Swami Keshvanand Instituteof Technology,

Management & Gramothan

(Accredited by NAAC with 'A++' Grade)

Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Affiliated to Rajasthan Technical University, Kota

1.1.2 Midterm Papers and Solutions (Sample) (2023-24)

: RAMNAGARIA (JAGATPURA), JAIPUR-302017 (RAJASTHAN), INDIA
: +91-141-3500300, 2752165, 2759609 | : 0141-2759555
: info@skit.ac.in | : www.skit.ac.in

Swami Keshvanand Institute of Technology, Management & Gramothan, Jaipur

II -Mid Term Examination, May-2024

Prog./Semester:	B. Tech.-VI	Branch:	CSE
Course Title:	Distributed Systems	Course Code:	6CS5-11
Duration:	1.5 Hours	Maximum Marks:	20
Session : First		Roll No.:	
Instructions if any:		Read all the questions carefully. Write neatly.	

PART -A (short answer type questions)
(All questions are compulsory)

Q. No.	Question	Max. Marks	CO(s)	Bloom's Level
1	Define Byzantine Agreement Problem with its solution	2	5	1
2	Discuss failures in a DS.	2	5	2
3	Discuss how mutual exclusion is handled in DS.	2	4	2

PART -B (Analytical/Problem solving questions)
(All questions are compulsory)

Q. No.	Question	Max. Marks	CO(s)	Bloom's Level
4	Illustrate Distributed Shared Memory with the help of a suitable diagram.	4	4	3
5	Illustrate fault tolerance. Explain how fault tolerance is ensured in DS.	4	5	3

PART- C (Descriptive/Analytical/Problem solving/Design questions)
(Attempt any one Question)

Q. No.	Question	Max. Marks	CO(s)	Bloom's Level
6	Illustrate Distributed File System. Explain the concept of transaction control and concurrency control in detail.	6	3	3
7	Examine dynamic load sharing and load balancing and its requirement in distributed process scheduling with justification.	6	3	4

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017

Solution of Question Paper
II Mid-Term Examination, May -2024

Prog./Sem.: B.Tech. VI Sem	Course Title: Distributed Systems	Course Code:6CS5-11
Duration: 1.5 hours	Date: 31/05/2024 Session: FIRST	Max Marks: 20
Submitted By: Dr. Nilam Choudhary/Ms. Astha Joshi		

S. No.	Course Outcome	Percentage Coverage
CO1	List basic components of distributed systems and describe various issues of DS.	NA
CO2	Illustrate and analyze process management, Inter-process Communication techniques.	NA
CO3	Compare Distributed File Systems, and Distributed Process Scheduling with Transaction Services and Concurrency Control.	30%
CO4	Analyze Distributed Shared Memory with memory consistency models and models of distributed computation.	30%
CO5	Demonstrate different algorithms and techniques for Distributed Agreement, Replicated Data Management.	40%

S. No.	Bloom's Level	Percentage Coverage
1	BL1- Remember	14%
2	BL2- Understand	29%
3	BL3- Apply	43%
4	BL4- Analyze	14%
5	BL5- Evaluate	NA
6	BL6- Create	NA

Solution of Question Paper
II Mid-Term Examination, May -2024

Prog./Sem.: B.Tech. VI Sem	Course Title: Distributed Systems	Course Code:6CS5-11
Duration: 1.5 hours	Date: 31/05/2024 Session: FIRST	Max Marks: 20
Submitted By: Dr. Nilam Choudhary/Ms. Astha Joshi		

A1> Mutual exclusion in distributed systems (DS) is a crucial aspect to ensure that concurrent processes or nodes do not access a shared resource simultaneously, potentially causing data corruption or inconsistencies. Unlike in single-system environments, mutual exclusion in DS has to address issues such as network latency, partial failures, and the lack of a global clock. Here are the main approaches to handle mutual exclusion in distributed systems:

1. Centralized Algorithms

Centralized Mutex Algorithm

A central coordinator node is responsible for granting access to the shared resource:

Process Request: A process sends a request to the coordinator when it wants to access the resource.

Grant Access: The coordinator grants access if no other process is currently accessing the resource; otherwise, it queues the request.

Release: The process notifies the coordinator when it releases the resource, prompting the coordinator to grant access to the next queued request.

2. Decentralized Algorithms

Token-Based Algorithms

These algorithms use a token, a unique object that grants the right to access the shared resource.

Token Ring Algorithm

Processes are arranged in a logical ring.

The token circulates around the ring.

A process can access the resource only if it holds the token. It releases the token after use, passing it to the next process in the ring.

3. Quorum-Based Algorithms

Majority Voting

Each process must request permission from a majority of the processes (a quorum).

To access the resource, a process must receive permission from a majority of processes in the quorum.

This ensures that no two processes can access the resource simultaneously, as their quorums will overlap.

Solution of Question Paper
II Mid-Term Examination, May -2024

Prog./Sem.: B.Tech. VI Sem	Course Title: Distributed Systems	Course Code:6CS5-11
Duration: 1.5 hours	Date: 31/05/2024 Session: FIRST	Max Marks: 20
Submitted By: Dr. Nilam Choudhary/Ms. Astha Joshi		

Use consensus algorithms like Paxos or Raft to manage locks in a fault-tolerant manner.

A process requests a lock from the service, which coordinates among nodes to ensure mutual exclusion.

A2> Failures in a distributed system (DS) are common and can have significant impacts on the reliability, availability, and correctness of the system. Understanding and handling these failures are crucial for designing robust distributed systems. Here are the main types of failures that can occur in a DS and strategies for handling them:

Types of Failures

Crash Failures

Description: A node or process crashes and stops functioning entirely. The node does not respond to any messages.

Handling: Techniques like heartbeats and timeouts can detect crash failures. Redundancy and replication can help maintain availability despite such failures.

Omission Failures

Description: A node or network component fails to send or receive messages. This can be due to network issues or software bugs.

Handling: Retransmission of messages, acknowledgments, and using reliable communication protocols can mitigate omission failures.

Timing Failures

Description: Nodes or network components do not adhere to timing assumptions. For instance, a node might respond too late or too early.

Handling: Implementing timeout mechanisms and time synchronization protocols like NTP can help manage timing failures.

Response Failures

Description: A node responds incorrectly to a request. This can happen due to bugs, misconfigurations, or transient errors.

Handling: Techniques like voting, where multiple nodes provide responses and the majority decision is taken, can help detect and correct response failures.

Byzantine Failures

Description: Nodes exhibit arbitrary or malicious behavior, sending incorrect or inconsistent messages. These are the

most challenging type of failures to handle.

Handling: Byzantine Fault Tolerance (BFT) algorithms like PBFT (Practical Byzantine Fault Tolerance) and blockchain consensus mechanisms can manage Byzantine failures by ensuring agreement among honest nodes.

Causes of Failures

Hardware Failures: Physical components like servers, disks, or network hardware can fail.

Software Bugs: Bugs in the operating system, application code, or middleware can cause failures.

Network Issues: Network partitions, congestion, or misconfigurations can lead to communication failures.

Resource Exhaustion: Running out of critical resources like CPU, memory, or disk space can cause nodes to fail.

Human Errors: Misconfigurations, accidental deletions, or other operator mistakes can lead to system failures.

A3>The Byzantine Agreement Problem is a fundamental issue in distributed systems and distributed computing. It addresses the challenge of achieving consensus among distributed processes or nodes in the presence of faulty or malicious participants. The problem is named after the Byzantine generals' problem, an allegory that describes the difficulty of achieving coordinated action among generals (or nodes) when some may be traitors (or faulty/malicious nodes).

Definition of Byzantine Agreement Problem

In the Byzantine Agreement Problem, we consider a distributed system with nn nodes, some of which may be faulty or malicious (called Byzantine nodes). The goal is to ensure that all non-faulty nodes agree on a single value despite the presence of Byzantine nodes. The requirements for a solution to the Byzantine Agreement Problem are:

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

Validity: If all non-faulty nodes propose the same value, they must agree on that value.

Termination: All non-faulty nodes must eventually reach a decision.

Solution to the Byzantine Agreement Problem

One of the well-known solutions to the Byzantine Agreement Problem is the Byzantine Fault Tolerance (BFT) algorithm. The most notable BFT algorithm is the Practical Byzantine Fault Tolerance (PBFT), proposed by Miguel Castro and Barbara Liskov in 1999. Here's an overview of the PBFT algorithm:

PBFT Algorithm

The PBFT algorithm is designed for asynchronous systems and can tolerate up to $\lfloor n-13 \rfloor \lfloor 3n-1 \rfloor$ Byzantine faults,

Prog./Sem.: B.Tech. VI Sem	Course Title: Distributed Systems	Course Code:6CS5-11
Duration: 1.5 hours	Date: 31/05/2024 Session: FIRST	Max Marks: 20
Submitted By: Dr. Nilam Choudhary/Ms. Astha Joshi		

meaning it can handle ff faulty nodes out of nn total nodes, where $n \geq 3f+1$n≥3f+1.

Phases of PBFT

Pre-prepare Phase:

A client sends a request to the primary node (leader).

The primary node multicasts a pre-prepare message to all replica nodes.

Prepare Phase:

Upon receiving the pre-prepare message, each replica node multicasts a prepare message to all other nodes.

Each node waits to receive $2f$2f prepare messages (including its own).

Commit Phase:

After receiving $2f$2f prepare messages, a node multicasts a commit message to all other nodes.

Each node waits to receive $2f$2f commit messages (including its own).

Reply Phase:

After receiving $2f$2f commit messages, the nodes execute the request and send a reply to the client.

The client waits for $f+1$f+1 consistent replies to consider the request complete.

Key Features

Fault Tolerance: The PBFT algorithm can tolerate up to $\lfloor n-13 \rfloor \lfloor 3n-1 \rfloor$ Byzantine faults, ensuring consensus despite malicious behavior.

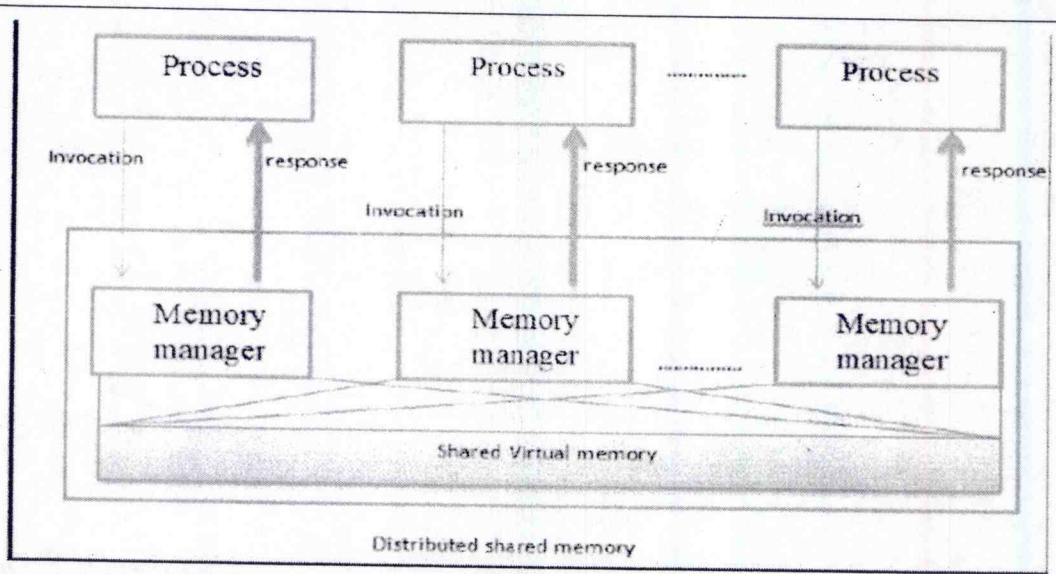
Safety and Liveness: The algorithm guarantees safety (agreement and validity) and liveness (termination) even in asynchronous environments.

Low Latency: PBFT achieves consensus with a lower message complexity and latency compared to earlier Byzantine agreement protocols.

A4>

Solution of Question Paper
II Mid-Term Examination, May -2024

Prog./Sem.: B.Tech. VI Sem	Course Title: Distributed Systems	Course Code:6CS5-11
Duration: 1.5 hours	Date: 31/05/2024 Session: FIRST	Max Marks: 20
Submitted By: Dr. Nilam Choudhary/Ms. Astha Joshi		



Distributed Shared Memory (DSM) is an abstraction that allows distributed systems to share a single memory space among multiple nodes. This abstraction enables processes on different machines to access shared data as if they were on the same physical memory. DSM simplifies the development of distributed applications by providing a shared memory model, abstracting away the complexities of message passing.

Key Concepts of DSM

Shared Memory Abstraction:

Provides a unified memory space accessible by all nodes in the system.

Allows processes to read and write to shared memory locations transparently.

Consistency Models:

Strict Consistency: Every read returns the most recent write (impractical in most DSM systems due to high latency and synchronization overhead).

Sequential Consistency: Operations appear in some sequential order that is consistent with the order observed by each process.

Causal Consistency: Ensures that causally related operations are seen by all nodes in the same order.

Eventual Consistency: Updates will eventually propagate to all nodes, and all nodes will converge to the same value if no new updates are made.

Memory Coherence:

Ensures that all nodes have a consistent view of the shared memory.

Typically implemented using coherence protocols such as invalidation or update protocols.

Diagram of DSM

Each node has its own local memory.

Nodes are connected to the shared memory space provided by DSM.

Distributed Shared Memory:

Acts as a shared space accessible by all nodes.

Manages shared pages that can be mapped into the local memory of each node.

Shared Pages:

Solution of Question Paper
II Mid-Term Examination, May -2024

Prog./Sem.: B.Tech. VI Sem	Course Title: Distributed Systems	Course Code:6CS5-11
Duration: 1.5 hours	Date: 31/05/2024 Session: FIRST	Max Marks: 20
Submitted By: Dr. Nilam Choudhary/Ms. Astha Joshi		

Segments of memory that are part of the DSM.
Each shared page can be accessed and modified by any node in the system.
Working of DSM
Memory Mapping:
The DSM system maps shared pages into the local memory of each node.
When a process accesses a shared memory location, the DSM ensures that the latest version of the page is available.
Synchronization:
DSM uses synchronization mechanisms like locks or semaphores to manage access to shared data.
Ensures consistency and prevents race conditions.
Consistency Protocols:
Invalidation Protocol: When a node modifies a shared page, it invalidates copies of that page on other nodes.
Update Protocol: When a node modifies a shared page, it updates the copies of that page on other nodes.
Fault Tolerance:
DSM systems often include mechanisms to handle node failures, ensuring data replication and recovery.
Use Cases of DSM
Parallel Computing:
DSM is used in parallel computing environments where multiple processors need to access shared data efficiently.
Distributed Databases:
Provides a shared memory space for distributed database systems, facilitating coordinated access to database records.
Collaborative Applications:
Enables multiple users to work on shared documents or resources in real-time, with consistent views of the shared data.

A5>Definition of Fault Tolerance

Fault tolerance is the property of a system that enables it to continue operating properly in the event of the failure of some of its components. A fault-tolerant system is designed to ensure that the system as a whole remains functional and maintains its performance and availability, even when certain parts of the system fail. Fault tolerance is crucial in distributed systems (DS) due to the inherent complexity and higher likelihood of component failures.

Ensuring Fault Tolerance in Distributed Systems

Fault tolerance in distributed systems is achieved through a combination of techniques designed to detect, isolate, and recover from failures. Here are the key strategies and mechanisms used to ensure fault tolerance in distributed systems:

1. Redundancy

Description: Redundancy involves duplicating critical components or data to ensure that if one component fails, another can take over without interrupting the system's functionality.

Data Redundancy: Data is replicated across multiple nodes to prevent data loss and ensure data availability. Examples

include RAID (Redundant Array of Independent Disks) and distributed databases like Cassandra, which replicate data across multiple nodes.

Component Redundancy: Critical system components, such as servers or network links, are duplicated. If one component fails, the redundant component takes over. This is common in high-availability setups.

2. Replication

Description: Replication involves creating copies of data or services and distributing them across multiple nodes or locations to ensure fault tolerance.

Master-Slave Replication: One node acts as the master (primary), and other nodes act as slaves (secondaries). If the master fails, a slave can be promoted to master.

Multi-Master Replication: Multiple nodes act as masters, allowing read and write operations. This ensures that even if one master fails, others can continue to serve requests.

3. Consensus Algorithms

Description: Consensus algorithms are used to achieve agreement among distributed nodes on a single data value or system state, even in the presence of some faulty nodes.

Paxos and Raft: These are consensus protocols used to ensure that multiple nodes agree on a single value or sequence of operations. They are commonly used in distributed databases and systems like etcd and Consul.

Byzantine Fault Tolerance (BFT): Algorithms like PBFT (Practical Byzantine Fault Tolerance) handle Byzantine failures, where nodes may behave arbitrarily or maliciously. These are used in blockchain systems and other critical applications.

4. Failure Detection

Description: Failure detection mechanisms identify when components or nodes have failed.

Heartbeats: Nodes periodically send heartbeat messages to indicate that they are alive. If a heartbeat is missed for a certain period, the node is considered to have failed.

Timeouts: If a node does not respond to a request within a specified time, it is assumed to have failed.

5. Failover Mechanisms

Description: Failover mechanisms automatically switch operations to a standby system or component when a primary system or component fails.

Active-Passive Failover: A standby system (passive) takes over when the primary system (active) fails. This is common in database clusters and web server setups.

Active-Active Failover: All systems are active and share the load. If one system fails, the remaining systems continue to

handle the workload. This is used in load-balanced server farms.

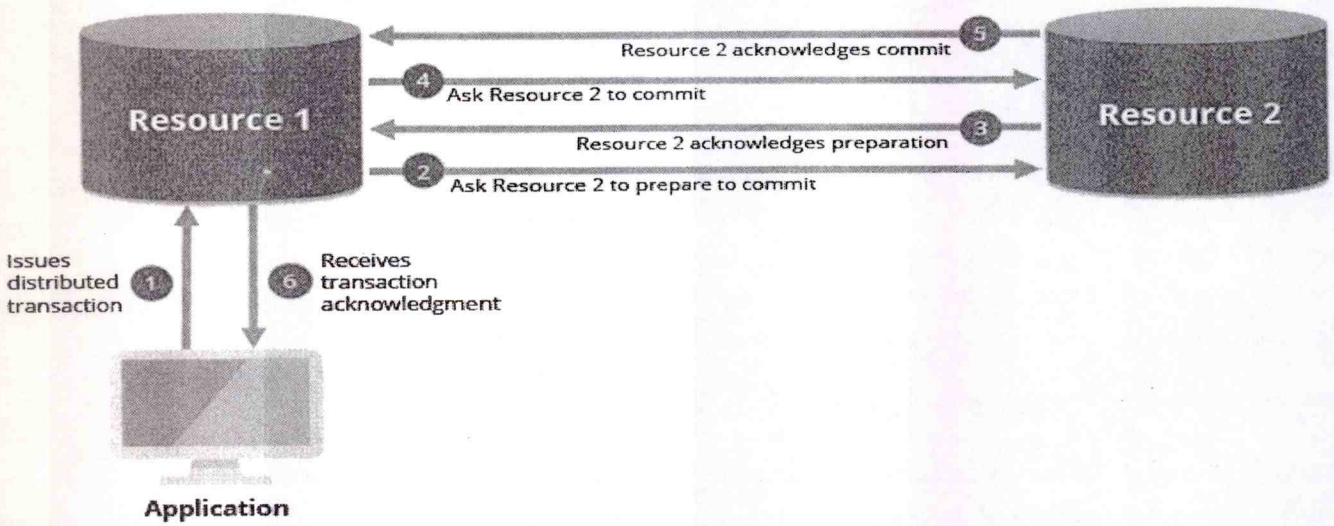
A6> Distributed File System (DFS)

A Distributed File System (DFS) is a system that allows files to be accessed and managed across multiple servers in a network, presenting a unified file system view to users and applications. The goal of a DFS is to provide reliable, efficient, and transparent access to files regardless of their physical location in the network.

A distributed transaction is a set of operations on data that is performed across two or more data repositories (especially databases). It is typically coordinated across separate nodes connected by a network, but may also span multiple databases on a single server.

There are two possible outcomes: 1) all operations successfully complete, or 2) none of the operations are performed at all due to a failure somewhere in the system. In the latter case, if some work was completed prior to the failure, that work will be reversed to ensure no net work was done. This type of operation is in compliance with the "ACID" (atomicity-consistency-isolation-durability) principles of databases that ensure data integrity. ACID is most commonly associated with transactions on a single database server, but distributed transactions extend that guarantee across multiple databases.

The operation known as a "two-phase commit" (2PC) is a form of a distributed transaction. "XA transactions" are transactions using the XA protocol, which is one implementation of a two-phase commit operation.



A distributed transaction spans multiple databases and guarantees data integrity.

How Do Distributed Transactions Work?

Distributed transactions have the same processing completion requirements as regular database transactions, but they must be managed across multiple resources, making them more challenging to implement for database developers. The multiple resources add more points of failure, such as the separate software systems that run the resources (e.g., the

database software), the extra hardware servers, and network failures. This makes distributed transactions susceptible to failures, which is why safeguards must be put in place to retain data integrity.

For a distributed transaction to occur, transaction managers coordinate the resources (either multiple databases or multiple nodes of a single database). The transaction manager can be one of the data repositories that will be updated as part of the transaction, or it can be a completely independent separate resource that is only responsible for coordination. The transaction manager decides whether to commit a successful transaction or rollback an unsuccessful transaction, the latter of which leaves the database unchanged.

First, an application requests the distributed transaction to the transaction manager. The transaction manager then branches to each resource, which will have its own "resource manager" to help it participate in distributed transactions. Distributed transactions are often done in two phases to safeguard against partial updates that might occur when a failure is encountered. The first phase involves acknowledging an intent to commit, or a "prepare-to-commit" phase. After all resources acknowledge, they are then asked to run a final commit, and then the transaction is completed.

We can examine a basic example of what happens when a failure occurs during a distributed transaction. Let's say one or more of the resources become unavailable during the prepare-to-commit phase. When the request times out, the transaction manager tells each resource to delete the prepare-to-commit status, and all data will be reset to its original state. If instead, any of the resources become unavailable during the commit phase, then the transaction manager will tell the other resources that successfully committed their portion of the transaction to undo or "rollback" that transaction, and once again, the data is back to its original state. It is then up to the application to retry the transaction to make sure it gets completed.

Why Do You Need Distributed Transactions?

Distributed transactions are necessary when you need to quickly update related data that is spread across multiple databases. For example, if you have multiple systems that track customer information and you need to make a universal update (like updating the mailing address) across all records, a distributed transaction will ensure that all records get updated. And if a failure occurs, the data is reset to its original state, and it is up to the originating application to resubmit the transaction.

Concurrency Control in Distributed Transactions

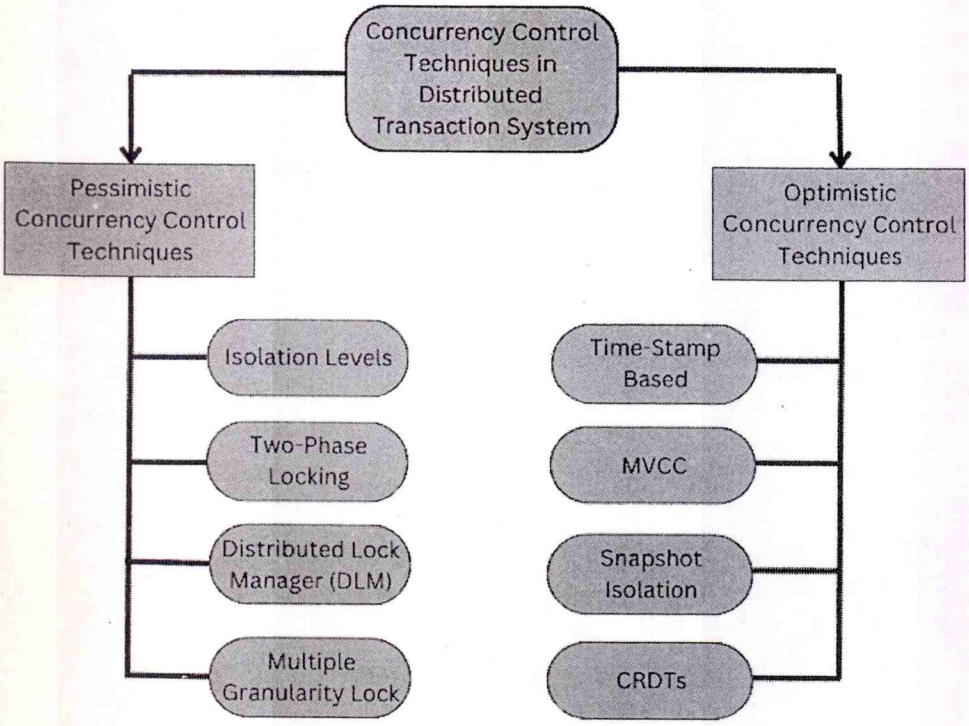
Concurrency control mechanisms provide us with various concepts & implementations to ensure the execution of any transaction across any node doesn't violate ACID or BASE (depending on database) properties causing inconsistency & mixup of data in the distributed systems. Transactions in the distributed system are executed in "sets", every set consists of various sub-transactions. These sub-transactions across every node must be executed serially to maintain data integrity & the concurrency control mechanisms do this serial execution.

Types of Concurrency Control Mechanisms

There are 2 types of concurrency control mechanisms as shown below diagram:

Solution of Question Paper
II Mid-Term Examination, May -2024

Prog./Sem.: B.Tech. VI Sem	Course Title: Distributed Systems	Course Code:6CS5-11
Duration: 1.5 hours	Date: 31/05/2024 Session: FIRST	Max Marks: 20
Submitted By: Dr. Nilam Choudhary/Ms. Astha Joshi		



Types of Concurrency Control Mechanism

Pessimistic Concurrency Control (PCC)

The Pessimistic Concurrency Control Mechanisms proceeds on assumption that, most of the transactions will try to access the same resource simultaneously. It's basically used to prevent concurrent access to a shared resource and provide a system of acquiring a Lock on the data item before performing any operation.

Optimistic Concurrency Control (OCC)

The problem with pessimistic concurrency control systems is that, if a transaction acquires a lock on a resource so that no other transactions can access it. This will result in reducing concurrency of the overall system.

The Optimistic Concurrency control techniques proceeds on the basis of assumption that, 0 or very few transactions will try to access a certain resource simultaneously. We can describe a system as FULLY OPTIMISTIC, if it uses No-Locks at all & checks for conflicts at commit time. It has following 4-phases of operation:

Read Phase: When a transaction begins, it reads the data while also logging the time-stamp at which data is read to verify for conflicts during the validation phase.

Execution Phase: In this phase, the transaction executes all its operation like create, read, update or delete etc.

Validation Phase: Before committing a transaction, a validation check is performed to ensure consistency by checking

Solution of Question Paper
II Mid-Term Examination, May -2024

Prog./Sem.: B.Tech. VI Sem	Course Title: Distributed Systems	Course Code:6CS5-11
Duration: 1.5 hours	Date: 31/05/2024 Session: FIRST	Max Marks: 20
Submitted By: Dr. Nilam Choudhary/Ms. Astha Joshi		

the last_updated timestamp with the one recorded at read_phase. If the timestamp matches, then the transaction will be allowed to be committed and hence proceed with the commit phase.

Commit phase: During this phase, the transactions will either be committed or aborted, depending on the validation check performed during previous phase. If the timestamp matches, then transactions are committed else they're aborted.

Pessimistic Concurrency Control Methods

Following are the four Pessimistic Concurrency Control Methods:

Isolation Level

The isolation levels are defined as a degree to which the data residing in Database must be isolated by transactions for modification. Because, if some transactions are operating on some data let's say transaction – T1 & there comes another transaction – T2 and modifies it further while it was under operation by transaction T1 this will cause unwanted inconsistency problems. Methods provided in this are: Read-Uncomitted, Read-Comitted, Repeatable Read & Serializable.

Two-Phase Locking Protocol

The two-phase locking protocol is a concurrency technique used to manage locks on data items in database. This technique consists of 2 phases:

Growing Phase: The transaction acquires all the locks on the data items that'll be required to execute the transaction successfully. No locks will be realease in this phase.

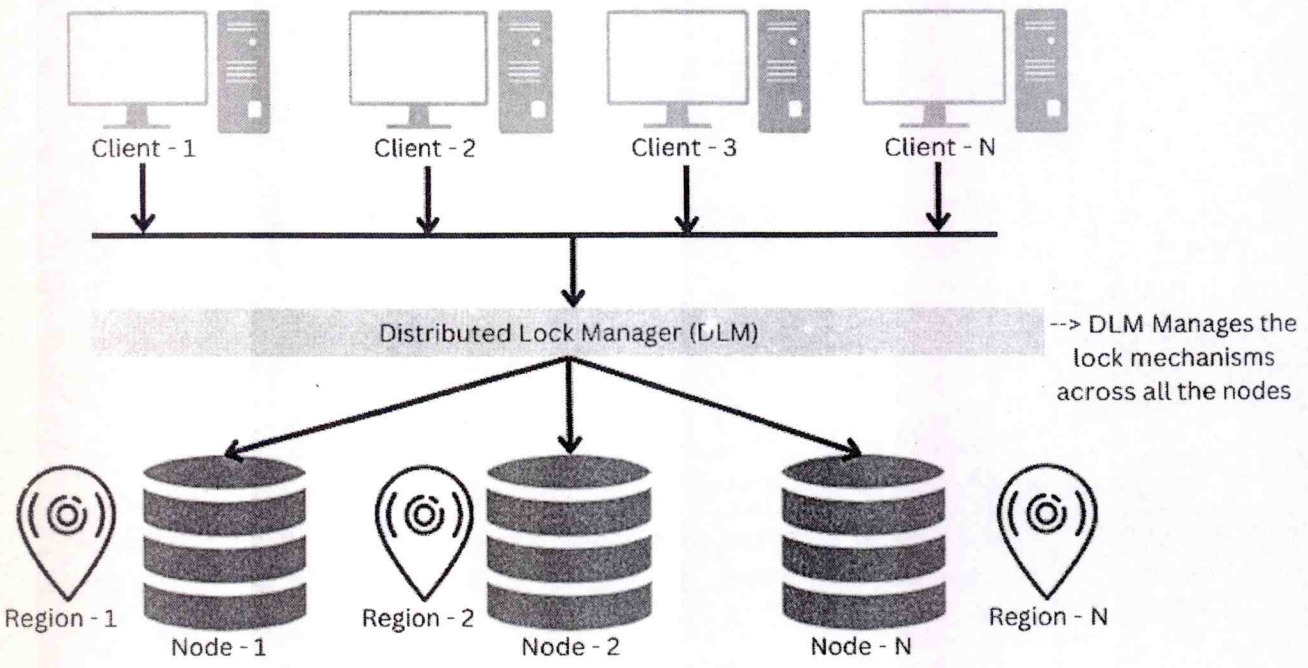
Shrinking Phase: All the locks acquired in previous phase will be released one by one and No New locks will be acquired in this phase.

Distributed Lock Manager

A distributed lock a critical component in the distributed transaction system, which co-ordinates the lock acquiring, and releasing operations in the transactions. It helps in synchronizing the transaction and their operation so that data integrity is maintained.

Solution of Question Paper
II Mid-Term Examination, May -2024

Prog./Sem.: B.Tech. VI Sem	Course Title: Distributed Systems	Course Code:6CS5-11
Duration: 1.5 hours	Date: 31/05/2024 Session: FIRST	Max Marks: 20
Submitted By: Dr. Nilam Choudhary/Ms. Astha Joshi		



Distributed Lock Manager (DLM)

Multiple Granularity Lock

A lock can be acquired at various granular level like: table level, row/record level, page level or any other resource's level. In transaction system a transaction can lock a whole table, or a specific row while performing some changes on it. This lock acquiring when done by various transactions simultaneously, this phenomena is called as multiple granularity locking.

Optimistic Concurrency Control Methods

Below are four Optimistic Concurrency Control Methods:

Timestamp Based (OCC)

In a timestamp based concurrency technique, each transaction in the system is assigned a unique timestamp which is taken as soon as the transaction begins, and its verified again during the commit phase. If there's new updated timestamp from a different transaction then based on some policy defined by the System Adminstrator the transaction will either be restarted or aborted. But if the times stamp is same & never modified by any other transaction then it will be committed.

Example: Let's say we have two transaction T1 and T2, they operate on data item – A. The Timestamp concurrency technique will keep track of the timestamp when the data was accessed by transaction T1 first time.

Solution of Question Paper
II Mid-Term Examination, May -2024

Prog./Sem.: B.Tech. VI Sem	Course Title: Distributed Systems	Course Code:6CS5-11
Duration: 1.5 hours	Date: 31/05/2024 Session: FIRST	Max Marks: 20
Submitted By: Dr. Nilam Choudhary/Ms. Astha Joshi		

Transaction	Data item and operation	Most_recent_Timestamp	Initial_timestamp of data item (A)
T1	Read(A)	12:00PM	12:00PM
T2	Write(A)	12:15PM	12:00PM
T1	Write(A)	12:30PM	12:00PM

Now, let's say this transaction T1 is about to commit, before committing, it will check the initial timestamp with the most recent timestamp. In our case, the transaction T1 won't be committed because a write operations by transaction T2 was performed.

if(Initial_timestamp == Most_recent_timestamp)
then 'Commit'
else
'Abort'

In our case, transaction will be aborted because T2 modified the same data item at 12:15PM.

Multi-Version Concurrency Control (MVCC)

In MVCC, every data item has multiple versions of itself. When a transaction starts, it reads the version that is valid at the start of the transaction. And when the transaction writes, it creates a new version of that specific data item. That way, every transaction can concurrently perform their operations.

Example: In a banking system two or more user can transfer money without blocking each other simultaneously.

A similar technique to this is : Immutable Data Structures. Every time a transaction performs a new operation, new data item will be created so that way transactions do not have to worry about consistency issues.

Snapshot Isolation

Snapshot isolation is basically a snapshot stored in an isolated manner when our database system was purely consistent. And this snapshot is read by the transactions at the beginning. Transaction ensures that the data item is not changed while it was executing operations on it. Snapshot isolation is achieved through OCC & MVCC techniques.

Conflict Free Replicated Data Types (CRDTs)

CRDTs is a data structure technique which allows a transaction to perform all its operation and replicate the data to

some other node or current node. After all the operations are performed, this technique offers us with merging methods that allows us to merge the data across distributed nodes (conflict-free) and eventually achieving consistent state (eventually consistent property).

A7> Dynamic Load Sharing and Load Balancing in Distributed Systems

In distributed systems, load sharing and load balancing are critical for ensuring that computational resources are utilized efficiently and that no single node is overwhelmed while others remain idle. These concepts play a key role in distributed process scheduling.

Load Sharing

Load Sharing refers to the distribution of tasks across multiple nodes in a distributed system to ensure that all nodes have a roughly equal amount of work. It does not necessarily aim to minimize the overall completion time of tasks but focuses on preventing any node from being idle while others are overloaded.

Dynamic Load Sharing involves making real-time decisions about where to allocate tasks based on the current load of each node. This is in contrast to static load sharing, where the decision is made at the start based on predefined criteria.

Key Features of Dynamic Load Sharing:

Real-Time Decision Making: Tasks are assigned to nodes based on their current load, which can change dynamically.

Adaptability: The system adapts to changing workloads and node availability.

Simple Mechanisms: Common methods include random allocation, round-robin, and threshold-based policies.

Load Balancing

Load Balancing aims to distribute tasks across nodes to minimize the overall completion time, maximize resource utilization, and improve the system's performance. It is more sophisticated than load sharing and often includes mechanisms to monitor node loads continuously and migrate tasks as necessary.

Dynamic Load Balancing involves real-time monitoring and adjustment of task distribution to ensure an even workload across all nodes.

Key Features of Dynamic Load Balancing:

Continuous Monitoring: Constantly monitors the load on each node.

Task Migration: Tasks can be moved between nodes to balance the load.

Complex Algorithms: Often employs more sophisticated algorithms like least connection, shortest expected delay, or weighted round-robin.

Requirement in Distributed Process Scheduling

Solution of Question Paper
II Mid-Term Examination, May -2024

Prog./Sem.: B.Tech. VI Sem	Course Title: Distributed Systems	Course Code:6CS5-11
Duration: 1.5 hours	Date: 31/05/2024 Session: FIRST	Max Marks: 20
Submitted By: Dr. Nilam Choudhary/Ms. Astha Joshi		

Distributed process scheduling is the task of deciding where to execute processes in a distributed system. Both dynamic load sharing and load balancing are essential for effective distributed process scheduling for several reasons:

Improved Performance:

By evenly distributing tasks, the system can ensure that all nodes are effectively utilized, reducing the overall completion time for processes.

Prevents bottlenecks where one node might become a performance limiter due to excessive load.

Scalability:

As the system grows, dynamic load balancing ensures that new nodes can be integrated seamlessly without overloading existing nodes.

Adapts to varying workloads, which is crucial for large-scale distributed systems.

Fault Tolerance and Reliability:

Dynamic load balancing can detect node failures and redistribute tasks from the failed node to other nodes, maintaining system reliability.

Enhances the system's ability to handle failures without significant performance degradation.

Resource Utilization:

Maximizes resource utilization by ensuring that all nodes are active and contributing to the computational workload.

Avoids scenarios where some nodes are idle while others are overloaded, leading to inefficient use of resources.

Energy Efficiency:

By distributing tasks more evenly, the system can prevent nodes from overheating and reduce energy consumption.

Dynamic strategies can adapt to energy-efficient policies, such as turning off idle nodes or reducing their power usage.

Justification

Why Dynamic Load Sharing and Balancing are Necessary:

Heterogeneous Environments: In many distributed systems, nodes can have different capacities and performance characteristics. Dynamic load balancing can take these differences into account, ensuring that more capable nodes handle larger workloads.

Variable Workloads: Workloads in distributed systems can be highly variable and unpredictable. Static load balancing cannot adapt to these changes, whereas dynamic load balancing can adjust in real-time to current conditions.

Solution of Question Paper
II Mid-Term Examination, May -2024

Prog./Sem.: B.Tech. VI Sem	Course Title: Distributed Systems	Course Code:6CS5-11
Duration: 1.5 hours	Date: 31/05/2024 Session: FIRST	Max Marks: 20
Submitted By: Dr. Nilam Choudhary/Ms. Astha Joshi		

User Experience: For systems that provide services to users (e.g., web servers, cloud services), maintaining a balanced load ensures faster response times and a better user experience.

Cost Efficiency: Efficiently utilizing resources can reduce operational costs, particularly in cloud environments where resources are billed based on usage.

Swami Keshvanand Institute of Technology, Management & Gramothan, Jaipur

II -Mid Term Examination, May-2024

Prog./Semester:	**B. Tech.-VI**	**Branch:**	**CSE(AI)/ IT**
Course Title:	**DIP**	**Course Code:**	**6CAI3-01/6IT3-01**
Duration:	**1.5 Hours**	**Maximum Marks:**	**20**
Session : First		**Roll No.:**	
Instructions if any:			

PART -A (short answer type questions)
(All questions are compulsory)

Q. No.	Question	Max. Marks	CO (s)	Bloom's Level
1	What are the common causes of image degradation in digital images?	2	3	1
2	What are the most common edge detection algorithms used in image processing?	2	5	1
3	Explain the term 'redundancy'? If an image is 10 MB before compression and 5 MB after compression, Find the compression ratio and saving percentage.	2	4	2

PART -B (Analytical/Problem solving questions)
(All questions are compulsory)

Q. No.	Question	Max. Marks	CO (s)	Bloom's Level
4	Compare the various types of noises in digital images with the help of suitable graphs.	4	3	4
5	Analyze the hough transform to find out whether the following points (1, 4), (2, 3), (3, 1), (4, 1), (5, 0) are collinear or not. Also find the equation of line.	4	5	4

PTO

PART- C (Descriptive/Analytical/Problem solving/Design questions)

(Attempt any one Question)

<table>
<tr><th>Q. No.</th><th>Question</th><th>Max. Marks</th><th>CO (s)</th><th>Bloom s Leve</th></tr>
<tr><td>6</td><td>Consider the probabilities p(a)=0.2, p(b)=0.3, p(c)=0.1, p(d)=0.4.Encode message "abcad", comprising a string by apply arithmetic coding to compress the message?</td><td>6</td><td>4</td><td>3</td></tr>
<tr><td>7</td><td>Determine the Huffman code assignment procedure for the following Data.
<table>
<tr><td>Symbol</td><td>A</td><td>B</td><td>C</td><td>D</td><td>E</td><td>F</td></tr>
<tr><td>Probability</td><td>0.1</td><td>0.4</td><td>0.06</td><td>0.1</td><td>0.04</td><td>0.3</td></tr>
</table></td><td>6</td><td>4</td><td>5</td></tr>
</table>

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017

Solution of Question Paper
II Mid-Term Examination, May -2024

Prog./Sem.: B. Tech.-VI	Course Title: DIP	Course Code: 6CAI3-01/6IT3-01
Duration: 1.5 hours	Date: 28.05.2024 Session : FIRST	Max Marks: 20
Submitted By: Dr. S.R. Dogiwal, Mr. Sohan Lal Gupta		

S. No.	Course Outcome	Percentage Coverage
CO1	**Identify** the fundamental elements of an Image and **Describe** the need of Digital Image Processing	0%
CO2	**Understand** different types of image transformation techniques and their properties.	0%
CO3	**Use** various noise models and **Calculate** the values for restoration and degradation.	23%
CO4	**Analyze** and **Evaluate** various image compression techniques.	54%
CO5	**Integrate** and **Demonstrate** various image transformation and segmentation techniques.	23%

S. No.	Bloom's Level	Percentage Coverage
1	BL1- Remember	15%
2	BL2- Understand	8%
3	BL3- Apply	23%
4	BL4- Analyze	31%
5	BL5- Evaluate	23%
6	BL6- Create	0

-- Question Paper Solution --

Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017

Solution of Question Paper
II Mid-Term Examination, May -2024

Prog./Sem.: B. Tech.-VI	**Course Title: DIP**	**Course Code: 6CAI3-01/6IT3-01**
Duration: 1.5 hours	**Date: 28.05.2024** **Session : FIRST**	**Max Marks: 20**
Submitted By: Dr. S.R. Dogiwal, Mr. Sohan Lal Gupta		

Q.1 What are the common causes of image degradation in digital images?

Solution: Image degradation in digital images can be caused by various factors, including:

1. Noise: This includes random variations in brightness or color in an image. Noise can be introduced during image capture, transmission, or processing. Common types of noise are:

Gaussian Noise: A statistical noise that has a probability density function equal to that of the normal distribution.

Salt-and-Pepper Noise: Occurs when pixels in an image are either very bright or very dark, giving the appearance of white and black dots.

2. Blur: This occurs due to the loss of sharpness and detail in an image and can result from:

Motion Blur: Caused by the movement of the camera or subject during exposure.

Out-of-Focus Blur: Occurs when the camera lens is not focused correctly.

Atmospheric Blur: Caused by particles or turbulence in the air affecting the clarity of the captured image.

3. Compression Artifacts: Distortions that occur when an image is compressed, especially using lossy compression algorithms like JPEG. These artifacts include:

4. Resolution Loss: This results from reducing the image size or using a lower resolution sensor during capture, leading to a loss of detail and clarity.

5. Color Distortion: Changes in the original colors of an image due to:

6. Digital Sampling: Errors introduced during the conversion of an analog image to a digital format, such as:

7. Quantization Errors: Loss of image detail resulting from reducing the number of bits used to represent each pixel's color value during compression or processing.

8. Environmental Factors: External conditions like dust, humidity, and temperature variations can affect the image sensor and degrade image quality.

Q.2 What are the most common edge detection algorithms used in image processing?

Solution: Edge detection is a critical step in image processing and computer vision, used to identify the boundaries of objects within images. Here are some of the most common edge detection algorithms:

Sobel Operator: Uses two 3x3 convolution kernels, one for detecting changes in the horizontal direction and one for the vertical direction. Highlights regions of high spatial frequency, typically corresponding to edges.

Prewitt Operator: Similar to the Sobel operator, but uses different kernels for gradient approximation.Often used in edge detection to approximate the gradient of the image intensity function.

Solution of Question Paper
II Mid-Term Examination, May -2024

Prog./Sem.: B. Tech.-VI	Course Title: DIP	Course Code: 6CAI3-01/6IT3-01
Duration: 1.5 hours	Date: 28.05.2024 Session : FIRST	Max Marks: 20
Submitted By: Dr. S.R. Dogiwal, Mr. Sohan Lal Gupta		

Canny Edge Detector: A multi-stage algorithm that involves smoothing the image with a Gaussian filter, finding intensity gradients, applying non-maximum suppression, and using double thresholding and edge tracking by hysteresis. Known for its effectiveness in detecting true edges and reducing noise.

Laplacian of Gaussian (LoG): Applies a Gaussian smoothing filter to the image and then uses the Laplacian operator to detect edges. Detects edges by looking for zero-crossings after the Laplacian filter is applied.

Roberts Cross Operator: Uses 2x2 convolution kernels to approximate the gradient of the image intensity function. One of the oldest edge detection algorithms, simple but sensitive to noise.

Scharr Operator: An improvement over the Sobel operator, providing better rotational symmetry and accuracy. Often preferred when high precision in edge detection is required.

Kirsch Operator: Uses eight convolution kernels to detect edges in all eight compass directions (N, NE, E, SE, S, SW, W, NW). Provides directional edge detection and can be useful in specific applications where directionality is important.

Marr-Hildreth (Laplacian of Gaussian): Combines Gaussian smoothing with the Laplacian operator to find edges, similar to LoG. Effective in detecting edges and reducing noise, although it can sometimes produce false edges due to the second derivative.

Q.3 Explain the term 'redundancy'? If an image is 10 MB before compression and 5 MB after compression, Find the compression ratio and saving percentage.

Solution: Redundancy in the context of information theory and image processing refers to the presence of unnecessary or repetitive information that can be removed or compressed without significantly affecting the quality or understanding of the data. It is a key concept in compression algorithms, which aim to reduce file sizes by eliminating this redundant information.

Types of Redundancy

Spatial Redundancy, Temporal Redundancy, Spectral Redundancy, Psycho-visual Redundancy

Compression Ratio=10/5 =2MB,
Percentage Reduction=50%

Q.4 Compare the various types of noises in digital images with the help of suitable graphs.

Solution: In digital images, various types of noise can degrade image quality. Each type of noise has distinct characteristics and impacts image processing differently. Below, I compare the common types of noise in digital images with descriptions and graphical representations:

1. Gaussian Noise: Gaussian noise is statistical noise with a probability density function equal to that of the normal distribution (Gaussian distribution). It is characterized by its mean (average intensity) and variance (spread).

Graphical Representation: A histogram of Gaussian noise typically shows a bell-shaped curve centered around the mean value.

2. Salt-and-Pepper Noise: Salt-and-pepper noise, also known as impulse noise, appears as sparsely occurring white and black pixels in an image. It is caused by sudden disturbances in the image signal.

Graphical Representation: The noise histogram shows two distinct peaks, one near the minimum value (black) and one near the maximum value (white), with the original data in between.

3. Poisson Noise (Shot Noise): Poisson noise, also known as shot noise, is related to the discrete nature of electronic charge. It is commonly seen in photon-counting devices like digital cameras.
Graphical Representation: The histogram for Poisson noise resembles the Poisson distribution, which is skewed and has a single peak.
4. Speckle Noise: Speckle noise is multiplicative noise commonly seen in radar and medical imaging. It appears as a granular pattern.
Graphical Representation: The noise histogram shows a multiplicative effect where noise is proportional to the intensity of the image pixels.
5. Uniform Noise: Uniform noise has a constant probability distribution, meaning each value in the range has an equal probability of occurring. It is often used to model quantization noise.
Graphical Representation: The histogram for uniform noise is flat, indicating that all values are equally probable.

Summary of Characteristics

Noise Type	Characteristics	Histogram Shape
Gaussian Noise	Bell-shaped distribution, mean and variance	Bell-shaped curve
Salt-and-Pepper	Sparse black and white pixels	Two peaks at extremes (black and white)
Poisson Noise	Related to signal intensity, skewed	Poisson distribution
Speckle Noise	Multiplicative noise, grainy appearance	Dependent on image intensity
Uniform Noise	Equal probability of all values	Flat distribution

Conclusion: Understanding the types of noise and their characteristics is essential for selecting appropriate noise reduction techniques in image processing. Each noise type requires different strategies for effective removal or reduction, thus preserving the quality and integrity of the digital image.

Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagaria, Jagatpura, Jaipur-302017

Solution of Question Paper
II Mid-Term Examination, May -2024

Prog./Sem.: B. Tech.-VI	Course Title: DIP	Course Code: 6CAI3-01/6IT3-01
Duration: 1.5 hours	Date: 28.05.2024 Session : FIRST	Max Marks: 20
Submitted By: Dr. S.R. Dogiwal, Mr. Sohan Lal Gupta		

Q.5 Analyze the hough transform to find out whether the following points (1, 4), (2, 3), (3, 1), (4, 1), (5, 0) are collinear or not. Also find the equation of line.

Solution:

Solution. The equation of line is $y = mx + c$

- In order to perform Hough transform, we need to convert line from (x, y) plane to (m, c) plane.
- equation of line in (m, c) plane is $\boxed{c = -mx + y}$

⇒ For $(x, y) = (1, 1)$

$c = -m + 1$

$\left\{\begin{array}{ll} \text{if } c = 0 & \text{then } m = 1 \\ \text{if } m = 0 & \text{then } c = 1 \end{array}\right\}$ So

Thus $(m, c) = (1, 1)$

⇒ For $(x, y) = (2, 2)$ $\boxed{c = -mx + y}$

$c = -2m + 2$

$\left\{\begin{array}{ll} \text{if } c = 0 & \text{then } m = 1 \\ \text{if } m = 0 & \text{then } c = 2 \end{array}\right\}$

Thus $(m, c) = (1, 2)$

⇒ For $(x, y) = (3, 3)$ $c = -3m + 3$

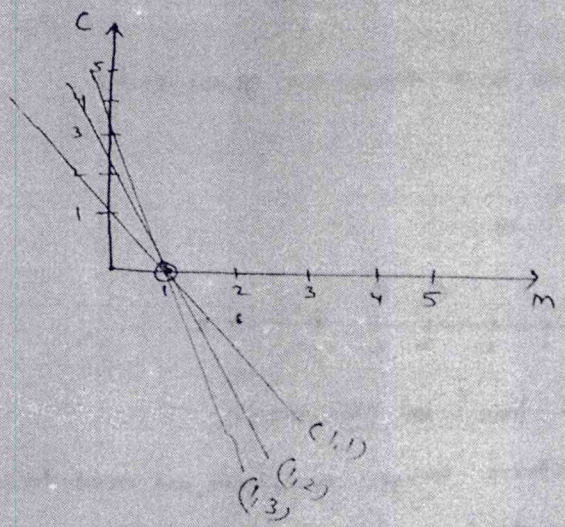
$\left\{\begin{array}{ll} \text{if } c = 0 & \text{then } m = 1 \\ \text{if } m = 0 & \text{then } c = 3 \end{array}\right\}$

Thus $(m, c) = (1, 3)$ (14)

Solution of Question Paper
II Mid-Term Examination, May -2024

Prog./Sem.: B. Tech.-VI	Course Title: DIP	Course Code: 6CAI3-01/6IT3-01
Duration: 1.5 hours	Date: 28.05.2024 Session : FIRST	Max Marks: 20
Submitted By: Dr. S.R. Dogiwal, Mr. Sohan Lal Gupta		

Finally $(m, c) = (1,1), (1,2), (1,3)$



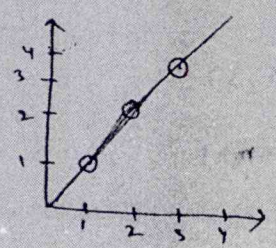
All three lines lie match at 1 point so $(m, c) = (1, 0)$

The original equation of line is $y = mx + c$

so put value of ~~m and c~~ $m = 1$ and $c = 0$ so

$\boxed{y = x}$ the ~~the~~ equation of line.

So we are show that point $(1,1), (2,2), (3,3)$ are collinear.



All three point on a single line, so it is colinear.

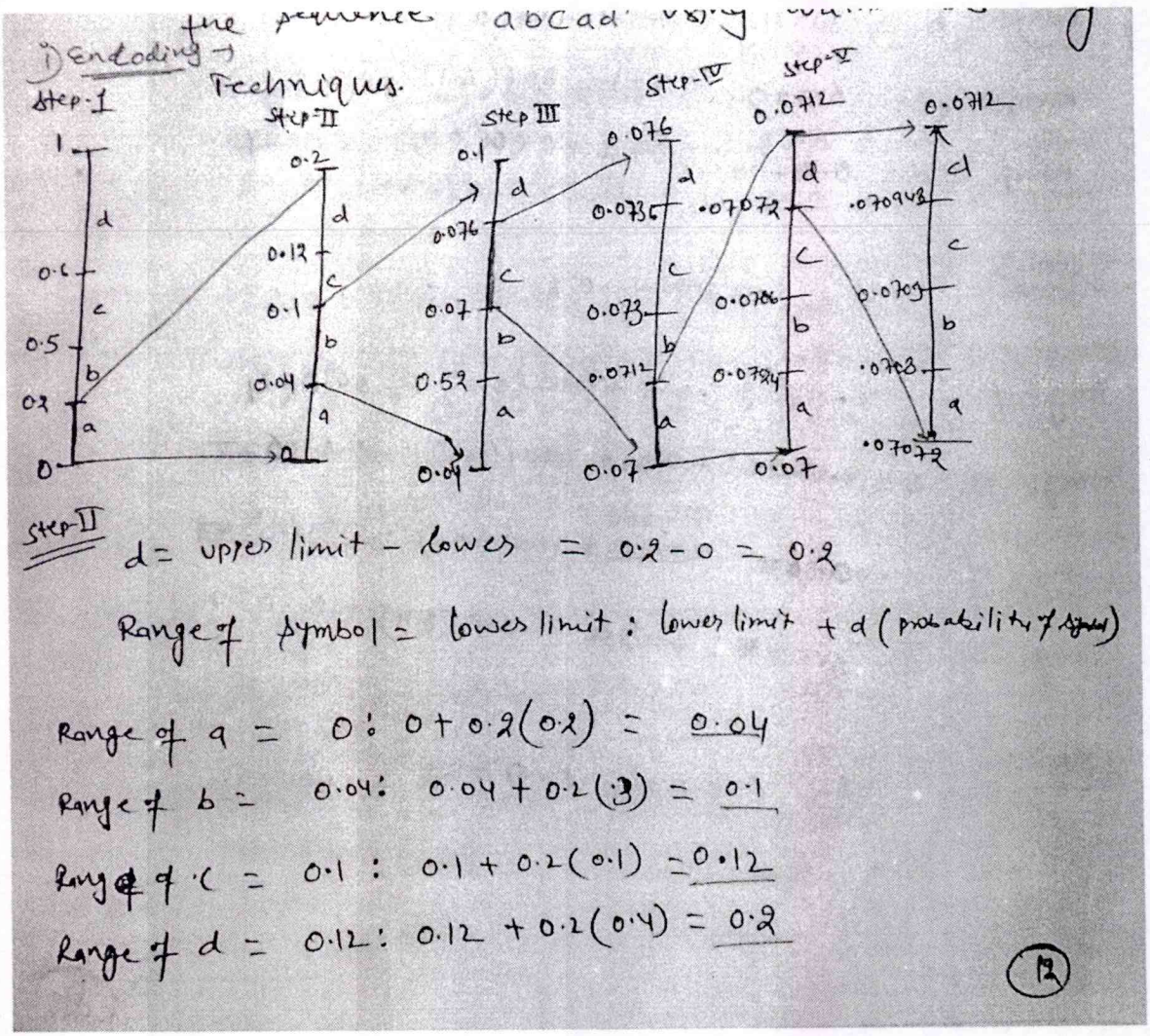
Swami Keshvanand Institute of Technology, Management & Gramothan, Ramnagaria, Jagatpura, Jaipur-302017

Solution of Question Paper
II Mid-Term Examination, May -2024

Prog./Sem.: B. Tech.-VI	Course Title: DIP	Course Code: 6CAI3-01/6IT3-01
Duration: 1.5 hours	Date: 28.05.2024 Session : FIRST	Max Marks: 20
Submitted By: Dr. S.R. Dogiwal, Mr. Sohan Lal Gupta		

Q.6 Consider the probabilities p(a)=0.2, p(b)=0.3, p(c)=0.1, p(d)=0.4. Encode message "abcad", comprising a string by apply arithmetic coding to compress the message?

Solution:



Solution of Question Paper
II Mid-Term Examination, May -2024

Prog./Sem.: B. Tech.-VI	Course Title: DIP	Course Code: 6CAI3-01/6IT3-01
Duration: 1.5 hours	Date: 28.05.2024 Session : FIRST	Max Marks: 20
Submitted By: Dr. S.R. Dogiwal, Mr. Sohan Lal Gupta		

Step III →

d = upper limit - lower limit = 0.1 − 0.04 = 0.06

Range of a = 0.04 : 0.04 + 0.06(0.2) = .052
Range of b = 0.052 : .052 + 0.06(0.3) = .07
Range of c = 0.07 : .07 + 0.06 (0.1) = 0.076
Range of d = 0.076 : 0.076 + 0.06 (0.4) = 0.08

Step IV

d = upper limit − lower limit = 0.076 − 0.070 = .006

Range of a = .070 : .07 + .006 (0.2) = 0.0712
Range of b = 0.0712 : 0.0712 + .006 (0.3) = 0.073
Range of c = 0.073 : 0.073 + .006(0.1) = 0.0736
Range of d = 0.0736 : 0.0736 + 0.006(0.4) = 0.076

Step V → d = .0712 − .07 = .0012

Range of a = 0.070 : .070 + .0012 (0.2) = 0.07024
Range of b = 0.07024 : .07024 + .0012 (0.3) = 0.07060
Range of c = 0.0706 : 0.0706 + .0012(0.1) = 0.070732
Range of d = 0.070732 : 0.070732 + .0012(0.4) = 0.0712

Step VI → d = 0.0712 − 0.070732 = .00048

Solution of Question Paper
II Mid-Term Examination, May -2024

Prog./Sem.: B. Tech.-VI	Course Title: DIP	Course Code: 6CAI3-01/6IT3-01
Duration: 1.5 hours	Date: 28.05.2024 Session : FIRST	Max Marks: 20
Submitted By: Dr. S.R. Dogiwal, Mr. Sohan Lal Gupta		

Range of a = .07072 : .07072 + .00048(0.2) = .0708

Range of b = .0708 : .0708 + .00048(0.3) = .0709

Range of c = .0709 : .0709 + .00048(0.1) = .070948

Range of d = .070948 : .070948 + .00048(0.4) = .0712

Now we will found out tag. and Code word.

→ Here Code Word is

0.070948 < Code word is < 0.0712

$$tag = \frac{\text{upper limit} + \text{lower limit}}{2}$$

$$= \frac{0.0712 + 0.07072}{2}$$

Tag. = 0.07096

Q.7 Determine the Huffman code assignment procedure for the following Data.

Symbol	A	B	C	D	E	F
Probability	0.1	0.4	0.06	0.1	0.04	0.3

Solution: Determine the Huffman code assignment for the given symbols and their probabilities, we can follow these steps:

Step 1: List the Symbols and Their Probabilities

Symbol	A	B	C	D	E	F
Probability	0.1	0.4	0.06	0.1	0.04	0.3

Step 2: Create a Min-Heap of All Symbols

Solution of Question Paper
II Mid-Term Examination, May -2024

Prog./Sem.: B. Tech.-VI	Course Title: DIP	Course Code: 6CAI3-01/6IT3-01
Duration: 1.5 hours	Date: 28.05.2024 Session : FIRST	Max Marks: 20
Submitted By: Dr. S.R. Dogiwal, Mr. Sohan Lal Gupta		

Create nodes for each symbol and their probabilities.
Insert all nodes into a min-heap (priority queue) based on their probabilities.

Step 3: Build the Huffman Tree

First Iteration:

Remove the two nodes with the smallest probabilities (E: 0.04 and C: 0.06).

Combine them to create a new node with probability 0.04+0.06=0.1

Insert the new node back into the min-heap.

Second Iteration:

Remove the two nodes with the smallest probabilities (A: 0.1 and D: 0.1).

Combine them to create a new node with probability 0.1+0.1=0.2.

Insert the new node back into the min-heap.

Third Iteration:

Remove the two nodes with the smallest probabilities (new node 0.1 and F: 0.3).

Combine them to create a new node with probability 0.1+0.3=0.4.

Insert the new node back into the min-heap.

Fourth Iteration:

Remove the two nodes with the smallest probabilities (B: 0.4 and new node 0.2).

Combine them to create a new node with probability 0.4+0.2=0.6.

Insert the new node back into the min-heap.

Fifth Iteration:

Remove the two remaining nodes (new node 0.4 and new node 0.6).

Combine them to create the root node with probability 0.4+0.6=1.0.

Step 4: Assign Huffman Codes:

Traverse the Huffman tree from the root, assigning binary codes to each symbol. Move left for '0' and right for '1'.

Assigning Codes: B: 1 A: 01 D: 00 F: 10 E: 110 C: 111

Summary of Huffman Codes:

Symbol	Probability	Huffman Code
A	0.1	01
B	0.4	1
C	0.06	111
D	0.1	00
E	0.04	110
F	0.3	10