock ? APR/MAY 2024

The limitations of logical clocks include:

1. Scalability: Logical clocks can become impractical in very large distributed systems, as the size of the clock value grows with the number of processes.
2. Clock synchronization: Logical clocks assume that all processes have a consistent view of the clock value, which can be challenging to maintain in practice.
3. Partial ordering: Logical clocks only provide a partial ordering of events, as they do not account for concurrent events or events that occur simultaneously.
4. Limited resolution: Logical clocks typically have a limited resolution, meaning they may not be able to distinguish between events that occur very close together in time.

29. If two events, E1 and E2, from different processes occur at the same time (i.e., concurrently) and independently, then the relationship between E1 and E2 is called "concurrent" or "simultaneous". NOV/DEC 2023

In the context of distributed systems, concurrent events are considered to be unrelated, meaning that neither event causes or affects the other. This is because the processes are executing independently, and the events are not synchronized in any way.

In terms of causality, concurrent events are considered to be "unrelated" or "independent", meaning that neither event is the cause of the other. This is denoted as E1 || E2 (read as "E1 parallel to E2").

.

PART B

1. **Write a brief notes on Physical clock synchronization: NTP. (Network Time Protocol) NOV/DEC 2023**

- **Need of Knowing Time**
  - The time of the day at which an event happened on a specific machine in the network.
  - The time interval between two events that happened on different machines in the network.
  - The relative ordering of events that happened on different machines in the network.
- **Need for synchronization**
  - In database systems, the order in which processes perform updates on a database is important to ensure a consistent, correct view of the database. To ensure the right ordering of events, a common notion of time between co-operating processes becomes imperative.
  - Improves the performance of distributed algorithms by replacing communication with local computation.
  - Distributed protocols use timeouts, and their performance depends on how well physically dispersed processors are time-synchronized. Design of such applications is simplified when clocks are synchronized.
  - Clock synchronization is the process of ensuring that physically distributed processors have a common notion of time. It has a significant effect on many problems like secure systems, fault diagnosis and recovery, scheduled operations, database systems, and real-world clock values.

Definitions:

1. Time: The time of a clock in a machine p is given by the function Cp(t),where Cp(t)= t for a perfect clock.

2. Frequency: Frequency is the rate at which a clock progresses. The frequency at time t of clock Ca is Ca'(t).

3. Offset: Clock offset is the difference between the time reported by a clock and the real time. The offset of the clock Ca is given by Ca(t)− t. The offset of clock C a relative to Cb at time $t \geq 0$ is given by Ca(t)- Cb(t)

4. Skew: The skew of a clock is the difference in the frequencies of the clock and the perfect clock. The skew of a clock Ca relative to clock Cb at timet is Ca ' (t)- Cb ' (t).

5. Drift (rate): The drift of clock Ca the second derivative of the clock value with respect to time. The drift is calculated as: C"a(t)- C"b(t)

2.Explain in detail with a neat diagram Clock inaccuracies in a distributed system.

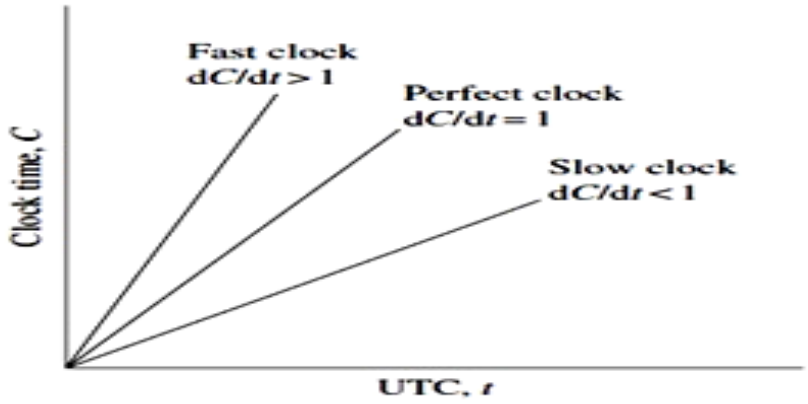


Fig 2.1 (a) The behavior of fast, slow, and perfect clocks with respect to UTC

1. Offset Delay Estimation using Network Transmission Protocol

- In practice, a source node cannot accurately estimate the local time on the target node due to varying message or network delays between the nodes in above Fig 2.1 (a)
- This protocol employs a very common practice of performing several trials and chooses the trial with the minimum delay.
- The design of NTP involves a hierarchical tree of time servers.
  - The primary server at the root synchronizes with the UTC.
  - The next level contains secondary servers, which act as a backup to the primary server.
  - At the lowest level is the synchronization subnet which has the clients in below Fig **2.1 (b)**

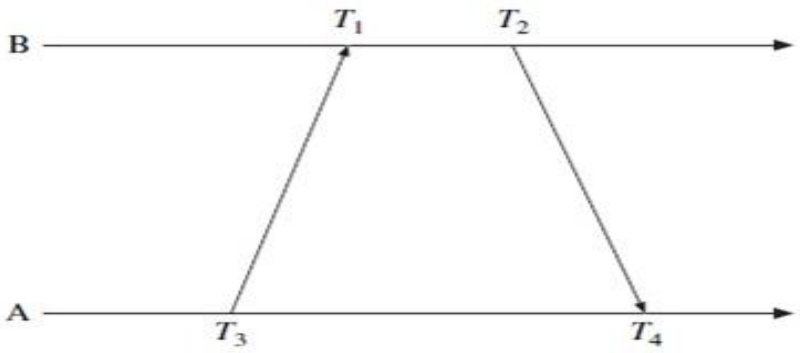


Fig 2.1 (b) Offset and delay estimation

2.NTP Synch Protocol

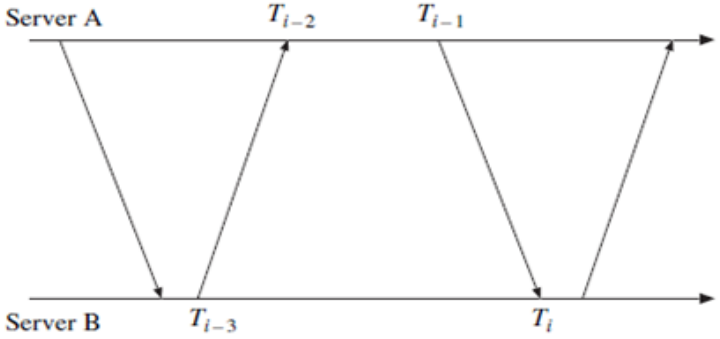


Fig 2.1 (c) NTP Synch Protocol

- A pair of severs in Symmteric mode exchange pairs of timing messages.
- A store of data is then built up about the relationship between the two servers.
- The offset corresponding to the minimum delay is chosen in above Fig 2.1 (c)

- The offset between A's clock and B's clock is O. If A's local clock time is $A(t)$ and B's local clock time is $B(t)$, we have

$$A(t) = B(t) + O.$$

Then,

$$T_{i-2} = T_{i-3} + t + O,$$
$$T_i = T_{i-1} - O + t'.$$

Assuming $t = t'$, the offset O_i can be estimated as

$$O_i = (T_{i-2} - T_{i-3} + T_{i-1} - T_i)/2.$$

The round-trip delay is estimated as

$$D_i = (T_i - T_{i-3}) - (T_{i-1} - T_{i-2}).$$

- The eight most recent pairs of (O_i, D_i) are retained.
- The value of O_i that corresponds to minimum D_i is chosen to estimate O.

3.Describe the logical clock synchronization and explain the framework of a system of logical clock NOV/DEC 2023

1. In distributed computing, logical time refers to a mechanism used to establish an ordering of events that occur across different nodes or processes in a distributed system.
2. It provides a way to reason about the causal relationships between events, even in the absence of a globally consistent physical time.
3. Logical time is crucial for coordinating activities, maintaining consistency, and enabling the implementation of various distributed algorithms and protocols.
4. It also ensures proper order of events.
5. Every event is assigned a timestamp and the causality relation between events can be generally inferred from their timestamps.
6. The timestamps assigned to events obey the fundamental monotonicity property; that is, if an event a causally affects an event b, then the timestamp of a is smaller than the timestamp of b

Methods of Representing Logical Time

1. Lamport's scalar clocks, the time is represented by non-negative integers
2. The time is represented by a vector of non-negative integers
3. The time is represented as a matrix of non-negative integers

Framework for System of Logical Clocks

- A system of logical clocks consists of a time domain T and a logical clock C.
- 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:

$$C : H \mapsto T,$$

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

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

Implementing Logical Clocks

Every process needs data structure(store logical time)+algorithms(updation)

Each process p_i maintains data structures that allow it the following two capabilities:

- A *local logical clock*, denoted by lc_i, that helps process p_i measure its own progress.
- A *logical global clock*, denoted by gc_i, that is a representation of process p_i's local view of the logical global time. It allows this process to assign consistent timestamps to its local events. Typically, lc_i is a part of gc_i.

- **R1** This rule governs how the local logical clock is updated by a process when it executes an event (send, receive, or internal).
- **R2** This rule governs how a process updates its global logical clock to update its view of the global time and global progress. It dictates what information about the logical time is piggybacked in a message and how this information is used by the receiving process to update its view of the global time.

4. Explain the Message ordering paradigms. April/May2022, Nov/Dec2022

1. Asynchronous executions –Non –FIFO

An asynchronous execution (or A-execution) is an execution (E, $\prec$) for which the causality relation is a partial order.

- Non-FIFO transmission : On any logical link between two nodes in the system, messages may be delivered in any order, not necessarily first-in first-out. Such executions are also known as non-FIFO executions in below Fig: 2.2 a
- Although each physical link typically delivers the messages sent on it in FIFO order due to the physical properties of the medium, a logical link may be formed as a composite of physical links and multiple paths may exist between the two end points of the logical link.

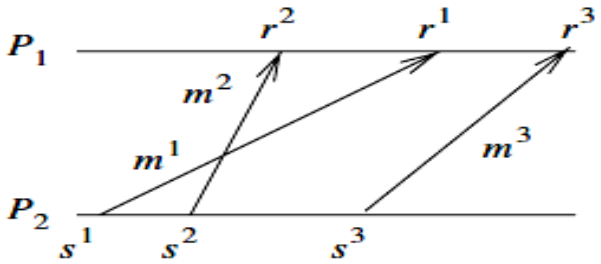


Fig: 2.2 a) FIFO executions

2. FIFO executions

A FIFO execution is an A-execution in which, for all (s, r) and $(s', r') \in \mathcal{T}$, $(s \sim s'$ and $r \sim r'$ and $s \prec s') \Longrightarrow r \prec r'$.

- On any logical link in the system, messages are necessarily delivered in the order in which they are sent. Although the logical link is inherently non-FIFO, most network protocols provide a connection-oriented service at the transport layer.

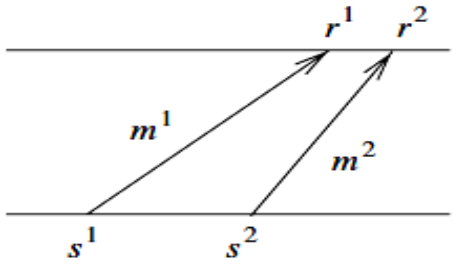


Fig: 2.2 b) non FIFO executions

- Design FIFO algo over non-FIFO channel:
  - separate numbering scheme to sequence the messages on each logical channel.
  - The sender assigns and appends a sequence_num, connection_id tuple to each message.
  - The receiver uses a buffer to order the incoming messages as per the sender's sequence numbers, and accepts only the "next" message in sequence in above Fig: 2.2 b

3. Causally ordered (CO) executions

A CO execution is an A-execution in which,

for all (s, r) and $(s', r') \in \mathcal{T}$, $(r \sim r' \text{ and } s \prec s') \Longrightarrow r \prec r'$.

- If two send events s and s are related by causality ordering (not physical time ordering), then a causally ordered execution requires that their corresponding receive events r and r occur in the same order at all common destinations.

4. Synchronous Execution

- When all the communication between pairs of processes uses synchronous send and receive primitives, the resulting order is the synchronous order.
- As each synchronous communication involves a handshake between the receiver and the sender, the corresponding send and receive events can be viewed as occurring instantaneously and atomically in below Fig 2.3 a & 2.3 b

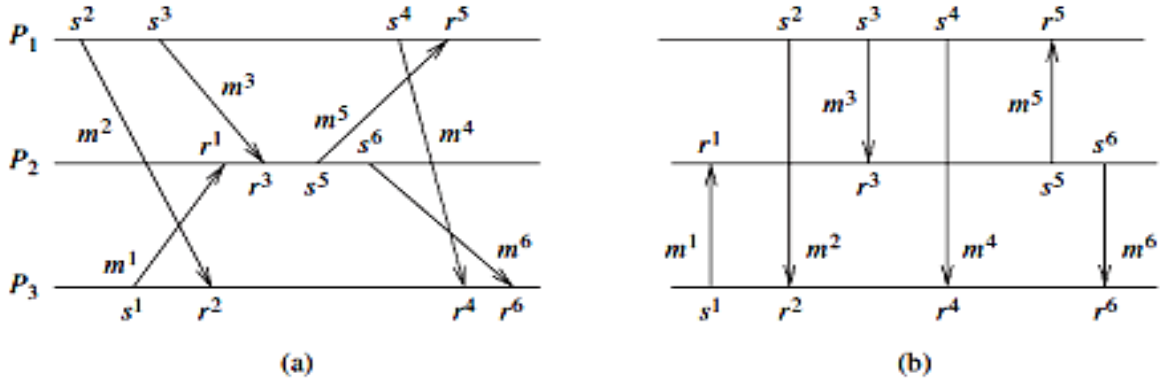


Fig 2.3 (a) Execution in an asynchronous system Fig 2.3 b) Equivalent instantaneous communication

5.Explain in detail about Asynchronous execution with synchronous communication

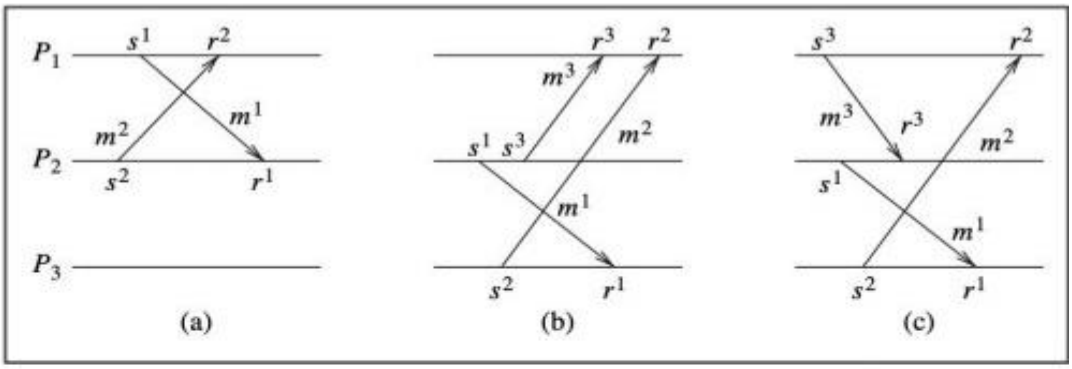
- A distributed algorithm designed to run correctly on asynchronous systems (called A-executions) may not run correctly on synchronous systems.
- An algorithm that runs on an asynchronous system may deadlock on a synchronous system.

Process i	**Process** j
...	...
Send(j)	*Send*(i)
Receive(j)	*Receive*(i)
...	...

Executions realizable with synchronous communication (RSC)

- In an A-execution, the messages can be made to appear instantaneous if there exists a linear extension of the execution, such that each send event is immediately followed by its corresponding receive event in this linear extension.
- Such an A-execution can be realized under synchronous communication and is called a realizable with synchronous communication (RSC) execution in below Fig 2.4 a

Crown



Definition 6.12 (Crown) Let E be an execution. A crown of size k in E is a sequence $\langle (s^i, r^i), i \in \{0, \ldots, k-1\} \rangle$ of pairs of corresponding send and receive events such that: $s^0 \prec r^1, s^1 \prec r^2, \ldots, s^{k-2} \prec r^{k-1}, s^{k-1} \prec r^0$.

Examples

- Figure 6.5(a): The crown is $\langle (s^1, r^1), (s^2, r^2) \rangle$ as we have $s^1 \prec r^2$ and $s^2 \prec r^1$. This execution represents the program execution in Figure 6.4.
- Figure 6.5(b): The crown is $\langle (s^1, r^1), (s^2, r^2) \rangle$ as we have $s^1 \prec r^2$ and $s^2 \prec r^1$.
- Figure 6.5(c): The crown is $\langle (s^1, r^1), (s^3, r^3), (s^2, r^2) \rangle$ as we have $s^1 \prec r^3$ and $s^3 \prec r^2$ and $s^2 \prec r^1$.

Fig 2.4 a Crown

Hierarchy of ordering paradigms

- For an A-execution, A is RSC if and only if A is an S-execution. • RSC ⊂ CO ⊂ FIFO ⊂ A.
- The above hierarchy implies that some executions belonging to a class X will not belong to any of the classes included in X. Thus, there are more restrictions on the possible message orderings in the smaller classes in below Fig 2.4 b
- A program using synchronous communication is easiest to develop and verify. A program using non-FIFO communication, resulting in an A execution, is hardest to design and verify.
- This is because synchronous order offers the most simplicity due to the restricted number of possibilities, whereas non-FIFO order offers the greatest difficulties because it admits a much larger set of possibilities that the developer and verifier need to account for.

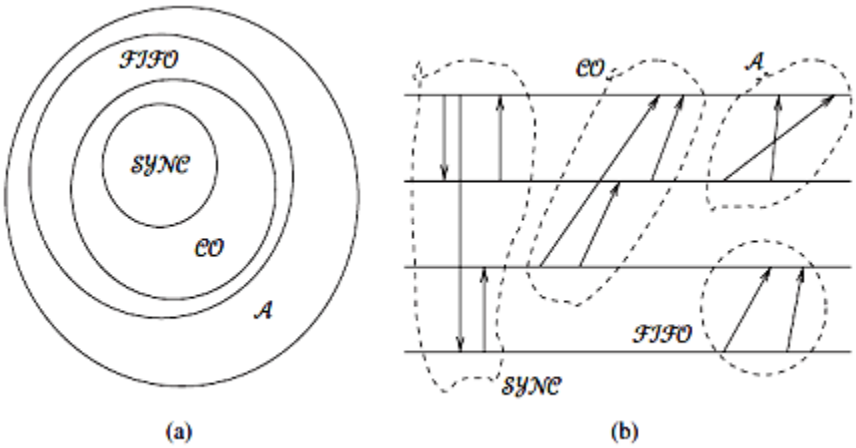


Fig : 2.4 b Hierarchy of execution classes

Simulation - Asynchronous programs on synchronous systems (How to make non RSC to RSC)

- Each channel Cij is modeled by a control process Pij that simulates the channel buffer.
- An asynchronous communication from i to j becomes a synchronous communication from i to Pij followed by a synchronous communication from Pij to j.
- This enables the decoupling of the sender from the receiver, a feature that is essential in asynchronous systems in below Fig 2.4 c

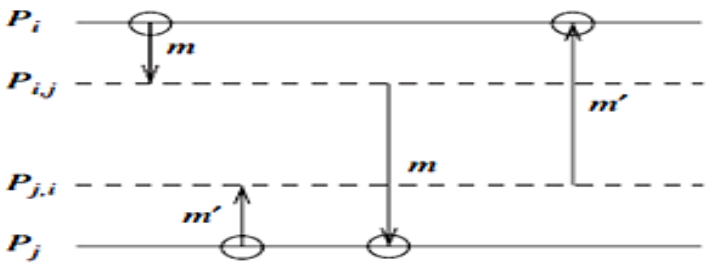


Fig : 2.4 c Modeling channels as processes to simulate an execution using asynchronous primitives on synchronous system

Synchronous programs on asynchronous systems

1. A (valid) S-execution can be trivially realized on an asynchronous system by scheduling the messages in the order in which they appear in the S execution.

2. The partial order of the S-execution remains unchanged but the communication occurs on an asynchronous system that uses asynchronous communication primitives.

3. Once a message send event is scheduled, the middleware layer waits for an acknowledgment; after the ack is received, the synchronous send primitive completes.

6.Discuss about the Synchronous program order on an asynchronous system

How non-determinism occurs in DC?

1. A receive call can receive a message from any sender who has sent a message, if the expected sender is not specified.

2. Multiple send and receive calls which are enabled at a process can be executed in an interchangeable order.

Rendezvous

- Multiway rendezvous:- synchronous communication among an arbitrary number of asynchronous processes.
- Binary rendezvous:- synchronous communication between a pair of processes at a time.
- Support for binary rendezvous communication was first provided by programming languages such as CSP and Ada. We consider here a subset of CSP in below **Fig 2.5 a & b**

Algorithm for binary rendezvous

Constraints:

● Schedule on-line, atomically, and in a distributed manner, i.e., the scheduling code at any process does not know the application code of other processes.

● Schedule in a deadlock-free manner (i.e., crown-free), such that both the sender and receiver are enabled for a message when it is scheduled.

● Schedule to satisfy the progress property (i.e., find a schedule within abounded number of steps) in addition to the safety (i.e., correctness) property.

Bagrodia algorithm for binary rendezvous

Assumptions

- Receives are always enabled
- Send,once enabled ,remains enabled.
- To break ,deadlock ,PIDs used to introduce asymmetry
- Each process schedules one send at a time.

Message Types :M, ack (M),request (M),permission (M),

Process blocks when it knows it can be successfully synchronize the current message.

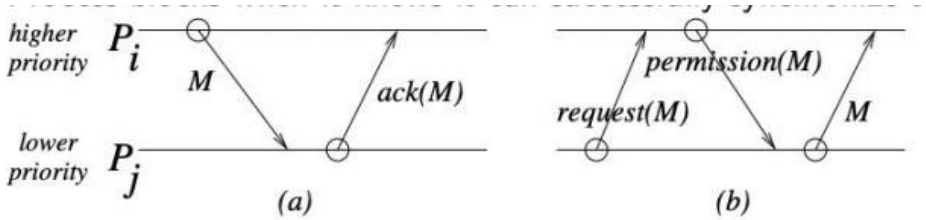


Fig 2.5 Rules to prevent message cycles
a) High priority process blocks b) Low priority process do not block

Algorithm in Simple Steps

- To send to a lower priority process

  a. The sender issues send(M)

  b. blocks until ack(M) arrives

  c. Thus, when sending to a lower priority process, the sender blocks waiting for the partner process to synchronize and send an acknowledgement.

- To send to a higher priority process

  a. The sender issues send(request(M)), does not block, and awaits permission.

  b. When permission(M) arrives, the sender issues send(M).

(1) Pi wants to execute SEND(M) to a lower priority process

Pj: Pi executes send(M) and blocks until it receives ack(M) from Pj . The send event SEND(M) now completes.

Any M' message (from a higher priority processes) and request(M') request for synchronization (from a lower priority processes) received during the blocking period are queued.

(2) Pi wants to execute SEND(M) to a higher priority process

Pj: (2a) Pi seeks permission from Pj by executing send(request(M)).

// to avoid deadlock in which cyclically blocked processes queue // messages. (2b) While Pi is waiting for permission, it remains unblocked.

(i) If a message M' arrives from a higher priority process Pk, Pi accepts M' by scheduling a RECEIVE(M') event and then executes send(ack(M')) to Pk.

(ii) If a request(M') arrives from a lower priority process Pk, Pi executes send(permission(M')) to Pk and blocks waiting for the messageM'. WhenM' arrives, the RECEIVE(M') event is executed.

(2c) When the permission(M) arrives, Pi knows partner Pj is synchronized and Pi executes send(M). The SEND(M) now completes.

(3) request(M) arrival at Pi from a lower priority process Pj:

At the time a request(M) is processed by Pi, process Pi executes send(permission(M)) to Pj and blocks waiting for the message M. When M arrives, the RECEIVE(M) event is executed and the process unblocks.

(4) Message M arrival at Pi from a higher priority process Pj:

At the time a message M is processed by Pi, process Pi executes RECEIVE(M) (which is assumed to be always enabled) and then send(ack(M)) to Pj .

(5) Processing when Pi is unblocked:

When Pi is unblocked, it dequeues the next (if any) message from the queue and processes it as a message arrival (as per rules 3 or 4).

7.Explain the Group communication and Causal order (CO) (or) Explain various ways of ordering messages in group communication (or) Also illustrate distributed algorithm to implement casual order of messages. April/May 2021,2022,2024

GROUP COMMUNICATION

- Processes across a distributed system cooperate to solve a joint task. Often, they need to communicate with each other as a group, and therefore there needs to be support for group communication.
- A message broadcast is the sending of a message to all members in the distributed system.
- If a multicast algorithm requires the sender to be a part of the destination group, the multicast algorithm is said to be a closed group algorithm.
- If the sender of the multicast can be outside the destination group, the multicast algorithm is said to be an open group algorithm in below Fig 2.6

CAUSAL ORDER (CO)

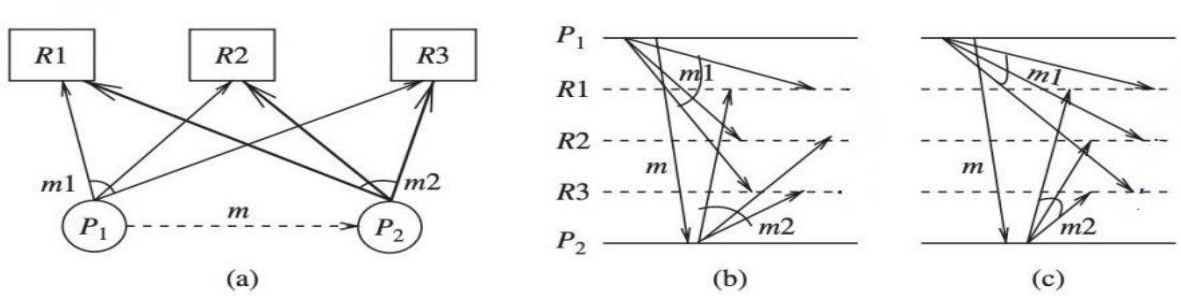


Fig 2.6 casual order

Criteria must be met by a causal ordering protocol

- **Safety**: In order to prevent causal order from being violated, a message M that arrives at a process may need to be buffered until all system wide messages sent in the causal past of the send (M) event to that same destination have already arrived. The arrival of a message is transparent to the application process. The delivery event corresponds to the receive event in the execution model .
- **Liveness**: A message that arrives at a process must eventually be delivered to the process.

The Raynal–Schiper–Toueg algorithm

(local variables)
array of int $SENT[1 \ldots n, 1 \ldots n]$
array of int $DELIV[1 \ldots n]$ // $DELIV[k]$ = # messages sent by k that are delivered locally

(1) **send event**, where P_i wants to send message M to P_j:
(1a) **send** $(M, SENT)$ to P_j;
(1b) $SENT[i, j] \longleftarrow SENT[i, j] + 1$.

(2) **message arrival,** when (M, ST) arrives at P_i from P_j:
(2a) **deliver** M to P_i when **for each** process x,
(2b) $DELIV[x] \geq ST[x, i]$;
(2c) $\forall x, y$, $SENT[x, y] \longleftarrow \max(SENT[x, y], ST[x, y])$;
(2d) $DELIV[j] \longleftarrow DELIV[j] + 1$.

8. Explain in detail about Total and casual Message Order (or) Also illustrate distributed algorithm to implement total order of messages. Nov/Dec 2022 / April/May 2024

For each pair of processes Pi and Pj and for each pair of messages Mx and My that are delivered to both the processes, Pi is delivered Mx before My if and only if Pj is delivered Mx before My.

Centralized Algorithm for total ordering

Each process sends the message it wants to broadcast to a centralized process, which relays all the messages it receives to every other process over FIFO channels.

(1) When process Pi wants to multicasts a message M to group G:(1a) send M(i, G) to central coordinator

(2) When M(i, G) arrives from Pi at the central coordinator(2a) send M(i, G) to all members of the group G.

(3) When M(i, G) arrives at pj from the central coordinator(3a) deliver M(i, G) to the application.

Sender side

Phase 1

- In the first phase, a process multicasts the message M with a locally unique tag and the local timestamp to the group members.

Phase 2

- The sender process awaits a reply from all the group members who respond with tentative proposal for a revised timestamp for that message M.
- The await call is non-blocking.

Phase 3

- The process multicasts the final timestamp to the group.

Receiver Side

Phase 1

- The receiver receives the message with a tentative timestamp. It updates the variable priority that tracks the highest proposed timestamp, then revises the proposed timestamp to the priority, and places the message with its tag and the revised timestamp at the tail of the queue temp_Q. In the queue, the entry is marked as undeliverable.

Phase 2

- The receiver sends the revised timestamp back to the sender. The receiver then waits in a non-blocking manner for the final timestamp.

Phase 3

- The final timestamp is received from the multicaster. The corresponding message entry in temp_Q is identified using the tag, and is marked as deliverable after the revised timestamp is overwritten by the final timestamp.

- The queue is then resorted using the timestamp field of the entries as the key. As the queue is already sorted except for the modified entry for the message under consideration, that message entry has to be placed in its sorted position in the queue.

- If the message entry is at the head of the temp_Q, that entry, and all consecutive subsequent entries that are also marked as deliverable, are dequeued from temp_Q, and enqueued in deliver_Q.

Complexity This algorithm uses three phases, and, to send a message to $n - 1$ processes, it uses $3(n - 1)$ messages and incurs a delay of three message hops.

9.Explain the Global state and snapshot recording Algorithms.

System Model

- The system consists of a collection of n processes P1, P2, ... Pn that are connected by channels.
- There are no globally shared memory and physical global clock and processes communicate by passing messages through communication channels.
- Cij denotes the channel from process p; to process p; and its state is denoted by SCij.

• The actions performed by a process are modeled as three types of events: Internal events, the message send event and the message receive event.

• For a message mij that is sent by process p; to process pj, let send(mij) and rec(mij) denote its send and receive events.

• At any instant, the state of process p;, denoted by (LSi, is a result of the sequence of all the events executed by p; till that instant.

• For an event e and a process state LS;, eELS; iff e belongs to the sequence of events that have taken process p; to state LS;.

• For an event e and a process state LS;, e LS; iff e does not belong to the sequence of events that have taken process p; to state LS;.

• For a channel Cij, the following set of messages can be defined based on the local states of the processes pi and pj;

Models of Communication

Recall, there are three models of communication: FIFO, non-FIFO, and Co.

• In FIFO model, each channel acts as a first-in first-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.

• A system that supports causal delivery of messages satisfies the following property: "For any two messages mij and mkj ,if send(mij) → send(mkj) then rec(mij) → rec(mkj)".

Consistent Global state

• The global state of a distributed system is a collection of the local states of the processes and the channels.

• Notationally, global state GS is defined as,

GS ={Ui LSi, U i,j SCij }

• A global state GS is a consistent global state iff it satisfies the following two conditions:

C1: send(mi) ELS; mij ESC; rec(mi;)ELS;.(is Ex-OR operator.)

C2: send(mi;)LS; ⇒ mij SC;; A rec(mij)LS;.

Key Points

• 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.

• A consistent global state corresponds to a cut in which every message received in the PAST of the cut was sent in the PAST of that cut. Such a cut is known as a consistent cut.

• For example, consider the space-time diagram for the computation illustrated in Figure 4.1.

- Cut C1 is inconsistent because message m1 is flowing from the FUTURE to the PAST.

• Cut C2 is consistent and message m4 must be captured in the state of channel C21.

Issues in recording a global state

The following two issues need to be addressed:

11: How to distinguish between the messages to be recorded in the snapshot from those not to be recorded.

-Any message that is sent by a process before recording its snapshot, must be recorded in the global snapshot (from C1). -Any message that is sent by a process after recording its snapshot, must not be recorded in the global snapshot (from C2).

12: How to determine the instant when a process takes its snapshot.

-A process p; must record its snapshot before processing a message mi; that was sent by process pi after recording its snapshot.

10. Explain in detail about Snapshot algorithms for FIFO channels (or) Write and describe the snapshot algorithm for FIFO channels (or) What is snapshot and what are the needs of taking snapshot in distributed systems? (or) outline chandy- Lamport algorithm for snapshot and illustrate the algorithm April/May2021,2022 ,NOV/DEC 2023 April/May2024

Chandy Lamport Algorithm –signal

Marker Sending Rule for process i

Process i records its state.

For each outgoing channel C on which a marker has not been sent, i sends a marker along C before i sends further messages along C.

Marker Receiving Rule for process j

On receiving a marker along channel C:
if j has not recorded its state then
Record the state of C as the empty set
Follow the "Marker Sending Rule"
else
Record the state of C as the set of messages received along
C after j's state was recorded and before j received the marker along C

Key Points

- 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.
- The algorithm can be initiated by any process by executing the "Marker Sending Rule" by which it records its local state and sends a marker on each outgoing channel.
- A process executes the "Marker Receiving Rule" on receiving a marker. If the process has not yet recorded its local state, it records the state of the channel on which the marker is received as empty and executes the "Marker Sending Rule" to record its local state.
- The algorithm terminates after each process has received a marker on all of its incoming channels. All the local snapshots get disseminated to all other processes and all the processes can determine the global state in below Fig 2.7

Correctness and Complexity

Correctness

• Due to FIFO property of channels, it follows that no message sent after the marker on that channel is recorded in the channel state. Thus, condition C2 is satisfied.

• When a process p; receives message m;; that precedes the marker on channel Cij, it acts as follows: If process p; has not taken its snapshot yet, then it includes mij in its recorded snapshot. Otherwise, it records mi; in the state of the channel Cij. Thus, condition C1 is satisfied.

Complexity

The recording part of a single instance of the algorithm requires O(e) messages and O(d) time, where e is the number of edges in the network and d is the diameter of the network.

Properties of Recorded Global State

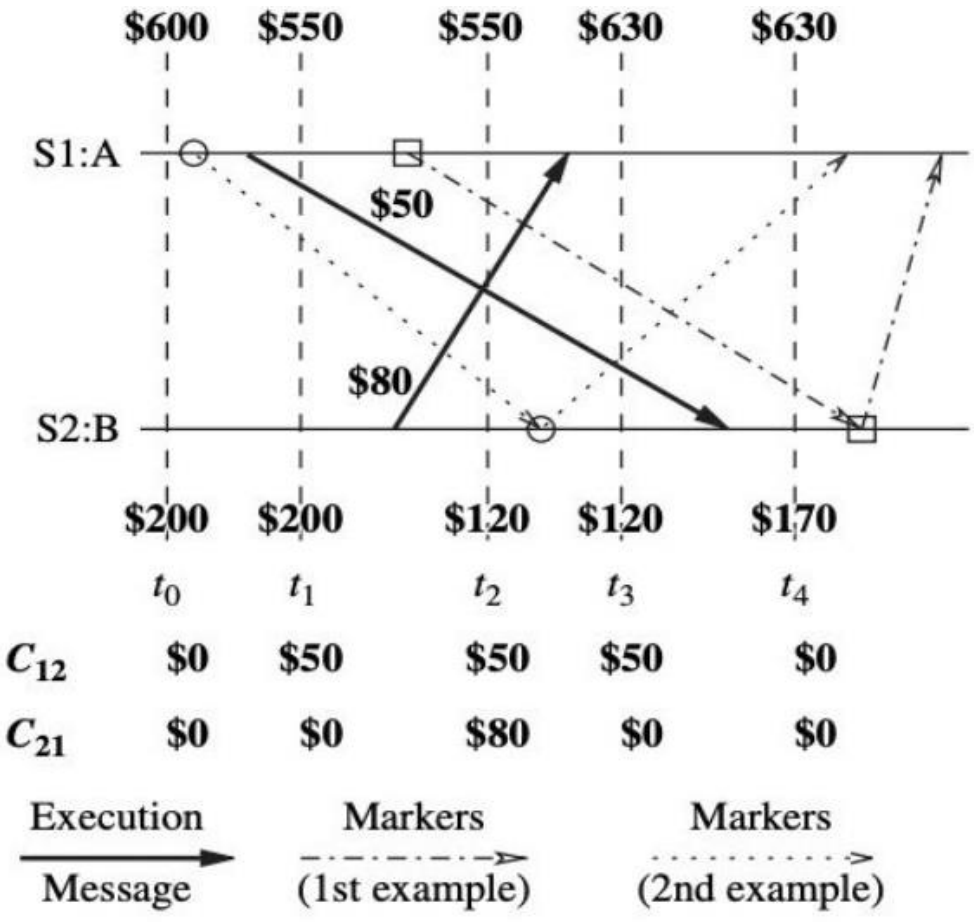


Fig 2.7 Recorded Global State

Explanation of Example

1. (Markers shown using dashed-and-dotted arrows.) Let site S1 initiate the algorithm just after t_1. Site S1 records its local state (account A = \$550) and sends a marker to site S2. The marker is received by site S2 after t_4. When site S2 receives the marker, it records its local state (account B = \$170), the state of channel C_{12} as \$0, and sends a marker along channel C_{21}. When site S1 receives this marker, it records the state of channel C_{21} as \$80. The \$800 amount in the system is conserved in the recorded global state,

$$A = \$550, B = \$170, C_{12} = \$0, C_{21} = \$80.$$

2. (Markers shown using dotted arrows.) Let site S1 initiate the algorithm just after t_0 and before sending the \$50 for S2. Site S1 records its local state (account A = \$600) and sends a marker to site S2. The marker is received by site S2 between t_2 and t_3. When site S2 receives the marker, it records its local state (account B = \$120), the state of channel C_{12} as \$0, and sends a marker along channel C_{21}. When site S1 receives this marker, it records the state of channel C_{21} as \$80. The \$800 amount in the system is conserved in the recorded global state,

$$A = \$600, B = \$120, C_{12} = \$0, C_{21} = \$80.$$

Properties of Recorded Global State

- The recorded global state may not correspond to any of the global states that occurred during the computation.
- This happens because a process can change its state asynchronously before the markers it sent are received by other sites and the other sites record their states.
  - ➢ But the system could have passed through the recorded global states in some equivalent executions.
  - ➢ The recorded global states is a valid state in an equivalent executions and if a stable property holds in the system before the snapshot algorithm begins, it hold in the recorded global snapshot.
  - ➢ Therefore, recorded global state is useful in detecting stable properties.

11. Give a real time scenario where FIFO message queue is used. NOV/DEC 2023

Here are some real-time scenarios where FIFO (First-In-First-Out) message queues are used:

1. Order processing: In e-commerce, FIFO queues are used to process orders in the order they are received, ensuring that orders are fulfilled in a timely and fair manner.

2. Job scheduling: In cloud computing, FIFO queues are used to schedule jobs (e.g., video encoding, data processing) in the order they are received, ensuring that jobs are executed in a fair and efficient manner.

3. Message processing: In messaging platforms (e.g., SMS, chat apps), FIFO queues are used to process messages in the order they are received, ensuring that messages are delivered in a timely and sequential manner.

4. Print job processing: In printing systems, FIFO queues are used to process print jobs in the order they are received, ensuring that documents are printed in the order they were submitted.

5. Bank transaction processing: In banking systems, FIFO queues are used to process transactions (e.g., deposits, withdrawals) in the order they are received, ensuring that transactions are processed in a timely and secure manner.

6. Network packet processing: In network routers, FIFO queues are used to process packets in the order they are received, ensuring that packets are forwarded in a timely and efficient manner.

7. Customer support ticketing: In helpdesk systems, FIFO queues are used to process customer support tickets in the order they are received, ensuring that tickets are addressed in a timely and fair manner.

8. Supply chain management: In supply chain management systems, FIFO queues are used to process orders and shipments in the order they are received, ensuring that goods are delivered in a timely and efficient manner.

These scenarios illustrate how FIFO message queues are used to ensure that tasks or messages are processed in a fair, timely, and efficient manner, maintaining the order in which they were received.

UNIT III DISTRIBUTED MUTEX & DEADLOCK

Distributed mutual exclusion algorithms: Introduction – Preliminaries – Lamport's algorithm – Ricart-Agrawala 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.

PART A

1. **What is Mutual exclusion? Nov/Dec2022**
   - ✓ **Mutual exclusion** is a fundamental problem in distributed computing systems.
   - ✓ Mutual exclusion ensures that concurrent access of processes to a shared resource or data is serialized, that is, executed in a mutually exclusive manner.
   - ✓ Mutual exclusion in a distributed system states that only one process is allowed to execute the critical section (CS) at any given time.
2. **What are the basic approaches for implementing Distributed Mutual Exclusion?**

   There are three basic approaches for implementing distributed mutual exclusion:
   1. Token-based approach.
   2. Non-token-based approach.
   3. Quorum-based approach.

3. **Define Token based approach.**
   - ✓ In the token-based approach, a unique token (also known as the **PRIVILEGE message**) is shared among the sites.
   - ✓ A site is allowed to enter its (Critical section) CS if it possesses the token and it continues to hold the token until the execution of the CS is over.
   - ✓ Mutual exclusion is ensured because the token is unique.

4. **Define Non-token based approach.**
   - ✓ In Non-Token based algorithm, there is no token even not any concept of sharing token for access.
   - ✓ Here, two or more successive rounds of messages are exchanged between sites to determine which site is to enter the Critical Section next.

5. **What are the Requirements of mutual exclusion algorithms? (or) What are the properties of mutual exclusion algorithms?**

   A mutual exclusion algorithm should satisfy the following properties:
   1. **Safety property**
      The safety property states that at any instant, only one process can execute the critical section. This is an essential property of a mutual exclusion algorithm.
   2. **Liveness property**
      This property states the absence of deadlock and starvation. Two or more sites should not endlessly wait for messages that will never arrive

6. **What is the performance of mutual exclusion algorithms?**
   - ✓ Message complexity
   - ✓ Synchronization delay
   - ✓ Response time
   - ✓ System throughput

7. Define Lamport's algorithm.

Lamport's Distributed Mutual Exclusion Algorithm is a permission based algorithm proposed by Lamport as an illustration of his synchronization scheme for distributed systems. In permission based timestamp is used to order critical section requests and to resolve any conflict between requests.

8. What is the performance of Lamport's algorithm?

For each CS execution, Lamport's algorithm requires

- (N −1) REQUEST messages,
- (N −1) REPLY messages, and
- (N −1) RELEASE messages.
- Thus, Lamport's algorithm requires **3(N −1)** messages per CS invocation. The synchronization delay in the algorithm is T.

9. Define Ricart–Agrawala algorithm.

- ✓ The Ricart-Agrawala algorithm is **an algorithm** for mutual exclusion on a distributed system.
- ✓ This algorithm is an extension and optimization of Lamport's Distributed Mutual Exclusion Algorithm, by removing the need for ack messages.
- ✓ The Ricart–Agrawala algorithm assumes that the communication channels are FIFO.
- ✓ The algorithm uses **two types of messages**: REQUEST and REPLY.

10. What is the Ricart–Agrawala algorithm Message Complexity? April/May2022

Message Complexity

Ricart–Agrawala algorithm requires invocation of 2(N – 1) messages per critical section execution. These 2(N – 1) messages involves

- ✓ (N – 1) request messages
- ✓ (N – 1) reply messages

11. What is the Performance of Ricart–Agrawala algorithm?

Performance

- ✓ For each CS execution, the Ricart–Agrawala algorithm requires
- ✓ (N – 1) REQUEST messages and
- ✓ (N – 1) REPLY messages.
- ✓ Thus, it requires 2(N −1) messages per CS execution. The synchronization delay in the algorithm is T.

12. Define quorum.

A site does not request permission from every other site but from a subset of sites which is called **quorum**.

13. Define Maekawa's algorithm.

Maekawa's algorithm is an algorithm for mutual exclusion on a distributed system. The basis of this algorithm is a quorum-like approach where any one site needs only to seek permissions from a subset of other sites.

14. What is the Message Complexity of Maekawa's algorithm?

MessageComplexity:

Maekawa's Algorithm requires invocation of 3√N messages per critical section execution as the size of a request set is √N. These 3√N messages involve.

- √N request messages
- √N reply messages
- √N release messages

15. What are the Drawbacks of Maekawa's Algorithm?

This algorithm is deadlock prone because a site is exclusively locked by other sites and requests are not prioritized by their timestamp.

16. What is the performance of Maekawa's Algorithm?

The size of a request set is √N. Therefore, an execution of the CS requires

- ✓ √N REQUEST,
- ✓ √N REPLY, and
- ✓ √N RELEASE messages, resulting in 3√N messages per CS execution.
- ✓ Synchronization delay in this algorithm is 2T.

17. What is the Suzuki–Kasami's broadcast algorithm?

The Suzuki–Kasami algorithm is a token-based algorithm for achieving mutual exclusion in distributed systems. The process holding the token is the only process able to enter its critical section.

18. What are the design issues of Suzuki–Kasami's broadcast algorithm?

- How to distinguishing an outdated REQUEST message from a current REQUEST message
- How to determine which site has an outstanding request for the CS

19. Define performance of Suzuki–Kasami's broadcast algorithm.

- Synchronization delay is 0 and no message is needed if the site holds the idle token at the time of its request.
- In case site does not holds the idle token, the maximum synchronization delay is equal to maximum message transmission time and a maximum of N message is required per critical section invocation.

20. What are the Drawbacks of Suzuki–Kasami Algorithm?

Non-symmetric Algorithm: A site retains the token even if it does not have requested for critical section. According to definition of symmetric algorithm "No site possesses the right to access its critical section when it has not been requested."

21. Define Message Complexity of Suzuki–Kasami Algorithm.

The algorithm requires 0 message invocation if the site already holds the idle token at the time of critical section request or maximum of N message per critical section execution. This N messages involves

- (N – 1) request messages
- 1 reply message.

22. Define Deadlock. Nov/Dec 2022

A Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource occupied by some other process.

23. Define Wait-for graph. April/May 2021

- ✓ In distributed systems, the state of the system can be modeled by directed graph, called **a *wait-for graph* (WFG).**
- ✓ In a WFG, nodes are processes and there is a directed edge from node P1 to mode P_2 if P_1 is blocked and is waiting for P_2 to release some resource.

24. What are the Strategies of Deadlock Handling?

There are three strategies for handling deadlocks

1. Deadlock prevention,
2. Deadlock avoidance, and
3. Deadlock detection.

25. What are the Issues in deadlock detection?

Deadlock handling using the approach of deadlock detection entails addressing two basic issues:

1. Detection of existing deadlocks and,
2. Resolution of detected deadlocks.

26. Define Correctness criteria of Deadlock Handling. (or) what are the condition satisfied to deadlock detection algorithm? April/May2022.

A deadlock detection algorithm must satisfy the following **two conditions**:

- ✓ Progress (no undetected deadlocks)
- ✓ Safety (no false deadlocks)

27. What are the Models of deadlocks? April/May 2021

Models of deadlocks

1. The single-resource model
2. The AND model
3. The OR model
4. The AND-OR model
5. The $\binom{p}{q}$ Model
6. Unrestricted model

28. Define Knapp's classification of distributed deadlock detection algorithms.(or) what is Distributed deadlock detection algorithms?

Distributed deadlock detection algorithms can be divided into four classes:

1. Path-pushing
2. Edge-chasing
3. Diffusion computation
4. Global state detection.

29. What are Path-pushing algorithms?

Path-pushing algorithms

- In path-pushing algorithms, distributed deadlocks are detected by maintaining an **explicit global WFG**.
- The basic idea is to build a global WFG for each site of the distributed system.
- In this class of algorithm, whenever deadlock computation is performed, each site sends its local WFG to all the neighboring sites.
- This feature of sending around the paths of the global WFG has led to the term path-pushing algorithms.

30. Define edge-chasing algorithm.

- ✓ In an edge-chasing algorithm, the presence of a cycle in a distributed graph structure is verified by propagating special messages called probes along the edges of the graph.
- ✓ These probe messages are different to the request and reply messages.
- ✓ The main advantage of edge-chasing algorithms is that **probes are fixed size** messages that are normally **very short**.

31. Define diffusing computation-based algorithms.

- In *diffusion computation*-based distributed deadlock detection algorithms, m deadlock detection computation is diffused through the WFG of the system.
- These algorithms make use of echo algorithms to detect deadlocks.

32. Define private label and public label.

- Each node of the WFG has two local variables, called **labels**:
  - private label
  - public label
- **A private label**, which is unique to the node at all times, though it is not constant, and **a public label**, which can be read by other processes and which may not be unique.

33. What are the phases of Mitchell and Merritt's algorithm for the single-resource model? (or) What are the phases of single-resource model?

This algorithm has **two phases**.

- The **first phase** is almost identical to the algorithm.
- In the **second phase** the smallest priority is propagated around the circle.

34. What is Chandy–Misra–Haas algorithm for the AND model?

- Chandy–Misra–Haas's distributed deadlock detection algorithm for the AND model, which is based on **edge-chasing**.
- The algorithm uses a special message called ***probe***, which is a triplet (i, j, k), denoting that it belongs to a deadlock detection initiated for process P_i and it is being sent by the home site of process P_j to the home site of process P_k.

35. Define data structure of Chandy–Misra–Haas algorithm for the AND model.

- ✓ Each process P_i maintains a boolean array, $dependent_i$, where $dependent_i(j)$ is true only if P_i knows that P_j is dependent on it.
- ✓ Initially, $dependent_i(j)$ is false for all i and j.

36. What is the performance analysis of Chandy–Misra–Haas algorithm for the AND model?
Performance analysis

- The algorithm exchanges at most m(n−1)/2 messages to detect a deadlock that involves m processes and spans over n sites.
- The size of messages is fixed and is very small (only three integer words). The delay in detecting a deadlock is O(n).

37. What is Chandy–Misra–Haas algorithm for the OR model

- Chandy–Misra–Haas's distributed deadlock detection algorithm for the OR model , which 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)**

38. What are the performance analysis of Chandy–Misra–Haas algorithm for the "OR" model?
Performance analysis

For every deadlock detection, the algorithm exchanges **e *query* messages** and**e *reply* messages,** where e=n(n-1) is the number of edges.

39. What are the three types of messages in Deadlock handling?

Deadlock handling requires the following **three types of messages:**

- ✓ **FAILED** A FAILED message from site Si to site Sj indicates that Si cannot grant Sj's request because it has currently granted permission to a site with a higher priority request.
- ✓ **INQUIRE** An INQUIRE message from Si to Sj indicates that Si would like to find out from Sj if it has succeeded in locking all the sites in its request set.
- ✓ **YIELD** A YIELD message from site Si to Sj indicates that Si is returning the permission to Sj (to yield to a higher priority request at Sj).

40. What is the Assumption Deadlock detection in distributed Systems?

- ✓ The systems have only reusable resources.
- ✓ Processes are allowed to make only exclusive access to resources.
- ✓ There is only one copy of each resource.

41.what are the various deadlock detection in distributed systems? state a common factor in detecting the deadlock. NOV/DEC 2023

Centralized Approach: A single resource is responsible for detecting deadlocks.

Distributed Approach: Various nodes work together to detect deadlocks.

Hierarchical Approach: This approach combines the centralized and distributed approaches, offering the benefits of both.

A common factor in detecting deadlocks is the need to detect all deadlocks in the system while avoiding the detection of false or phantom deadlocks

41.How beneficial is the Chandy-Misra-Haas algorithm in the AND model and OR model? NOV/DEC 2023

AND Model:

- Efficient detection: The algorithm can detect deadlocks efficiently in the AND model, where a process waits for multiple resources.

- Reduced false positives: The algorithm reduces the number of false positive detections, which can lead to unnecessary process termination or resource preemption.

OR Model:

- Flexibility: The algorithm can handle the OR model, where a process waits for any one of multiple resources, making it more flexible than other deadlock detection algorithms.

- Scalability: The algorithm can handle large distributed systems with many processes and resources, making it a good choice for complex systems.

42. List the advantages and limitations of token based algorithm for implementing mutual exclusion. APR/MAY 2024

Advantages of Token-Based Algorithm for Mutual Exclusion:

1. Efficient: Token-based algorithms are efficient in terms of message passing, as only a single token is passed around the nodes.

2. Simple to implement: The algorithm is relatively simple to implement, as it only requires a token and a simple protocol for passing it.

3. Low overhead: The algorithm has low overhead in terms of memory and computation, making it suitable for distributed systems.

Limitations of Token-Based Algorithm for Mutual Exclusion:

1. Single point of failure: If the node holding the token fails, the entire system comes to a halt.

2. Token loss: If the token is lost or corrupted, the system must be restarted.

3. Limited scalability: As the number of nodes increases, the token passing overhead increases, leading to decreased performance.

43. what is false deadlock and when does it occurs? APR/MAY 2024

False deadlocks can occur in the following scenarios:

1. Transient cycles: Temporary cycles in the resource allocation graph that disappear quickly, but are misinterpreted as deadlocks by the detection algorithm.

2. Race conditions: Concurrent events that lead to temporary inconsistencies in the resource allocation graph, causing the algorithm to mistakenly detect a deadlock.

3. Incomplete information: The detection algorithm may not have access to complete information about the system's state, leading to incorrect conclusions about deadlocks.

PART B

1. Explain the Distributed mutual exclusion algorithms. April/May 2021

Requirements of Mutual Exclusion Algorithm:

1.Safety Property

The safety property states that at any instant, only one process can execute the critical section. This is an essential property of mutual exclusion algorithm.

2.Liveness Property

This property states the absence of deadlock and starvation Two or more sites should not endlessly wait for message that will never arrive. In addition, a site must not wait indefinitely to execute the CS while other sites are repeatedly executing the CS in finite time.

3.Fairness

Fairness in the context of mutual exclusion means that each process gets a fair chance to execute the CS. In mutual exclusion algorithm, the fairness property generally means that the CS execution requests are executed in order of their arrival in the system .

There are three fundamental approaches for executing distributed mutual exclusion:

1. Token Based Algorithm:

- In the token-based approach, a unique token (also known as the PRIVILEGE message) is shared among the sites.
- A site is allowed to enter its CS if it possesses the token and it continues to hold the token until the execution of the CS is over.
- Mutual exclusion is ensured because the token is unique.

Example : Suzuki–Kasami Algorithm

2. Non-token based approach:

- two or more successive rounds of messages are exchanged among the sites to determine which site will enter the CS next.
- A site enters the critical section (CS) when an assertion, defined on its local variables, becomes true.
- Mutual exclusion is enforced because the assertion becomes true only at one site at any given time.

Example : Ricart–Agrawala Algorithm

3. Quorum based approach:

- Each site requests permission to execute the CS from a subset of sites (called a quorum).
- The quorums are formed in such a way that when two sites concurrently request access to the CS, at least one site receives both the requests and this site is responsible to make sure that only one request executes the CS at any time in below Fig 3.1 & 3.1.2

Example : Maekawa's Algorithm

Performance Metrics

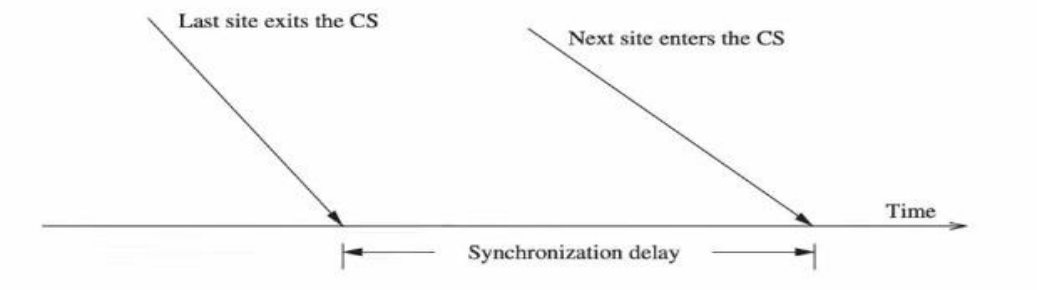


Fig 3.1 Synchronization delay

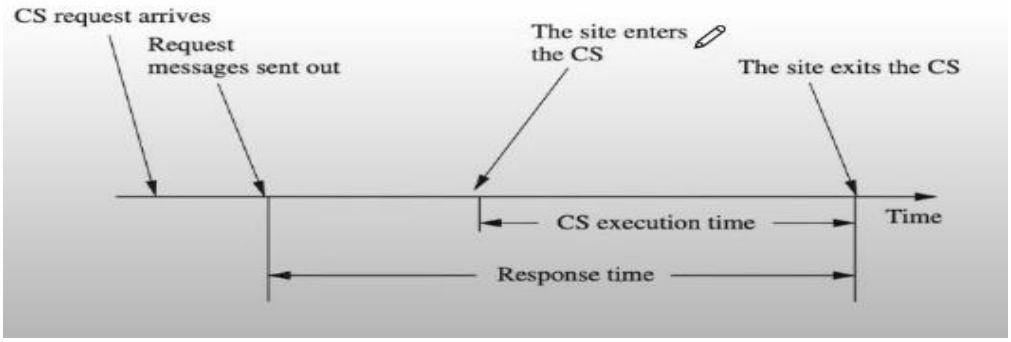


Fig 3.1.2 Response Time

The performance of mutual exclusion algorithm is generally measured by the following four metrics

- **Message complexity** : This is the number of messages that are required per CS execution by a site.

- **Synchronization delay**: After a site leaves the CS ,it is the time required and before the next site enters the CS .Note that normally one or more sequential message exchanges may be required after a site exist the Cs and before the next site can enter the CS
- **Response time** : This is the time interval a request wait for its CS execution to be over after its request message have been sent out .Thus, response time does not include the time a request waits at a site before its request message have been sent out .
- **System throughput** ; This is the rate at which the system executes request for the CS .

Low and High Load Performance

- Under low load conditions, there is rarely more than one request for the critical section present in the system simultaneously.
- Under heavy load conditions, there is always a pending request for critical section at a site. Thus, in heavy load conditions, after having executed a request, a site immediately initiates activities to execute its next CS request.
- A site is seldom in the idle state in heavy load conditions

2. Explain in detail about Lamport's algorithm? (Or) State the Lamport's algorithm and its use and also the limitations and benefits. Compare this with any two token nased algorithms. NOV/DEC 2023

Lamport's Distributed Mutual Exclusion Algorithm is a permission based algorithm proposed by Lamport as an illustration of his synchronization scheme for distributed systems. In permission based timestamp is used to order critical section requests and to resolve any conflict between requests in below Fig 3.2.2 & 3.2.3

Requesting the Critical section:

- When a site S_i wants to enter the critical section, it sends a request message **Request(ts_i, i)** to all other sites and places the request on **request_queue$_i$**. Here, Ts_i denotes the timestamp of Site S_i
- When a site S_j receives the request message **REQUEST(ts_i, i)** from site S_i, it returns a timestamped REPLY message to site S_i and places the request of site S_i on **request_queue$_j$**

Executing the critical section:

Site S_i enters the critical section when the following two conditions hold :

L1: S_i has received a message than with larger time stamp **(ts_i, i)** from all other sites

L2: S_i is request is at the top of **$request\_queue_i$**

Releasing the critical section:

- When a site S_i exits the critical section, it removes its own request from the top of its request queue and sends a timestamped **RELEASE** message to all other sites
- When a site S_j receives the timestamped **RELEASE** message from site S_i, it removes the request of S_i from its request queue

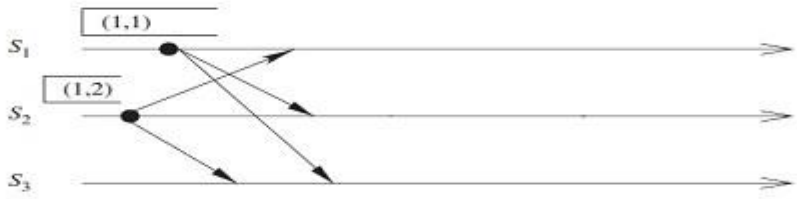


Figure3.2.1 Sites S_1 and S_2 are Making Requests for the CS.

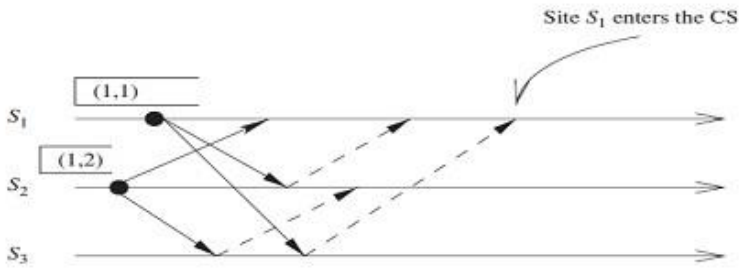


Figure3.2.2 Site S_1 enters the CS.

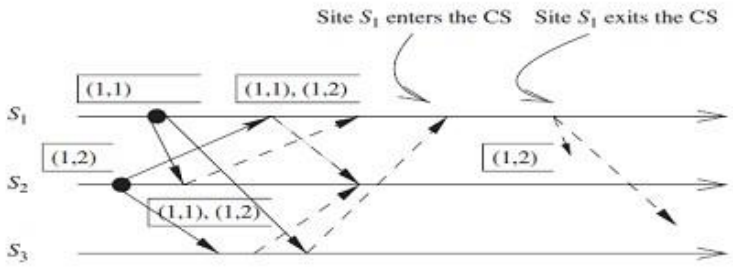


Figure3.2.3 Site S_1 exits the CS and sends RELEASE messages.

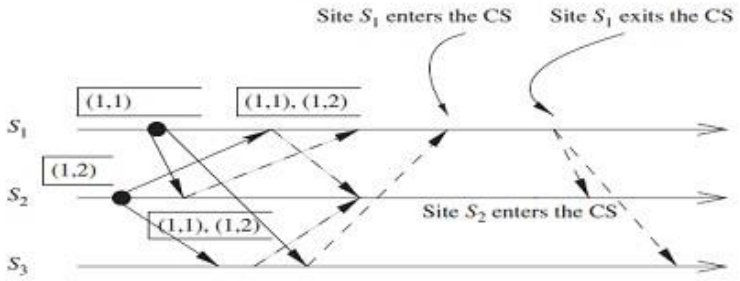


Figure3.2.4 Site S_2 enters the CS.

Key Points

- The algorithm is fair in the sense that a request for CS are executed in the order of their timestamps and time is determined by logical clocks in above Fig 3.2.3 & 3.2.4
- When a site processes a request for the CS, it updates its local clock and assigns the request a timestamp.
- The algorithm executes CS requests in the increasing order of timestamps.
- Every site Si keeps a queue, request_queue i , which contains mutual exclusion requests ordered by their timestamps. (Note that this queue is different from the queue that contains local requests for CS execution awaiting their turn.)
- This algorithm requires communication channels to deliver messages in FIFO order.
- When a site removes a request from its request queue, its own request may come at the top of the queue, enabling it to enter the CS. Clearly, when a site receives a REQUEST, REPLY, or RELEASE message, it updates its clock using the timestamp in the message

Use:

Lamport's algorithm is used in distributed systems where multiple processes need to access a shared resource, such as a file or a printer.

Limitations:

1. High message overhead: Each process sends messages to all other processes, leading to high communication costs.
2. Starvation: A process may be denied access to the resource indefinitely if other processes have smaller RTs.

Benefits:

1. Simple to implement
2. Works in a distributed environment without a central coordinator

Comparison with Token-Based Algorithms:

Let's compare Lamport's algorithm with two token-based algorithms: Suzuki-Kasami's algorithm and Raymond's algorithm.

Suzuki-Kasami's Algorithm:

1. A token circulates among processes, granting access to the shared resource.
2. Each process maintains a request queue.
3. When a process receives the token, it checks its request queue and grants access to the resource if it has a pending request.

Raymond's Algorithm:

1. A token circulates among processes, granting access to the shared resource.
2. Each process maintains a request timestamp.
3. When a process receives the token, it grants access to the resource if its request timestamp is the smallest.

3. Explain the Ricart–Agrawala algorithm with example (or)Outline Ricart and Agrawala's algorithm to detect deadlock in OR model and illustrate the algorithm with an example Nov/Dec 2022, April/May 2024

Ricart–Agrawala algorithm is an algorithm for mutual exclusion in a distributed system proposed by Glenn Ricart and Ashok Agrawala. This algorithm is an extension and optimization of Lamport's Distributed Mutual Exclusion Algorithm. Like Lamport's Algorithm, it also follows permission-based approach to ensure mutual exclusion.

Algorithm:

Requesting the Critical section:

a) When a site S_i wants to enter the critical section, it broadcast a timestamped **REQUEST** message to all other sites.

b) When site S_j receives a **REQUEST** message from site S_i, It sends a **REPLY** message to site S_i if Site S_j is neither requesting nor executing the critical section or if the Site S_j is requesting, the timestamp of Site S_i's request is smaller than its own request timestamp.

Executing the critical section:

c) Site S_i enters the critical section if it has received the **REPLY** message from every site it sent a REQUEST message to.

Executing the critical section:

d) Upon exiting site S_i sends **REPLY** message to all the deferred requests.

Key Points

- A process sends a REQUEST message to all other processes to request their permission to enter the critical section.
- A process sends a REPLY message to a process to give its permission to that process. Processes use Lamport-style logical clocks to assign a timestamp to critical section requests.
- Timestamps are used to decide the priority of requests in case of conflict – if a process pi that is waiting to execute the critical section receives a REQUEST message from process pj, then if the priority of pj's request is lower, pi defers the REPLY to pj and sends a REPLY message to pj only after executing the CS for its pending request.
- Otherwise, pi sends a REPLY message to pj immediately, provided it is currently not executing the CS.
- Thus, if several processes are requesting execution of the CS, the highest priority request succeeds in collecting all the needed REPLY messages and gets to execute the CS.
- When a site receives a message, it updates its clock using the timestamp in the message.
- Also, when a site takes up a request for the CS for processing, it updates itslocal clock and assigns a timestamp to the request.
- In this algorithm, a site's REPLY messages are blocked only by sites that are requesting the CS with higher priority (i.e., smaller timestamp).
- Thus, when a site sends out deferred REPLY messages, the site with the next highest priority request receives the last needed REPLY message and enters the CS. Execution of the CS requests in this algorithm is always in the order of their timestamps in below Fig 3.3.1 & 3.3.2

Figure 3.3.1 Sites S_1 and S_2 each make a request for the CS.

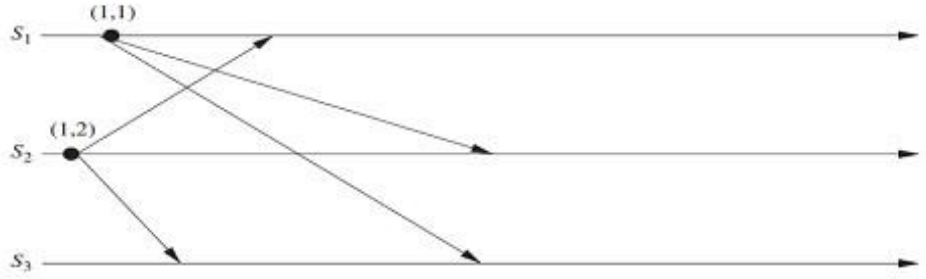


Figure 3.3.2 Site S_1 enters the CS.

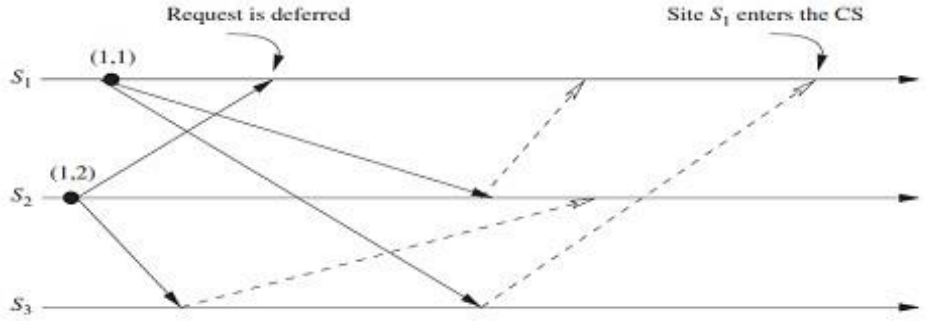


Figure 3.3.3 Site S_1 exits the CS and sends a REPLY message to S_2's deferred request.

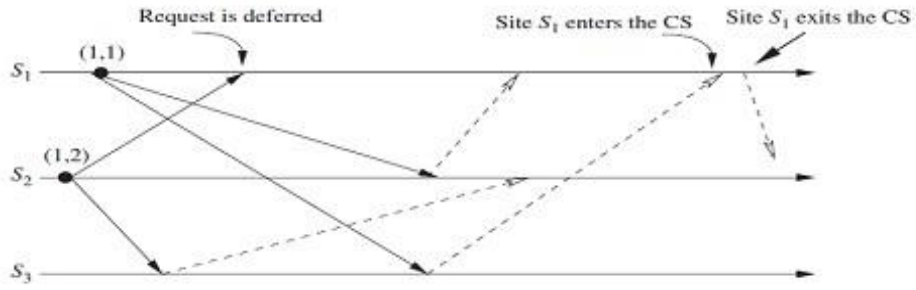
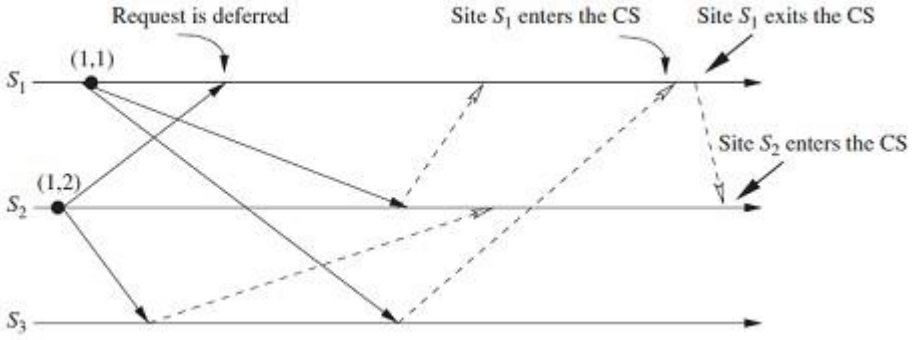


Figure 3.3.4 Site S_2 enters the CS.



Theorem : Algorithm achieves Mutual Exclusion

Proof Proof is by contradiction. Suppose two sites S, and S, are executing the CS concurrently and S,'s request has higher priority (i.e., smaller times- tamp) than the request of S,. Clearly, S; received S,'s request after it has made its own request. (Otherwise, S,'s request will have lower priority.)

Thus, S; can concurrently execute the CS with S, only if S, returns a REPLY to S; (in response to S,'s request) before S; exits the CS. However, this is impossible because S's request has lower priority. Therefore, the Ricart-Agrawala algorithm achieves mutual exclusion in above Fig 3.3.3 & 3.3.4

4. Explain the Suzuki–Kasami's broadcast algorithm? Nov/Dec 2022

Token Based Algorithm

1. a unique token is shared among the sites.
2. A site is allowed to enter its CS if it possesses the token.
3. A site holding the token can enter its CS repeatedly until it sends the token to some other site.
4. First, token-based algorithms use sequence numbers instead of timestamps.

a. Every request for the token contains a sequence number and the sequence numbers of sites advance independently.

b. A site increments its sequence number counter every time it makes a request for the token.

5. algorithm guarantees mutual exclusion because site holds the token during the execution of the CS..

Requesting the Critical section:

- When a site S_i wants to enter the critical section and it does not have the token then it increments its sequence number $\mathbf{RN_i[i]}$ and sends a request message **REQUEST(i, sn)** to all other sites in order to request the token.
  Here **sn** is update value of $\mathbf{RN_i[i]}$
- When a site S_j receives the request message **REQUEST(i, sn)** from site S_i, it sets $\mathbf{RN_j[i]}$ to maximum of $\mathbf{RN_j[i]}$ and **sn** i.e $\mathbf{RN_j[i]} = \max(\mathbf{RN_j[i]}, \mathbf{sn})$.
- After updating $\mathbf{RN_j[i]}$, Site S_j sends the token to site S_i if it has token and $\mathbf{RN_j[i]} = \mathbf{LN[i]} + 1$

Executing the critical section:

- Site S_i executes the critical section if it has acquired the token.

Releasing the critical section

After finishing the execution Site S_i exits the critical section and does following:

- sets $\mathbf{LN[i]} = \mathbf{RN_i[i]}$ to indicate that its critical section request $\mathbf{RN_i[i]}$ has been executed
- For every site S_j, whose ID is not present in the token queue **Q**, it appends its ID to **Q** if $\mathbf{RN_i[j]} = \mathbf{LN[j]} + 1$ to indicate that site S_j has an outstanding request.

- After above updation, if the Queue **Q** is non-empty, it pops a site ID from the **Q** and sends the token to site indicated by popped ID.
- If the queue **Q** is empty, it keeps the token

Problem Explanation

1. How to distinguishing an outdated REQUEST message from a current REQUEST message

a. Due to variable message delays, a site may receive a token request message after the corresponding request has been satisfied.

b. If a site cannot determined if the request corresponding to a token request has been satisfied,

it may dispatch the token to a site that does not need it. This will not violate the correctness,

however, but it may seriously degrade the performance by wasting messages and increasing

the delay at sites that are genuinely requesting the token.

c. Therefore, appropriate mechanisms should implemented to determine if a token request

message is outdated.

2. How to determine which site has an outstanding request for the CS?

a. After a site has finished the execution of the CS, it must determine what sites have an outstanding request for the CS so that the token can be dispatched to one of them.

b. The problem is complicated because when a site Si receives a token request message from a

site Sj, site Sj may have an outstanding request for the CS.

c. However, after the corresponding request for the CS has been satisfied at Sj, an issue is howto inform site Si (and all other sites) efficiently about it.

5. Explain in detail about Deadlock detection in distributed Systems Deadlock and Wait-for graph (WFG) (Or) Explain the system models with its preliminaries and hoo is it characterized in the models of deadlock. April/May 2022 , NOV/DEC 2023

What is deadlock?

1. A deadlock can be defined as a condition where a set of processes request resources that are held by other processes in the set.

2. Handling Deadlocks:

a. **Deadlock prevention** is commonly achieved by either having a process acquire all the needed resources simultaneously before it begins execution or by pre-empting a process that holds the needed resource.

b**. In the deadlock avoidance** approach to distributed systems, a resource is granted to a process if the resulting global system is safe.

c. **Deadlock detection** requires an examination of the status of the process–resources interaction for the presence of a deadlock condition. To resolve the deadlock, we have to abort a deadlocked process

System Model

1. A distributed system consists of a set of processors that are connected by a communication network.

2. The communication delay is finite but unpredictable.

3. A distributed program is composed of a set of n asynchronous processes P1, P2, , Pi, , Pn that communicate by message passing over the communication network.

4. Without loss of generality we assume that each process is running on a different processor.

5. The processors do not share a common global memory and communicate solely by passing messages over the communication network.

6. There is no physical global clock in the system to which processes have instantaneous access.

7. The communication medium may deliver messages out of order, messages may be lost, garbled, or duplicated due to timeout and retransmission, processors may fail, and communication links may go down.

8. The system can be modeled as a directed graph in which vertices represent the processes and edges represent unidirectional communication channels

- assumptions:
  - The systems have only reusable resources.
  - Processes are allowed to make only exclusive access to resources.
  - There is only one copy of each resource.
- Running state (also called active state), a process has all the needed resources and is either executing or is ready for execution.
- Blocked state, a process is waiting to acquire some resource.

Wait for Graph

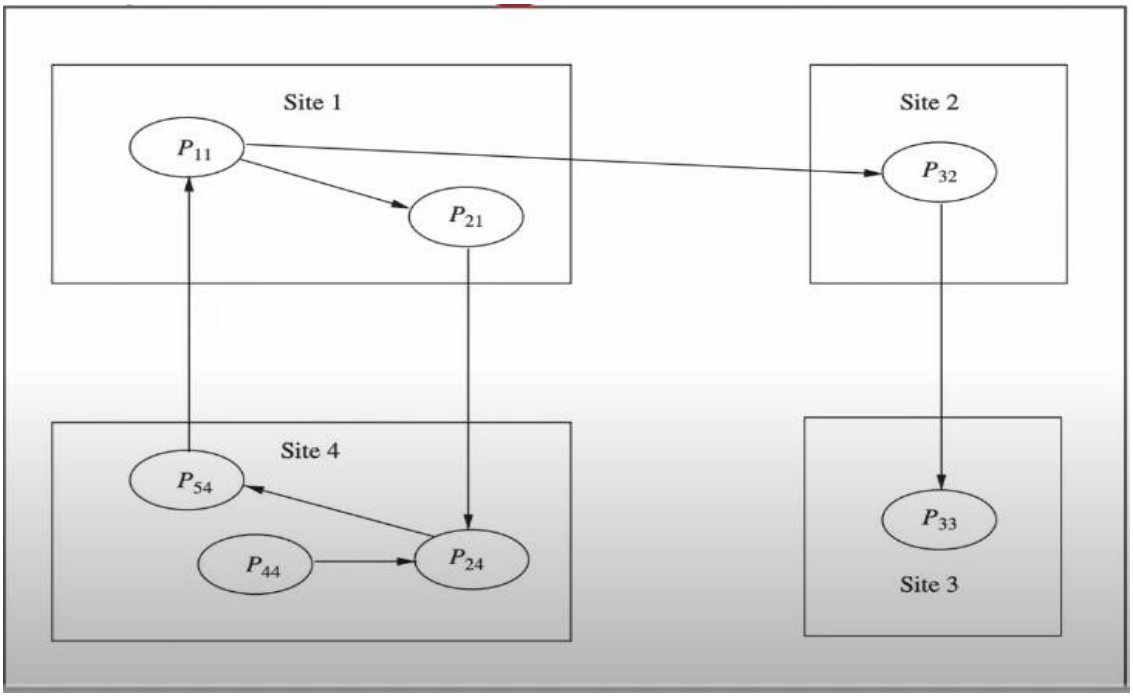


Fig 3.4 Wait for Graph

- In distributed systems, the state of the system can be modeled by directed graph, called a wait-for graph (WFG). In a WFG, nodes are processes and there is a directed edge from node P, to mode P2 if P1 is blocked and is waiting for P2 to release some resource. A system is deadlocked if and only if there exists a directed cycle or knot in the WFG in above Fig 3.4
- Figure shows a WFG, where process P1 of site 1 has an edge to process P21 of site 1 and an edge to process P32 of site 2. Process P32 of site 2 is waiting for a resource that is currently held by process P33 of site 3. At the same time process P21 at site 1 is waiting on process P24 at site 4 to release a resource, and so on. If P33 starts waiting on process P24, then processes in the WFG are involved in a deadlock depending upon the request model.

Issues in Deadlock Detection

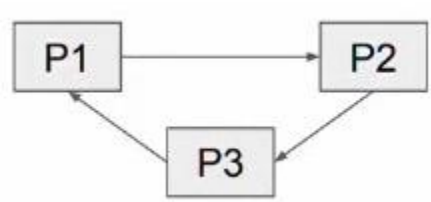
- **Correctness criteria**

  A deadlock detection algorithm must satisfy the following two conditions:

  Progress (no undetected deadlocks) The algorithm must detect all exist- ing deadlocks in a finite time. Once a deadlock has occurred, the dead- lock detection activity should continuously progress until the deadlock is detected. In other words, after all wait-for dependencies for a deadlock have formed, the algorithm should not wait for any more events to occur to detect the deadlock.

- **Safety (no false deadlocks)** The algorithm should not report deadlocks that do not exist (called phantom or false deadlocks). In distributed systems where there is no global memory and there is no global clock, it is difficult to design a correct deadlock detection algorithm because sites may obtain an out-of-date and inconsistent WFG of the system. As a result, sites may detect a cycle that never existed but whose different segments existed in the system at different times. This is the main reason why many deadlock detection algorithms reported in the literature are incorrect.
- Detection of deadlocks involves addressing two issues: maintenance of the WFG and searching of the WFG for the presence of cycles (or knots).
- Since, in distributed systems, a cycle or knot may involve several sites, the search for cycles greatly depends upon how the WFG of the system is represented across the system. Many algorithms are there for the same.

Issues in Deadlock - Resolution of Deadlocks



- Deadlock resolution involves breaking existing wait-for dependencies between the processes to resolve the deadlock.
- It involves rolling back one or more deadlocked processes and assigning their resources to blocked processes so that they can resume execution.
- Note that several deadlock detection algorithms propagate information regarding wait-for dependencies along the edges of the wait-for graph.
- Therefore, when a wait-for dependency is broken, the corresponding information should be immediately cleaned from the system. If this information is not cleaned in a timely manner, it may result in detection of phantom deadlocks.

6. Explain about the different Models of deadlocks? Nov/Dec 2022

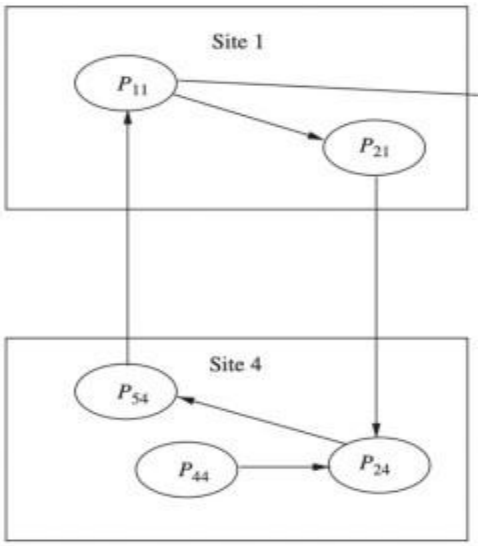
1.Single Resource Model
2.AND Model
3.OR Model
4. $\binom{p}{q}$ **Model (p out of q model)**
5.Unrestricted Model

1. Single Resource Model

a. a process can have at most one outstanding request for only one unit of a resource.
b. 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.

2. AND Model

a. a process can request more than one resource simultaneously and the request is satisfied only n after all the requested resources are granted to the process.
b. The requested resources may exist at different locations.
c. The out degree of a node in the WFG for AND model can be more than 1.
d. The presence of a cycle in the WFG indicates a deadlock in the AND model. Each node of the WFG in such a model is called an AND node in below Fig 3.5.1



Cycle => Deadlock but not vice versa

Fig 3.5.1 AND Model

: $P_{11} \rightarrow P_{21} \rightarrow P_{24} \rightarrow P_{54} \rightarrow P_{11}$, ١

OR Model

- a process can make a request for numerous resources simultaneously and the request is satisfied if any one of the requested resources is granted.
- The requested resources may exist at different locations. If all requests in the WFG are OR requests, then the nodes are called OR nodes.
- Presence of a cycle in the WFG of an OR model does not imply a deadlock in the OR model.
- In the OR model, the presence of a knot indicates a deadlock . In a WFG, a vertex v is in a knot if for all u :: u is reachable from v : v is reachable from u.

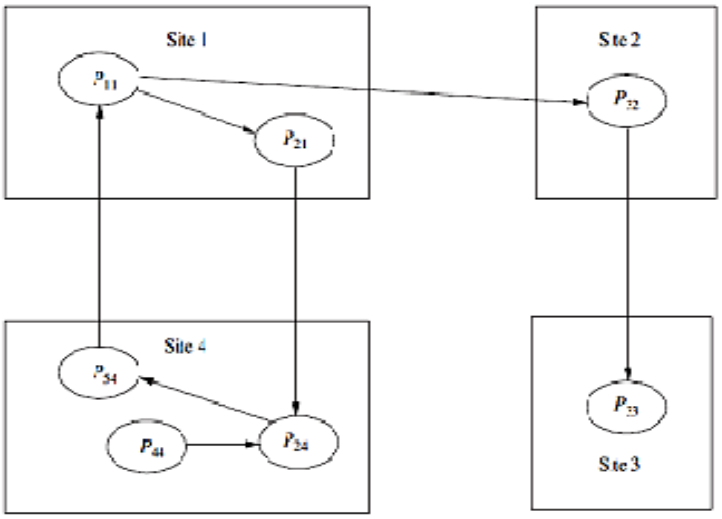


Fig 3.5.2 OR Model

When deadlock occurs in OR Model?

- a process Pi is blocked if it has a pending OR request to be satisfied.
- With every blocked process, there is an associated set of processes called dependent set.
- A process shall move from an idle to an active state on receiving a grant message from any of the processes in its dependent set.
- A process is permanently blocked if it never receives a grant message from any of the processes in its dependent set.
- Intuitively, a set of processes S is deadlocked if all the processes in S are permanently blocked in above Fig 3.5.2
- To formally state that a set of processes is deadlocked, the following conditions holdtrue:
  - Each of the process is the set S is blocked.
  - The dependent set for each process in S is a subset of S.
  - No grant message is in transit between any two processes in set S

AND-OR Model

- a request may specify any combination of and and or in the resource request.
- For example, in the ANDOR model, a request for multiple resources can be of the form x and (y or z).
- The requested resources may exist at different locations.
- 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 (i.e., once it exists, it does not go away by itself), this property can be exploited and a deadlock in the AND-OR model can be detected by repeated application of the test for OR-model deadlock.

The $\binom{p}{q}$ model

Another form of the AND-OR model is the $\binom{p}{q}$ model (called the P-out-of-Q model), which allows a request to obtain any k available resources from a pool of n resources. Both the models are the same in expressive power. However, $\binom{p}{q}$ model lends itself to a much more compact formation of a request. Every request in the $\binom{p}{q}$ model can be expressed in the AND-OR model and vice-versa.

Unrestricted Model

- In this model, only one assumption that the deadlock is stable is made and hence it is the most general model .concerns about properties of the problem (stability and deadlock) are separated from underlying distributed systems computations (e.g., message passing versus synchronous communication).
- Hence, these algorithms can be used to detect other stable properties as they deal with this general model. But, these algorithms are of more theoretical value for distributed systems since no further assumptions are made about the underlying distributed systems computations which leads to a great deal of overhead.

9. Explain the Chandy–Misra–Haas algorithm for the AND model and Chandy–Misra–Haas algorithm for the OR model with performance analysis or Outline the Chandy–Misra –Haas algorithm to detect deadlock in OR model and illustrate the algorithm with an example. APR/MAY 2024

CHANDY–MISRA–HAAS ALGORITHM FOR THE AND MODEL

Algorithm

if P_i, is locally dependent on itself
then declare a deadlock
else for all P_j, and P_k, such that
(a) P_i , is locally dependent upon P_j. and
(b) P_j, is waiting on P_k , and
(c) P_j and P_k , are on different sites,
send a probe (i, j, k) to the home site of P_k
On the receipt of a probe (i, j, k), the site takes the following actions:
if
(d) P_k, is blocked, and
(e) dependent $_k$ (i) is false, and
(f) P_k has not replied to all requests P_j ,
then
begin
dependent, (i) = true;
if k=i
then declare that P_i, is deadlocked

else for all P_m, and P_n, such that
(a) P_k is locally dependent upon P_m , and
(b) P_m is waiting on P_n, and
(c) P_m and P_n, are on different sites,
send a probe (i, m, n) to the home site of P_n end.

- This algorithm uses a special message called probe, which is a triplet (i, j,k), denoting that it belongs to a deadlock detection initiated for process Pi and it is being sent by the home site of process Pj to the home site of process Pk.
- Each probe message contains the following information:
  - ❖ the id of the process that is blocked (the one that initiates the probe message);
  - ❖ the id of the process is sending this particular version of the probe message;
  - ❖ the id of the process that should receive this probe message.

- A probe message travels along the edges of the global WFG graph, and a deadlock is detected when a probe message returns to the process that initiated it.
- A process Pj is said to be dependent on another process Pk if there exists a sequence of processes Pj, Pi1 , Pi2 , . . . , Pim, Pk such that each process except Pk in the sequence is blocked and each process, except the Pj, holds a resource for which the previous process in the sequence is waiting.
- Process Pj is said to be locally dependent upon process Pk if Pj is dependent upon Pk and both the processes are on the same site.

CHANDY–MISRA–HAAS ALGORITHM FOR THE OR MODEL

Algorithm

Initiate a diffusion computation for a blocked process P_i:
send query(i, i, j) to all processes P_j; in the dependent set DS_i; of P_i;
num, (i) = $|DS_i|$; $wait_i$, (i) := true;
When a blocked process P_k receives a query(i, j, k):
if this is the engaging query for process P_i; then
send query(i, k, m) to all P_m, in its dependent set DS_k;
num $_k$ (i) = $|DS_k|$; wait $_k$ (i) := true
else if $wait_k$, (i) then send a reply(i, k, j) to Pj.
When a process P_k receives a reply(i, j, k):
if wait $_k$ (i) then
num $_k$ (i) := num $_k$ (i) — 1;
if num $_k$, (i) = 0 then
if i=k then **declare a deadlock**

else send reply(i, k, m) to the process P_m
which sent the engaging query.

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) , reply(i, j, k) denoting that they belong to a diffusion computation initiated by a process pi and are being sent from process pj to process pk.

Basic Idea

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 Pk receives a query(i, j, k) message, it takes the following actions:

1. If this is the first query message received by Pk for the deadlock detection initiated by Pi, then it propagates the query to all the processes in its dependent set and sets a local variable num k (i) to the number of query messages sent.
2. If this is not the engaging query, then Pk returns a reply message to it immediately provided Pk has been continuously blocked since it received the corresponding engaging query. Otherwise, it discards the query.

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. **Check pointing and rollback recovery:** Introduction – Background and definitions – Issues in failure recovery – Checkpoint-based recovery –Coordinated check pointing algorithm – Algorithm for asynchronous check pointing and recovery.

PART – A

1. **What is Consensus Problem (all processes have an initial value)?Nov/Dec 2022**
   Agreement: All non-faulty processes must agree on the same (single) value.
   Validity: If all the non-faulty processes have the same initial value, then the agreed upon value by all the non-faulty processes must be that same value.
   Termination: Each non-faulty process must eventually decide on a value.

2. **What is Byzantine Agreement (single source has an initial value) ?April/May 2022**
   Agreement: All non-faulty processes must agree on the same value.
   Validity: If the source process is non-faulty, then the agreed upon value by all the non-faulty processes must be the same as the initial value of the source.
   Termination: Each non-faulty process must eventually decide on a value.

3. **What is Interactive Consistency (all processes have an initial value)?**
   Agreement All non-faulty processes must agree on the same array of values $A[v_1 \ldots v_n]$
   Validity If process i is non-faulty and its initial value is ϑ_i , then all non-faulty processes agree on vi as the ith element of the array A. If process j is faulty, then the non-faulty processes can agree on any value for A[j].
   Termination Each non-faulty process must eventually decide on the array A.

4. **Define Phase King Algorithm.**
   Each phase has a unique"phase king" derived, say, from PID.
   Each phase has **two rounds**:
   - ✓ 1st round, each process sends its estimate to all other processes.
   - ✓ 2nd round, the "Phase king" process arrives at an estimate based on the values it received in 1st round, and broadcasts its new estimate to all others.

5. **What is Terminating Reliable Broadcast (TRB)?**
   A correct process always gets a message, even if sender crashes while sending (in which case the process gets a null message).
   Validity: If the sender of a broadcast message m is non-faulty, then all correct processes eventually deliver m.
   Agreement: If a correct process delivers a message m, then all correct processes deliver m.

Integrity: Each correct process delivers at most one message. Further, if it delivers a message different from the null message, then the sender must have broadcast m.
Termination: Every correct process eventually delivers some message.

6. What is check point and rollback recovery? April/May 2022

- The saved state is called **a checkpoint**, and the procedure of restarting from a previously check pointed state is called **rollback recovery.**
- A checkpoint can be saved on either the stable storage or the volatile storage depending on the failure scenarios to be tolerated.
- In distributed systems, rollback recovery is complicated because messages induce inter-process dependencies during failure-free operation.

7. Define rollback propagation.

A failure of one or more processes in a system, these dependencies may force some of the processes that did not fail to roll back, creating what is commonly called rollback **propagation.**

8. What are the Different types of messages?

- ✓ In-transit messages
- ✓ Lost messages
- ✓ Delayed messages
- ✓ Orphan messages
- ✓ Duplicate messages.

9. Define In-transit messages and lost messages.

In-transit messages: the global state $\{C_{1,8}, C_{2,9}, C_{3,8}, C_{4,8}\}$ shows that message m1 has been **sent but not yet received.** We call such a message an in-transit message. Message m2 is also an **in-transit message**.
Lost messages: Messages whose **send is not undone but receive is undone due to rollback** are called **lost messages.**

10. What is delayed messages?

Messages whose receive is not recorded because the receiving process was either down or the message arrived after the rollback of the receiving process, are called **delayed messages.**

11. Define Duplicate messages and Orphan messages.

- ✓ Duplicate messages arise due to message logging and replaying during process recovery.
  Orphan messages
- ✓ Messages with receive recorded but message send not recorded are called **orphan messages.** Orphan messages do not arise if processes roll back to a consistent global state.

12. What is consistent global checkpoint?

- ✓ **A consistent global checkpoint** is a global checkpoint such that no message is sent by a process after taking its local checkpoint that is received by another process before taking its local checkpoint.

13. Define a local checkpoint and global checkpoint. April/May 2021

- ➢ In distributed systems, all processes save their local states at certain instants of time. This saved state is known as **a local checkpoint**.
- ➢ A local checkpoint is **a snapshot of the state** of the process at a given instance and the event of n recording the state of a process is called **local check pointing.**
- ➢ The contents of a checkpoint depend upon the application context and the check pointing method being used.
- ➢ **A global checkpoint** is a set of local checkpoints, one from each process.

14. What are three categories of Checkpoint-based rollback-recovery?

Checkpoint-based rollback-recovery techniques can be classified into three categories:

1. Uncoordinated check pointing,
2. Coordinated check pointing, and
3. Communication-induced check pointing.

15. Define uncoordinated check pointing.

Uncoordinated check pointing

- ✓ In uncoordinated check pointing, each process has autonomy in deciding when to take checkpoints.
- ✓ This eliminates the synchronization overhead as there is no need for coordination between processes and it allows processes to take checkpoints when it is most convenient or efficient.

16. Define coordinated check pointing.

Coordinated check pointing

- In coordinated check pointing, processes arrange their check pointing activities so that all local checkpoints form a consistent global state.
- Coordinated check pointing simplifies recovery and is not susceptible to the domino effect, since every process always restarts from its most recent checkpoint.

17. What are the advantages of Uncoordinated check pointing?

Advantages:

- ✓ The main advantage is the lower runtime overhead during normal execution, because no coordination among processes is necessary.
- ✓ Autonomy in taking checkpoints also allows each process to select appropriate checkpoints positions.

18. What are the disadvantages of Uncoordinated check pointing?

- ✓ First, there is the possibility of the **domino effect during a recovery**, which may cause the loss of a large amount of useful work.
- ✓ Second, recovery from a failure is slow because processes need to iterate to find a consistent set of checkpoints.
- ✓ Third, uncoordinated check pointing forces each process to maintain multiple checkpoints, and to periodically invoke a garbage collection algorithm to reclaim the checkpoints that are no longer required.
- ✓ Fourth, it is not suitable for applications with frequent output commits because these require global coordination to compute the recovery line, negating much of the advantage of autonomy.

19. Define Communication-induced check pointing.

- ➢ Communication-induced check pointing is another way to avoid the domino effect, while allowing processes to take some of their checkpoints independently.
- ➢ Communication-induced check pointing reduces or completely eliminates the useless checkpoints.
- ➢ In communication-induced check pointing, processes take two types of checkpoints, namely, **autonomous and forced checkpoints**.

20. What is the problem of coordinated check pointing?

Disadvantage:

- ✓ A problem with this approach is that the computation is blocked during the check pointing and therefore, non-blocking check pointing schemes are preferable.

21. What are the types of communication-induced check pointing?

There are two types of communication-induced check pointing:

1. Model-based check pointing and
2. Index-based check pointing.

22. Define Log-based rollback recovery.

Log-based rollback recovery • combines check pointing with logging of non-deterministic events • relies on piecewise deterministic (PWD) assumption.

23. Define Consistent global state.

- ➢ A global state that may occur during a failure-free execution of distribution of distributed computation
- ➢ If a process's state reflects a message receipt, then the state of the corresponding sender must reflect the sending of the message.

24. Define blocking coordinated check pointing.

- ✓ A straightforward approach to coordinated check pointing is to block communications while the check pointing protocol executes.

- ✓ After a process takes a local checkpoint, to prevent orphan messages, it remains blocked until the entire check pointing activity is complete.

25. Define Non-blocking checkpoint coordination.

- ✓ In this approach the processes need not stop their execution while taking checkpoints.
- ✓ A fundamental problem in coordinated check pointing is to prevent a process from receiving application messages that could make the checkpoint inconsistent.

26. What is model-based check pointing?

- ➢ The system maintains checkpoints and communication structures that prevent the domino effect or achieve some even stronger properties.
- ➢ Model-based check pointing prevents patterns of communications and checkpoints that could result in inconsistent states among the existing checkpoints.
- ➢

27. What is index-based check pointing?

- ➢ The system uses an indexing scheme for the local and forced checkpoints, such that the checkpoints of the same index at all processes form a consistent state.
- ➢ Index-based communication-induced check pointing assigns monotonically increasing indexes to checkpoints, such that the checkpoints having the same index at different processes form a consistent state.

28. What is Koo–Toueg coordinated check pointing algorithm?

- ➢ Koo and Toueg's coordinated check pointing and recovery technique takes a consistent set of checkpoints and avoids the **domino effect and live lock problems** during the recovery.
- ➢ Processes coordinate their local check pointing actions such that the set of all checkpoints in the system is consistent.

29. What are the two kinds of checkpoints on the stable storage?

- ➢ The checkpoint algorithm takes **two kinds of checkpoints on the stable storage**: permanent and tentative.
- ➢ **A permanent checkpoint** is **a local checkpoint** at a process and is a part of a consistent global checkpoint.
- ➢ **A tentative checkpoint** is **a temporary checkpoint** that is made a permanent checkpoint on the successful termination of the checkpoint algorithm.

30. What are the two phases of Check pointing algorithm?

1. The initiating process takes a tentative checkpoint and requests all other processes to take tentative checkpoints. Every process can not send messages after taking tentative checkpoint. All processes will finally have the single same decision: do or discard.
2. All processes will receive the final decision from initiating process and act accordingly.

31. What is rollback recovery algorithm? Nov/Dec 2022

- The rollback recovery algorithm restores the system state to a consistent state after a failure.
- The rollback recovery algorithm assumes that a single process invokes the algorithm. It also assumes that the checkpoint and the rollback recovery algorithms are not invoked concurrently.

32.Outline the issues in failure recovery. APR/MAY 2024

I.Detection

1. Identifying the failure
2. Determining the scope and impact
3. Notifying relevant teams and stakeholders

II. Assessment

1. Evaluating the root cause
2. Assessing the damage and affected systems
3. Determining the resources needed for recovery

III. Containment

1. Isolating the affected systems or components
2. Preventing further damage or propagation
3. Stabilizing the environment

33.What are the advantages and disadvantages of asynchronous check pointing? APR/MAY 2024

Asynchronous checkpointing is a technique used in distributed systems to ensure fault tolerance and recoverability. Here are its advantages and disadvantages:

Advantages:

- Improved system availability: Asynchronous checkpointing allows the system to continue operating even if one or more nodes fail.
- Reduced downtime: By checkpointing data asynchronously, the system can quickly recover from failures without waiting for synchronous checkpointing to complete.

Disadvantages:

- Increased complexity: Asynchronous checkpointing is more complex to implement than synchronous checkpointing, requiring additional logic to manage checkpoints and handle failures.
- Higher storage requirements: Asynchronous checkpointing requires more storage space to store multiple checkpoints, which can increase storage costs.

34.What is a checkpoint in the recovery system? How is it useful? NOV/DEC 2023

A checkpoint in a recovery system is a saved state of a process or system at a specific point in time. It captures the current state of the system, including data, variables, and context, allowing the system to restart from that exact point in case of a failure or crash.

Checkpoints are useful in several ways:

1. Fault tolerance: Checkpoints enable the system to recover from failures, such as power outages, hardware failures, or software crashes.
2. Error recovery: Checkpoints allow the system to roll back to a previous state, undoing any changes made since the last checkpoint, and restarting from a known good state**.**

35.State the issues in failure recovery. State the means to identify the failure at an early stage. NOV/DEC 2023

Issues in Failure Recovery:

1. Detection: Identifying the failure quickly and accurately.
2. Assessment: Evaluating the scope and impact of the failure.
3. Containment: Preventing further damage or propagation.
4. Recovery: Restoring the system to a stable state.

Means to Identify Failure at an Early Stage:

1. Monitoring: Real-time monitoring of system performance and logs.
2. Anomaly detection: Identifying unusual patterns or behavior.
3. Predictive analytics: Using machine learning to predict potential failures.

PART –B

1. What is the Problem definition of Consensus and Agreement in detail?

What are the key assumptions underlying while designing agreement algorithms and brief them? April/May 2021, 2022

Assumptions

1. Failure models

a. Among the n processes in the system, at most f processes can be faulty.

b. A faulty process can behave in any manner allowed by the failure model assumed.

c. In fail-stop model, a process may crash in the middle of a step, which could be the execution of a local operation or processing of a message for a send or receive event.

d. In the Byzantine failure model, a process may behave arbitrarily

2. Synchronous/asynchronous communication

a. If a failure-prone process chooses to send a message to process Pi but fails, then Pi cannot detect the non-arrival of the message in an asynchronous system

b. In synch systems, message which has not been sent can be recognized by the intended recipient, at the end of the round.

3. Network connectivity

a. The system has full logical connectivity, i.e., each process can communicate with any other by direct message passing.

4. Channel reliability

a. The channels are reliable, and only the processes may fail.

5. Sender identification

a. A process that receives a message always knows the identity of the sender process.

b. When multiple messages are expected from the same sender in a single round, we implicitly assume a scheduling algorithm that sends these messages in sub-rounds, so that each message sent within the round can be uniquely identified.

6. Authenticated vs. non-authenticated messages

a. With unauthenticated messages, when a faulty process relays a message to other processes,

(i) it can forge the message and claim that it was received from another process, and

(ii) it can also tamper with the contents of a received message before relaying it.

b. When a process receives a message, it has no way to verify its authenticity.An unauthenticated message is also called an oral message or an unsigned message.

c. Using authentication via techniques such as digital signatures, it is easier to solve the agreement problem.

7. Agreement variable

a. The agreement variable may be boolean or multivalued, and need not be an integer

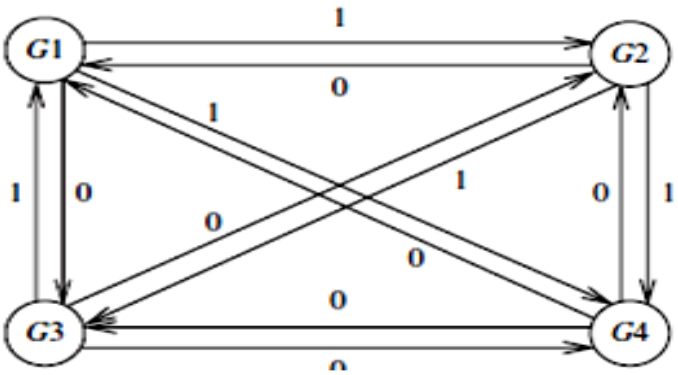


Fig 4.1 Byzantine Empire

Case Study

- Four camps of the attacking army, each commanded by a general, are camped around the fort of Byzantium in above Fig 4.1
- They can succeed in attacking only if they attack simultaneously. Hence, they need to reach agreement on the time of attack.
- The only way they can communicate is to send messengers among themselves.
- The messengers model the messages. An asynchronous system is modeled by messengers taking an unbounded time to travel between two camps.
- A lost message is modeled by a messenger being captured by the enemy. A Byzantine process is modeled by a general being a traitor.
- The traitor will attempt to subvert the agreement-reaching mechanism, by giving misleading information to the other generals. For example, a traitor may inform one general to attack at 10 a.m., and inform the other generals to attack at noon. Or he may not send a message at all to some general.
- Likewise, he may tamper with the messages he gets from other generals, before relaying those messages.

- The various generals are conveying potentially misleading values of the decision variable to the other generals, which results in confusion. In the face of such Byzantine behavior, the challenge is to determine whether it is possible to reach agreement, and if so under what conditions.
- If agreement is reachable, then protocols to reach it need to be devised.

The Byzantine agreement problem

The Byzantine agreement problem requires a designated process, called the source process, with an initial value, to reach agreement with the other processes about its initial value,

- Agreement All non-faulty processes must agree on the same value.
- Validity If the source process is non-faulty, then the agreed upon value by all the non-faulty processes must be the same as the initial value of the source.
- Termination Each non-faulty process must eventually decide on a value.

The consensus problem

- Agreement All non-faulty processes must agree on the same (single) value.
- Validity If all the non-faulty processes have the same initial value, then the agreed upon value by all the non-faulty processes must be that same value.
- Termination Each non-faulty process must eventually decide on a value.

The interactive consistency problem

- Agreement All non-faulty processes must agree on the same array of values A[v1... vn].
- Validity If process i is non-faulty and its initial value is v;, then all non- faulty processes agree on v; as the ith element of the array A. If process j is faulty, then the non-faulty processes can agree on any value for A[j].
- Termination Each non-faulty process must eventually decide on the array A.

2. Explain the Agreement in a failure-free system (synchronous or asynchronous).

- In a failure-free system, consensus can be reached by
  - collecting information from the different processes
  - arriving at a "decision,"
  - 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.

- The decision can be reached by using an application specific function – some simple examples being the majority, max, and min functions.
- Algorithms to collect the initial values and then distribute the decision may be based on the token circulation on a logical ring, or the three-phase tree-based broadcast–convergecast–broadcast, or direct communication with all nodes.
- In a synchronous system, this can be done simply in a constant number of rounds.
- In an asynchronous system, consensus can similarly be reached in a constant number of message hops.

3. Explain in detail about Agreement in (message-passing) synchronous systems with failures (or) What is agreement in a failure free system? Explain how agreement is reached in message passing synchronous system with failures Nov/Dec 2022 , APR/MAY 2024

In a failure-free system, agreement refers to the ability of all nodes or processes to reach a common decision or consensus on a particular value, action, or state. This is a fundamental concept in distributed systems, where multiple nodes work together to achieve a common goal.

1. Consensus algorithm for crash failures (synchronous system)
2. Upper bound on Byzantine processes
3. Byzantine agreement tree algorithm: exponential (synchronous system)

a. Recursive

b. Iterative

4. Phase-king algorithm for consensus:

1. Consensus algorithm for crash failures (synchronous system)

Algorithm

(global constants)

Integer :f ;

(local variables)

Integer :x ← local value;

1) Process P_i $(1 \leq i \leq n)$ execute the consensus algorithm for up to f crash failures:

(1a) for round from 1 to f + 1 do

(1b) if the current value of x has not been broadcast then

(1c) broadcast(x);

(1d) yi ← value (if any) received from process j in this round;

(1e) x ← min$\forall$j (x, y j) ;

(1f) output x as the consensus value.

Validity of Conditions

- The agreement condition is satisfied because in the f+1 rounds, there must be at least one round in which no process failed. In this round, say round r, all the processes that have not failed so far succeed in broadcasting their values, and all these processes take the minimum of the values broadcast and received in that round.
- The validity condition is satisfied because processes do not send fictitious values in this failure model. For all i, if the initial value is identical, then the only value sent by any process is the value that has been agreed upon as per the agreement condition.
- The termination condition is seen to be satisfied.

2. Upper bound on Byzantine processes

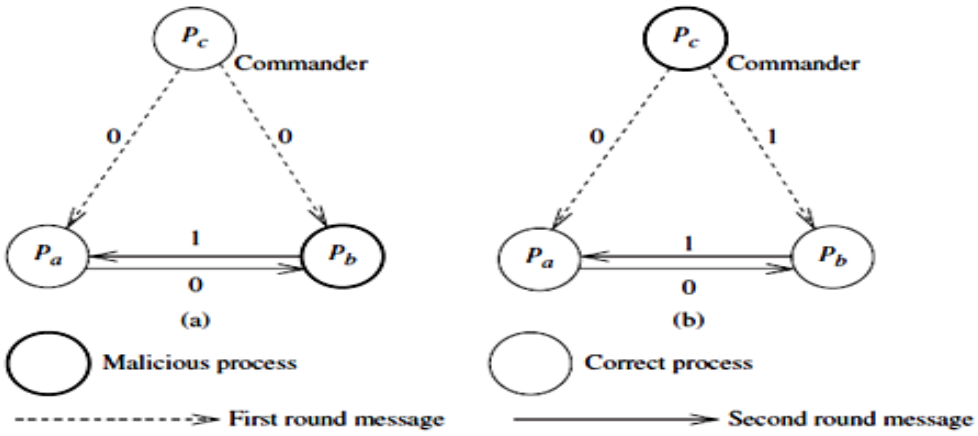


Fig 4.2 a, b Upper bound on Byzantine processes

1. Context: Byzantine Agreement Problem with n-3 processes and f=1 Byzantine process.
2. Scenario 1 (Figure 4.2 (a)):
   - Processes: Commander Pc, Lieutenants Pa and Pb.
   - Malicious Process: Pb (disloyal lieutenant).
   - Objective: Pa should agree on the value of the loyal commander Pc, which is 0.
   - Challenge: Pa cannot distinguish between scenarios, making it impossible to make a correct decision.
3. Scenario 2 (Figure 4.2(b)):
   - Processes: Commander Pc, Lieutenants Pa and Pb.
   - Malicious Process: Pc (disloyal commander), Pb (loyal lieutenant).
   - Situation: Pa receives identical values from Pb and Pc.
   - Dilemma: Pa needs to agree with Pb, but distinguishing between scenarios is not possible.

• Message Exchange: Further communication doesn't help as each process has already shared its information.

4. In both scenarios:
   • Pa receives different values from the other two processes.
   • Default values (0 or 1) lead to incorrect decisions in at least one of the scenarios.
5. Conclusion:
   • Agreement is impossible when n=3 processes and f=1 Byzantine process.

4. Explain in detail about Check pointing and rollback recovery.

What is Domino Effect?

- To see why rollback propagation occurs, consider the situation where the sender of a message m rolls back to a state that precedes the sending of m.
- The receiver of m must also roll back to a state that precedes m's receipt; otherwise, the states of the two processes would be inconsistent because they would show that message m was received without being sent, which is impossible in any correct failure-free execution.
- This phenomenon of cascaded rollback is called the domino effect.
- In some situations, rollback propagation may extend back to the initial state of the computation, losing all the work performed before the failure.
- Independent or uncoordinated checkpointing : - If each participating process takes its checkpoints independently, then the system is susceptible to the domino effect.

How to avoid domino effect?

- Coordinated checkpointing :
  - processes coordinate their checkpoints to form a system-wide consistent state.
  - In case of a process failure, the system state can be restored to such a consistent set of checkpoints, preventing the rollback propagation.
- 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.
- Logbased rollback recovery:
  - combines checkpointing with logging of nondeterministic events.

- Log-based rollback recovery relies on the piecewise deterministic (PWD) assumption, which postulates that all non-deterministic events that a process executes can be identified and that the information necessary to replay each event during recovery can be logged in the event's determinant.
- By logging and replaying the non-deterministic events in their exact original order, a process can deterministically recreate its pre-failure state even if this state has not been checkpointed.

Key Points

- Rollback recovery treats a distributed system application as a collection of processes that communicate over a network.
- It achieves fault tolerance by periodically saving the state of a process during the failure-free execution, enabling it to restart from a saved state upon a failure to reduce the amount of lost work.
- The saved state is called a checkpoint, and the procedure of restarting from a previously checkpointed state is called rollback recovery.
- A checkpoint can be saved on either the stable storage or the volatile storage depending on the failure scenarios to be tolerated.
- Challenges for Recovery:
  - on a failure of one or more processes in a system, these dependencies may force some of the processes that did not fail to roll back, creating what is commonly called a rollback propagation

5. Explain in detail about a local checkpoint and different types of messages. April/May 2021

1. **System Model**
2. **Local Checkpoint**
3. **Consistent system states**
4. **Interactions with the outside world**
5. **Different types of messages**

1. System Model

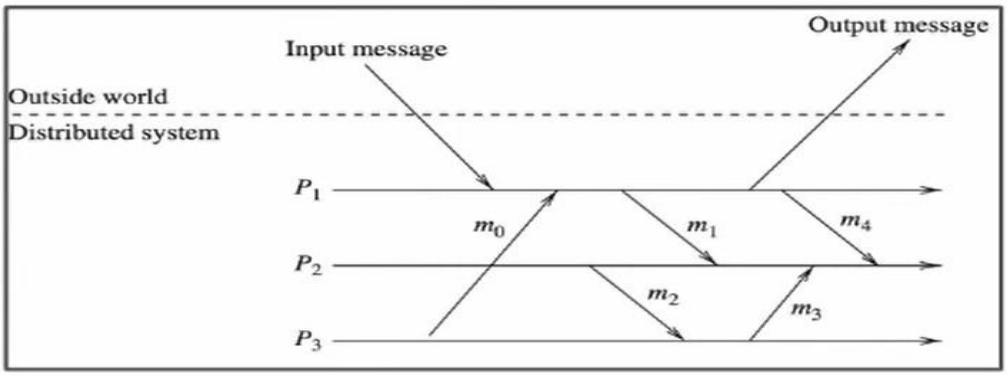


Fig 4.3.1 System Model

- A distributed system consists of a fixed number of processes, P1, P2 PN , which communicate only through messages in above Fig 4.3.1
- Processes cooperate to execute a distributed application and interact with the outside world by receiving and sending input and output messages, respectively.
- Some protocols assume that the communication subsystem delivers messages reliably, in first-in-first-out (FIFO) order, while other protocols assume that the communication subsystem can lose, duplicate, or reorder messages.
- a system recovers correctly if its internal state is consistent with the observable behavior of the system before the failure.

2. A local checkpoint

1. A 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.
2. The contents of a checkpoint depend upon the application context and the checkpointing method being used.
3. Depending upon the checkpointing method used, a process may keep several local checkpoints or just a single checkpoint at any time
4. a process stores all local checkpoints on the stable storage so that they are available even if the process crashes.
5. We also assume that a process is able to roll back to any of its existing local checkpoints and thus restore to and restart from the corresponding state

3.Consistent system states

- ✓ **A global state of a distributed system** is a collection of the individual states of all participating processes and the states of the communication channels.

  A consistent global state is one that may occur during a failure-free execution of a distributed computation in below Fig 4.3.2
- ✓ More precisely, **a *consistent system state*** is one in which a process's state reflects a message receipt, then the state of the corresponding sender must reflect the sending of that message.

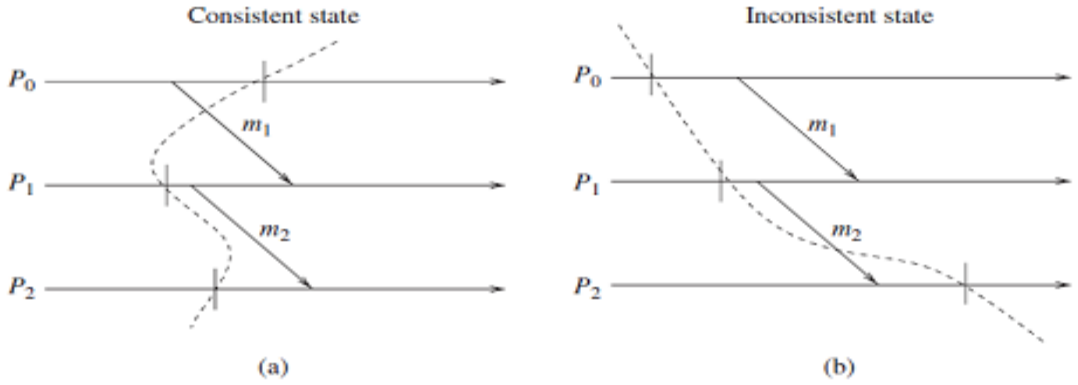


Fig 4.3.2 Examples of consistent and inconsistent states

4.Interactions with the outside world (OWP)

● a printer cannot roll back the effects of printing a character, and an automatic teller machine cannot recover the money that it dispensed to a customer

● A distributed application often interacts with the outside world to receive input data or deliver the outcome of a computation. If a failure occurs, the outside world cannot be expected to roll back.

● the outside world see a consistent behavior of the system despite failures

● Output Commit- before sending output to the OWP, the system must ensure that the state from which the output is sent will be recovered despite any future failure.

● Input messages :

○ Received messages from the OWP may not be reproducible during recovery, because it may not be possible for the outside world to regenerate them.

○ Thus, recovery protocols must arrange to save these input messages so that they can be retrieved when needed for execution replay after a failure

Types of Messages

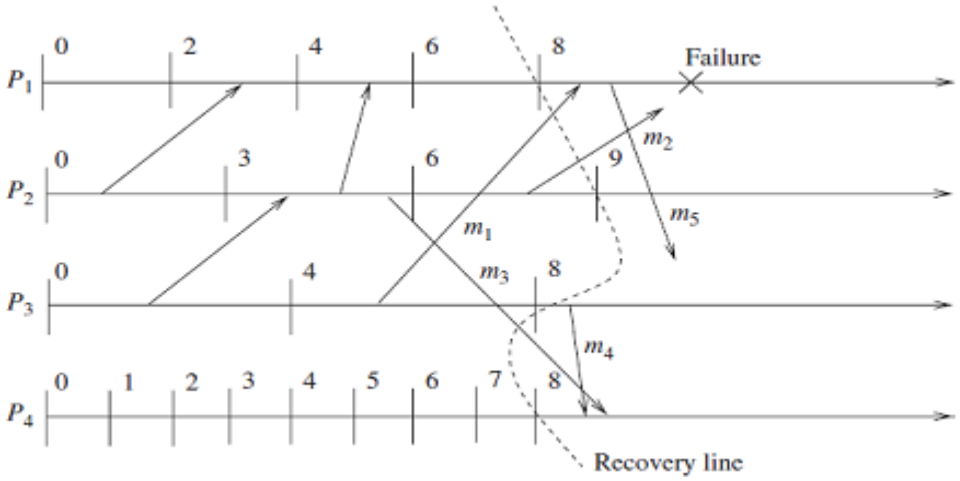


Fig 4.3.3 Different types of messages

1. In-transit (m1,m2)

a. Messages that has been sent but not yet received

b. When in-transit messages are part of a global system state, these messages do not cause any inconsistency in above Fig 4.3.3

c. For reliable communication channels, a consistent state must include in-transit messages because they will always be delivered to their destinations in any legal execution of the system.

d. On the other hand, if a system model assumes lossy communication channels, then in-transit messages can be omitted from system state.

2. Lost Messages(m1)

a. Messages whose send is not undone but receive is undone due to rollback are called lost messages.

b. This type of messages occurs when the process rolls back to a checkpoint prior to reception of the message while the sender does not rollback beyond the send operation of the message

3. Delayed Messages (m2,m5)

a. Messages whose receive is not recorded because the receiving process was either down or the message arrived after the rollback of the receiving process

4. Orphan Messages

a. Messages with receive recorded but message send not recorded are called orphan messages.

b. For example, a rollback might have undone the send of such messages, leaving the receive event intact at the receiving process.

c. Orphan messages do not arise if processes roll back to a consistent global state.

5. Duplicate Message(m4,m5)

a. Duplicate messages arise due to message logging and replaying during process recovery

6. Explain in detail about Issues in failure recovery(OR) What are the different check pointing based recovery system available? Explain the Coordinated checkpointing algorithm with illustration April/May 2021 , NOV/DEC 2023

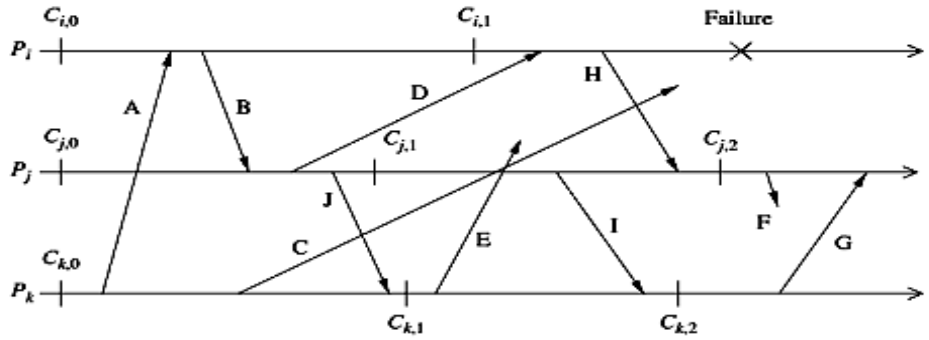


Figure 4.4 Illustration of issues in failure recovery.

Messages A, B, D, G, H, I, and J had been received at the points indicated in the figure and messages C, E, and F were in transit when the failure occurred. Restoration of system state to checkpoints (C1, C,1,C,1} automatically handles messages A, B, and J because the send and receive events of messages A, B, and J have been recorded, and both the events for G, H, and I have been completely undone. These messages cause no problem and we call messages A, B, and J normal messages and messages G, H, and I vanished messages in above Fig 4.4

Messages C, D, E, and F are potentially problematic. Message C is in transit during the failure and it is a delayed message. The delayed message C has several possibilities: C might arrive at process P_i, before it recovers, it might arrive while P_i, is recovering, or it might arrive after P_i, has completed recovery. Each of these cases must be dealt with correctly.

Message D is a lost message since the send event for D is recorded in the restored state for process P_j,, but the receive event has been undone at process P_i. Process P_j; will not resend D without

an additional mechanism, since the send D at P_j, occurred before the checkpoint and the communication system successfully delivered D.

Messages E and F are delayed orphan messages and pose perhaps the most serious problem of all the messages. When messages E and F arrive at their respective destinations, they must be discarded since their send events have been undone. Processes, after resuming execution from their checkpoints, will generate both of these messages, and recovery techniques must be able to distinguish between messages like C and those like E and F.

7. Explain the Checkpoint-based recovery and its types. April/May 2021

1. Uncoordinated checkpointing

2. Coordinated checkpointing

a. Blocking coordinated checkpointing

b. Non-blocking checkpoint coordination

3. Impossibility of min-process non-blocking checkpointing

4. Communication-induced checkpointing

a. Model-based checkpointing

b. Index-based checkpointing

1. Uncoordinated Checkpointing

- Each process has autonomy in deciding when to take checkpoints.
- Synchronization overhead is minimal as there is no need for coordination between processes. (Lower runtime overhead).
- Autonomy in taking checkpoints also allows each process to select appropriate checkpoints positions

Drawbacks:

1. Domino effect may occur during a recovery.
2. Recovery is slow because processes need to iterate to find a consistent set of checkpoints.

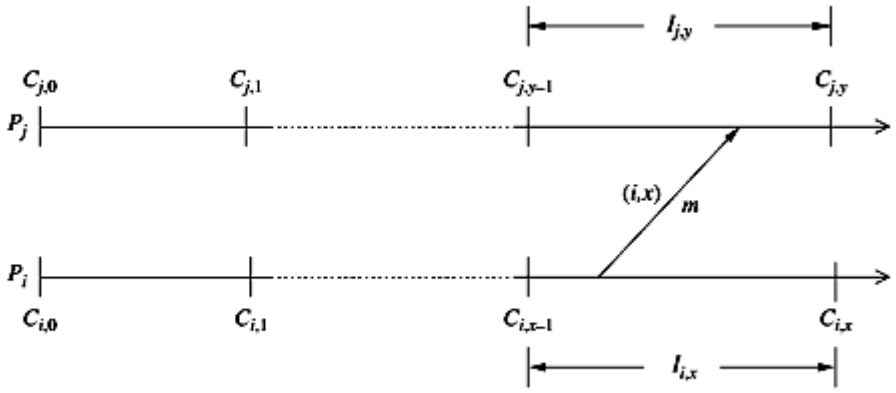


Figure 4.5 Checkpoint index and checkpoint interval [13].

Steps:

1. When a failure occurs, the recovering process initiates rollback by broadcasting a dependency request message to collect all the dependency information maintained by each process in above Fig 4.5
2. When a process receives this message, it stops its execution and replies with the dependency information saved on the stable storage as well as with the dependency information, if any, which is associated with its current state.
3. The initiator then calculates the recovery line based on the global dependency information and broadcasts a rollback request message containing the recovery line.
4. Upon receiving this message, a process whose current state belongs to the recovery line simply resumes execution; otherwise, it rolls back to an earlier checkpoint as indicated by the recovery line.

2. Coordinated checkpointing

1. Definition: Processes coordinate checkpointing to ensure consistent global state.

2. Benefits: Simplifies recovery, avoids the domino effect, and each process restarts from its latest checkpoint.

3. Storage: Requires each process to maintain only one checkpoint, reducing storage overhead and eliminating the need for garbage collection.

4. Drawbacks: Involves large latency for committing output, and delays/overhead occur with each new global checkpoint.

Clock Synchronization:

1. Ideal Scenario: If perfectly synchronized clocks exist, a simple method involves all processes agreeing on checkpoint instants triggered by their clocks.
2. Reality: Perfectly synchronized clocks are not available.
3. Approaches for Consistency:

- Block message sending during the protocol to ensure consistency.
- Piggyback checkpoint indices on messages to avoid blocking.

a.Blocking coordinated checkpointing

1. Approach: Coordinated checkpointing involves blocking communications during the protocol execution.

2. Procedure:

• After a process takes a local checkpoint, it remains blocked to prevent orphan messages until the entire checkpointing activity is complete.The coordinator initiates the process by taking a checkpoint and broadcasting a request message to all processes.

• Upon receiving the message, each process stops execution, flushes communication channels, takes a tentative checkpoint, and sends an acknowledgment back to the coordinator.

• After receiving acknowledgments from all processes, the coordinator broadcasts a commit message, completing the two-phase checkpointing protocol.

• Upon receiving the commit message, each process removes the old permanent checkpoint, makes the tentative checkpoint permanent, and resumes execution and message exchange.

b. Non-blocking checkpoint coordination

● In this approach the processes need not stop their execution while taking checkpoints.

● A fundamental problem in coordinated checkpointing is to prevent a process from receiving application messages that could make the checkpoint inconsistent.

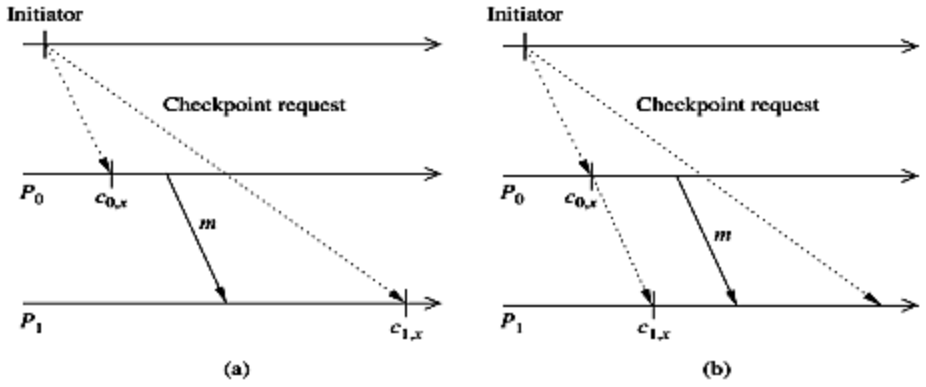


Figure 4.6 Non-blocking coordinated checkpointing: (a) checkpoint inconsistency; (b) a solution with FIFO channels [13].

3. Impossibility of min-process non-blocking checkpointing

● A min-process, non-blocking checkpointing algorithm is one that forces only a minimum number of processes to take a new checkpoint, and at the same time it does not force any

process to suspend its computation in above Fig 4.6

● Clearly, such checkpointing algorithms will be very attractive. Cao and Singhal showed that it is impossible to design a min-process, non-blocking checkpointing algorithm.

4.Communication-induced checkpointing

Communication-induced checkpointing prevents the domino effect while allowing processes to take some checkpoints indenpendently.

a. Model-based checkpointing

Model-based checkpointing prevents inconsistent states among checkpoints caused by communications patterns.

b.Index-based checkpointing

● Index-based communication-induced checkpointing assigns monotonically increasing indexes to checkpoints, such that the checkpoints having the same index at different processes form a consistent state.

● Inconsistency between checkpoints of the same index can be avoided in a lazy fashion if indexes are piggybacked on application messages to help receivers decide when they should take a forced a checkpoint.

8. Explain in detail about coordinated check pointing algorithm (or)
Explain (i) check pointing algorithm (ii) rollback recovery algorithm.
Koo–Toueg coordinated checkpointing algorithm (Or) Outline the Koo–Toueg coordinated checkpointing and recovery technique and apply this algorithm to a specific case study to demonstrate how this algorithm avoids the domino effect and live lock problems during the recovery Nov/Dec 2022 , APR/MAY 2024

Objective:

- Takes a consistent set of checkpoints and avoids the domino effect and livelock problems during the recovery.
- Processes coordinate their local checkpointing actions such that the set of all checkpoints in the system is consistent.

Assumptions of Checkpointing Algorithm:

- Processes communicate by exchanging messages through communication channels. Communication channels are FIFO.
- It is assumed that end-to-end protocols (such as the sliding window protocol) exist to cope with message loss due to rollback recovery and communication failure.
- Communication failures do not partition the network.

Permanent vs Tentative:

1. A permanent checkpoint is a local checkpoint at a process and is a part of a consistent global checkpoint.

2.A tentative checkpoint is a temporary checkpoint that is made a permanent checkpoint on the successful termination of the checkpoint algorithm.

Checkpoint - Phase 1

1. An initiating process Pi takes a tentative checkpoint and requests all other processes to take tentative checkpoints.

2. Each process informs Pi whether it succeeded in taking a tentative checkpoint.

3. A process says "no" to a request if it fails to take a tentative checkpoint, which could be due to several reasons, depending upon the underlying application.

4. If Pi learns that all the processes have successfully taken tentative checkpoints, Pi decides that all tentative checkpoints should be made permanent; otherwise, Pi decides that all the tentative checkpoints should be discarded.

Checkpoint - Phase 2

1. Pi informs all the processes of the decision it reached at the end of the first phase.

2. A process, on receiving the message from Pi, will act accordingly.

3. Therefore, either all or none of the processes advance the checkpoint by taking permanent checkpoints.

4. The algorithm requires that after a process has taken a tentative checkpoint, it cannot send messages related to the underlying computation until it is informed of Pi's decision.

Correctness

- A set of permanent checkpoints taken by this algorithm is consistent because of the following two reasons:
  - Either all or none of the processes take permanent checkpoints;
  - no process sends a message after taking a tentative checkpoint until the receipt of the initiating process's decision, as by then all processes would have taken checkpoints.
- Thus, a situation will not arise where there is a record of a message being received but there is no record of sending it in below Fig 4.7.1

Optimization

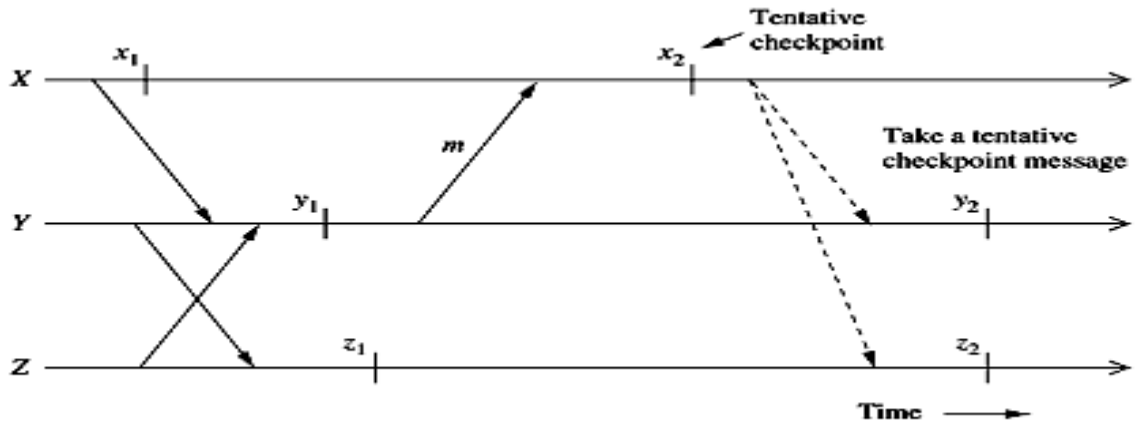


Fig 4.7.1 Examples of checkpoints taken unecessarily

Rollback Recovery Algorithm

Phase 1

- An initiating process Pi sends a message to all other processes to check if they all are willing to restart from their previous checkpoints.
- A process may reply "no" to a restart request due to any reason (e.g., it is already participating in a checkpoint or recovery process initiated by some other process).
- If Pi learns that all processes are willing to restart from their previous checkpoints, Pi decides that all processes should roll back to their previous checkpoints. Otherwise,

Pi aborts the rollback attempt and it may attempt a recovery at a later time.

Phase 2

- Pi propagates its decision to all the processes.
- On receiving Pi's decision, a process acts accordingly.
- During the execution of the recovery algorithm, a process cannot send messages related to the

underlying computation while it is waiting for Pi's decision in below Fig 4.7.2

Correctness

- All processes restart from an appropriate state because, if they decide to restart, they resume execution from a consistent state (the checkpointing algorithm takes a consistent set of checkpoints).

Optimization

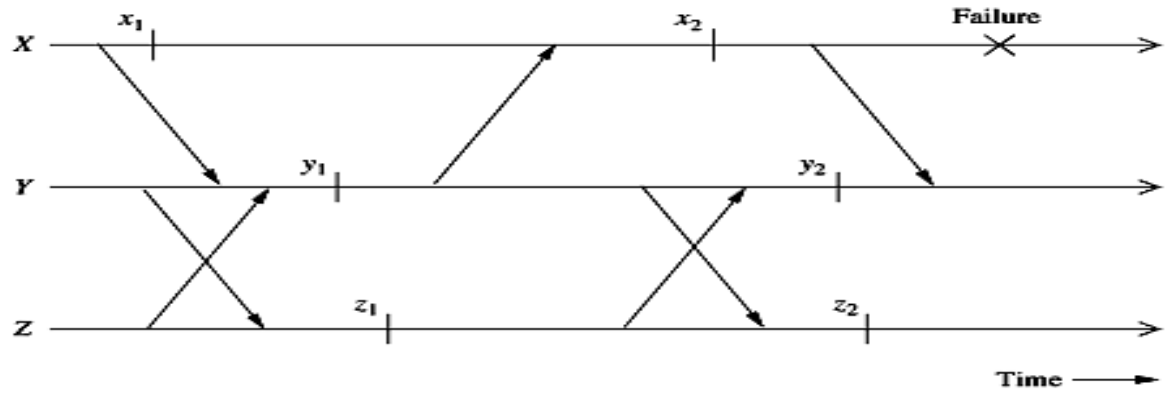


Fig 4.7.2 Examples of an unecessarily rollback

9. Explain the algorithm for asynchronous checkpointing and recovery with Example (OR) Illustrate the algorithm proposed by Juang and venkatesan for asynchronous check pointing and recovery. APR/MAY 2024

System Model and Assumptions:

- Communication channels are reliable, deliver the messages in FIFO order, and have infinite buffers.
- The message transmission delay is arbitrary, but finite.
- The processors directly connected to a processor via communication channels are called its neighbors.
- The underlying computation or application is assumed to be event-driven: a processor P waits until a message m is received, it processes the message m, changes its state from s to s , and sends zero or more messages to some of its neighbors.
- Then the processor remains idle until the receipt of the next message.
- The new state s and the contents of messages sent to its neighbors depend on state s and the contents of message m. The events at a processor are identified by unique monotonically increasing numbers, ex0, ex1, ex2,
- Storage can be:
  - Volatile Log - Less time but data lost when power is lost
  - Stable Storage - More time but data not lost.

Asynchronous Checkpointing

- After executing an event, a processor records a triplet (s, m, msgs_sent) in its volatile storage,
  - s is the state of the processor before the event
  - m is the message (including the identity of the sender of m, denoted as m.sender) whose arrival caused the event
  - msgs_sent is the set of messages that were sent by the processor during the event.
- Therefore, a local checkpoint at a processor consists of the record of an event occurring at the processor and it is taken without any synchronization with other processors.
- Periodically, a processor independently saves the contents of the volatile log in the stable storage and clears the volatile log in below Fig 4.8

Recovery Algorithm

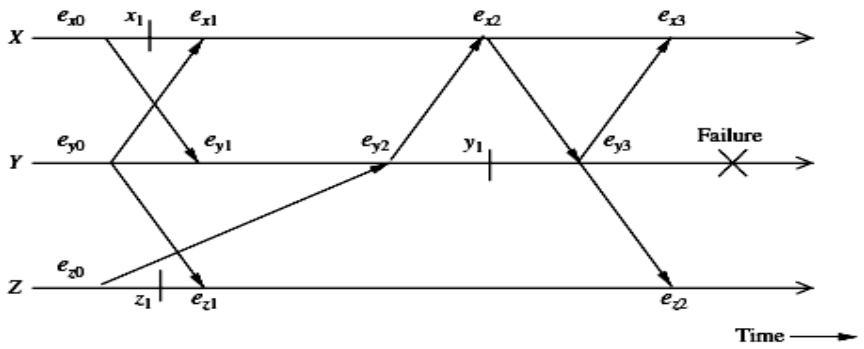


Fig 4.8 An example of the juang –venkatesan algorithm

Procedure RollBack_Recovery: processor p_i, executes the following:

STEP (a)

if processor p_i, is recovering after a failure then

$CkPt_i$; := latest event logged in the stable storage

else

$CkPt_i$, := latest event that took place in p_i, {The latest event at p_i, can be either in stable or in volatile storage.}

end if

STEP (b)

for k = 1 to N {N is the number of processors in the system} do

for each neighboring processor p_j; do

compute $SENTi \rightarrow j$ ($CkPt_i$)

send a ROLLBACK(i, $SENTi \rightarrow j$, ($CkPt_i$)) message to p_j end for

for every ROLLBACK(j, c) message received from a neighbor j do if RCVD i ← j (CkPt) > c {Implies the presence of orphan messages}

then

find the latest event e such that RCVD i ← j;_; (e) = c {Such an event e may be in the volatile storage or stable storage.}

$CkPt_i$, e

end if

10. what is checkpointing and why it is needed in distributed system? APR/MAY 2024

Checkpointing is a technique used in distributed systems to save the current state of a process or application at regular intervals, so that if a failure occurs, the system can recover from the last checkpoint instead of starting over from the beginning.

In a distributed system, where multiple nodes or processes work together to achieve a common goal, failures can occur due to various reasons such as:

- Node crashes
- Network partitions
- Software errors

Checkpointing is needed in distributed systems for several reasons:

1. Fault tolerance: Checkpointing allows the system to recover from failures without losing significant progress.
2. Reduced recovery time: By restoring from a checkpoint, the system can quickly recover and resume operation, reducing downtime.
3. Improved availability: Checkpointing helps ensure that the system remains available even in the presence of failures.
4. Simplified error handling: Checkpointing can simplify error handling by allowing the system to roll back to a known good state.

11. Explain the facts associated with agreement in failure free systems of both asynchronous and asynchronous system. NOV/DEC 2023

Asynchronous Systems:

1. No guarantee of termination: Asynchronous systems may not guarantee termination, making agreement more challenging.

2. Lack of global clock: Asynchronous systems don't have a global clock, making it difficult to coordinate actions.

3. Message delays: Messages may be delayed or lost, affecting agreement.

4. Non-determinism: Asynchronous systems can exhibit non-deterministic behavior, making agreement harder.

5. Leader election challenges: Electing a leader is more challenging in asynchronous systems.

6. Consensus algorithms: Algorithms like Paxos, Raft, and PBFT are designed for asynchronous systems.

Synchronous Systems:

1. Guaranteed termination: Synchronous systems guarantee termination, making agreement easier.

2. Global clock: Synchronous systems have a global clock, simplifying coordination.

3. Bounded message delays: Message delays are bounded, ensuring timely communication.

4. Determinism: Synchronous systems exhibit deterministic behavior, facilitating agreement.

5. Leader election simplicity: Leader election is simpler in synchronous systems.

6. Consensus algorithms: Algorithms like Two-Phase Commit (2PC) and Byzantine Agreement are designed for synchronous systems.

Common Facts:

1. Agreement is still challenging: Agreement remains a challenging problem in both asynchronous and synchronous systems.

2. Coordination is key: Coordination among nodes is crucial for achieving agreement.

3. Fault tolerance: Agreement algorithms must tolerate faults to ensure system reliability.

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.

PART A

1. Define Cloud Computing. (NOV/DEC 2021)

- Cloud computing is the delivery of various hardware and software services over the internet through remote servers. These servers are busy storing, managing, and processing data that enables users to expand or upgrade their infrastructure and retrieve files on demand.

2. Why Cloud Computing?

- ✓ Small as well as large IT companies, follow the traditional methods to provide the IT infrastructure. That means **for any IT company, we need a Server Room that is the basic need of IT companies**.
- ✓ In that server room, there should be a database server, mail server, networking, firewalls, routers, modem, switches, QPS (Query Per Second means how much queries or load will be handled by the server), configurable system, high net speed, and the maintenance engineers.
- ✓ To establish such IT infrastructure, we need to spend lots of money. To overcome all these problems and to reduce the IT infrastructure cost, Cloud Computing comes into existence.

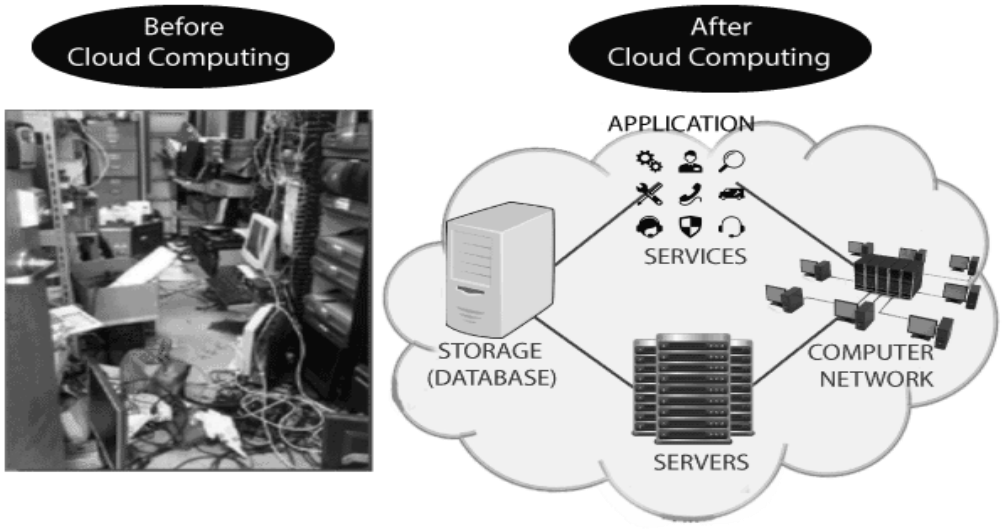


Fig: 5.1 Why Cloud Computing?

3. List the Characteristics of Cloud computing. (Nov/Dec 2020) , APR/MAY 2024

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

- ✓ No up-front commitments
- ✓ On-demand access, Universal access
- ✓ Pay-Per-Use scenario
- ✓ On-Demand self services
- ✓ Efficient resource allocation
- ✓ Energy efficiency
- ✓ Collaboration, Scalable Services

4. **What is the different deployment model of cloud computing?**

Various deployment models of cloud computing are

- ✓ Public Cloud
- ✓ Private Cloud
- ✓ Hybrid Cloud
- ✓ Community Cloud.

5. **Write short notes on Public cloud?**

Public cloud

- Services and Infrastructure are hosted on premise of cloud provider and are provisioned for open **use by general public**.
- The end users can access the services via **public network like internet**.

6. **Write short notes on Private cloud?**

Private cloud

- Private clouds are **designed and maintained by a single enterprise** to meet the specific needs of that enterprise.
- Private clouds need to set up a structure that is entirely built for a single business cloud solutions and that are either **hosted on-site** or in a specific **service provider's data center**.

7. **Write short notes on Hybrid cloud?**

Hybrid cloud

- Hybrid cloud computing is an environment that **combines public clouds and private clouds** by allowing data and applications to be shared between them.
- A hybrid cloud is ideal for scalability, flexibility, and security.

8. **Write short notes on Community cloud?**

Community cloud

- ✓ Community cloud is a cloud infrastructure that allows systems and services to be accessible by a **group of several organizations** to share the information.
- ✓ It is a mutually shared model between organizations that belong to a particular community such as **banks, government organizations, or commercial enterprises.**

9. **Bring out the differences between private cloud and public cloud.**

Private cloud	Public cloud
Single Client	Multiple Client
Hosted at Providers/ Orgs Location	Hosted at Providers Location
Access Over Internet/ Private Network	Access Over Internet
High cost	Low cost
High security	Shared infrastructure

10. How do you Compare Cloud Computing Deployment models.

Deployment Model	Scope of Services	Managed by	Security Level
Public model	General public and large industry groups	Cloud service provider	Low
Private model	Single organization	Single organization	High
Community model	Organization those share the same policy, mission and same security aspects	Several organization or Cloud service providers	High
Hybrid model	Organization and public	Organization and public	Medium.

11. What are the Cloud Service Models?

There are the following three types of cloud service models

1. Infrastructure as a Service (IaaS)
2. Platform as a Service (PaaS)
3. Software as a Service (SaaS).

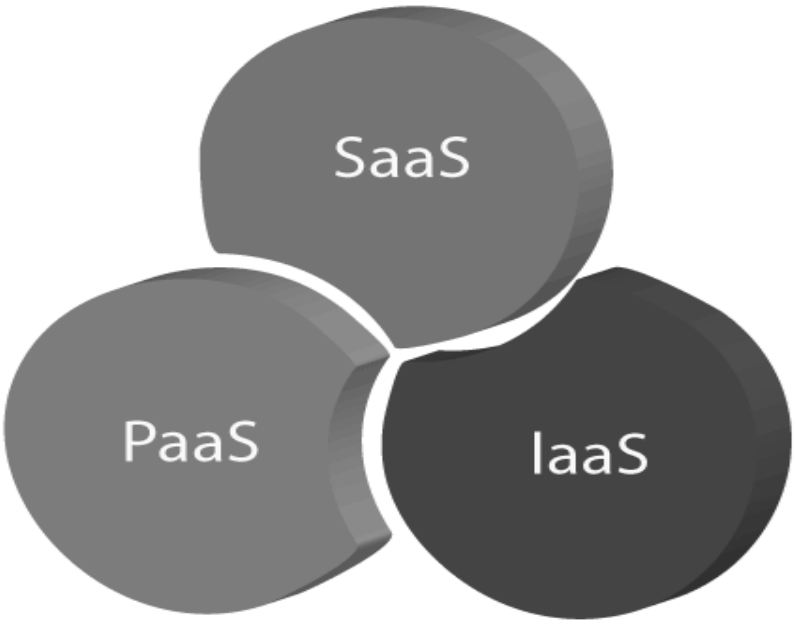


Fig: 5.2 - Cloud Service Models

12. What is Infrastructure as a Service (IaaS)?

- ✓ This model puts together infrastructures demanded by users—namely servers, storage, networks, and the data center fabric.
- ✓ The user can deploy and run on multiple VMs running guest OSes on specific applications.
- ✓ The user does not manage or control the underlying cloud infrastructure, but can specify when to request and release the needed resources.

13. What is Platform as a Service (PaaS)?

- ✓ This model enables the user to deploy user-built applications onto a virtualized cloud platform. PaaS includes middleware, databases, development tools, and some runtime support such as Web 2.0 and Java.

14. What is Software as a Service (SaaS)?

- ✓ This refers to browser-initiated application software over thousands of paid clouds customers. The SaaS model applies to business processes, industry applications, consumer relationship management (CRM), enterprise resources planning (ERP), human resources (HR), and collaborative applications.

15. What are the Differences between IaaS, PaaS, and SaaS?

IaaS	Paas	SaaS
It provides a virtual data center to store information and create platforms for app development, testing, and deployment.	It provides virtual platforms and tools to create, test, and deploy apps.	It provides web software and apps to complete business tasks.
It provides access to resources such as virtual machines, virtual storage, etc.	It provides runtime environments and deployment tools for applications.	It provides software as a service to the end-users.
It is used by network architects.	It is used by developers.	It is used by end users.
IaaS provides only Infrastructure.	PaaS provides Infrastructure+Platform.	SaaS provides Infrastructure+Platform +Software.

16. What are the advantages and disadvantages (Challenges) of Cloud Computing?

Advantages

- ✓ Lower-Cost Computers for Users
- ✓ Improved Performance
- ✓ Lower IT Infrastructure Costs
- ✓ Fewer Maintenance Issues
- ✓ Lower Software Costs
- ✓ Instant Software Updates

Disadvantages

- ✓ Requires a Constant Internet Connection
- ✓ Doesn’t Work Well with Low-Speed Connections
- ✓ Can Be Slow
- ✓ Features Might Be Limited
- ✓ Stored Data Might Not Be Secure.

17. What is meant by virtualization? (Or) Define Virtualization and why it is required in the larger organization (April/ May 2019) , (April/ May 2024)(NOV/DEC 2023)

Virtualization is the "creation of a virtual (rather than actual) version of something, such as a server, a desktop, a storage device, an operating system or network resources". In other words, Virtualization is a technique, which allows to share a single physical instance of a resource or an application among multiple customers and organizations.

Virtualization is required in larger organizations for several reasons:

- **Flexibility and Scalability**: Virtualization enables easy deployment, scaling, and management of virtual resources, making it ideal for dynamic environments.
- **Disaster Recovery:** Virtualization simplifies disaster recovery by allowing for easy replication and failover of virtual resources.
- **Security:** Virtualization provides an additional layer of security by isolating virtual resources from each other and the physical infrastructure.
- **Cost Savings:** Virtualization reduces hardware costs, power consumption, and maintenance requirements.

18. What is the concept behind the Virtualization?

Creation of a virtual machine over existing operating system and hardware is known as Hardware Virtualization. A Virtual machine provides an environment that is logically separated from the underlying hardware.

19. What are the Types of Virtualization?

There are six types of Virtualization

1. Application Virtualization
2. Network Virtualization
3. Desktop Virtualization
4. Storage Virtualization
5. Server Virtualization
6. Data virtualization.

20. What is Hardware Virtualization?

When the virtual machine software or virtual machine manager (VMM) is directly installed on the hardware system is known as hardware virtualization.

The main job of hypervisor is to control and monitoring the processor, memory and other hardware resources.

21. What is Operating System Virtualization?

When the virtual machine software or virtual machine manager (VMM) is installed on the Host operating system instead of directly on the hardware system is known as operating system virtualization.

Usage:

Operating System Virtualization is mainly used for testing the applications on different platforms of OS.

22. What is Server Virtualization?

When the virtual machine software or virtual machine manager (VMM) is directly installed on the Server system is known as server virtualization.

Usage:

Server virtualization is done because a single physical server can be divided into multiple servers on the demand basis and for balancing the load.

23. What is Storage Virtualization?

Storage virtualization is the process of grouping the physical storage from multiple network storage devices so that it looks like a single storage device.

Storage virtualization is also implemented by using software applications.

Usage:

Storage virtualization is mainly done for back-up and recovery purposes.

24. What are the main benefits of both scalability and elasticity?

- **Cost-effectiveness:** Cloud scalability and cloud elasticity features constitute an effective resource management strategy
- **The pay-per-use model:** makes cloud elasticity the proper answer for sudden surges of workload demand (vital for streaming services and marketplaces)
- **The pay-as-you-expand model:** allows to plan out gradual growth of the infrastructure in sync with growing requirements (especially handy for ad tech systems)

25. What are the types of cloud scalability?

There are several types of cloud scalability.

Vertical: aka Scale-Up - the ability to handle an increasing workload by adding resources to the existing infrastructure. It is a short term solution to cover immediate needs.

Horizontal: aka Scale-Out - the expansion of the existing infrastructure with new elements to tackle more significant workload requirements.

Diagonal scalability is a more flexible solution that combines adding and removal of resources according to the current workload requirements. It is the most cost-effective scalability solution by far.

26. What is Cloud Scalability?

Cloud scalability is the ability of the system's infrastructure to handle growing workload

- ✓ Requirements while retaining a consistent performance adequately.

27. What is the difference between Cloud Elasticity and Cloud Scalability?

Cloud Elasticity	Cloud Scalability
Cloud Elasticity is a tactical resource allocation operation. It provides the Necessary resources required for the current Task and handles varying loads for short periods. For example, running as entailment Analysis algorithm, doing database backups Or just taking on user website	Cloud Scalability is a strategic resource Allocation operation. Scalability handles the Increase and decrease of resources according To the system's workload demands.

28. Define Replication in Cloud Computing.

Replication in Cloud Computing refers to multiple storage of the same data copy of a file (copy) to several different locations by usually synchronization of these data sources. Replication in Cloud Computing is partly done for backup and on the other hand to reduce response times, especially for reading data requests.

29. What is Cloud Monitoring?

Cloud monitoring is the process of reviewing and managing the operational workflow and processes within a cloud infrastructure or asset. It's generally implemented through automated monitoring software that gives central access and control over the cloud infrastructure. Admin can review the operational status and health of cloud servers and components.

30. How Cloud Monitoring Works?

The cloud has many moving parts, and it's important to ensure everything works together seamlessly to optimize performance.

- **Website monitoring**: Tracking the processes, traffic, availability and resource utilization of cloud-hosted websites
- **Virtual machine monitoring:** Monitoring the virtualization infrastructure and individual virtual machines

31. What are the Cloud Monitoring Capabilities?

Cloud monitoring makes it easier to identify patterns and discover potential security risks in the infrastructure.

- ✓ Ability to monitor large volumes of cloud data across many distributed locations
- ✓ Gain visibility into application, user, and file behavior to identify potential attacks or compromises

32. What is a cloud service?

The term "cloud services" refers to a wide range of services delivered on demand to companies and customers over the internet. These services are designed to provide easy, affordable access to applications and resources, without the need for internal infrastructure or hardware.

33. What are the types of Cloud service model?

- ✓ Infrastructure as a Service
- ✓ Platform as a Service
- ✓ Software as a Service.

34. What is Infrastructure as a Service (IaaS)?

- ➢ This model puts together infrastructures demanded by users—namely servers, storage, networks, and the data center fabric.
- ➢ The user can deploy and run on multiple VMs running guest OS on specific applications.

35. What is Platform as a Service (PaaS)?

- ➢ This model enables the user to deploy user-built applications onto a virtualized cloud platform.
- ➢ PaaS includes middleware, databases, development tools, and some runtime support such as Web 2.0 and Java.

36. What is Software as a Service (SaaS)?

- ➢ This refers to browser-initiated application software over thousands of paid cloud customers.
- ➢ The SaaS model applies to business processes, industry applications, consumer relationship management (CRM), enterprise resources planning (ERP), human resources (HR), and collaborative applications.

37. What are compute services?

Compute services are also known as Infrastructure-as-a-Service (IaaS). Compute platforms, such as AWS Compute, supply a virtual server instance and storage and APIs that let users migrate workloads to a virtual machine.

38. Define storage service in cloud computing.

- ➢ A cloud storage service is a business that maintains and manages its customers' data and makes that data accessible over a network, usually the internet.
- ➢ Most of these types of services are based on a utility storage model.

39. What are the types of storage services?

- ✓ Public cloud storage
- ✓ Private cloud storage
- ✓ Hybrid cloud storage.

40. Define Public cloud storage.

Public cloud storage is a service owned and operated by a provider. It is usually suitable for unstructured data that is not subject to constant change. The infrastructure usually consists of inexpensive storage nodes attached to commodity drives.

41. Define Private cloud storage.

Private cloud storage services address the data safety and performance concerns of public cloud storage by bringing cloud storage inside an organization. A private cloud storage service is more suitable for actively used data and data that an organization needs more control over.

42. Define Hybrid cloud storage.

Hybrid cloud storage model that stores unstructured data for backup and archiving purposes, for example and less sensitive data with a public cloud provider, while a private cloud is used for active, structured and more sensitive data.

43. How do you define the term Load Balancing in Cloud Computing?

Load balancing is the method that allows you to have a proper balance of the amount of work being done on different pieces of device or hardware equipment.

Cloud load balancing is defined as dividing workload and computing properties in cloud computing..

44. What are the different types of Load Balancing?

- Network Load Balancing
- HTTP(S) load balancing
- Internal Load Balancing

45. What are the different Types of Load Balancing Algorithms used in Cloud Computing?

1. Static Algorithm
2. Dynamic Algorithm
3. Round Robin Algorithm
4. Weighted Round Robin Load Balancing Algorithm
5. Opportunistic Load Balancing Algorithm
6. Minimum To Minimum Load Balancing Algorithm.

46. Why load balancing is important in cloud computing?

- ✓ Offers better performance
- ✓ Helps Maintain Website Traffic
- ✓ Can Handle Sudden Bursts in Traffic
- ✓ Greater Flexibility.

47. What is Scalability?

In the context of cloud computing, scalability is the ability of a system to add, remove, or reconfigure the hardware, software, and other resources to handle an increase or decrease in usage. This allows a system to meet the demands of a variable workload.

48. What is Elasticity?

Elasticity, on the other hand, refers to a system's ability to automatically scale up or down resources to meet user demands. This scalability can occur without manual intervention, meaning that a system can expand or contract resources independently when needed.

49. What are the cloud service platforms?

There are several cloud service platforms some of the following:

- ✓ Amazon Web Services (AWS),
- ✓ Microsoft Azure,
- ✓ Google Cloud Platform (GCP),
- ✓ Alibaba Cloud,
- ✓ Oracle Cloud,
- ✓ IBM Cloud (Kyndryl),

50. State the relation of compute service with storage service in a cloud environment Nov/Dec 2023

In a cloud environment, Compute and Storage services are closely related and interdependent. Here's how:

1. Data Processing: Compute services (e.g., virtual machines, containers) process data, which is typically stored in Storage services (e.g., object storage, block storage).

2. Data Storage: Storage services provide a repository for data, which is then accessed and processed by Compute services.

3. Input/Output Operations: Compute services perform input/output (I/O) operations on Storage services, reading and writing data as needed.

PART B

1. Define cloud computing. List out characteristics of cloud computing.

- Cloud computing is the delivery of different services through the Internet. These resources include tools and applications like data storage, servers, databases, networking, and software.
- Rather than keeping files on a proprietary hard drive or local storage device, cloud-based storage makes it possible to save them to a remote database/server.
- Cloud computing is a popular option for people and businesses for a number of reasons including cost savings, increased productivity, speed and efficiency, performance, and security.

Characteristics of Cloud Computing

1. On demand self service
2. Broad Network Access
3. Resource Pooling
4. Rapid Elasticity - Horizontal & Vertical Scaling
5. Measured Service
6. Performance
7. Reduced Costs

On-demand self service

Cloud computing resources can be provisioned on-demand by the users, without requiring interactions with the cloud service provider. The process of provisioning resources is automated.

Broad network access

Cloud computing resources can be accessed over the network using standard access mechanisms that provide platform-independent access through the use of heterogeneous client platforms such as workstations, laptops, tablets and smartphones.

Resource pooling

The computing and storage resources provided by cloud service providers are pooled to serve multiple users using multi-tenancy. Multi-tenant aspects of the cloud allow multiple users to be served by the same physical hardware. Users are assigned virtual resources that run on top of the physical resources. Various forms of virtualization approaches such as full virtualization, para-virtualization and hardware virtualization are described in Chapter 2.

Rapid elasticity

Cloud computing resources can be provisioned rapidly and elastically. Cloud resources can be rapidly scaled up or down based on demand. Two types of scaling options exist:

- Horizontal Scaling (scaling out): Horizontal scaling or scaling-out involves launching and provisioning additional server resources.
- Vertical Scaling (scaling up): Vertical scaling or scaling-up involves changing the computing capacity assigned to the server resources while keeping the number of server resources constant.

Measured service

Cloud computing resources are provided to users on a pay-per-use model. The usage of the cloud resources is measured and the user is charged based on some specific metric. Metrics such as amount of CPU cycles used, amount of storage space used, number of network I/O requests, etc.

Performance

Cloud computing provides improved performance for applications since the resources available to the applications can be scaled up or down based on the dynamic application workloads.

Reduced costs

Cloud computing provides cost benefits for applications as only as much computing and storage resources as required can be provisioned dynamically, and upfront investment in purchase of computing assets to cover worst case requirements is avoid. This saves significant cost for organizations and individuals. For example, e-Commerce applications typically experience higher workloads in holiday seasons. To ensure market readiness of such applications, adequate resources need to be provisioned so that the applications can meet the demands of specified workload levels and at the same time ensure that service level agreements are met.

2. Discuss in detail about Cloud deployment models. (Or) Outline the various Cloud deployment models of cloud with neat sketch and identify which among them could be applied to formulate cloud structure for a small firm (Or) Explain the types of cloud computing deployment models with their relative advantages and disadvantages (Nov/Dec 2021) , (APR/MAY 2024)

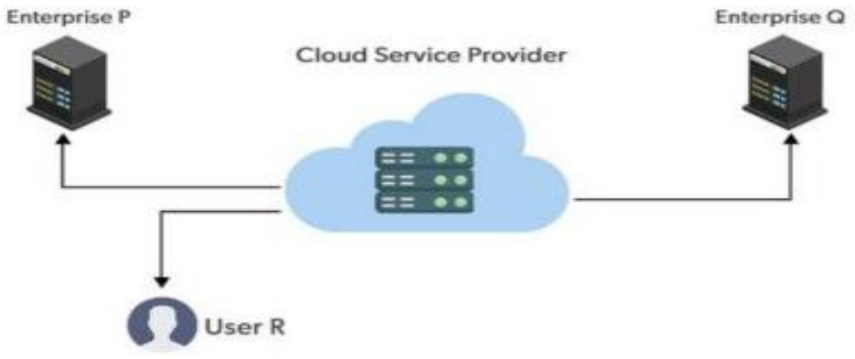


Fig 5.1.1 Deployment Models - Public Cloud

Advantages of the Public Cloud Model

- Minimal Investment: Because it is a pay-per-use service, there is no substantial upfront fee, making it excellent for enterprises that require immediate access to resources.
- No setup cost: The entire infrastructure is fully subsidized by the cloud service providers, thus there is no need to set up any hardware.
- Infrastructure Management is not required: Using the public cloud does not necessitate infrastructure management.
- No maintenance: The maintenance work is done by the service provider (not users).
- Dynamic Scalability: To fulfill your company's needs, on-demand resources are accessible in above Fig 5.1.1

Disadvantages of the Public Cloud Model

- Less secure: Public cloud is less secure as resources are public so there is no guarantee of high-level security.
- Low customization: It is accessed by many public so it can't be customized according to personal requirements.

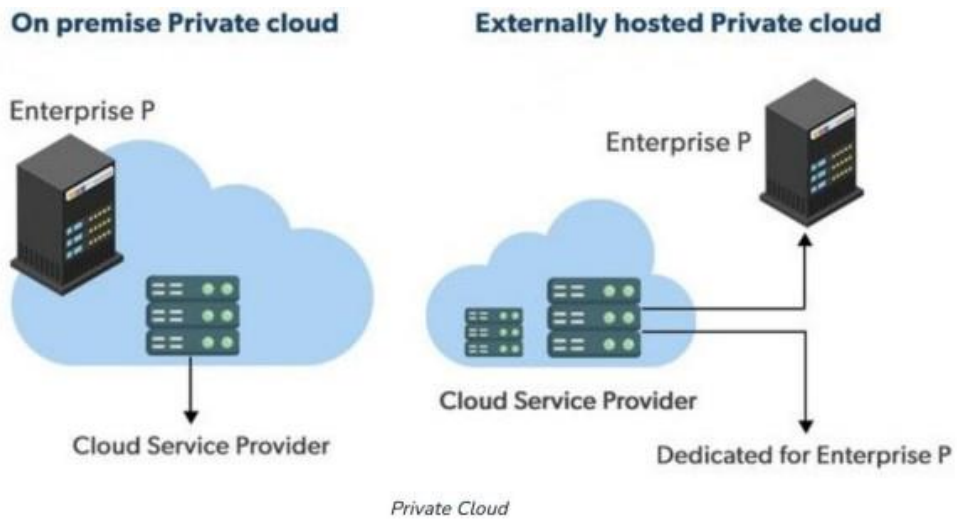


Fig 5.1.2 Deployment Models - Private Cloud

Advantages of the Private Cloud Model

- Better Control: You are the sole owner of the property. You gain complete command over service integration, IT operations, policies, and user behavior.

- Data Security and Privacy: It's suitable for storing corporate information to which only authorized staff have access. By segmenting resources within the same infrastructure, improved access and security can be achieved in above Fig 5.1.2
- Supports Legacy Systems: This approach is designed to work with legacy systems that are unable to access the public cloud.
- Customization: Unlike a public cloud deployment, a private cloud allows a company to tailor its solution to meet its specific needs.

Disadvantages of the Private Cloud Model

- Less scalable: Private clouds are scaled within a certain range as there is less number of clients.
- Costly: Private clouds are more costly as they provide personalized facilities.

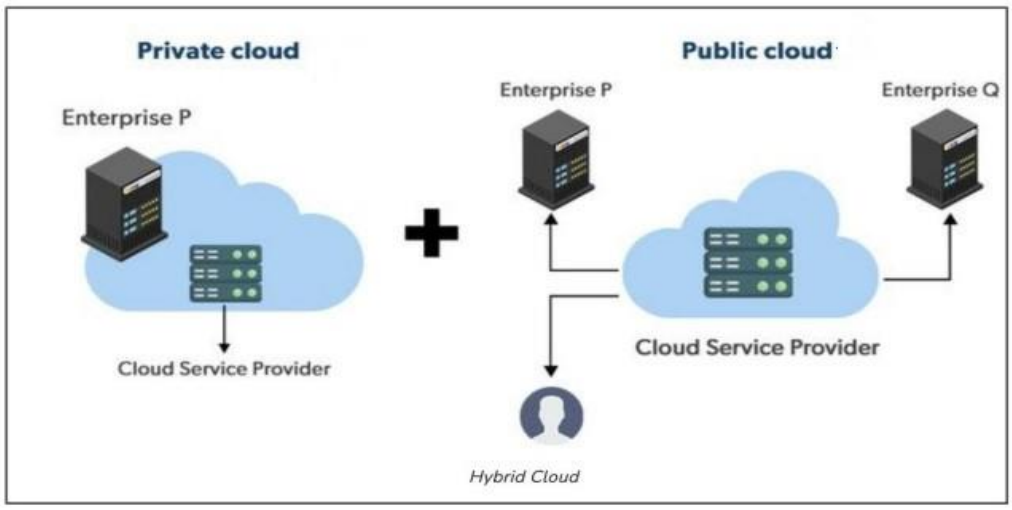


Fig 5.1.3 Deployment Models - Hybrid Cloud

Advantages of the Hybrid Cloud Model

- Flexibility and control: Businesses with more flexibility can design personalized solutions that meet their particular needs.
- Cost: Because public clouds provide scalability, you'll only be responsible for paying for the extra capacity if you require it.
- Security: Because data is properly separated, the chances of data theft by attackers are considerably reduced in above Fig 5.1.3

Disadvantages of the Hybrid Cloud Model

- Difficult to manage: Hybrid clouds are difficult to manage as it is a combination of both public and private cloud. So, it is complex.
- Slow data transmission: Data transmission in the hybrid cloud takes place through the public cloud so latency occurs.

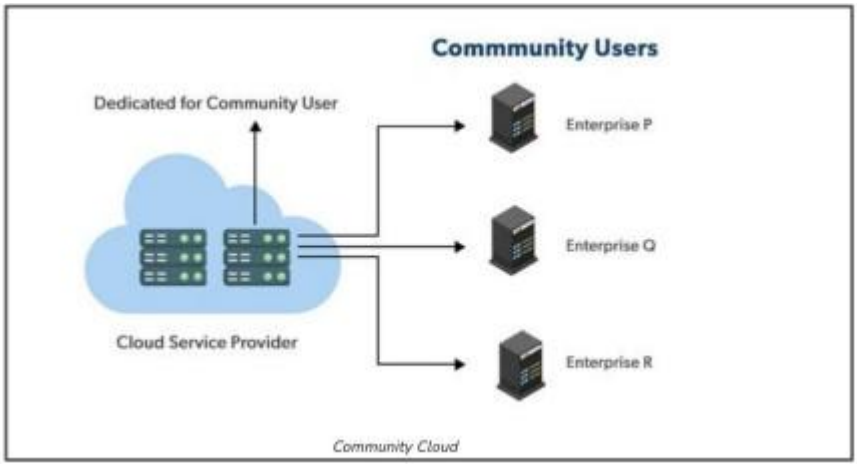


Fig 5.1.4 Deployment Models - Community Cloud

Advantages of the Community Cloud Model

- Cost Effective: It is cost-effective because the cloud is shared by multiple organizations or communities in above Fig 5.1.4
- Security: Community cloud provides better security.
- Shared resources: It allows you to share resources, infrastructure, etc. with multiple organizations.
- Collaboration and data sharing: It is suitable for both collaboration and data sharing.

Disadvantages of the Community Cloud Model

- Limited Scalability: Community cloud is relatively less scalable as many organizations share the same resources according to their collaborative interests.
- Rigid in customization: As the data and resources are shared among different organizations according to their mutual interests if an organization wants some changes according to their needs they cannot do so because it will have an impact on other organizations.

3. How the service models are categorized in cloud computing? (or) Explain in detail about service models in cloud computing.

There are the following three types of cloud service models

1. Infrastructure as a Service (IaaS)
2. Platform as a Service (PaaS)
3. Software as a Service (SaaS).

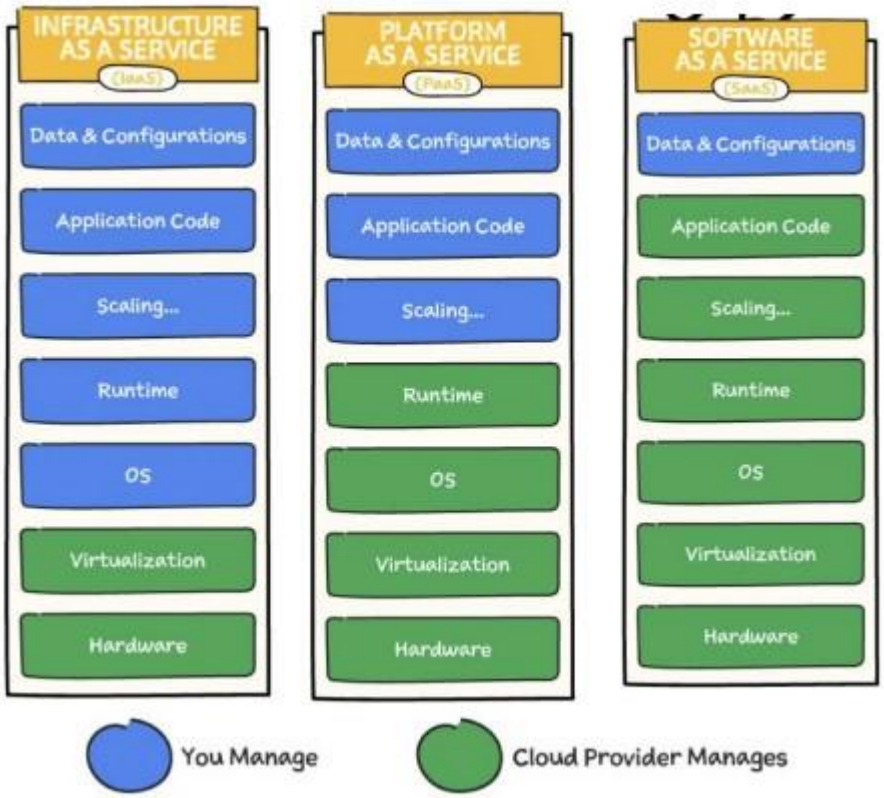


Fig: 5.2 - Cloud Service Models

Software as a Service (SaaS)

Software-as-a-Service (SaaS) is a way of delivering services and applications over the Internet. Instead of installing and maintaining software, we simply access it via the Internet, freeing ourselves from the complex software and hardware management

SaaS pros

- Easy to set up and start using
- The provider manages and maintains everything, from hardware to software
- Software is accessible over any internet connection on any device

SaaS cons

- No control over any of the infrastructure or security controls Integration issues with your existing tools and applications
- Vendor lock-in may be an issue depending on the cloud service provider
- Little to no customization

Platform as a Service

PaaS is a category of cloud computing that provides a platform and environment to allow developers to build applications and services over the internet. PaaS services are hosted in the cloud and accessed by users simply via their web browser.

PaaS pros

- Instant access to a complete, easy-to-use development platform
- Cloud service provider is responsible for maintenance and securing infrastructure
- Available over any internet connection on any device
- On-demand scalability

PaaS cons

- Application stack can be limited to the most relevant compo • Vendor lock-in may be an issue depending on the cloud service provider
- Less control over operations and the overall infrastructure
- More limited customizations

Infrastructure as a Service

Infrastructure as a service (IaaS) is a service model that delivers computer infrastructure on an outsourced basis to support various operations.

laaS pros

- Highest level of control over infrastructure
- On-demand scalability
- No single point of failure for higher reliability
- Reduced upfront capital expenditures (for example, pay-as-you-go pricing)
- Fewer provisioning delays and wasted resources
- Accelerated development and time to market

laaS cons

- Responsible for your own data security and recovery
- Requires hands-on configuration and maintenance
- Difficulties securing legacy applications on cloud-based infrastructure

4. What are the Driving Factors or the key advantages of cloud computing?

There are several driving factors in cloud computing. Some of following

- ✓ Saving costs
- ✓ Safety
- ✓ System mobility
- ✓ Improved insight
- ✓ Boost collaboration
- ✓ Quality control
- ✓ Loss & recovery
- ✓ Faster time to market.

- **Saving costs:** By adopting modern CSPs and methods, you will drive down the costs associated with business operations and IT infrastructure. While it may bear a notable cost to set everything up, the ROI you will enjoy as a result of safety, fluency, efficiency, and operational vision will pay back tenfold over the years.
- **Safety:** As we mentioned earlier, cyber security is on the rise. But if you can overcome any roadblocks related to data breaches, you will create a network that will fortify every sensitive area of your business, protecting yourself from the threat of devastating cyber attacks.
- **System mobility:** Another key benefit of using modern CSP functionality is the ability to access key data and system functionality from multiple devices wherever you are in the world, 24/7. Gaining this increased mobility will increase productivity, make your organization more responsive, and boost employee satisfaction by giving them the tools to perform better at their jobs.
- **Improved insight:** In the modern age, knowledge equals power, and data equals money. By working with CSP-based technologies, you can gain improved access to data-driven insights in a way that is centralized, efficient, and digestible. As such, with CSPs and modern IT-based capabilities, it's possible to drive innovation by achieving greater business intelligence (BI).
- **Boost collaboration:** A big part of the success of an organization relies on collaboration. After all, the different departments and teams are all working towards the same general company goals. Cloud computing can help boost collaboration through more innovative and engaging channels. This way, they'll avoid tedious in-presence meetings and develop joint strategies in an online environment.

- **Quality control:** If you are a regular reader of this blog, then you must be aware of the consequences bad quality data and reporting can bring to an organization's growth efforts. Cloud-based services store all relevant data in one secure location with a unique format. This way, you can ensure consistency, avoid manual errors, and keep close control of any accesses or manipulations of critical business data.

- **Loss & recovery:** No matter how efficient your company is at managing all its data-related processes, accidents and issues can still happen. On-premise solutions often require long recovery processes, and even then, it is not a given that you'll be able to recover what you lost. On the other hand, cloud-based solutions offer quick data recovery processes for multiple emergency situations, such as server cyber attacks or regular outages.

- **Faster time to market:** Last but not least, cloud computing sets the perfect environment for innovation at a faster and more efficient pace. You can think of a new idea, develop it, test it, and launch it, or retire it immediately if it doesn't work. There is no need to design complex applications or go through tedious processes.

5. What are the Challenges/ disadvantages/issues Of Cloud Computing?

1. Cyber security issues
2. Cost management and containment
3. Lack of resources/expertise
4. Governance/Control
5. Compliance
6. Network dependence
7. Managing multiple clouds
8. Performance

1. Cyber security issues

Security is a pressing concern in the world of cloud-based computing, as you are unable to see the exact location where your data is stored or being processed. This increases the risks that can arise during the implementation or management process.

The main concerns surrounding cyber threats

- ✓ Compromised credentials
- ✓ Broken authentication
- ✓ Human error
- ✓ Mass sensitive data breaches
- ✓ Hacked interfaces and APIs
- ✓ Account hijacking.

2. Cost management and containment

The next part of our cloud computing risks list involves costs. As mentioned before, expenses have been cited as the biggest cloud-related issue for organizations of all sizes today. It is not a surprise that the cloud industry is still growing at an increasing pace, with new tools and technologies emerging every day.

3. Lack of resources/expertise

One of the cloud computing challenges companies and enterprises are facing today is a lack of resources and/or expertise. This was already an issue before the pandemic and became an even bigger one after it as cloud technologies rapidly advanced during that period, leading organizations to lose revenue and even market share because of the lack of technical skills to manage their demands.

4. Governance/Control

Moving on with the problems of cloud computing, we have control in place four. Proper IT governance should ensure IT assets are implemented and used according to agreed-upon policies and procedures, ensure that these assets are properly controlled and maintained, and ensure that these assets are supporting your organization's strategy and goals.

5. Compliance

One of the risks of cloud computing is compliance. That is an issue for anyone using backup services or storage. Every time a company moves data from internal storage to a cloud, it is faced with being compliant with official regulations and laws.

6. Network dependence

Next, in our list of challenges and risks in cloud computing, we have network dependence. In order for businesses to successfully transfer massive volumes of data in real-time to and from the cloud internet bandwidth plays a key role. However, this is not always a given, as internet can present problems and unexpected outages that can significantly harm the business.

7. Managing multiple clouds

Cloud computing challenges haven't just been concentrated in one single cloud.

The state of multi-cloud has grown exponentially in recent years. Companies are shifting or combining public and private clouds, and, as mentioned earlier, tech giants like Alibaba and Amazon are leading the way.

8. Performance

When an organization moves to the cloud, it becomes dependent on the service providers. The next prominent challenge expands on this partnership. Nevertheless, this partnership often provides businesses with innovative technologies they wouldn't otherwise be able to access.

6. Explain in detail about Virtualization in Cloud Computing.

● Virtualization is a technique how to separate a service from the underlying physical delivery of that service. It is the process of creating a virtual version of something like computer hardware.

● It was initially developed during the mainframe era. It involves using specialized software to create a virtual or software-created version of a computing resource rather than the actual version of the same resource.

● With the help of Virtualization, multiple operating systems and applications can run on the same machine and its same hardware at the same time, increasing the utilization and flexibility of hardware in below Fig 5.3.1

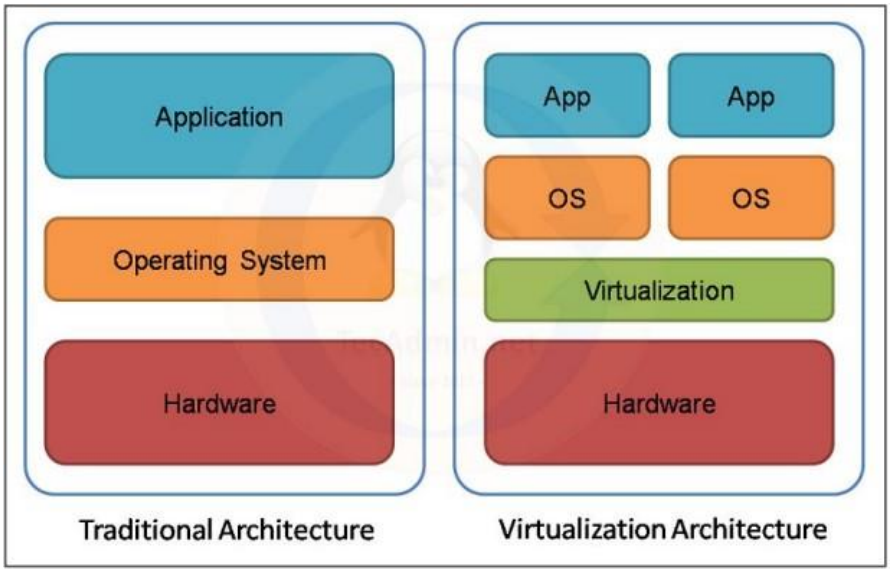


Fig 5.3.1 Architecture

Hypervisor

- A hypervisor, also known as a virtual machine monitor or VMM, is software that creates and runs virtual machines (VMs).
- A hypervisor allows one host computer to support multiple guest VMs by virtually sharing its resources, such as memory and processing.
- Type 1 and Type 2

- The type 1 hypervisor sits on top of the bare metal server and has direct access to the hardware resources. Because of this, the type 1 hypervisor is also known as a bare metal hypervisor.
- In contrast, the type 2 hypervisor is an application installed on the host operating system.

1. Full Virtualization:

Full Virtualization was introduced by IBM in the year 1966. It is the first software solution for server virtualization and uses binary translation and direct approach techniques. In full virtualization, guest OS is completely isolated by the virtual machine from the virtualization layer and hardware. Microsoft and Parallels systems are examples of full virtualization in below Fig 5.3.2

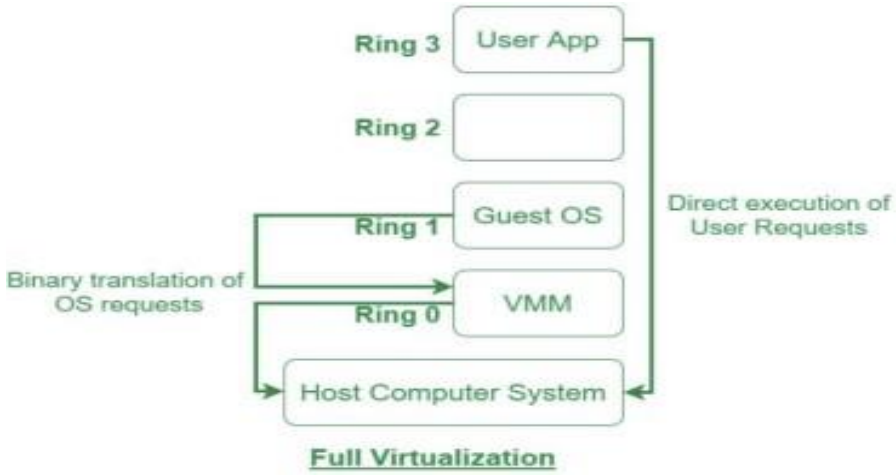


Fig 5.3.2 Full virtualization

2. Paravirtualization:

Para virtualization is the category of CPU virtualization which uses hypercalls for operations to handle instructions at compile time. In para virtualization, guest OS is not completely isolated but it is partially isolated by the virtual machine from the virtualization layer and hardware. VMware and Xen are some examples of para virtualization in below Fig 5.3.3

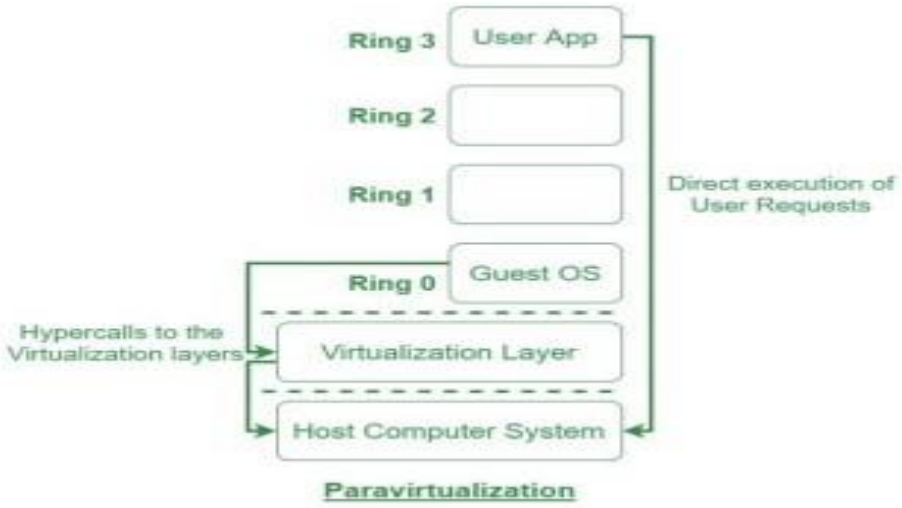


Fig 5.3.3 Para virtualization

Difference between Full Virtualization and Paravirtualization

S.No.	Full Virtualization	Paravirtualization
1.	In Full virtualization, virtual machines permit the execution of the instructions with the running of unmodified OS in an entirely isolated way.	In paravirtualization, a virtual machine does not implement full isolation of OS but rather provides a different API which is utilized when OS is subjected to alteration.
2.	Full Virtualization is less secure.	While the Paravirtualization is more secure than the Full Virtualization.
3.	Full Virtualization uses binary translation and a direct approach as a technique for operations.	While Paravirtualization uses hypercalls at compile time for operations.
4.	Full Virtualization is slow than paravirtualization in operation.	Paravirtualization is faster in operation as compared to full virtualization.
5.	Full Virtualization is more portable and compatible.	Paravirtualization is less portable and compatible.
6.	Examples of full virtualization are Microsoft and Parallels systems.	Examples of paravirtualization are Microsoft Hyper-V, Citrix Xen, etc.
7.	It supports all guest operating systems without modification.	The guest operating system has to be modified and only a few operating systems support it.

7. What are the types of Virtualization?

There are six types of Virtualization

1. Application Virtualization
2. Network Virtualization
3. Desktop Virtualization
4. Storage Virtualization
5. Server Virtualization
6. Data virtualization

1. Application Virtualization: Application virtualization helps a user to have remote access to an application from a server. The server stores all personal information and other characteristics of the application but can still run on a local workstation through the internet. An example of this would be a user who needs to run two different versions of the same software. Technologies that use application virtualization are hosted applications and packaged applications.

2. Network Virtualization: The ability to run multiple virtual networks with each having a separate control and data plan. It co-exists together on top of one physical network. It can be managed by individual parties that are potentially confidential to each other. Network virtualization provides a facility to create and provision virtual networks, logical switches, routers, firewalls, load balancers, Virtual Private Networks (VPN), and workload security within days or even weeks.

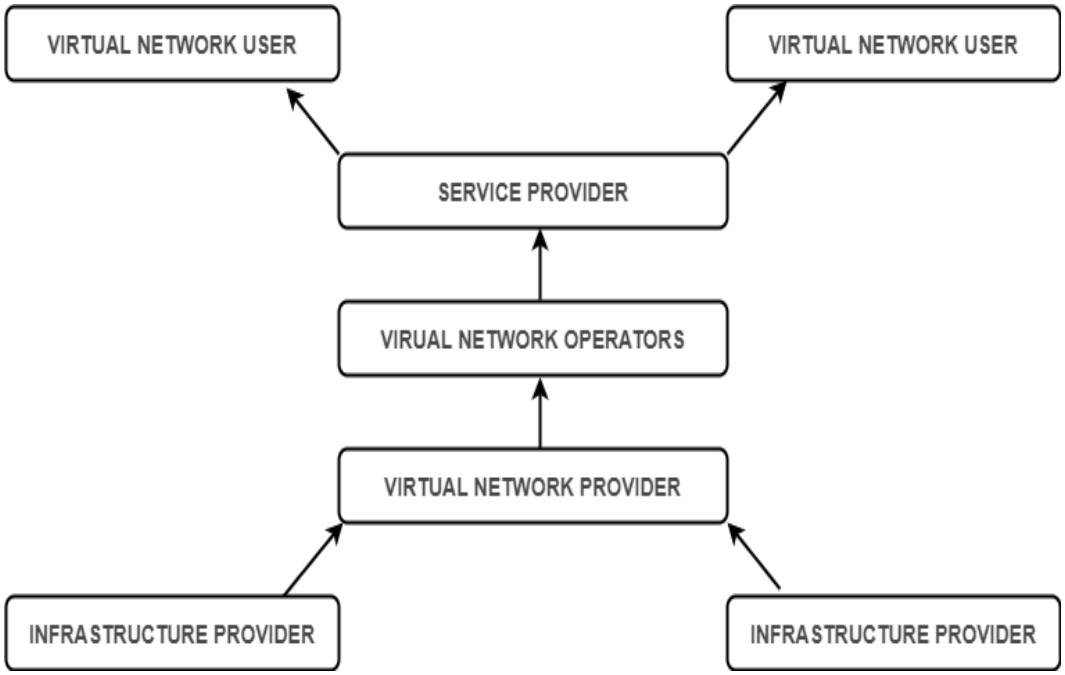


Fig 5.4.1 Network Virtualization

3. Desktop Virtualization: Desktop virtualization allows the users' OS to be remotely stored on a server in the data center. It allows the user to access their desktop virtually, from any location by a different machine. Users who want specific operating systems other than Windows Server will need to have a virtual desktop. The main benefits of desktop virtualization are user mobility, portability, and easy management of software installation, updates, and patches in above Fig 5.4.1

4. Storage Virtualization: Storage virtualization is an array of servers that are managed by a virtual storage system. The servers aren't aware of exactly where their data is stored and instead function more like worker bees in a hive. It makes managing storage from multiple sources be managed and utilized as a single repository. Storage virtualization software maintains smooth operations, consistent performance, and a continuous suite of advanced functions despite changes, breaks down, and differences in the underlying equipment.

5. Server Virtualization: This is a kind of virtualization in which the masking of server resources takes place. Here, the central server (physical server) is divided into multiple different virtual servers by changing the identity number, and processors. So, each system can operate its operating systems in an isolated manner. Where each sub-server knows the identity of the central server. It causes an increase in performance and reduces the operating cost by the deployment of main server resources into a sub-server resource. It's beneficial in virtual migration, reducing energy consumption, reducing infrastructural costs, etc.

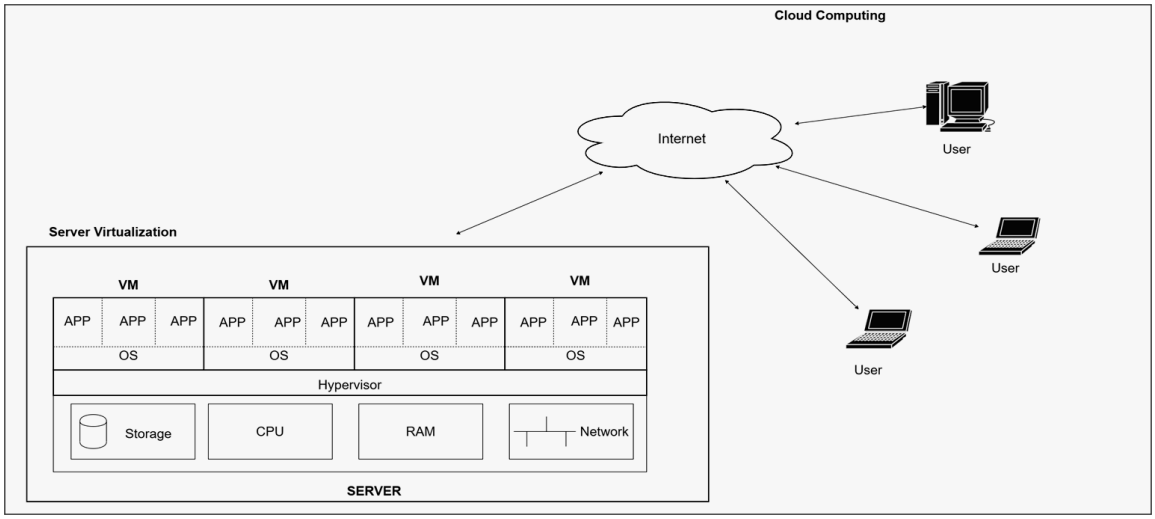


Fig 5.4.2 Server Virtualization

6. Data Virtualization: This is the kind of virtualization in which the data is collected from various sources and managed at a single place without knowing more about the technical information like how data is collected, stored & formatted then arranged that data logically so that its virtual view can be accessed by its interested people and stakeholders, and users through the various cloud

services remotely. Many big giant companies are providing their services like Oracle, IBM, At scale, Cdata, etc in above Fig 5.4.2

8. Explain in detail about Cloud Elasticity and Cloud Scalability.

- Elasticity refers to the ability of a cloud to automatically expand or compress the infrastructural resources on a sudden up and down in the requirement so that the workload can be managed efficiently.
- This elasticity helps to minimize infrastructural costs.
- This is not applicable for all kinds of environments, it is helpful to address only those scenarios where the resource requirements fluctuate up and down suddenly for a specific time interval.
- It is not quite practical to use where persistent resource infrastructure is required to handle the heavy workload.

	Cloud Elasticity	Cloud Scalability
1	Elasticity is used just to meet the sudden up and down in the workload for a small period of time.	Scalability is used to meet the static increase in the workload.
2	Elasticity is used to meet dynamic changes, where the resources need can increase or decrease.	Scalability is always used to address the increase in workload in an organization.
3	Elasticity is commonly used by small companies whose workload and demand increases only for a specific period of time.	Scalability is used by giant companies whose customer circle persistently grows in order to do the operations efficiently.
4	It is a short term planning and adopted just to deal with an unexpected increase in demand or seasonal demands.	Scalability is a long term planning and adopted just to deal with an expected increase in demand.

9. What are the Cloud Services and Platforms? (Or) Explain the different cloud services and the platform that support these services. Give an example for application service adapted by any organization NOV/DEC 2023

There are several cloud service platforms some of the following:

- ✓ Amazon Web Services (AWS),
- ✓ Google AppEngine
- ✓ Microsoft Azure,
- ✓ Hadoop

- ✓ Force.com and Salesforce.com.

Amazon Web Services (AWS**)**

- AWS provides different wide-ranging clouds IaaS services, which ranges from virtual compute, storage, and networking to complete computing stacks.
- AWS is well known for its storage and compute on demand services, named as Elastic Compute Cloud (EC2) and Simple Storage Service (S3).
- EC2 offers customizable virtual hardware to the end user which can be utilize as the base infrastructure for deploying computing systems on the cloud.
- It is likely to choose from a large variety of virtual hardware configurations including GPU and cluster instances. Either the AWS console, which is a wide-ranged Web portal for retrieving AWS services, or the web services API available for several programming language is used to deploy the EC2 instances.
- EC2 also offers the capability of saving an explicit running instance as image, thus allowing users to create their own templates for deploying system.
- S3 stores these templates and delivers persistent storage on demand. S3 is well ordered into a bucket which contains objects that are stored in binary form and can be grow with attributes.
- End users can store objects of any size, from basic file to full disk images and have them retrieval from anywhere. In addition, EC2 and S3, a wide range of services can be leveraged to build virtual computing system including: networking support, caching system, DNS, database support, and others.

Google AppEngine

- Google AppEngine is a scalable runtime environment frequently dedicated to executing web applications.

- These utilize benefits of the large computing infrastructure of Google to dynamically scale as per the demand. AppEngine offers both a secure execution environment and a collection of which simplifies the development if scalable and high-performance Web applications.

- These services include: in-memory caching, scalable data store, job queues, messaging, and corn tasks. Developers and Engineers can build and test applications on their own systems by using the AppEngine SDK, which replicates the production runtime environment, and helps test and profile applications.

- On completion of development, Developers can easily move their applications to AppEngine, set quotas to containing the cost generated, and make it available to the world. Currently, the supported programming languages are Python, Java, and Go.

Microsoft Azure

- Microsoft Azure is a Cloud operating system and a platform in which user can develop the applications in the cloud. Generally, a scalable runtime environment for web applications and distributed applications is provided.
- Application in Azure is organized around the fact of roles, which identify a distribution unit for applications and express the application's logic.
- Azure provides a set of additional services that complement application execution such as support for storage, networking, caching, content delivery, and others.

Hadoop

- Apache Hadoop is an open source framework that is appropriate for processing large data sets on commodity hardware. Hadoop is an implementation of MapReduce, an application programming model which is developed by Google.

- This model provides two fundamental operations for data processing: map and reduce. Yahoo! Is the sponsor of the Apache Hadoop project, and has put considerable effort in transforming the project to an enterprise-ready cloud computing platform for data processing.

- Hadoop is an integral part of the Yahoo! Cloud infrastructure and it supports many business processes of the corporate. Currently, Yahoo! Manges the world's largest Hadoop cluster, which is also available to academic institutions.

Force.com and Salesforce.com

- Force.com is a Cloud computing platform at which user can develop social enterprise applications. The platform is the basis of SalesForce.com – a Software-as-a-Service solution for customer relationship management.

- Force.com allows creating applications by composing ready-to-use blocks: a complete set of components supporting all the activities of an enterprise are available.

- From the design of the data layout to the definition of business rules and user interface is provided by Force.com as a support.

- This platform is completely hostel in the Cloud, and provides complete access to its functionalities, and those implemented in the hosted applications through Web services technologies.

10. What are the Advantages and Disadvantages of Cloud Computing?

Advantages of Cloud Computing

As we all know that Cloud computing is trending technology. Almost every company switched their services on the cloud to raise the company growth.

Here, we are going to discuss some important advantages of Cloud Computing

1) Back-up and restore data
2) Improved collaboration
3) Excellent accessibility
4) Low maintenance cost
5) Mobility
6) IServices in the pay-per-use model
7) Unlimited storage capacity
8) Data security

1) Back-up and restore data

Once the data is stored in the cloud, it is easier to get back-up and restore that data using the cloud.

2) Improved collaboration

Cloud applications improve collaboration by allowing groups of people to quickly and easily share information in the cloud via shared storage.

3) Excellent accessibility

Cloud allows us to quickly and easily access store information anywhere, anytime in the whole world, using an internet connection. An internet cloud infrastructure increases organization productivity and efficiency by ensuring that our data is always accessible.

4) Low maintenance cost

Cloud computing reduces both hardware and software maintenance costs for organizations.

5) Mobility

Cloud computing allows us to easily access all cloud data via mobile.

6) IServices in the pay-per-use model

Cloud computing offers Application Programming Interfaces (APIs) to the users for access services on the cloud and pays the charges as per the usage of service.

7) Unlimited storage capacity

Cloud offers us a huge amount of storing capacity for storing our important data such as documents, images, audio, video, etc. in one place.

8) Data security

Data security is one of the biggest advantages of cloud computing. Cloud offers many advanced features related to security and ensures that data is securely stored and handled.

Disadvantages of Cloud Computing

A list of the disadvantage of cloud computing is given below

1) Internet Connectivity
2) Vendor lock-in
3) Limited Control
4) Security

1) Internet Connectivity

As you know, in cloud computing, every data (image, audio, video, etc.) is stored on the cloud, and we access these data through the cloud by using the internet connection. If you do not have good internet connectivity, you cannot access these data. However, we have no any other way to access data from the cloud.

2) Vendor lock-in

Vendor lock-in is the biggest disadvantage of cloud computing. Organizations may face problems when transferring their services from one vendor to another. As different vendors provide different platforms, that can cause difficulty moving from one cloud to another.

3) Limited Control

As we know, cloud infrastructure is completely owned, managed, and monitored by the service provider, so the cloud users have less control over the function and execution of services within a cloud infrastructure.

4) Security

Although cloud service providers implement the best security standards to store important information. But, before adopting cloud technology, you should be aware that you will be sending all your organization's sensitive information to a third party, i.e., a cloud computing service provider.

While sending the data on the cloud, there may be a chance that your organization's information is hacked by Hackers.

11.Discuss about the cloud storage services and application services. APR/MAY 2024

In distributed computing, cloud storage services and application services play crucial roles in enabling scalable, flexible, and on-demand access to resources.

Cloud Storage Services in Distributed Computing:

Cloud storage services in distributed computing enable:

1. Data Distribution: Store and manage large datasets across multiple locations, ensuring data availability and accessibility.

2. Decentralized Data Management: Allow multiple nodes or devices to access and share data, promoting collaboration and data sharing.

3. Scalability: Dynamically adjust storage capacity to accommodate growing data demands.

4. Fault Tolerance: Ensure data availability even in case of node failures or network partitions.

Examples of cloud storage services in distributed computing include:

- Hadoop Distributed File System (HDFS)

- Amazon S3

- Google Cloud Storage

- Microsoft Azure Blob Storage

Application Services in Distributed Computing:

Application services in distributed computing provide:

1. Scalable Computing: Offer on-demand computing resources to handle large-scale computations.

2. Load Balancing: Distribute workload across multiple nodes to ensure efficient resource utilization.

3. Parallel Processing: Enable simultaneous execution of tasks across multiple nodes, accelerating processing times.

4. Fault Tolerance: Ensure application availability despite node failures or network partitions.

Examples of application services in distributed computing include:

- Apache Spark

- Hadoop MapReduce

- Google Cloud Dataflow

- Amazon Elastic MapReduce (EMR)

12. A company like to have an advanced collaboration services like video, chat and web conferences for their employees, but their system does not support any of the IT resources due to insufficient infrastructure. If they could leverage cloud computing technology in their system, suggest a suitable cloud type with proper justification. List the characteristics of cloud computing. NOV/DEC 2023

Fig 5.5 Deployment Models - Public Cloud

Advantages of the Public Cloud Model

- Minimal Investment: Because it is a pay-per-use service, there is no substantial upfront fee, making it excellent for enterprises that require immediate access to resources.
- No setup cost: The entire infrastructure is fully subsidized by the cloud service providers, thus there is no need to set up any hardware.
- Infrastructure Management is not required: Using the public cloud does not necessitate infrastructure management.
- No maintenance: The maintenance work is done by the service provider (not users).
- Dynamic Scalability: To fulfill your company's needs, on-demand resources are accessible in above Fig 5.5

Disadvantages of the Public Cloud Model

- Less secure: Public cloud is less secure as resources are public so there is no guarantee of high-level security.
- Low customization: It is accessed by many public so it can't be customized according to personal requirements.

Characteristics of Cloud Computing

- 1. On demand self service
- 2. Broad Network Access
- 3. Resource Pooling
- 4. Rapid Elasticity - Horizontal & Vertical Scaling

- 5. Measured Service
- 6. Performance
- 7. Reduced Costs

13. Consider an ABC IT company wanted to provide services like scientific converter, data converter, currency converter and some of the business logic as a server side component to the developer to tailor make the application. Explain the various adaptable technologies to implement in an distributed environment. State its merit and demerits. NOV/DEC 2023

To implement the required services in a distributed environment, ABC IT company can consider the following adaptable technologies:

1. Microservices Architecture:

   - Merits: Scalability, flexibility, fault tolerance, and ease of maintenance.

   - Demerits: Complexity, inter-service communication overhead, and increased operational costs.

2. Serverless Computing (e.g., AWS Lambda, Azure Functions):

   - Merits: Cost-effective, scalable, and reduced administrative burden.

   - Demerits: Limited control, vendor lock-in, and potential cold start delays.

3. Containerization (e.g., Docker, Kubernetes):

   - Merits: Lightweight, portable, and efficient resource utilization.

   - Demerits: Steep learning curve, orchestration complexity, and security concerns.

4. API Gateway (e.g., NGINX, AWS API Gateway):

   - Merits: Centralized management, security, and scalability.

   - Demerits: Additional latency, vendor lock-in, and potential single point of failure.

5. Message Queueing (e.g., RabbitMQ, Apache Kafka):

   - Merits: Decoupling, fault tolerance, and scalability.

   - Demerits: Complexity, message ordering issues, and potential message loss.

6. Cloud Functions (e.g., Google Cloud Functions, Azure Functions):

   - Merits: Event-driven, scalable, and cost-effective.

- Demerits: Limited control, vendor lock-in, and potential cold start delays.

7. Service-Oriented Architecture (SOA):

- Merits: Reusability, scalability, and flexibility.

- Demerits: Complexity, inter-service communication overhead, and potential vendor lock-in.

To tailor-make applications, developers can leverage:

1. APIs (RESTful, GraphQL): For integrating services and business logic.

2. SDKs (Software Development Kits): For accessing services and implementing business logic.

3. Webhooks: For event-driven interactions and real-time notifications.

By adopting these technologies, ABC IT company can create a flexible, scalable, and maintainable distributed environment, enabling developers to build customized applications efficiently.

14. An organization has planned to implement a client module that emulates a conventional file system interface for application programs, and server modules, that perform operations for clients on directories and on files. State the best distributed architecture to deploy and deliver the system. Also explain the suitable system model to avoid failures and adapt a recovery system for the same in case of unavoidable failures NOV/DEC 2023

The best distributed architecture to deploy and deliver the system is a Client-Server Architecture with a Distributed File System (DFS) approach. This architecture consists of:

1. Client Modules: Emulate a conventional file system interface for application programs.

2. Server Modules:

- Directory Servers: Manage directory operations (e.g., create, delete, list).

- File Servers: Manage file operations (e.g., read, write, delete).

- Metadata Servers: Store metadata about files and directories (e.g., permissions, ownership).

To avoid failures and adapt a recovery system, consider the following:

System Model:

1. Replication: Duplicate data and metadata across multiple servers to ensure availability.

2. Redundancy: Use redundant components (e.g., power supplies, network connections) to minimize single points of failure.

3. Fault Tolerance: Design the system to continue operating even if some components fail.

Recovery System:

1. Backup and Restore: Regularly backup data and metadata, and have a restore process in place.

2. Checkpoints: Periodically save the system's state to facilitate recovery.

3. Rollback Recovery: Allow the system to roll back to a previous checkpoint in case of failure.

4. Failure Detection and Recovery: Implement mechanisms to detect failures and automatically recover from them (e.g., restart failed components).

Additional considerations:

1. Load Balancing: Distribute workload across servers to prevent overload and ensure responsiveness.

2. Consistency Protocols: Implement protocols (e.g., Paxos, Raft) to maintain data consistency across replicas.

3. Security: Ensure secure communication between clients and servers, and authenticate/authorize access to data.

By adopting this architecture and system model, the organization can build a robust and resilient distributed file system that minimizes the risk of failures and ensures efficient recovery in case of unavoidable failures.

Reg. No. : 4 2 1 6 2 1 2 0 5 0 5 9

Question Paper Code : 20874

B.E./B.Tech. DEGREE EXAMINATIONS, NOVEMBER/DECEMBER 2023.

Fifth Semester

Computer Science and Design

CS 3551 – DISTRIBUTED COMPUTING

(Common to : Computer Science and Engineering/ Computer Science and Engineering (Artificial Intelligence and Machine Learning)/ Computer Science and Engineering (Cyber Security)/ Computer and Communication Engineering/ Artificial Intelligence and Data Science and Information Technology)

(Regulations – 2021)

Time : Three hours | Maximum : 100 marks

Answer ALL questions.

PART A — (10 × 2 = 20 marks)

1. State the differences from synchronous and asynchronous communication along with the buffering strategy adapted in both.
2. List the role of shared memory systems.
3. How do you define the scalar time and the vector time?
4. If the two events e_1 and e_2 of two different processes occur at same time, independently. What kind of relationship exist between e_1 and e_2?
5. What are the various deadlock detection in distributed systems? State a common factor in detecting the deadlock.
6. How beneficial is the Chandy-Misra-Haas algorithm in the AND model and OR model?
7. What is a checkpoint in the recovery system? How is it useful?
8. State the issues in failure recovery. State the means to identify the failure at an early stage.
9. What is virtualization? Why is it required in the larger organization?
10. State the relation of compute service with storage service in a cloud environment.

PART B — (5 × 13 = 65 marks)

11. (a) Explain message passing systems and discuss on the message oriented middleware and its types. Also explain their functionality in distributed computing.

Or

(b) Analyze the concepts of heterogeneity, openness, security, scalability impact of the distributed systems. How is the standardization of them make them effective?

12. (a) Give a real time scenario where FIFO message queue is used. Write and describe the snapshot algorithms for FIFO channels.

Or

(b) Describe the physical clock synchronization and logical clock synchronization and explain the framework of a system of logical clock.

13. (a) State the Lamport's algorithm and its use and also the limitations and benefits. Compare this with any two token based algorithms.

Or

(b) Explain the system models with its preliminaries and how is it characterized in the models of deadlock.

14. (a) Explain the facts associated with agreement in failure-free systems of both synchronous and asynchronous systems.

Or

(b) What are the different check pointing based recovery systems available? Explain the coordinated check pointing algorithm with illustration.

15. (a) A company like to have an advanced collaboration services like video, chat and web conferences for their employees, but their system does not support any of the IT resources due to insufficient infrastructure. If they could leverage cloud computing technology in their system, suggest a suitable cloud type with proper justification. List the characteristic of cloud computing.

Or

(b) Explain the different cloud services and the platform that support these services. Give an example for application service adapted by any organization.

PART C — (1 × 15 = 15 marks)

16. (a) Consider an ABC IT company wanted to provide services like scientific converter, data converter, currency converter and some of the business logic as a server side component to the developer to tailor make the application. Explain the various adaptable technologies to implement in an distributed environment. State its merit and demerits.

Or

(b) An organization has planned to implement a client module that emulates a conventional file system interface for application programs, and server modules, that perform operations for clients on directories and on files. State the best distributed architecture to deploy and deliver the system. Also explain the suitable system model to avoid failures and adapt a recovery system for the same in case of unavoidable failures.

———————

Reg. No. : | | | | | | | | | | | | |

Question Paper Code : 50907

B.E./B.Tech. DEGREE EXAMINATIONS, APRIL/MAY 2024

Fifth Semester

Computer Science and Design

CS 3551 – DISTRIBUTED COMPUTING

(Common to: Computer Science and Engineering/ Computer Science and Engineering (Artificial Intelligence and Machine Learning/ Computer Science and Engineering (Cyber Security/ Computer and Communication Engineering/ Artificial Intelligence and Data Science/ Information Technology)

(Regulations 2021)

Time : Three hours Maximum : 100 marks

Answer ALL questions.

PART A — (10 × 2 = 20 marks)

1. Distinguish between message passing systems and shared memory systems.

2. What are the motivations for distributed systems?

3. Define global state and consistent global state.

4. What are the limitations of logical clock?

5. List the advantages and limitations of token-based algorithms for implementing mutual exclusion.

6. What is a false deadlock and when does it occurs?

7. Outline the issues in failure recovery.

8. What are the advantages and disadvantages of asynchronous checkpointing?

9. What is virtualization in cloud computing?

10. List the characteristics of cloud computing.

PART B — (5 × 13 = 65 marks)

11. (a) Explain the design issues and challenges in the design of distributed operating systems.

Or

(b) Explain blocking, non-blocking, synchronous and asynchronous primitives for distributed communication. Also summarize major libraries and standards for building distributed applications.

12. (a) What is snapshot and what are the needs of taking snapshot in distributed systems? Outline Chandy–Lamport algorithm for snapshot and illustrate the algorithm.

Or

(b) Explain various ways of ordering messages in group communications. Also illustrate distributed algorithm to implement total order and causal order of messages.

13. (a) Outline the Chandy–Misra–Haas algorithm to detect deadlock in OR model and illustrate the algorithm with an example.

Or

(b) Outline Ricart and Agrawala's algorithm for implementing mutual exclusion in distributed systems and illustrate the algorithm.

14. (a) What is check pointing and why it is needed in distributed systems? Illustrate the algorithm proposed by Juang and Venkatesan for asynchronous check pointing and recovery.

Or

(b) What is agreement in a failure free system? Explain how agreement is reached in message–passing synchronous systems with failures.

15. (a) Explain the types of cloud computing deployment models with their relative advantages and disadvantages.

Or

(b) Discuss about the Cloud storage services and Application services.

PART C — (1 × 15 = 15 marks)

16. (a) The management of an autonomous education institution decided to migrate its website and database from on-premises infrastructure to the Cloud. As an expert of cloud computing you are contacted by management to identify the best suited cloud deployment model and service model for the institution. Describe the important factors you can consider for selecting the cloud deployment model and service model for the institution. Also justify, in what way the selected models are best suited for the institution compared to other models.

Or

(b) Outline the Koo and Toueg coordinated check pointing and recovery technique and apply this algorithm to a specific case study to demonstrate how this algorithm avoids the domino effect and live lock problems during the recovery.
