Fault Tolerance

Chapter 8

Fault Tolerance

- An important goal in distributed systems design is to construct the system in such a way that it can automatically recover from partial failure without seriously affecting the overall performance.
- A distributed system should tolerate faults and continue to operate to some extent even in their presence.

Basic Concepts

- Requirement of Fault Tolerance :-
  - **Availability:** a system is ready to be used immediately.
  - **Reliability:** a system can run continuously without failure.
  - **Safety:** when a system temporarily fails to operate, nothing catastrophic happens.
  - **Maintainability:** how easy a failed system can be repaired.

Basic Concepts

- A system is said to **fail** when it cannot meet its promises.
- An **error** is a part of a system's state that may lead to a failure.
- The cause of an error is a **fault.**
- **Fault tolerance** means that a system can provide its services even in the presence of faults.
- A **transient fault** occurs once and then disappears.
- An **intermittent fault** occurs, then vanishes of its own agreement, then reappears, and so on.
- A **permanent fault** is one that continues to exist until the faulty component is repaired.

Failure Models

Type 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 messages A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure *Value failure* *State transition failure*	The server's response is incorrect The value of the response is wrong The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times

Different types of failures.

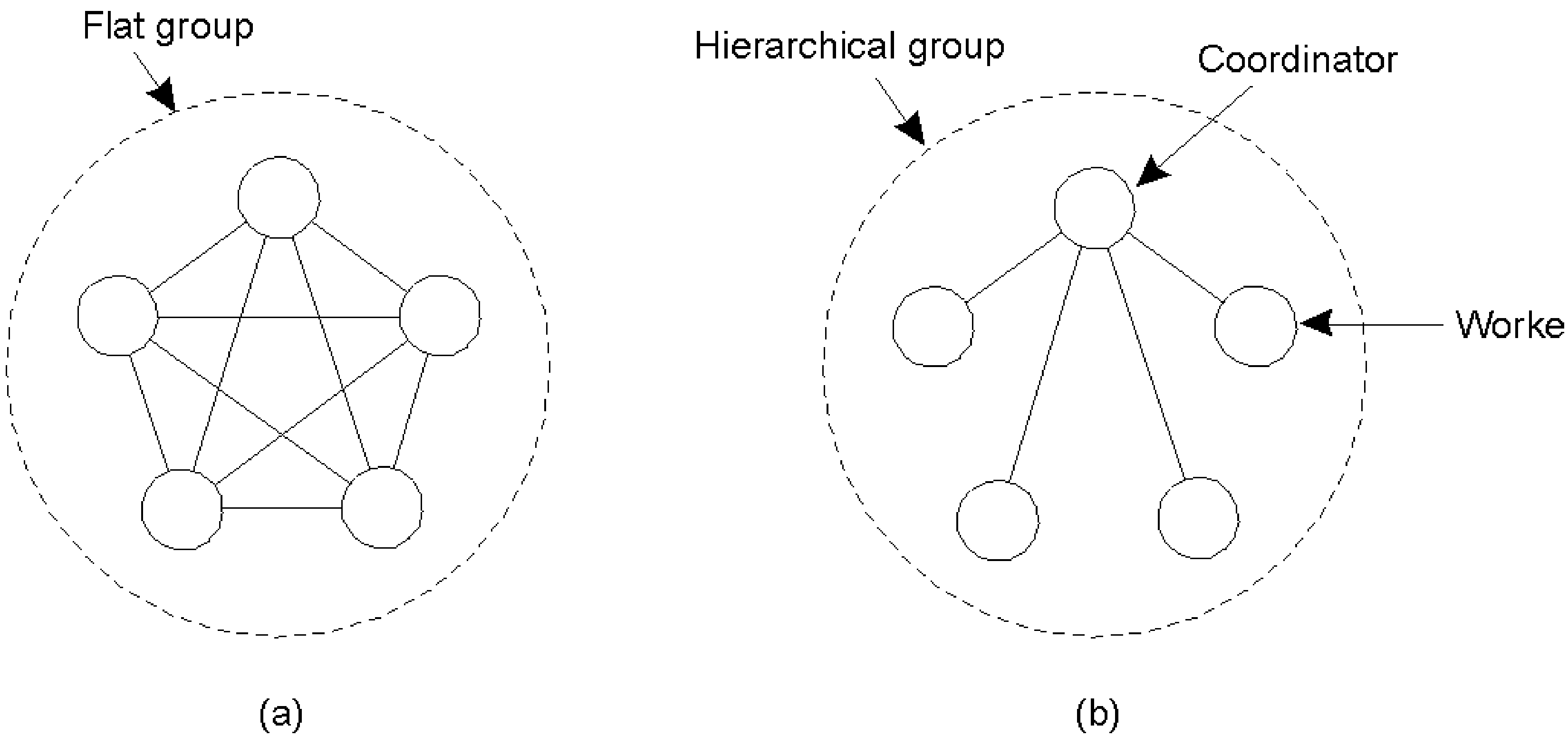
Process resilience

- The first topic we discuss is protection against process failures, which is achieved by replicating processes into groups.

Resilience by process groups

- The key approach to tolerating a faulty process is to organize several identical processes into a group.
- The key property that all groups have is that when a message is sent to the group itself, all members of the group receive it.
- In this way, if one process in a group fails, hopefully some other process can take over for it

Flat Groups versus Hierarchical Groups



a) Communication in a flat group.
b) Communication in a simple hierarchical group

Flat group and Hierarchical

- The flat group is symmetrical and 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.
- The hierarchical group has the opposite properties. Loss of the coordinator brings the entire group to a grinding halt, but as long as it is running, it can make decisions without bothering everyone else. 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

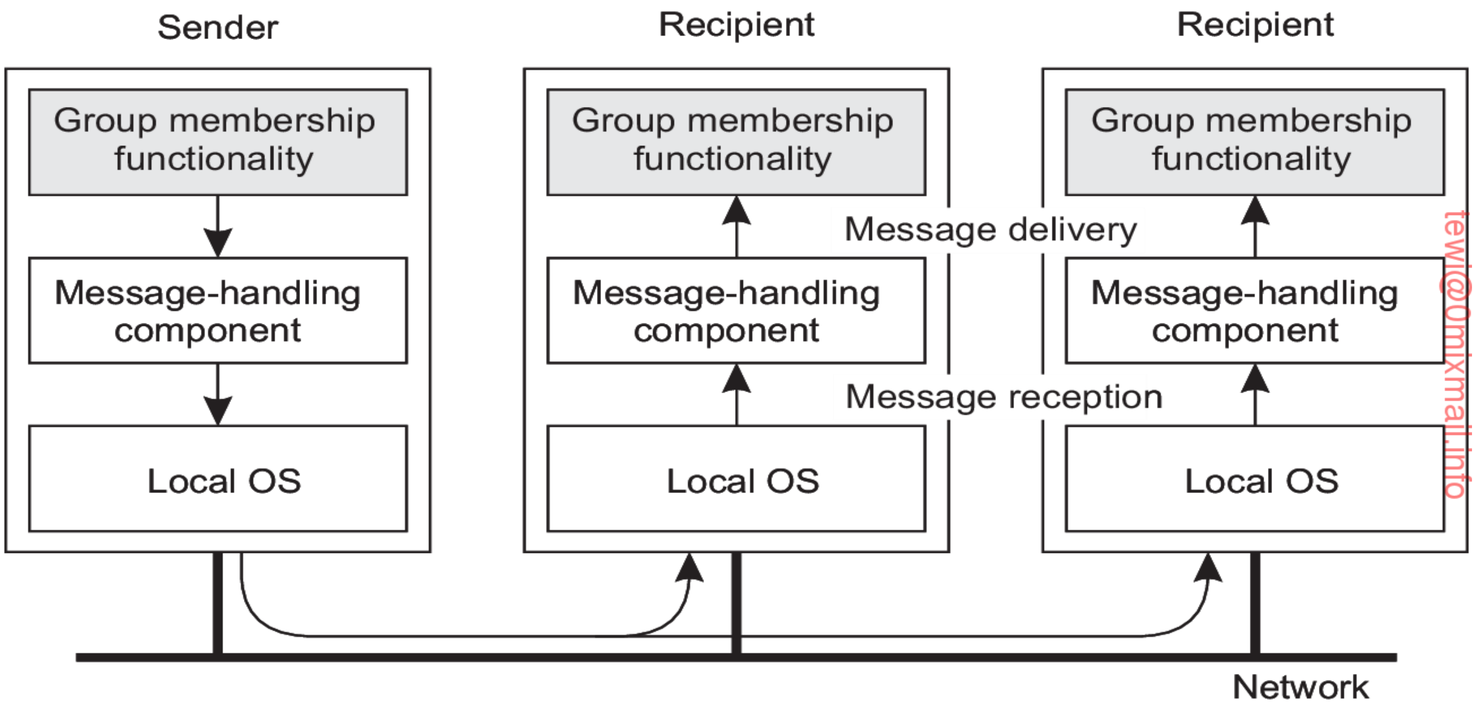
- The five different classes of failures that can occur are as follows:-

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.

Reliable group communication

- Reliable group communication means that a message that is sent to a process group should be delivered to each member of that 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* messages and *delivering* messages.
- 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.
- As an example, ensuring that messages from the same sender are delivered in the same order as they were sent, is typically taken care of by a message- handling component.

Reliable group communication



Distributed commit

- Distributed commit is often established by means of a coordinator.
- In a simple scheme, this coordinator tells all other processes that are also involved, called participants, whether or not to (locally) perform the operation in question.
- The distributed commit problem involves having an operation being performed by each member of a process group, or none at all.
- In the case of reliable multicasting, the operation is the delivery of a message.

Two-Phase Commit

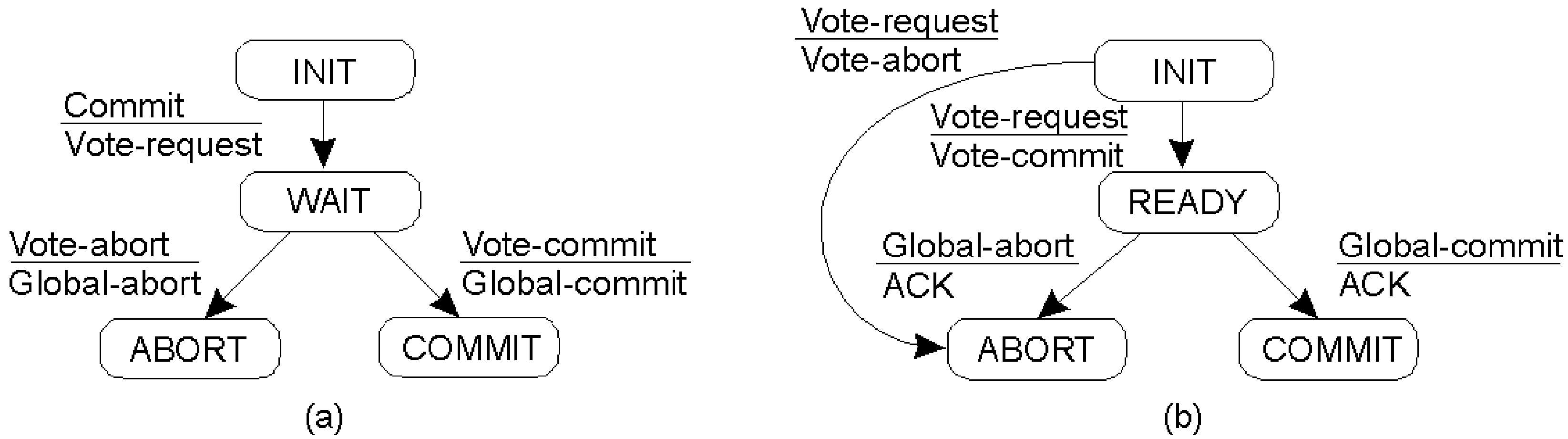
Phase I (Voting):

1. The coordinator sends a VOTE_REQUEST message to all participants.
2. When a participant receives a VOTE_REQUEST message, it returns either a VOTE_COMMIT message to the coordinator telling the coordinator that it is prepared to locally commit its part of the transaction, or otherwise a VOTE_ABORT message.

Phase II (Decision):

1. The coordinator collects all votes from the participants. If all have voted to commit, then so will the coordinator. In that case, it sends a GLOBAL_COMMIT message to all participants. However, if one participant had voted to abort the transaction, the coordinator will also decide to abort and multicast a GLOBAL_ABORT message.
2. Each participant that voted for a commit waits for the final reaction from the coordinator. If a participant receives a GLOBAL_COMMIT , it locally commits the transaction. Otherwise, when receiving a GLOBAL_ABORT, it locally aborts the transaction as well.

Two-Phase Commit



a) The finite state machine for the coordinator in 2PC.
b) The finite state machine for a participant.

Recovery

Two forms of error recovery:

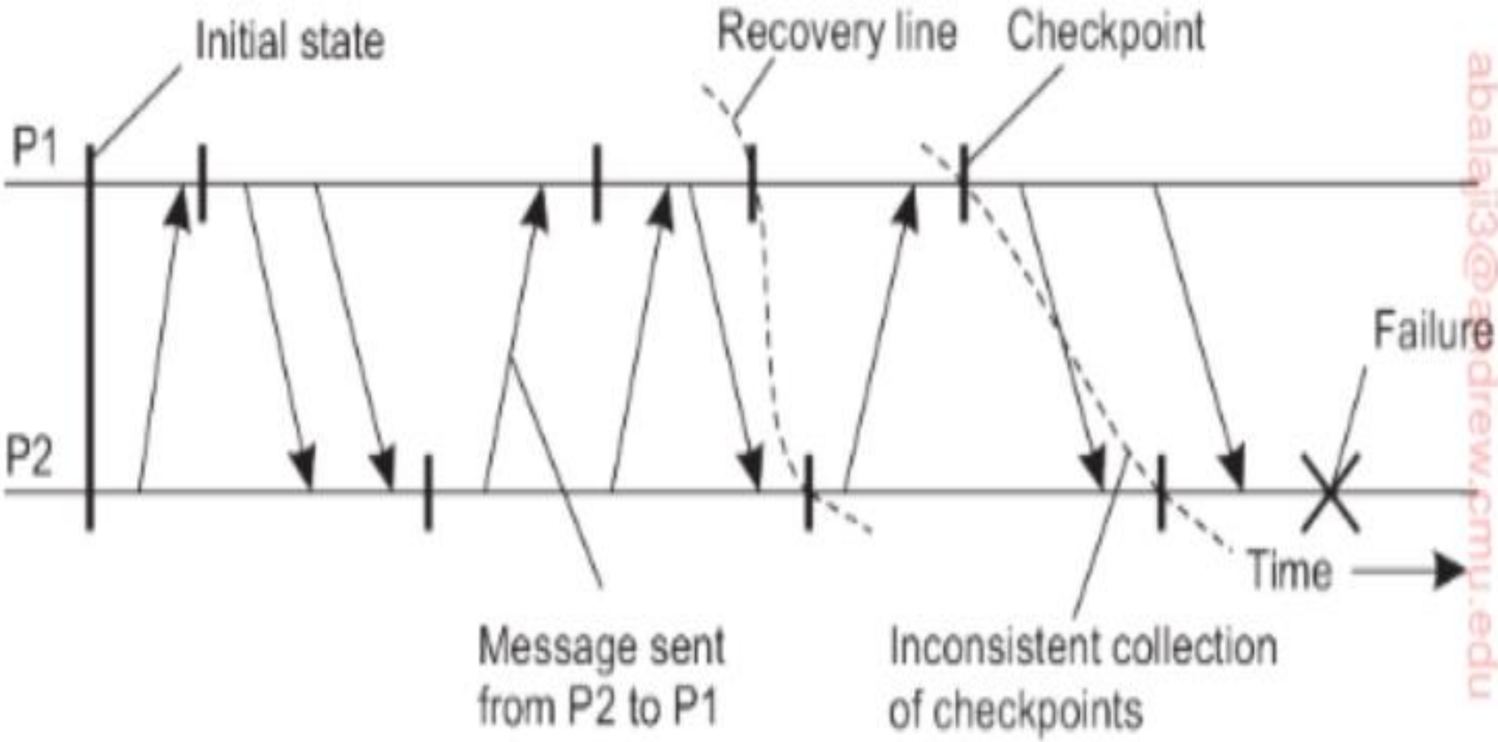
a) **Backward recovery:** bring the system from its present erroneous state back into a previously correct state. It is necessary to record the system's state from time to time (by **state checkpointing** and **message logging**). E.g., lost message retransmission.

b) **Forward recovery:** bring the system from its present erroneous state forward to a correct new state from which it can continue to execute. It has to know in advance which errors may occur. E.g., error correction by special encoding of messages.

Backward recovery is the most widely used error recovery technique due to its generality, but it also has the following drawbacks:

- Checkpointing is costly in terms of performance.
- Not all errors are reversible.

Checkpointing



A recovery line.

Message logging

- The basic idea underlying message logging is that if the transmission of messages can be *replayed*, we can still reach a globally consistent state but without having to restore that state from local storages.
- Instead, a checkpointed state is taken as a starting point, and all messages that have been sent since are simply retransmitted and handled accordingly.
- Considering that message logs are necessary to recover from a process crash so that a globally consistent state is restored, it becomes important to know precisely when messages are to be logged.