Fault Tolerance

Failure handling in Distributed Systems

- A characteristic feature of distributed systems that distinguishes them from single-machine systems is the notion of **partial failure**.

- One component may fail, while the rest of the systems keeps running

- A **partial failure** may happen when one component in a distributed system fails, while **some components are affected and some are unaffected**.

- An **important goal in distributed systems design** is to construct the system in such a way that it can automatically recover from partial failures without seriously affecting the overall performance.

Introduction to Fault tolerance

- **Fault tolerance** is the property that enables a system to continue operating properly in the failure.

- Being fault tolerant is strongly related to what are called **dependable systems**.

- Dependability is a term that covers a number of useful requirements for distributed systems including the following

1. Availability
2. Reliability
3. Safety
4. Maintainability

Availability

- Availability is defined as the property that a **system is ready to be used immediately**.

Reliability

- Reliability refers to the property that **a system can run continuously without failure.**
- If a system goes down for one millisecond every hour, it has an **availability of over 99.9999 percent, but is still highly unreliable.**
- Similarly, a system that never crashes but is shut down for two weeks every August is 100 percent **reliable ,but only 96 percent availability**.

Safety

- A measurement of how safe failures are
- System fails, nothing serious happens
- high degree of safety is required for systems controlling nuclear power plants.

maintainability

- maintainability refers to how easy a failed system can be repaired.
- A highly maintainable system may also show a high degree of availability.

- A system is said to ***fail*** when does not behave as promised.

  if a distributed system is designed to provide its users with a number of services, the system has failed when one or more of those services cannot be (completely) provided.

- An ***error*** is a part of a system state that might have caused a failure .

  when transmitting packets across a network, it is to be expected that some packets have been damaged when they arrive at the receiver. (reading a 1 instead of a 0 or may even be unable to detect that something has arrived.)

- The cause of an error is a ***fault***
  - finding out what caused an error is important.
  - For example, a wrong or bad transmission medium may easily cause packets to be damaged.

- A crashed program is clearly **failure**.

- Which may have happened because the program entered a branch of code containing a programming bug.(**error**)

- The cause of that bug is typically a programmer.(**fault**)

Programmer is the fault of the **error(programming bug),**in turn leading to **failure (a crashed program)**

$$\begin{tikzcd}\boxed{\text{Fault}} \arrow[r] & \boxed{\text{Error}} \arrow[r] & \boxed{\text{Failure}}\end{tikzcd}$$

Types of faults

1. Transient
2. Intermittent
3. Permanent

1. **Transient fault**

- Transient fault is a fault that happens once, and then doesn’t ever happen again.
- For example, a fault in the network might result in a request that is being sent from one node to another to time out or fail.
- However, if the same request is made between the two nodes again and succeeds, that fault has disappeared.

2. Intermittent fault

- Intermittent fault is one that occurs once, seems to go away, and then occurs again!
- Intermittent faults are some of the hardest ones to debug and deal with, since they masquerade as transient faults at first, but then come back — sometimes with inconsistency.
- example :loose connections in hardware, where sometimes it seems like the connection works, but occasionally the connection just stops working for a bit.

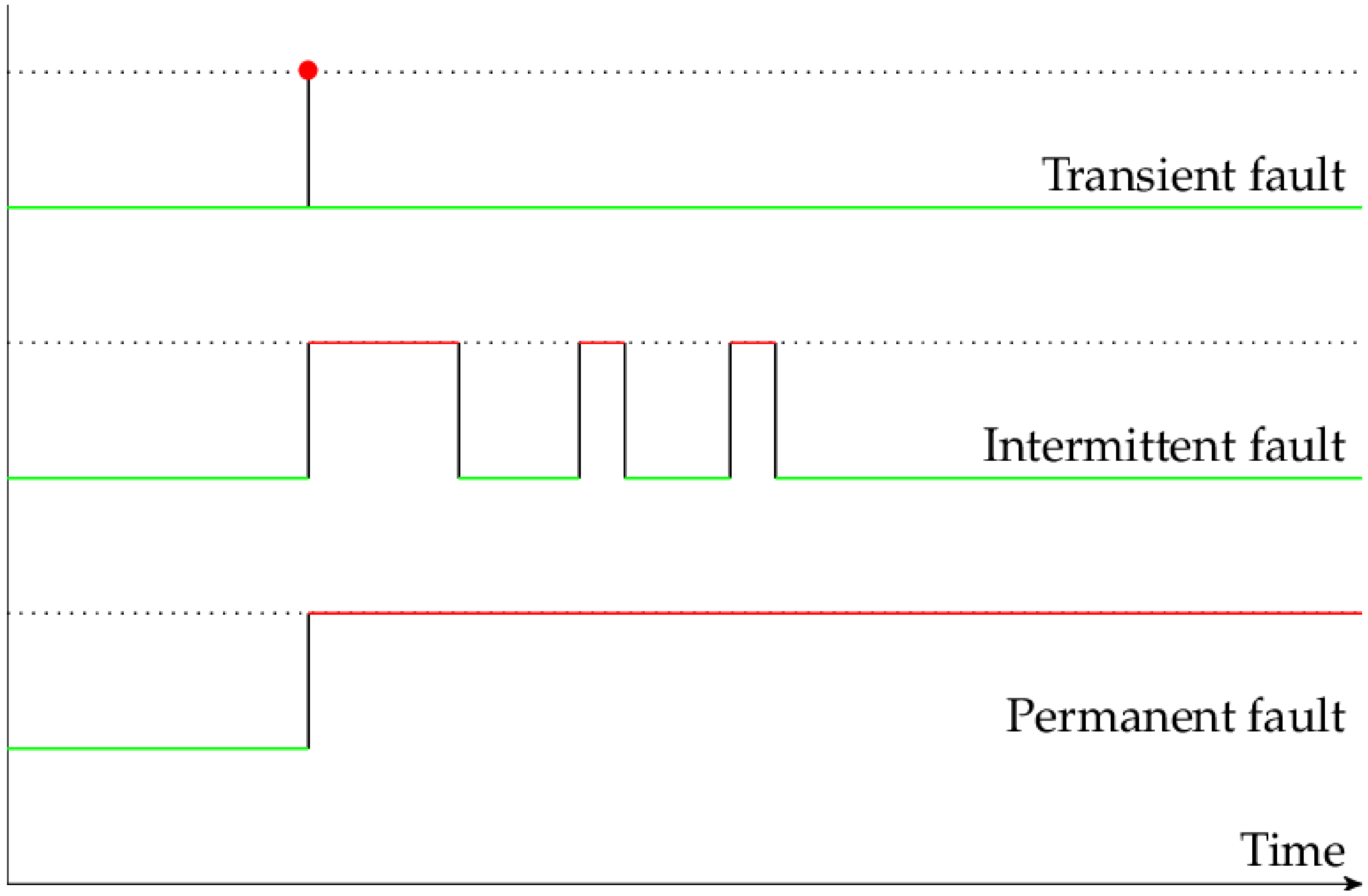
3. Permanent fault

- Permanent fault is one that continues to exist until the faulty component is replaced.

- A permanent fault occurs once, and then continues to persist until it has been addressed

- Burnt-out chips, software bugs, and disk head crashes are examples of permanent faults.

- For example, if part of a system runs out of memory, hits an infinite loop, or crashes unexpectedly, that “broken” state will just continue to be the same until someone (or some part of the system) fixes it or replaces it entirely.


Transient fault
Intermittent fault
Permanent fault
Time

Types of failures

1. Crash Failure
2. Omission failure
3. Timing failure
4. Response failure
5. Arbitrary failure

1. Crash Failure

- A server halts ,but it is working correctly until it halts.

2. Omission failure

- An omission failure occurs when a **server fails to respond to a request**.
- In **receive omission failure**,the server never got the request in the first place.
- The connection between a client and a server has been correctly established, but there was no thread listening to incoming requests.
- receive omission failure will generally **not affect the current state of the server**, as the server is unaware of any message sent to it.

- a **send omission failure** happens when the server has done its work, but somehow fails in sending a response.

- example, when a send buffer overflows while the server was not prepared for such a situation.

- In contrast to a receive omission failure, the server may now be in a state reflecting that it has just completed a service for the client.

- As a consequence, if the sending of its response fails, the server has to be prepared for the client to reissue its previous request.

3. Timing Failure

- Timing failures occur when the response lies outside a specified real-time interval.

4.Response failure

- A serious type of failure is a response failure, by which the **server's response is simply incorrect.**

- Two types of response failure

    1. Value failure:

        - a server simply provides the wrong reply to a request.

        - For example, a search engine that systematically returns Web pages not related to any of the search terms used. has failed.

2. state transition failure

- The server deviates from the correct flow of control.

5. Arbitrary failure

- The most serious are arbitrary failures, also known as Byzantine failures.
- In effect, when arbitrary failures occur, clients should be prepared for the worst.
- unpredictable behaviour of a node (complete arbitrary)
- Server may produce arbitrary responses at arbitrary times

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure	A server fails to respond to incoming requests
Receive omission	A server fails to receive incoming messages
Send omission	A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure	A server's response is incorrect
Value failure	The value of the response is wrong
State transition failure	The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times

Figure 8-1. Different types of failures.

Failure Masking by Redundancy

Idea

- Hiding failures from other processes
- The key technique for masking faults is **redundancy**

Three kinds of redundancy

1. Information redundancy
2. Time redundancy
3. Physical redundancy

1. **<u>Information redundancy</u>**

- Extra bits are added to allow recovery from garbled bits.
- For example, a Hamming code can be added to transmitted data to recover from noise on the transmission line.

2. **<u>Time redundancy</u>**

- An action is performed, and then. if need be, it is performed again.
- Transactions use this approach. If a transaction aborts, it can be redone with no harm.
- Time redundancy is especially helpful when the faults are transient or intermittent.

3. Physical redundancy

- **Extra equipment or processes are added** to make it possible for the system as a whole to tolerate the loss or malfunctioning of some components.

- Physical redundancy can thus be done either in hardware or in software.

- For example, extra processes can be added to the system so that if a small number of them crash, the system can still function correctly.

PROCESS RESILIENCE

- **Resilience**: is about preventing faults turning into failure

Problem

- how fault tolerance can actually be achieved in distributed systems?

Solution

- protection against process failures, which is achieved by **replicating processes into groups.**

Resilience by process groups

- Tolerating a faulty process is to organize **several identical processes into a group**.
- Consider collections of process as a single abstraction.
- when a message is sent to the group itself, all members of the group receive it.
- All members of the group receive the same message, if one process fails, the others can take over for it.

- Process groups may be dynamic.
- New groups can be created and old groups can be destroyed.
- A process can join a group or leave one during system operation.
- A process can be a member of several groups at the same time.
- Consequently, mechanisms are needed for managing groups and group membership.

Group organization

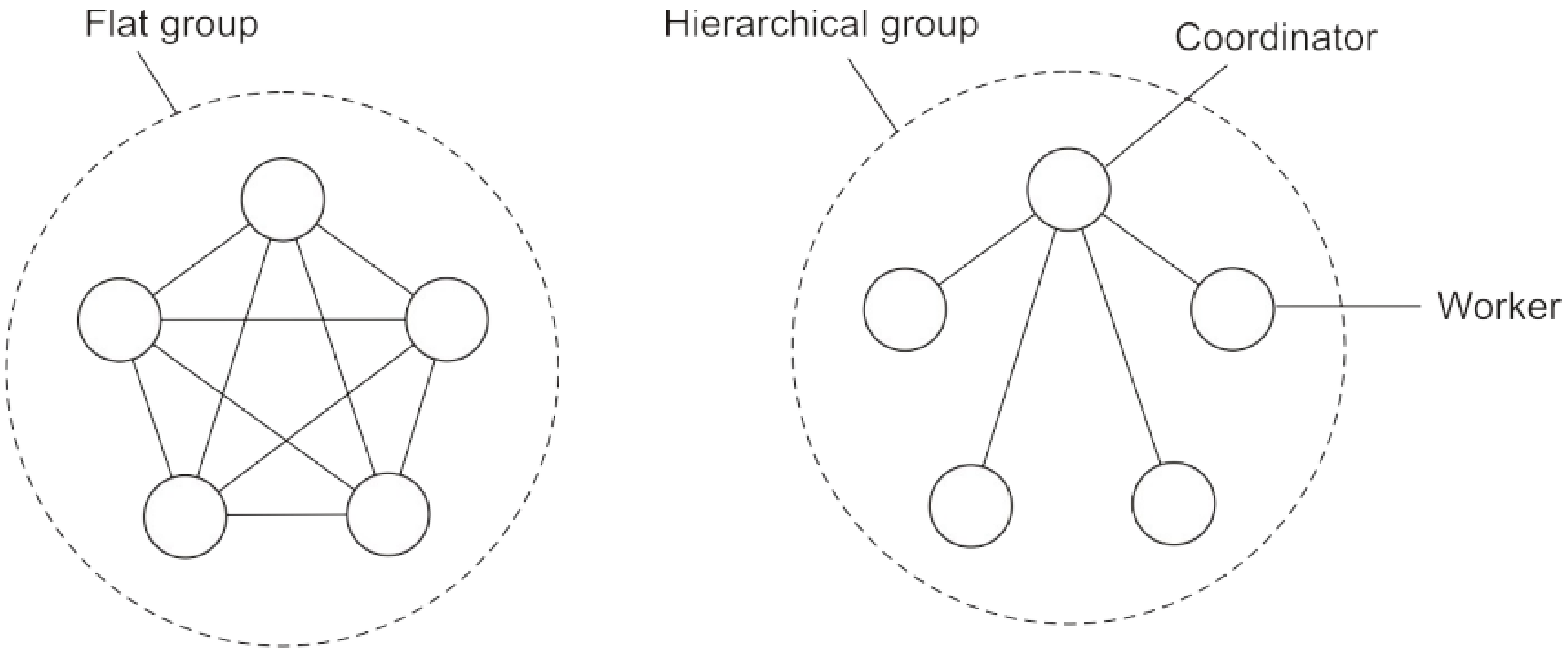
- Group internal structure is different
  1. Flat group
  2. Hierarchical group

- In **flat group**, all processes are equal and all decisions are made collectively.

- In **Hierarchical group** , one process is coordinator and all the others are workers

- In Hierarchical group, when a request for work is generated, either by an external client or by one of the workers, it is sent to the coordinator.

- The coordinator then decides which worker is best suited to carry it out, and forwards it there.


Flat group
Hierarchical group
Coordinator
Worker

<u>Flat group</u>

- The flat group is symmetrical and has no single point of failure.
- If one of the process crashes, the group 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

Hierarchical group

- It can make decisions without bothering everyone else.
- Loss of the coordinator brings the entire group to a grinding halt.
- When coordinator fails, its role will need to taken over, one of the workers is elected as coordinator.

Group Membership Management

- When group communication is present, some method is **needed for creating and deleting groups**, as well as for **allowing processes to join and leave groups**.
- One possible approach is to have a group server to which all these requests can be sent.
- The group server can then maintain a complete data base of all the groups and their exact membership
- major disadvantage: a single point of failure.
- If the group server crashes, group management ceases to exist.
- Probably most or all groups will have to be reconstructed from scratch, possibly terminating whatever work was going on.

Group Membership Management

- Another approach is is to manage group membership in a distributed way.
- For example, an outsider can send a message to all group members announcing its wish to join the group
- To leave a group, a member just sends a goodbye message to everyone.
- The trouble is, there is no polite announcement that a process crashes as there is when a process leaves voluntarily.
- The other members have to discover this experimentally by noticing that the crashed member no longer responds to anything.
- Once it is certain that the crashed member is really down (and not just slow), it can be removed from the group.

- leaving and joining have to be synchronous with data messages being sent.

- starting at the instant that a process has joined a group, it must receive all messages sent to that group.

- Similarly, as soon as a process has left a group, it must not receive any more messages from the group, and the other members must not receive any more messages from it.

Failure Masking and Replication

- Having a group of identical processes allows us to mask one or more faulty processes in that group.

- we can replicate processes and organize them into a group to replace a single (vulnerable) process with a (fault tolerant) group.

- There are two approaches to arranging the replication

    1. Primary based replication
    2. Replicated write protocols

Primary based replication

- hierarchical organization

- a group of processes is organized in a hierarchical fashion in which a primary coordinates all write operations.

- The primary is fixed, although its role can be taken over by one of the backups. if need be.

- when the primary crashes, execute election among backups to select a new primary

Replicated write protocols

- Flat process groups

- organizing a collection of identical processes into a flat group.

- The main advantage is that such groups have no single point of failure