IAF0030
Arvutitehnika erikursus I

Advanced Issues of Fault Tolerance
FT in Distributed Systems

Tallinn University of Technology

Gert Jervan

Arvutitehnika instituut (ATI)
Tallinna Tehnikaülikool

Lecture Outline

Some slides:
Chris Clifton
Andrew S. Tanenbaum
Maarten van Steen

- **Introduction**
- **Fault Tolerance in Distributed Systems**
- **Case Study presentatsiooni näide**

Distributed Systems

- Why:
  - Sharing of resources
  - Fault Tolerance
  - Cooperative work
  - Parallelism
- Problems:
  - Software complexity
  - Network saturation
  - Failures
  - Inconsistencies

Fault Tolerance in Distributed Systems

- Perfect world: No Failures
  - We don't live in a perfect world
- Non-distributed system
  - Crash, you're dead
- Distributed system: Redundancy
  - Should result in less down time
  - But does it?
- Distributed systems according to Butler Lampson (Microsoft & MIT)
  - A distributed system is a system in which I can't get my work done because a computer that I've never heard of has failed.

Problems with Distributed Systems

- Failures more frequent
  - Many places to fail
  - More complex
    - More pieces
    - New challenges: Asynchrony, communication
- Potential problems of failure greater
  - Single system – everything stops
  - Distributed – some parts continue

Problems with Distributed Systems

- Ensuring globally consistent state of the system
- Uncertainty - cannot distinguish whether a node or the communication sub-system that has failed
- Synchronization of normal execution and re-synchronization after a failure should be accomplished in spite of uncertainties

Physical Model

- Each node is an autonomous processor
  - Private volatile memory
  - Private clock
- Network interface that connects to the communication network
- Non-volatile storage
- Network topologies

Logical Model

- Processes
  - Cooperating
  - Finite progress
- Fully connected logical network
  - Channels represent logical connection between processes
  - Channels have infinite buffer and error-free
  - Deliver the messages in the order they are sent
  - No total ordering, only partial ordering of messages

First Step: Goals

- Availability
  - Can I use it now?
- Reliability
  - Will it be up as long as I need it?
- Safety
  - If it fails, what are the consequences?
- Maintainability
  - How easy is it to fix if it breaks?

Failure Model (Flaviu Cristian, 1991)

- Dependency
  - Proper operation of Database depends on proper operation of processor, disk
- Failure Classification
  - Type of response to failure
- Failure semantics
  - State of system after given class of failure
- Failure masking
  - High-level operation succeeds even if they depend on failed services

Failure Models

- Different types of failures.

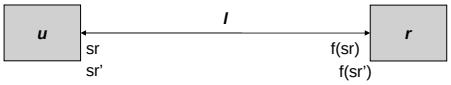
Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure *Receive omission* *Send omission*	A server fails to respond to incoming requests A server fails to receive incoming messages A server fails to send messages
Timing failure	A server's response 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

Crash Failure types

- Based on recovery behavior:
  - Amnesia
    - Server recovers to predefined state independent of operations before crash
  - Partial amnesia
    - Some part of state is as before crash, rest to predefined state
  - Pause
    - Recovers to state before omission failure
  - Halting
    - Never restarts

Failure Semantics

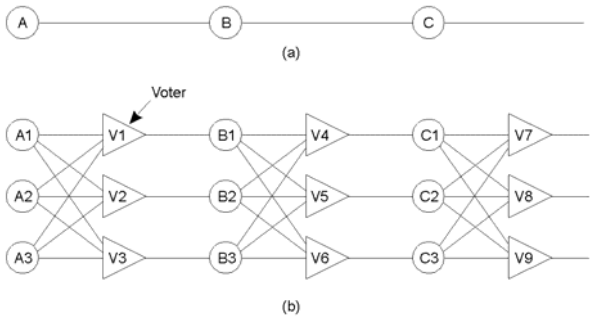
- Max delay on link: d; Max service time: p
  - Should get response in 2d+p
- Assume omission failure only
  - If no response in 2d+p, resend request
- What if performance failure possible?
  - Must distinguish between response to sr and sr'



Failure Semantics

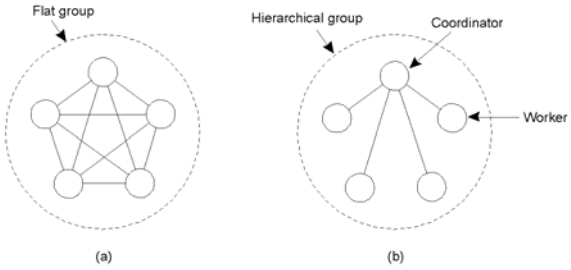
- Specification for service must include
  - Failure-free (normal) semantics
  - Failure semantics (likely failure behaviors)
- Multiple semantics
  - Combine to give (weaker) semantics
  - Arbitrary failure semantics: Weakest possible
- Choice of failure semantics
  - Is class of failure likely?
    - Probability of type of failure
  - What is the cost of failure
    - Catastrophic?

Failure Masking by Redundancy



Flat Groups versus Hierarchical Groups

- Communication in a flat group.
- Communication in a simple hierarchical group



Group Failure Masking

- Redundant servers
  - Failed server masked by others in group
  - Allows failure semantics of group to be higher than individuals
- k-fault tolerant
  - Group can mask k concurrent group member failures from client
- May "upgrade" failure semantics
  - Example: Group detects non-responsive server, other member picks up the slack
  - Omission failure becomes performance failure

Hierarchical Failure Masking

- Hierarchical failure masking
  - Dependency: Higher level gets (at best) failure semantics of lower level
  - Can compensate for lower level failure to improve this
- Example: TCP fixes communication errors, so some failure semantics not propagated to higher level

Additional Issues

- Tradeoff of failure probabilities / semantics
  - Which is better?
    - P[Catastophic] = 0.01 & P[Performance] = 0.5
    - P[Catastrophic] = 0.1 & P[Performance] = 0.01
- Techniques for managing failure
- Recovery mechanisms

What is Reliable?

- Guaranteed message delivery
  - When?
- Guaranteed delivery order
  - Lost?
- Guaranteed delivery time
  - Just drop late messages?
- All of the above?
  - Is this enough?
  - Is this possible?

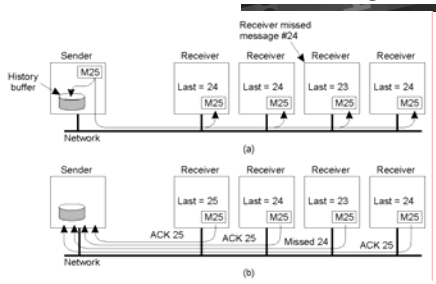
Doesn't TCP provide reliability?

- TCP: Guarantees
  - Message delivery
  - Message order
- How does it work?
  - Sequence number
  - Request resend for missing
- What are the limitations?

How do we get reliable communications?

- Guaranteed message delivery
  - Acknowledgement
- Guaranteed delivery order
  - Sequence number
- Guaranteed delivery time
  - QoS research
- Corruption / interception
  - Cryptographic techniques

Basic Reliable-Multicasting Schemes



- A simple solution to reliable multicasting when all receivers are known and are assumed not to fail
- Message transmission
- Reporting feedback

Limits of Reliable Communication

- Guaranteed message delivery
  - Message lost: Acknowledge, resend if no acknowledgement
  - Failed link leads to known loss
    - Delivery of last message unknown
  - One-way link failure: Delivered, but not known to sender
  - Transient partition: No guarantees (Byzantine Generals)
- Guaranteed delivery order
  - Okay for point to point
  - What about order among multiple senders/receivers?
    - Lamport clocks

End-to-End Argument

- Can't trust reliability of underlying mechanisms
  - Don't handle the right failure classes
  - Failure between mechanism and application
- Thus applications need to implement reliability
  - Are underlying reliability mechanisms needed at all?

What about Multicast?

- Guaranteed message delivery
  - Either all or none
- Guaranteed delivery order
  - Multicasts from different sources ordered same at all recipients

Classes of Reliable Multicast

- Sender-initiated: Acknowledge all packets
  - Sender resends if ACK not received
- Receiver-initiated: Request missing packets
  - Receiver sends NAK if packet missing
- Problem: Scalability

Sender-Initiated

- Acknowledgement required from each receiver
  - Scaling problems
- Sender resends if acknowledgement not received in time
  - Wall-clock time
  - Number of packets
- Old packets must be kept until acknowledged by everyone

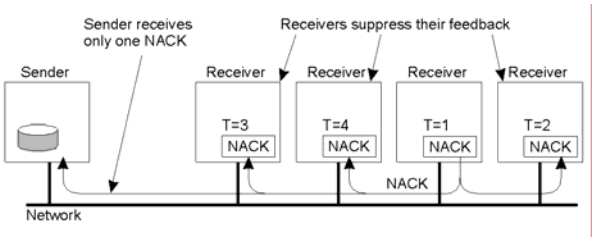
Receiver-initiated

- Receiver detects failure and requests resend
  - Error from lower level
  - Skip in sequence numbers
  - Timeout
- Scales well under normal operation
  - Floods sender on failure
- How long must sender keep old packets?

Receiver-initiated with NAK avoidance

- Receiver-initiated floods sender if general failure
- Solution: Multicast NAK
  - Wait random time first
  - Don't NAK if somebody else does
  - Sender multicasts retransmit

Nonhierarchical Feedback Control



- Several receivers have scheduled a request for retransmission, but the first retransmission request leads to the suppression of others.

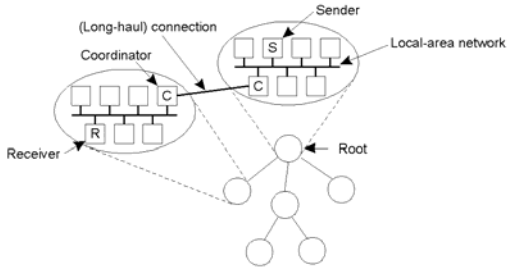
Advantages of receiver-initiated protocols

- Scalability in normal operation
- Receivers pace source
  - Retransmit takes priority, slows sending
- Sender doesn't even need to know multicast group members
  - Existing solutions to unbounded memory problem do require this knowledge

Tree-based Protocols

- Organize multicast group into tree
  - Children acknowledge to parent
  - Parent acknowledges when all children have acknowledged
- Advantages
  - Sender doesn't need to know full group
  - Solves unbounded memory
  - Scalable
- Disadvantages
  - Rate paced by slowest acknowledgement path in tree

Hierarchical Feedback Control



- The essence of hierarchical reliable multicasting.
- Each local coordinator forwards the message to its children.
- A local coordinator handles retransmission requests.

Ring-based protocols

- Idea: Token site responsible for retransmit
  - Sender multicasts
  - Token site multicasts ACK
  - Receivers request retransmit from token site if ACK doesn't match what they have
- Can only accept token if you've received everything acknowledged
  - Keep packets since last time you had token
- Advantages:
  - Space
  - Low load on sender

Questions?

Tallinn University of Technology

Gert Jervan

Department Of Computer Engineering
Tallinn University of Technology
Estonia