Acknowledgement

- The slides for this lecture are based on ideas and materials from the following sources:
  - ID2203 Distributed Systems Advanced Course by Prof. Seif Haridi from KTH - Royal Institute of Technology (Sweden)
  - CS5410/514: Fault-tolerant Distributed Computer Systems Course by Prof. Ken Birman from Cornell University
  - Course from F. Bongiovanni
  - A few slides from SARDAR MUHAMMAD SULAMAN

System models

- synchronous distributed system
  - each message is received within bounded time
  - each step in a process takes $1b \lt time \lt ub$
  - each local clock’s drift has a known bound

- asynchronous distributed system
  - no bounds on process execution
  - no bounds on message transmission delays
  - arbitrary clock drifts

the Internet is an asynchronous distributed system
Failure model

* First we must decide what do we mean by failure?
  * Different types of failures
    * Crash-stop (fail-stop)
      * A process halts and does not execute any further operations
    * Crash-recovery
      * A process halts, but then recovers (reboots) after a while
* Crash-stop failures can be detected in synchronous systems
* Next: detecting crash-stop failures in asynchronous systems

What's a Failure Detector?

1. Ping-ack protocol

   Needs to know about $P_j$’s failure

   ![Ping-ack protocol diagram]

   If $P_i$ fails, within $T$ time units, $P_i$ will send it a ping message, and will time out within another $T$ time units. Detection time = $2T$

   - $P_i$ queries $P_j$ once every $T$ time units
   - if $P_i$ does not respond within $T$ time units, $P_i$ marks $P_j$ as failed

2. Heart-beating protocol

   Needs to know about $P_j$’s failure

   ![Heart-beating protocol diagram]

   - if $P_i$ has not received a new heartbeat for the past $T'$ time units, $P_i$ declares $P_j$ as failed
   - $P_i$ maintains a sequence number
   - $P_i$ sends $P_j$ a heartbeat with incremented seq. number after $T' (= T)$ time units
Failure Detectors

- Basic properties
  - Completeness
    - Every crashed process is suspected
  - Accuracy
    - No correct process is suspected

Both properties come in two flavours: Strong and Weak

- Strong Completeness
  - Every crashed process is eventually suspected by every correct process
- Weak Completeness
  - Every crashed process is eventually suspected by at least one correct process

- Strong Accuracy
  - No correct process is ever suspected
- Weak Accuracy
  - There is at least one correct process that is never suspected

Perfect failure detector P

- Assume synchronous system
  - Max transmission delay between 0 and $\delta$ time units
  - Every $\gamma$ time units, each node:
    - Sends <heartbeat> to all nodes
  - Each node waits $\gamma+\delta$ time units
    - If did not get <heartbeat> from pi
      Detect <crash | pi>

Correctness of P

- PFD1 (strong completeness)
  - A crashed node doesn’t send <heartbeat>
    - Eventually every node will notice the absence of <heartbeat>

- PFD2 (strong accuracy)
  - Assuming local computation is negligible
  - Maximum time between 2 heartbeats
    - $\gamma+\delta$ time units
  - If alive, all nodes will recv hb in time
    - No inaccuracy

An algorithm for P

Upon event (HBTimeout)
  For all pi in $P$
    Send HeartBeat to pi
    startTimer ($\gamma$, HBTimeout)

Upon event Receive HeartBeat from pj
  alive := alive $\cup$ pj

Upon event (DetectTimeout)
  crashed := $P \setminus$ alive
  for all pi in crashed Trigger (crashed, pi)
  alive := $\emptyset$
  startTimer ($\delta+\gamma$, DetectTimeout)
Eventually perfect failure detector \(<>P\)

- For asynchronous system
  - We suppose there is an unknown maximal transmission delay -- *partially synchronous system*
- Every \(\gamma\) time units, each node:
  - Sends \(<\text{heartbeat}>\) to all nodes
- Each node waits \(T\) time units
  - If did not get \(<\text{heartbeat}>\) from \(pi\)
    - Indicate \(<\text{suspect} | \pi>\) if \(pi\) is not in suspected
    - Put \(pi\) in suspected set
  - If get \(<\text{heartbeat}>\) from \(pi\) and \(pi\) is suspected
    - Indicate \(<\text{restore} | \pi>\)
    - remove \(pi\) from suspected
    - Increase timeout \(T\)

Correctness of \(<>P\)

- **PFD1 (strong completeness)**
  - Idem
- **PFD2 (eventual strong accuracy)**
  - Each time \(p\) is inaccurately suspected by a correct \(q\)
    - Timeout \(T\) is increased at \(q\)
    - Eventually system becomes synchronous, and \(T\) becomes larger than the unknown bound \(\delta\) \((T > \gamma + \delta)\)
  - \(q\) will receive HB on time, and never suspect \(p\) again

Homework

Eventually Perfect Failure Detector: an alternative algorithm (questions next slide)
Exercise: is this a good algorithm?

What is the delay between two heartbeats? At the beginning? At any point in time? Can you find a formula for this depending on the number of failures suspected/recovered.

Is there a maximal time before a failure is detected? (supposing there is a bound Delta on maximal communication time)

Consensus (agreement)

- In the consensus problem, the processes propose values and have to agree on one among these values

- Solving consensus is key to solving many problems in distributed computing (e.g., total order broadcast, atomic commit, terminating reliable broadcast)

Consensus - basic properties

- Termination
  - Every correct node eventually decides

- Agreement
  - No two correct processes decide differently

- Validity
  - Any value decided is a value proposed

- Integrity:
  - A node decides at most once

- A variant: UNIFORM CONSENSUS
  - Uniform agreement: No two processes decide differently

Consensus algorithm I

- A P-based (fail-stop) consensus algorithm

- The processes exchange and update proposals in rounds and decide on the value of the non-suspected process with the smallest id [Gue95]

Consensus algorithm II

- A P-based (i.e., fail-stop) uniform consensus algorithm

- The processes exchange and update proposal in rounds, and after n rounds decide on the current proposal value [Lyn96]
Consensus algorithm I

- The processes go through rounds incrementally (1 to n): in each round, the process with the id corresponding to that round is the leader of the round.
- The leader of a round decides its current proposal and broadcasts it to all.
- A process that is not leader in a round waits (a) to deliver the proposal of the leader in that round to adopt it, or (b) to suspect the leader.

Consensus algorithm I

- Implements: Consensus (cons).
- Uses:
  - BestEffortBroadcast (beb).
  - PerfectFailureDetector (P).

Upon event < Init > do
- suspected := empty;
- round := 1; currentProposal := nil;
- broadcast := delivered[] := false;

Upon event < crash, pi > do
  suspected := suspected U {pi};
Upon event < Propose, v> do
  if currentProposal = nil then
    currentProposal := v;
Upon event < bebDeliver, p_round, value > do
  currentProposal := value;
  delivered[round] := true;
Upon event delivered[round] = true or
  p_round ∈ suspected do
  round := round + 1;
Upon event p_round = self and broadcast=false and
  currentProposal=nil do
  trigger <Decide, currentProposal>;
  trigger <bebBroadcast, p_round currentProposal>;
  broadcast := true;

Consensus algorithm I

p1
propose(0) decide(0)

p2
propose(0)

p3
propose(0)
Correctness argument

• Let \( p_i \) be the correct process with the smallest id in a run \( R \).
• Assume \( p_i \) decides \( v \).
  • If \( i = n \), then \( p_n \) is the only correct process.
  • Otherwise, in round \( i \), all correct processes receive \( v \) and will not decide anything different from \( v \). They are all located after \( i \).

Question: How do you ensure that a message does not arrive too late? (in the wrong round)

Algorithm II: Uniform consensus

• The “Hierarchical Uniform Consensus” algorithm uses a perfect failure-detector, a best-effort broadcast to disseminate the proposal, a perfect link abstraction to acknowledge the receipt of a proposal, and a reliable broadcast to disseminate the decision.
• Every process maintains a single proposal value that it broadcasts in the round corresponding to its rank. When it receives a proposal from a more importantly ranked process, it adopts the value.
• In every round of the algorithm, the process whose rank corresponds to the number of the round is the leader.
Algorithm II: Uniform consensus (2)

- A round here consists of two communication steps: within the same round, the leader broadcasts a PROPOSAL message to all processes, trying to impose its value, and then expects to obtain an acknowledgment from all correct processes.
- Processes that receive a proposal from the leader of the round adopt this proposal as their own and send an acknowledgment back to the leader of the round.
- If the leader succeeds in collecting an acknowledgment from all processes except detected as crashed, the leader can decide. It disseminates the decided value using a reliable broadcast communication abstraction.

Example - no failure
Example - failure (1)

Example - failure (2)

Correctness ???

- Validity and Integrity follows from the properties of the underlying communication, and the algorithm
- Agreement
  Assume two processes decide differently, this can happen if two decisions were rbBroadcast
  Assume pi and pj, j > i, rbBroadcast two decisions vi and vj, because of accuracy of P, pj must have adopted the value vi

Exercise: uniform consensus
What if process 2 fails?
  draw an example
Is the reliable broadcast necessary?
Can you write a distributed algorithm now?
  Exercise: write the algorithm for the CIC protocol

Could you study properties of a given algorithm now?
Note: formal methods (e.g. model checking) can help
Exercise

Study the algorithm on the next slides:
1 - Show a failure free execution and 2 execution with faults
2 - is it a correct consensus? Why?
3 - is it a uniform consensus? Why?