LTAT.06.007 Distributed Systems
Lecture 11 – Fault tolerance I

Huber Flores, PhD
ASSOCIATE PROFESSOR

Tartu, Estonia 26/04/2021
Recap

- Learned the relation between consistency and replication
  - Levels of consistency
  - Replication models
- Replica management and quorum-based protocols
Roadmap (so far)

Logical order of events

Global clock

Internal clock in components
Agenda

Logical order of events

Global clock

Internal clock in components

Printer1

DNS

amazon.com

ut.ee
Agenda

• **Goal**: To understand how to handle failures in distributed systems

• **Content**:
  - Fault tolerance and dependability
  - Failure models
  - Process resilience
  - Consensus algorithms

**After this lecture, you should be able to:**

• Identify different types of failures in a system and ways to overcome them
## Introduction to fault tolerance

Distributed systems are suspect to partial failures

**Design goal:** recovery from partial failures without seriously affecting overall performance

### Note
- Being fault tolerant is strongly related to what are called **dependable systems**
- A key technique to handle failures is redundancy
Introduction to fault tolerance
Introduction to fault tolerance

Web application
Introduction to fault tolerance
A component provides services to clients. To provide services, the component may require the services from other components → a component may depend on some other component.

Specifically

A component $C$ depends on $C^*$ if the correctness of $C$’s behavior depends on the correctness of $C^*$’s behavior. (Components are processes or channels.)
Requirements of dependable systems

- **Availability**
  - Probability that the system is operating correctly at any given moment and is available to perform its functions on behalf of its users

- **Reliability**
  - Probability that the system can run continuously without a failure for an interval of time

- **Safety**
  - When a system temporarily fails to operate correctly, nothing catastrophic should happen

- **Maintainability**
  - Refers to how easily a failed system can be repaired
Reliability versus availability

Reliability $R(t)$ of component $C$
Conditional probability that $C$ has been functioning correctly during $[0, t)$ given $C$ was functioning correctly at time $T = 0$.

Traditional metrics
- **Mean Time To Failure** (MTTF): The average time until a component fails.
- **Mean Time To Repair** (MTTR): The average time needed to repair a component.
- **Mean Time Between Failures** (MTBF): Simply MTTF + MTTR.
Reliability versus availability

Availability $A(t)$ of component C
Average fraction of time that C has been up-and-running in interval $[0, t)$.

- Long-term availability $A$: $A(\infty)$

- Note: \[ A = \frac{MTTF}{MTBF} = \frac{MTTF}{MTTF + MTTR} \]

Observation

Reliability and availability make sense only if we have an accurate notion of what a failure actually is.
Terminology (fault tolerance)

Failure, error, fault

- **Failure**: A component is not living up to its specifications.
- **Error**: part of system’s state that may lead to failure
- **Fault**: The cause of an error
  - **Transient fault**: occurs once and then disappears
  - **Intermittent fault**: occurs, vanishes of its own accord, reappears,
    ...
  - **Permanent fault**: continues to exist until repaired

Fault tolerance

system can provide its services even in the presence of faults
### Failure models

#### Different types of failures

<table>
<thead>
<tr>
<th>Type of failure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crash failure</td>
<td>A server halts, but is working correctly until it halts</td>
</tr>
<tr>
<td>Omission failure</td>
<td></td>
</tr>
<tr>
<td>Receive omission</td>
<td>A server fails to respond to incoming requests</td>
</tr>
<tr>
<td>Send omission</td>
<td>A server fails to receive incoming messages</td>
</tr>
<tr>
<td>Timing failure</td>
<td>A server’s response lies outside the specified time interval</td>
</tr>
<tr>
<td>Response failure</td>
<td></td>
</tr>
<tr>
<td>Value failure</td>
<td>A server’s response is incorrect</td>
</tr>
<tr>
<td>State transition failure</td>
<td>The value of the response is wrong</td>
</tr>
<tr>
<td>Arbitrary failure</td>
<td>A server may produce arbitrary responses at arbitrary times</td>
</tr>
</tbody>
</table>

#### Observation

- **Fail-stop failures**: Server crash is announced by the server or properly detected by other processes
- **Fail-silent failures**: Server crash is only detected by other processes, possibly incorrectly
- **Fail-safe failures**: Server exhibits arbitrary failures in a benign way
Redundancy is key technique for masking faults

- **Information redundancy**
  - Extra bits are added to allow recovery from garbled bits
- **Time redundancy**
  - (Trans)action is redone for no harm if needed
- **Physical redundancy**
  - Extra hardware (e.g., server) or processes added to tolerate loss
## Process resilience

### Process replication

- Organizing several identical processes into a group allows masking a faulty process
  - When a message is sent to the group, all processes receive 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 may be a member of several groups at the same time

- Allows treating a collection of processes as a single abstraction

- Need mechanisms for managing groups and group memberships
## Process resilience

### Flat groups
- All processes are equal
- No single point of failure
- Complicated decision making

![Flat group](image)

### Hierarchical groups
- Coordinator delegates requests to workers
- Single point of failure
- Clear decision making

![Hierarchical group](image)
# Process resilience

## Group membership
- Method needed for creating and deleting groups
- Method needed for allowing processes to join and leave groups

## Group server
- Handles all requests, maintains database of groups
- Pros: Straightforward, efficient, easy to implement
- Cons: Single point of failure

## Distributed group membership management with multicasting (Challenges)
- Discovery of crashed group members
- Synchronizing leaving/joining with message delivery
- Rebuilding group in case of many crashes
Process resilience

Group (G1)

Group (G2)
Process resilience

Group (G1)

Group (G2)
Process resilience

Group (G1)  

Change of membership

Group (G2)
## Process resilience

### Group membership
- Method needed for creating and deleting groups
- Method needed for allowing processes to join and leave groups

### Group server
- Handles all requests, maintains database of groups
- Pros: Straightforward, efficient, easy to implement
- Cons: Single point of failure

### Distributed group membership management with multicasting (Challenges)
- Discovery of crashed group members
- Synchronizing leaving/joining with message delivery
- Rebuilding group in case of many crashes
Important assumptions

- All members are identical
- All members process commands in the same order

Result: We can now be sure that all processes do exactly the same thing
## Groups and failure masking

### Process replication via primary-backup or replicated-write protocol

- **Primary-backup (hierarchical group):** single point of failure
- **Replicated-write (flat group):** no single point of failure, cost of distributed coordination

### k-fault tolerant group

When a group can mask any k concurrent member failures (k is called **degree of fault tolerance**).

### How large does a k-fault tolerant group need to be?

- **With halting failures (crash/omission/timing failures):** we need a total of k +1 members as **no member will produce an incorrect result, so the result of one member is good enough.**
- **With arbitrary failures:** we need 2k +1 members so that the correct result can be obtained through a majority vote.
Consensus

Example on sequential consistency
In a fault-tolerant process group, each nonfaulty process executes the same commands, and in the same order, as every other nonfaulty process.

Reformulation
Nonfaulty group members need to reach consensus on which command to execute next.
Flooding-based consensus

System model
- A process group $P = \{P_1, \ldots, P_n\}$
- Fail-stop failure semantics, i.e., with reliable failure detection
- A client contacts a $P_i$ requesting it to execute a command
- Every $P_i$ maintains a list of proposed commands

Basic algorithm (based on rounds)
- In round $r$, $P_i$ multicasts its known set of commands $C_i^r$ to all others
- At the end of $r$, each $P_i$ merges all received commands into a new $C_i^{r+1}$
- Next command $cmd_i$ selected through a globally shared, deterministic function: $cmd_i \ select(C_i^{r+1})$. 
Flooding-based consensus

$P_1$

$P_2$

$P_3$

$P_4$
Flooding-based consensus
Flooding-based consensus
Flooding-based consensus
Flooding-based consensus

**Summary**

- $P_2$ received all proposed commands from all other processes → makes decision.
- $P_3$ may have detected that $P_1$ crashed, but does not know if $P_2$ received anything, i.e., $P_3$ cannot know if it has the same information as $P_2$) cannot make decision (same for $P_4$).
Realistic consensus

Implementations

- Paxos
  - it depicts the implementation of flood-based consensus.
- Raft
More on distributed consensus
More on distributed consensus
More on distributed consensus
More on distributed consensus

Client

3

3
More on distributed consensus
Raft
Raft

Follower

Candidate

Leader
Raft
Raft
Raft
Raft
Raft
Raft

set 3
Raft
Raft
Raft
Raft in action
Summary

What we learned?
• The importance of fault tolerance
• Failure models
• Resilience to failures
Next lecture(s)
Fault tolerance II
Questions?

E-mail: huber.flores@ut.ee