

Sections 7.1 & 7.2

Fault Tolerance & Process Resilience

John P. Daigle

Department of Computer Science
Georgia State University

05.16.06

Outline

- 1 Fault Tolerance
 - Basic Concepts
 - Failure Models
 - Failure Masking by Redundancy
- 2 Process Resilience
 - Design Issues
 - Failure Masking and Replication
 - Agreement in Faulty Systems
- 3 Conclusion

Outline

- 1 **Fault Tolerance**
 - Basic Concepts
 - Failure Models
 - Failure Masking by Redundancy
- 2 **Process Resilience**
 - Design Issues
 - Failure Masking and Replication
 - Agreement in Faulty Systems
- 3 **Conclusion**

Dependability

System Failure

When a process hangs or a portion of a machine physically breaks, operation of the system is compromised. Fault tolerance is keeping operations normal when something goes wrong. Distributed systems, compared to single machine systems, have more places where faults can take place, but also more potential for fault tolerance.

Dependable systems meet 4 criteria

Availability Is the system running right now?

Reliability How long can the system run without failure?

Safety What will happen when the system fails? Will we notice?

Maintainability How easy is it to fix the system?

Faults I

failure implies a fault

Fault → Error → Failure

Fault prevention: prevent the occurrence of a fault

Fault tolerance: build a component in such a way that it can meet its specifications in the presence of faults (i.e., **mask** the presence of faults)

Fault removal: reduce the presence, number, seriousness of faults

Fault forecasting: estimate the present number, future incidence, and the consequences of faults

Faults II

Types of Fault	
Transient Fault	Occur once and disappear No fix necessary
Intermittent Fault	Occur, disappear, reappear hard to diagnose
Permanent Fault	Most serious, always there Easiest to trace and fix

Failure Classification I

Crash Failure A server halts suddenly and completely. ex. Blue Screen of Death.

Omission Failure A server fails to respond to incoming requests.

Receive Omission The server fails to receive a message. This can be a problem with the connection or with the server software. The server state is usually not affected.

Send Omission The server fails to send a message. For example, if there is a buffer overflow. The server could now be in an incorrect state.

Failure Classification II

- Timing Failure** The server response lies outside of the acceptable interval. In streaming, can happen if the server provides information too soon *or* too late.
- Response Failure** The server may respond at the correct time, but it is the wrong response.
- Value Failure** The server provides the wrong reply.
- State Transition Failure** The server reacts unexpectedly to a request.
- Arbitrary or Byzantine Failure** Malicious or unexpected failures, for example, a server may provide incorrect output that appears to be correct.

Failure Behavior

I wonder if the server crashed?

It is not always clear why a system is not responding. It could be

- bad channel
- transient problem
- bad timing

How do we know?

fail-stop failures The server stops producing output, but does produce an error message for other systems.

fail-silent system The system simply halts, but produces no warning or error

fail-safe The system is producing arbitrary output, but it is clearly junk.

Types of Redundancy

Redundancy

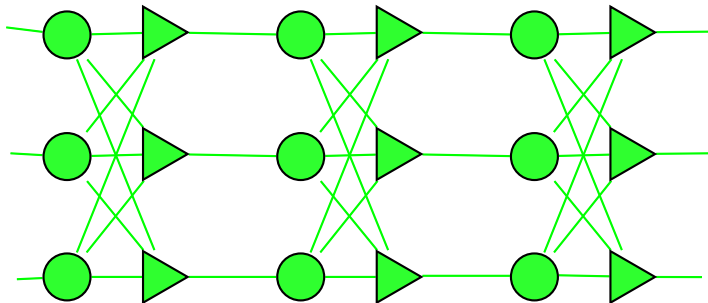
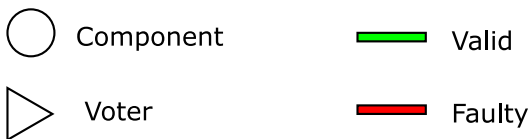
The key method for masking failure is **redundancy**, having duplicate processes or information available to cover mistakes.

Information Redundancy Extra bits are added to allow recovery from garbled bits

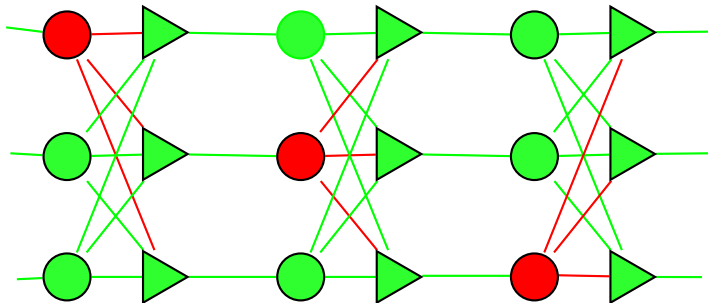
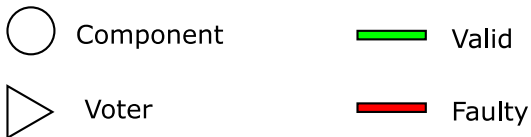
Time Redundancy An action is performed, but if need be can be performed again, as in the transactions model from chapter 5.

Physical Redundancy Extra equipment or processes are added to allow continuing function.

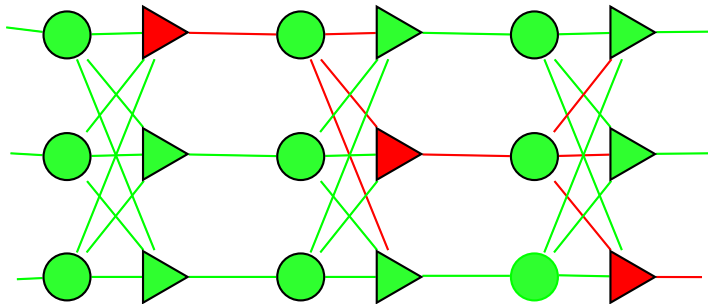
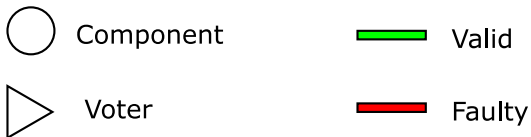
Triple Modular Redundancy



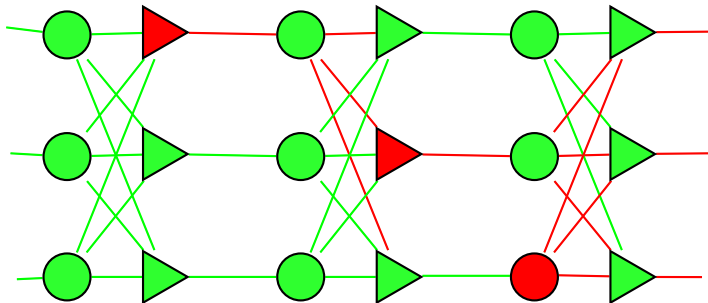
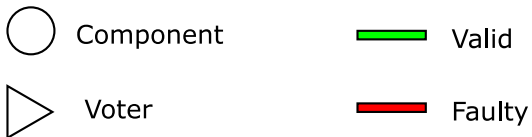
Triple Modular Redundancy



Triple Modular Redundancy



Triple Modular Redundancy



Fault Tolerance Review

Basic Concepts

- Four dimensions of **Dependability**, *Availability, Reliability, Safety, Maintainability*
- Four Types of **Faults**, *transient, intermittent, byzantine, permanent*

Failure Models, Failure Masking

- 5 **types** of **Failure**, *Crash, Omission, Timing, Response, Arbitrary*
- 3 **methods** of failure, *fail-stop, fail-silent, fail-safe*
- 3 ways to achieve **redundancy**, *information, time, physical*

Outline

- 1 Fault Tolerance
 - Basic Concepts
 - Failure Models
 - Failure Masking by Redundancy
- 2 Process Resilience
 - Design Issues
 - Failure Masking and Replication
 - Agreement in Faulty Systems
- 3 Conclusion

Overview

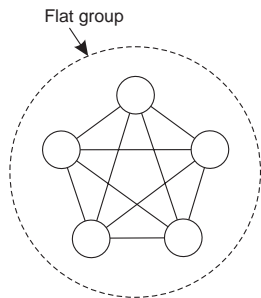
- Protection Against Failure through Groups
- Design Issues With Groups
- How to achieve Consensus

Design Issues

Key Properties of Groups

- Several identical processes
- A message to the group goes to every member
- Groups can be dynamic, members can join or leave
- Groups allow abstraction, the sender doesn't need info about the group
- Groups are non-exclusive, a process can belong to more than one group

Flat Groups vs. Hierarchal Groups I

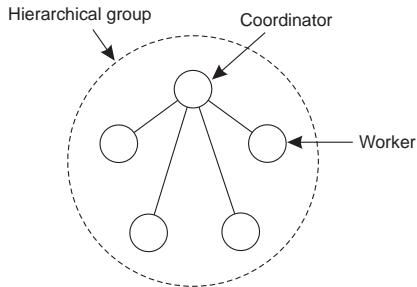


(a)

Flat Groups

Pro: Symmetry, Robustness

Con: Voting Delay



(b)

Hierarchal Groups

Pro: Quick Decisions

Con: Single Point of Failure

Group Membership

Implementation

Processes can enter and leave groups, groups can be created and destroyed. How do we keep track?

Group Server

- One server to track all membership and groups
- Easy to implement, simple
- Problem: Single point of failure

Member driven

- Members ask group to enter
- Members announce exit
- Problem: No notification on Crash
- Problem: Synchronization with Messages.
 - On entrance, needs all messages
 - On exit, must no longer send/receive as member

How Much Replication?

k fault tolerant

A system is *k fault tolerant* if it can have *k* faulty components and still meet its specifications.

- Silent Failure

$k + 1$ components will provide *k* coverage, because only one component needs to be correct.

- Byzantine Failure

$2k + 1$ components are needed, so that the $k + 1$ non-faulty components can outvote the *k* faulty components.

It is unlikely that *k* components will fail, but $k + 1$ will not fail. A large scale failure is not always predictable.

Introduction to Agreement

Finding Consensus

Many cases need agreement.

- Committing a transaction
- dividing tasks
- synchronization

Having the group find agreement protects against fault, but causes its own problems in communication.

Perfect Process, Bad Communication

Example (Dude, where's my car?)

I lent my car to my friend Sven the other day. Sven has the keys with him, but I'm not sure where he is. So, we begin text messaging back and forth: but we both know that the messages are unreliable. I text Sven I can meet him at his place around 7, but I don't want to go there without confirmation—I don't have my car. He confirms, but he doesn't know if I got his confirmation. Sven is a busy guy who works late, so he wants me to confirm I got the confirmation... and so on...

Will I ever get my car back?

This problem is classically stated as the **two-army problem**, and is insoluble. The agreed upon action will never take place, because the last sender will never be certain that the last confirmation went through.

Perfect Communication, Bad Processes

Example (Unreliable Henchmen)

Our hero is surrounded! The Evil Genius' Henchmen are prepared to attack. But if they attack one-at-a-time, they will lose. They all need to attack at once.

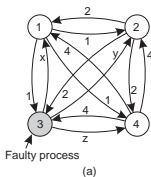
- All the henchmen can communicate simultaneously with all the other henchmen
- But if too few of them attack at once, the hero will win.
- Some of these henchmen are not reliable. They always attack one at a time.

How many reliable henchmen does the Evil Genius need?

Perfect Communication, Bad Processes I

This problem is classically stated as the **Byzantine Generals Problem**

- n Generals exchange troop strength
- m Generals are traitors, and lie
- If there are enough troops, loyal Generals attack the enemy



1 Got(1, 2, x, 4)
 2 Got(1, 2, y, 4)
 3 Got(1, 2, 3, 4)
 4 Got(1, 2, z, 4)

(b)

1 Got	2 Got	4 Got
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

(c)

- 1 Exchange with all generals
- 2 Place results in a vector
- 3 Exchange vectors

Perfect Communication, Bad Processes II

If at the end of step three, there is a majority opinion, this is selected. In the example, it is (1,2,UNKNOWN,4). As it turns out, if there are m faulty components, than to be m tolerant, the system must have $3m + 1$ processes.

Wrap Up

Group Design

- Groups are dynamic, homogeneous, redundant
- Flat vs. Hierarchical Groups
- How membership is handled

Failure Masking, Agreement in Faulty Systems

- k fault tolerance
- two-general problem
- byzantine generals problem

Outline

- 1 Fault Tolerance
 - Basic Concepts
 - Failure Models
 - Failure Masking by Redundancy
- 2 Process Resilience
 - Design Issues
 - Failure Masking and Replication
 - Agreement in Faulty Systems
- 3 Conclusion

Final Thoughts

- Distributed systems have many points of failure
- Redundancy is the key to creating reliable systems
- Communication is key for redundancy