Goal of today’s lecture
– Some summary/review
– Some practical considerations/implications of what we’ve already learned
Why not “stand-alone” machines?

• Why not use “stand-alone” redundant machines, rather than distribute fault tolerance schemes?
  – E.g., Stratus, Tandem, …
  – Some disadvantages…

• Often require lots of special-purpose hardware
  – Depends on your failure assumptions and configuration

• If disaster recovery later required, have to “add-on” distribution
  – With distributed fault tolerance, geographic separation is simply another configuration parameter

• Stand-alone machines often use tight synchronization
  – Reduced tolerance to transient faults that can affect all redundant channels

• Tightly-coupled redundancy does not provide as much tolerance of SW design faults
  – Heisenbugs
  – OS crashes
Implications of Fail-Silent Failure Model

• Any message sent by a fail-silent host is correct: time + value
• Fail-silent deals with external perceptions of node activity
• Fail-silent simplifies things in many ways
  – All messages correct, so no “two-faced” behavior
    ➢ Distributed consensus problems are much simpler
  – Node sends correct message within time bounds, or does not send
    ➢ Other nodes can detect failure with simple interrogation and timeout, or absence of a “heartbeat” message
  – Data and/or code replication techniques to tolerate $t$ faulty nodes only need rely on $t+1$ replicas
  – No garbage messages may be produced
    ➢ Faults in communication network can be dealt with independently of faults in nodes [Fig 2]
• Downside: “coverage” of fail-silent assumption may not be very high
Implications of Fail-Uncontrolled Model

- Fail-uncontrolled may behave quite arbitrarily:
  - Send messages late or never
  - Send sooner than expected
  - Send with incorrect content
  - Send unexpected “impromptu” messages
- Good news: “assumption coverage” is 1
- Bad news:
  - Very malicious behavior, including “two-faced” (relaying different values to different replicas in same round)
    - Need $3t+1$ nodes to tolerate $t$ faulty nodes if they can be “two-faced”
  - Can’t use timeout and interrogation to detect if it is faulty
    - It can still reply correctly but send incorrect messages to others!
    - Can only detect by comparing activity of different nodes
  - Even if no “two-faced” behavior, can be wrong time and wrong content
    - Need $2t+1$ replicas to tolerate $t$ faulty nodes
  - Faulty node may generate infinite # of impromptu messages
    - Saturate channel(s) other nodes need to use
    - May also lie about address fields in messages
    - Need to rely on interconnection topology (or signed messages) to identify sender
    - Need to rely on interconnection topology to eliminate possibility of saturation
  - Nodes must not all share communication channels [Fig 3]
Network Attachment Controllers (NACs)

- Previous topology was not
  - Affordable
  - Flexible
  - Extensible
- Split node into two parts [Fig 4]
  - COTS component (HW+SW), may be fail-uncontrolled
  - NAC, assumed to be fail-silent (enforced by self-checking HW)
- Goal: allow use of
  - Simplified interconnect topologies
  - Less elaborate consensus protocols
  yet still tolerate arbitrary behavior of COTS components
NACs (cont.)

• Break up host/NAC interface, so each can observe other’s behavior
  – NH: host-to-NAC link, with messages ("service items")
    • Application messages (requests or replies)
    • Handshakes (requests or ACKs) for service items sent over link
  – NH: NAC-to-host link, with service items
    • Application messages received from the communication network
    • Handshakes (requests or ACKs) for service items sent over the HN link

• Key observation: instead of defining fail-silent or fail-uncontrolled behavior of hosts in terms of network messages, define it in terms of service items on the HN link!
• These are locally observable by the (fail-silent) NAC!
NACs (cont.)

- Faulty fail-uncontrolled host may behave in different ways
  a) Send messages or handshakes that are late (or omitted)
  b) Send messages or handshakes too early
  c) Send messages or handshakes with erroneous content
  d) Send “impromptu” messages or handshakes, ones unspecified by protocol

- Handshakes with a) are important for multicasting, because two possible causes
  - Receiving host entity could block for a logical reason
    - NAC must request “network” (all nodes) to stop sending
  - Host could have failed
    - Sending nodes should be able to continue (don’t need handshake from failed node)

- Cases b) and c) and d) suggest, to provided fail-silent assumption, NAC must shelter it from unexpected or wrong service items

- Solution: NAC plays master role in HN interface
  - Host may only transfer info at times and locations (in memory) NAC dictates

- Leads to architecture of [Fig5]
  - Network partitions not tolerated…. (why not….philosophically/practically)
Replicated Software Components

- Observation: distribution of a computation without replication leads to a decrease in dependability
  - Why?
  - A huge motivator for replication, given how widely computations are being distributed in the last few years!

- A replica is deterministic if, in the absence of faults, any execution of the replica
  - Starting from same initial state
  - Consuming same ordered set of input messages
  produces the same ordered set of output messages

- A replica group is deterministic if, in the absence of faults, given
  - Same initial state for each replica
  - Same set of input messages
  each replica produces same ordered set of output messages

- Note: if replicas consume input messages in same order, replica determinism guarantees replica group determinism
General Sources of Nondeterminism

• General Sources of nondeterminism (explain each):
  – Shared memory or semaphore synchronization
  – Dependency on order of reception of messages
  – Time in execution stream when data/message arrives
  – System clock
  – Random numbers
  – Thread scheduling order
  – Amount of data in the pipe/buffer
  – Preemption
  – Non-deterministic language constructs
  – Decision based on site-specific info
  – Local timers
  – …
  – (others??)
Replication Domains

• An object’s replication domain is the set of nodes on which its replicas may reside

• How to choose or restrict?
  – Possess the necessary resources to execute the replica (Q: examples?)
    • Type
    • Quantity
  – Guarantee replica determinism (if required by replica)
  – Failure mode reasonably assumed of node
  – Equivalence of execution speed
    • Wide range causes problems in detecting timing problems
    • Group would proceed at the pace of the slowest
  – Note: above are mainly static measures

• Other dynamic factors for replica management (mapping the required #replicas for the object to nodes)
  – Load on a node
  – Load on communication links connecting a node
  – Other topological/geographic considerations (Q: examples?)
Notes on Active Replication

• The output consistency condition: all replicas produce same output messages in same order (over same output ports)

• Sufficient conditions for output consistency
  – Input consistency: set of input messages delivered to correct replicas are identical
  – Replica group determinism: when
    • starting from same initial states and
    • processing identical sets of input messages
  each replicas produces
    • identical output messages
    • in same order
Fail-Uncontrolled Active Replicas

• Kinds of errors to worry about in error processing
  – Omission errors
  – Late-timing errors
  – Value errors
  – Early-timing errors
  – Impromptu errors

• Masking $t$ early-timing errors takes $2t+1$ replicas
  – If $t$ messages arrive, cannot distinguish between
    • First $t$ messages of a set of $2t+1$
    • Messages sent too early
    • Impromptu messages
  – Solution: rep_entity ensures $t+1$ messages have arrived within a specified time interval

• Masking value errors
  – **Message validation**: checking data sent by all replicas
  – To mask $t$ value errors in Delta-4, must ensure have $t+1$ equal values (Q: why?)(Q: why not $3t+1$?)
Passive Replication Checkpointing Strategies

• Goal: prevent domino effect

• Example 1
  – A checkpoints
  – A sends msg$_1$ to B
  – A crashes
  – A restarted from checkpoint
  – A resends msg$_1$
  – B’s state can be incorrect (Q: how?)

• Example 2
  – A checkpoints
  – A receives msg$_2$ from B
  – A crashes
  – A restarted from checkpoint
  – A now needs to receive msg$_2$ from B (oops…)

• In either case, B would have to be rolled back to checkpoint to guarantee correctness
Passive Repl. Checkpointing Strategies (cont.)

• Checkpointing thus must be performed so that a restarted (or substituted) replica re-executing from last checkpoint never
  – Requests re-sending of previously received input messages
  – Sends duplicate output messages

• Avoiding resending of previously received input messages:
  – Backup replicas must maintain queue of input messages same as received by primary since last checkpoint, or
  – Each time primary receives a message, a checkpoint is taken

• Avoiding sending duplicate output messages:
  – Systematic checkpointing: each time the primary sends a message
  – Periodic checkpointing: checkpoint every \( n \) message sends.
    • During recovery: any output messages generated by substitute are checked against log of ones previously sent (if there, don’t send)
Passive Repl. Checkpointing Strategies (cont.)

• Correct recovery in periodic checkpointing requires
  – Replicas be deterministic (Q: no big deal?)
  – Messages be received in same order

• Correct recovery in systematic checkpointing required atomic delivery of
  – Checkpoints to backup replicas
  – Data messages to destination(s)

This can be done with a compound atomic multicast protocol:
Passive Replication Performance Issues

- Passive replication has a few *potential* advantages over active
  - Lower computational load: no redundant computations
  - Primary can have message delivered to it once arrives off of network
- But two different time overheads
  - *Permanent* communication and processing overhead to send checkpoints to backup replicas
  - *Temporary* processing overhead due to rollback and re-execution from a checkpoint, when a fault occurs
- Large internal states can be a killer. Some ways to mitigate:
  - Decrease frequency of checkpointing
  - Compress “unused” parts of state
  - Only send states from processes which have run since checkpoint
  - Only send memory pages modified (with OS hooks)
  - Q: any costs of above?
- Bottom line: passive replication will usually perform worse than active