Unit- 8

Fault Tolerance

- **Introduction:**

Fault means defect with hardware or software. Fault tolerance is defined as the characteristics by which a system can mask the occurrence and recovery from failures.

Requirement of Fault Tolerant System

The useful requirements for distributed systems include the following:

- **Availability:** It is defined as the property that a system is ready to be used immediately.
- **Reliability:** It refers to the property that a system can run continuously without failure.
- **Safety:** It refers to the situation that when a system temporarily fails to operate correctly.
- **Maintainability:** It refers to how easy a failed system can be repaired.A highly maintainable system may also show a high degree of availability, especially if failures can be detected and repaired automatically.

- **<u>Types of Faults:</u>**

Faults are generally classified as following:

- **Node fault:** A fault that occurred at an individual node participating in the distributed system. Example, a machine in a distributed system fails due to some error in database configuration of that machine.
- **Program fault:** A fault that occurred due to some logical or syntactical errors in the code. Example: a program that is designed to filter the data by category but instead filtered by other means.
- **Communication faults:** A fault that occur due to unreliable communication channels connecting the nodes. Example: a node sending data 010110 to another node but due to some problem in communication channel the receiver receives "010111".
- **Timing fault:** The fault that occurred due to mismatch on testing of any particular response. Example: a server replies too late or a server is provided with data too soon that it has no enough buffer to hold the data.

- **Failure models:**

Types of failure	**Description**
Crash failure	A server halts, but is working correctly until it halts.
Omission Failure receive omission send omission	A server fails to respond to incoming requests A server fails to receive incoming message A server fails to send message
Timing failure	A server’s response lies outside the specified time interval
Response failure value failure State transition failure	A server’s respond to incorrect The value of the response is wrong The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary response at arbitrary times.

- **Process Resilience:**

Process resilience is a mechanism to protect against faulty process by replicating and distributing computations in a group. The group can be of two types.

Flat Group:

In flat group all the process with in a group have equal roles and control is completely distributed to all process.

The flat group has no single point of failure.

If one of the processes crashes, the group simply becomes smaller, but can otherwise continue.

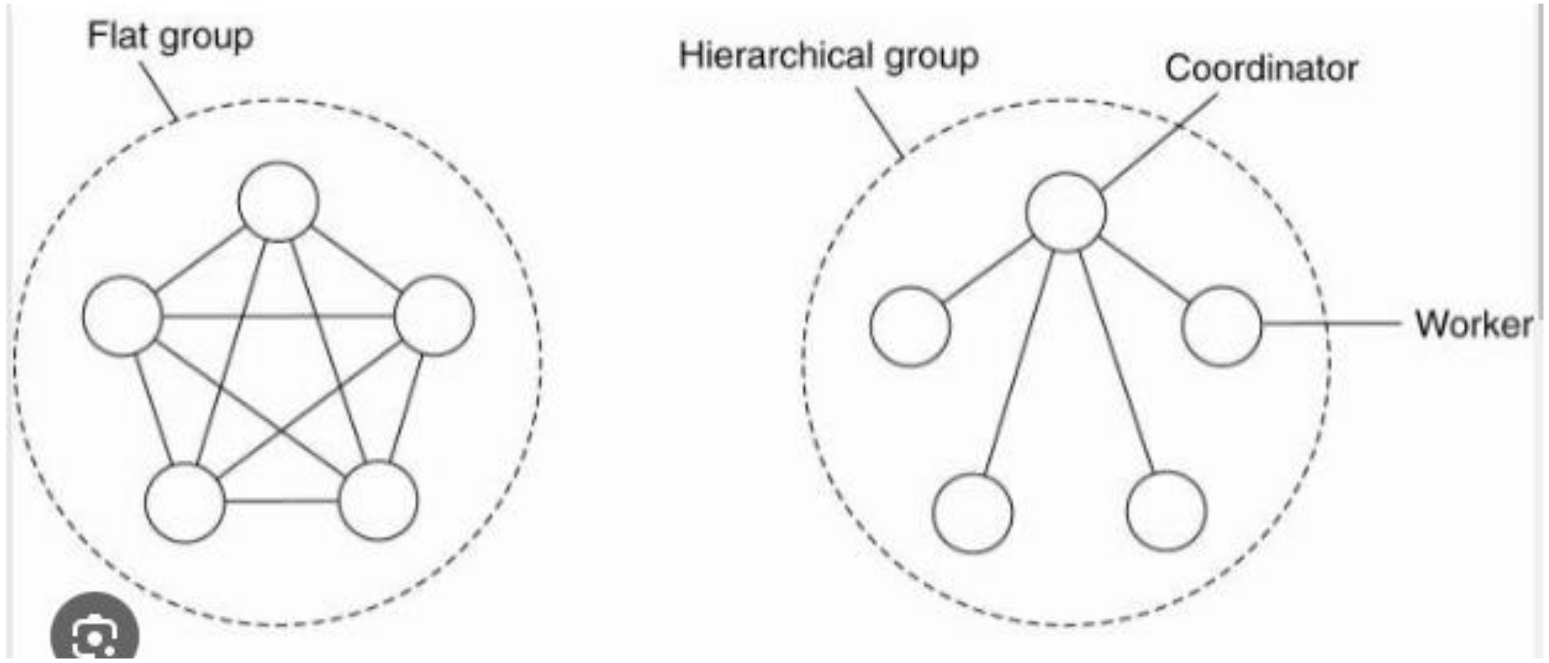
A disadvantage is that decision making is more complicated. For example: to decide anything , a vote often has to be taken, incurring some delay and overhead.

- **<u>Hierarchical Group:</u>**

The hierarchical group has the opposite properties . All the communication are handled by a single process designed as coordinator.

Loss of the coordinator brings the entire group to a grinding halt.

In practice, when the coordinator in a hierarchical group fails, its role will need to be taken over and one of the workers is elected as new coordinator.



- **Reliable Client server Communication:**

In many cases fault tolerance in distributed system concentrates on faulty process. However, we also need to consider communication failures because a communication channel may exhibit crash, omission, timing and arbitrary failures.

The five different classes of failures that can occur as follows:

1. The client is unable to locate the server.
2. The request message from the client to the server is lost.
3. The server crashes after receiving a request
4. The reply message from the server to the client is lost.
5. The client crashes after sending a request.

Each of these categories posed different problem and requires different solution. The solution are:

1. Report back to client
2. Resend the message
3. Use of remote procedure call(RPC) semantics
4. Operations should be idempotent (send multiple requests)
5. Kill the orphan computation

- **Reliable Group Communication:**

Reliable group communication means that a message that is sent to a process group should be delivered to each member of the group. If we separate the logic of handling messages from the core functionality of a group member, we can conveniently make the distinction between receiving message and delivering message.

A message is received by a message handling component, which, in turn, delivers a message to the component containing the core functionality of a group member, a message that is received by process P will also be delivered by p.

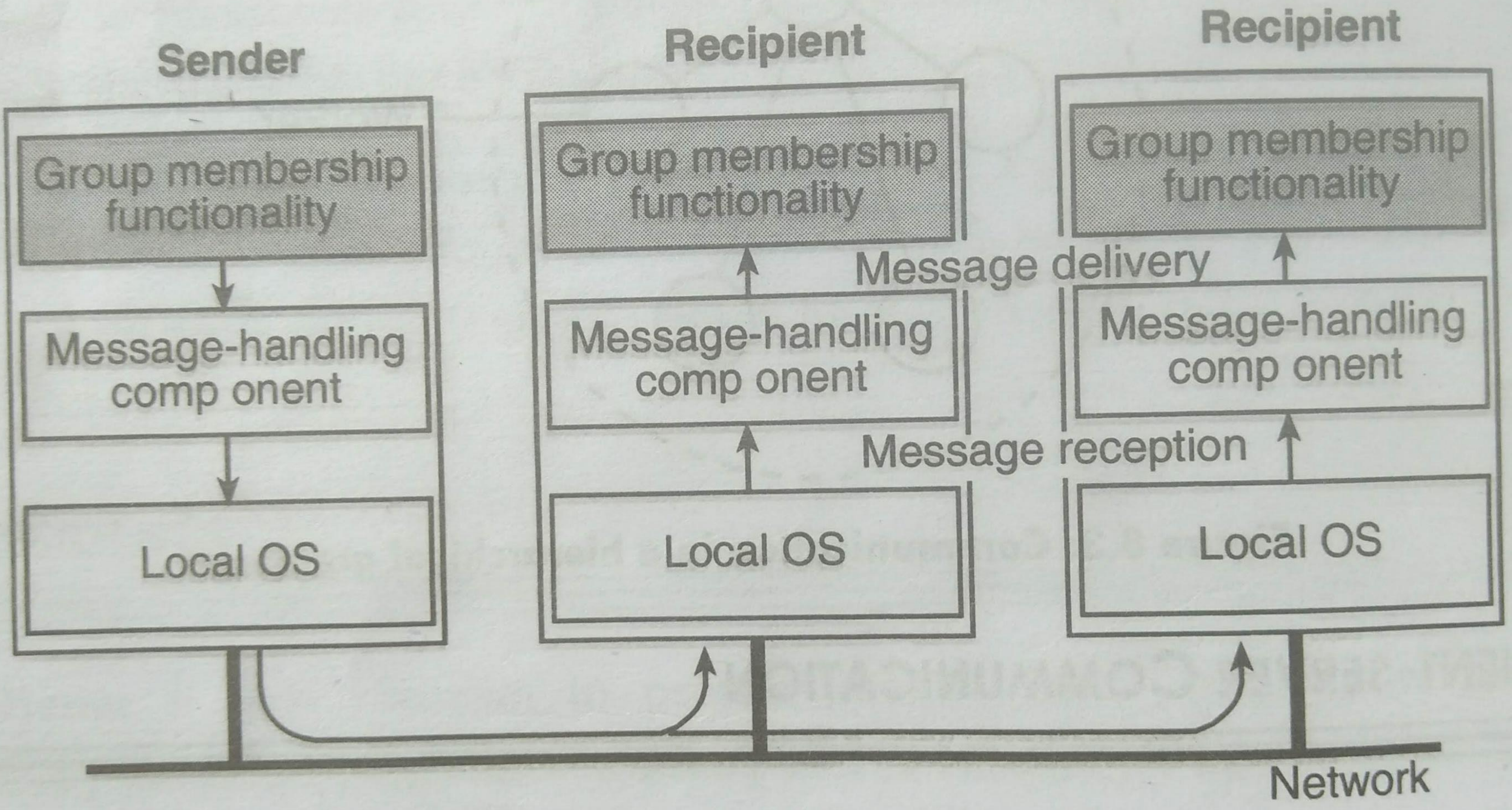

Sender
Recipient
Recipient
Group membership functionality
Group membership functionality
Group membership functionality
Message delivery
Message-handling comp onent
Message-handling comp onent
Message-handling comp onent
Message reception
Local OS
Local OS
Local OS
Network

Figure 8.4: The distinction between receiving and delivering messages.

- <u>**Distributed Commit:**</u>
- **Commit = Saving**
- **Goals:**An operation to be performed by all group members or none at all.

Distributed Commit deals with methods to ensure the either all the process commit to the final result or none of them do. This ensures consistency of the distributed system.

Types of Distributed Commit:

One phase:

An elected coordinators tells all the other processes to perform that are involved.

But if a process cannot perform the operation –there is no way to tell.

- **Two Phase:**

One coordinator responsible for initiating protocol.

Other entities called participants

If coordinator or participants unable to commit all parts of transaction are aborted.

Two phase:

Phas1: The coordinator sends a can commit message to all participants in transaction.

Phase2: Implement that decision at all sites.

- <u>**Algorithm:**</u>
- **Coordinator:**

1. Multicast Commit request to all participants.
2. If all commits are not collected, repeat:

    - wait for any incoming commit

    - If timeout, write GLOBAL-COMMIT and Multicat to all participants
3. If all participants send COMMIT, write GLOBAL-COMMIT and multicast to all participants; else write, GLOBAL-ABORT and multicast to all participants.

- **Participants:**

1. Wait for COMMIT-REQUEST from coordinator
2. If timeout, write VOTE-ABORT and exit.
3. If participants votes ABORT, write VOTE-ABORT and send ABORT to coordinator.
4. If participants votes COMMIT

   - Write VOTE-COMMIT and send COMMIT to coordinator

   - Wait for DECISION from coordinator

   - If time out , multicast DECISION-REQUEST to other participants, waits for decision and write decision.

   - if coordinator DECISION is COMMIT write GLOBAL-COMMIT else write GLOBAL-ABORT.

- **Handling Decision request:**

1. Repeat until true:

- Wait until any incoming DECISION-REQUEST is received.
- Reads most recently recorded state.
- If state is GLOBAL-COMMIT send GLOBAL-COMMIT
- Else If state is GLOBAL-ABORT send GLOBAL-ABORT
- ELSE,SKIP.

Recovery:

Recovery is the mechanism to handles failure. It helps to recover correct state of the system after failure. The main idea of error recovery is to replace an erroneous state with an error state. There are essentially two forms of error recovery.

In **backward recovery:** The main issue is to bring the system from its present erroneous state back into a previously correct state. To do so, it will be necessary to record the system's state from time to time, and to restore such a recorded state when things go wrong. Each time the system's present state is recorded, a checkpoint is said to be made.

Forward recovery:

When the system has entered an erroneous state, instead of moving back to a previous, check pointed state, an attempt is made to bring the system in a correct new state from which it can continue to execute. The main problem with forwarded error recovery mechanism is that it has to be known in advance which errors may occur . Only in that case it possible to correct those errors and move to a new state. Some of the recovery techniques are:

Coordinated Checkpoint:

- A coordinated recovery checkpoint is a mechanism used to ensure fault tolerance and recoverability in the event of failures. It involves capturing the system's state at regular intervals, known as checkpoints, and storing this information in a stable storage medium. In the event of a failure, the system can be restored to a consistent state using the information from the most recent checkpoint.

The coordinated recovery checkpoint process typically involves the following steps:

- Checkpoint Creation: At regular intervals, all processes in the distributed system agree to take a checkpoint. Each process saves its state, including its local variables, data structures, and any relevant system information. This state is then stored in stable storage, which is usually a durable and reliable storage medium, such as a disk.
- Checkpoint Coordination: To ensure consistency, the distributed system needs to coordinate the checkpoint process across all participating processes. This coordination can be achieved using various algorithms, such as the Chandy-Lamport algorithm or the Maekawa's algorithm. These algorithms ensure that all processes take checkpoints in a coordinated and consistent manner.
- Checkpoint Storage: Once a process completes its checkpoint, it stores the checkpoint data in stable storage. Stable storage provides durability, meaning that even in the event of a failure, the checkpoint data remains intact and can be accessed for recovery purposes.

- Failure Detection: The distributed system continuously monitors the status of processes to detect failures. If a process fails, the failure is detected through timeout mechanisms.
- Recovery: In the event of a failure, the system needs to recover to a consistent state. The recovery process involves rolling back the system state to a previous checkpoint and then replaying the recorded events from the checkpoint onward. This process restores the system to a known and consistent state before the failure occurred.

- **Independent Checkpoint:**
- An independent checkpoint in recovery refers to a mechanism where each individual process or component in the system takes checkpoints independently without coordinating with other processes. This approach allows for faster checkpoint creation and reduces the coordination overhead compared to coordinated recovery checkpoints. However, it also introduces additional challenges in achieving a consistent global system state during recovery.

In the case of independent checkpoints in distributed system recovery, the process typically follows these steps:

- **Checkpoint Creation:** Each process independently decides when to take a checkpoint based on its internal criteria or predefined policies. When a process decides to take a checkpoint, it saves its local state and relevant system information without coordinating with other processes.
- **Checkpoint Storage:** After a process takes a checkpoint, it stores the checkpoint data in stable storage. This could be a durable and reliable storage medium such as a disk or any other persistent storage mechanism. The process ensures that the checkpoint data is safely written and can be accessed for recovery purposes.

- **Failure Detection:** Similar to coordinated recovery, the distributed system continuously monitors the status of processes to detect failures. Failures can be detected through timeout mechanisms.
- **Recovery:** When a failure occurs, the recovery process involves restoring the system to a consistent state. Each process independently recovers by rolling back to its most recent checkpoint and then replaying the recorded events from that checkpoint onwards. As each process recovers independently, there is no coordination required among processes during recovery.

- **Message logging:**
- Message logging recovery is a technique used in distributed systems to achieve fault tolerance and recoverability by recording and replaying messages exchanged between processes. It involves logging messages sent and received by processes in stable storage, allowing for recovery and system restoration in the event of failures. This approach ensures that the system can recover its state and continue operation from a known consistent point.
- The message logging recovery process typically involves the following steps:

- **Message Logging:** Each process in the distributed system logs the messages it sends and receives. When a process sends a message, it logs the message along with any relevant metadata, such as the sender, receiver, timestamp, and message contents. Similarly, when a process receives a message, it logs the received message. The logged messages are stored in stable storage, ensuring durability and availability even in the event of failures.
- **Checkpoint Creation:** Along with message logging, periodic checkpoints are taken to capture the system's state. These checkpoints include the process's local state, such as variables, data structures, and any relevant system information. Checkpoints can be taken using coordinated recovery checkpoints or independent checkpoint mechanisms, as discussed in the previous response.
- **Failure Detection:** The distributed system continuously monitors processes to detect failures. Failures can be detected through timeouts. When a failure is detected, the recovery process is triggered.
- **Recovery and Message Replay:** In the event of a failure, the recovery process begins. The system rolls back to a consistent checkpoint, restoring the state of each process to that point. After the rollback, the logged messages from the stable storage are replayed in the order they were originally logged. By replaying the messages, the system restores the interprocess communication and brings the system back to the state it was in before the failure occurred.