Implementable Failure Detectors in Asynchronous Systems

Vijay K. Garg
J. Roger Mitchell
Parallel and Distributed Systems Laboratory,
Electrical and Computer Engineering Department
University of Texas at Austin,
Austin, TX 78712

http://maple.ece.utexas.edu/~vijay/

Outline of the talk

- Motivation
- Properties of Failure Detectors
  - Completeness (Strong and Weak)
  - Accuracy (Eventual weak)
- Properties introduced in this paper
  - Infinitely Often Accuracy
  - Conditional Accuracy
  - Global Accuracy
- Implementation of IO detectors
- Application: Server Maintenance Problem

Failure Detection: Motivation

- The most basic module for many fault-tolerant systems
- Function: Which entities have failed ?
  - Processes, channels, messages
  - Focus on process failures
- Converting blocking program to a non-blocking one
  - wait for a message from P_j becomes
  - wait for (a message from P_j) or (P_j suspected)
- Desirable Properties of Failure Detectors
  - Completeness: failed entity is suspected
  - Accuracy: unfailed entity is not suspected

Model of a Distributed Computation

- messages:
  - asynchronous (no upper bound on message delays)
  - reliable, no FIFO assumption
- no shared clock or memory
- Model of a process failure
  - crash: ceases all its activities
  - does not announce its crash
  - no malicious behavior

Model of a run

- One distributed program has multiple runs possible depending on
  - external inputs
  - message ordering
  - failure patterns of processes
  - time taken by messages
- Each run results in a sequence of states at each process
  - finite if the process fails
  - infinite otherwise
- Notation
  - P_i, P_j: processes
  - s,t: local states (totally ordered for a single process)

Failure Detection: Completeness

- Predicates
  - $suspects(s, i) \equiv P_i$ is suspected in state s
  - $permsusp(s, i) \equiv \forall t : s \leq t : suspects(t, i)$
  - $failed(i) \equiv P_i$ has failed
- Strong Completeness: a failed process is permanently suspected by all correct processes
  - $failed(i) \wedge \neg failed(j) \Rightarrow \exists s \in P_j : permsusp(s, i)$
- Weak Completeness: a failed process is permanently suspected by some correct process
  - $failed(i) \Rightarrow \exists s : permsusp(s, i)$
- Note: all properties implicitly universally quantified for runs.

Failure Detection: Accuracy

- Eventual Weak Accuracy
  - Eventually some correct process is never suspected by any correct process
- EW Detector: weak completeness + eventual weak accuracy
- Consensus problem can be solved in an asynchronous system given EW detector [CT 96].
- Consensus problem is impossible to solve in an asynchronous system [FLP 85] (even when at most one process fails).
- Therefore, there is no failure detector which provides weak completeness and eventual weak accuracy.

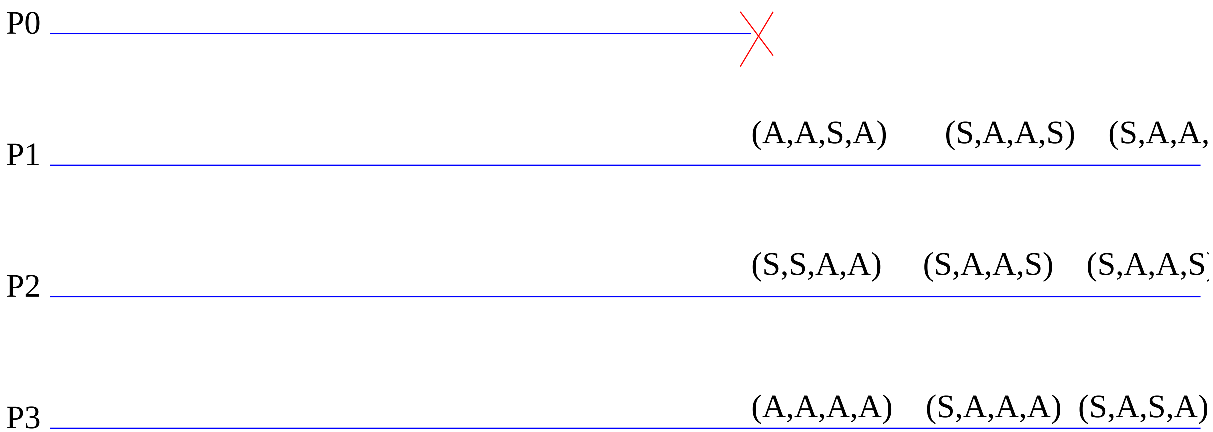
Implementable Accuracy Property

Infinitely Often Accuracy: No correct process is permanently suspected.

- $\neg failed(i) \Rightarrow \forall s : \neg permsusp(s, i)$
- Alternatively, any wrong suspicion is discovered in finite time (eventually).
  You will make mistakes in an asynchronous system. Just ensure that you discover your mistakes (eventually).

- IO Detector = Strong Completeness + Infinitely Often Accuracy

Example



A = Alive S = Suspected

Implementation of failure detectors

- Algorithm that does not satisfy IO accuracy (similar to watchd on Unix[HuaKin 96]).
  - Every k units broadcast "are you alive"
    - wait for timeout period $t < k$.
    - suspect = processes that did not respond
- Another example [Beck91]
  - multicast polling messages periodically
  - processes are expected to reply with "I am alive" messages immediately
  - If the answer is missing for three times consecutively, it assumes that this process has crashed.

Correct Implementation of IO failure detectors

- An algorithm that satisfies IO accuracy
  - Broadcast “alive” message every k units
  - On receiving “alive” from P_i
    - unsuspect P_i
    - reset the timer for P_i
  - On expiry of the timer for P_i
    - suspect P_i
- Key idea: there should not be any interval of time in which messages are ignored

Conditional Properties

- Motivation: better properties when the computation is well-behaved
- Partial Synchrony:
  - A run is partially synchronous if there exists a state s in a correct process P_i and a bound δ such that all messages sent by P_i after s take at most δ units of time.
  - P_i, s, δ may not be known in advance
- Conditional Eventual Weak Accuracy
  - A failure detector satisfies conditional eventual weak accuracy if for all partially synchronous runs it satisfies eventual weak accuracy.

IO detector at P_j with the conditional accuracy

$IO.suspects$: set of processes initially $\emptyset$;
$timeout$: array[1..N] of integer initially t;
$watch$: timer initially set to timeout;

(A1) send "alive" to all processes after every t units;
(A2) On receiving "alive" from P_i;
if $i \in IO.suspects$ then
$IO.suspects := IO.suspects - \{i\}$;
$timeout$[i]++;
Set $watch[i]$ timer for $timeout[i]$;
(A3) on expiry of watch[i]
$IO.suspects := IO.suspects \cup \{i\}$;

Infinitely Often Global Accuracy

- Stronger property than IO accuracy
  Example: Whenever P1 is accurate about P2, it is mistaken about P3 and vice versa
- GIO accuracy Every process is accurate aboue the entire system infinitely often
- Algorithm at P_i: main idea
  $timestamp$: array[1..N] of integer initially 0;
  /* when was the last time P_i received a message from P_j */
  $IOG.suspects$: set of processes initially $\emptyset$;
  $timeout$: array[1..N] of integer initially t;
  $watch$: array[1..N] of timer initially set to $timeout$;

GIO detector

(A1) send "alive" to all processes after every t units;

(A2) On receiving "alive" from P_i;

$timestamp[i]$:= time; // any increasing counter will do

if $i \in IOG.suspects$ then

$timeout[i]$++;

for $k \in \{1, .., n\}$ do

if $k \in IOG.suspects \wedge timestamp[i] \leq timestamp[k]$

$IOG.suspects := IOG.suspects - \{k\}$;

Set $watch[k]$ timer for $timeout[k]$;

(A3) on expiry of $watch[i]$

$IOG.suspects := IOG.suspects \cup \{i\}$;

Summary of IO detectors

- Strong Completeness
  - Every failed process is permanently suspected (eventually).
- Infinitely Often Accuracy
  - No unfailed process is permanently suspected.
  - (global) every process has accurate view of the system infinitely often.
- Conditional Accuracy
  - In a partially synchronous run, there exists a correct process which is eventually never suspected by anybody.

Synchronous vs Asynchronous Systems

- Argument: Just use a large value of timeout
  - Latency in failure detection high
  - trade-off between response time and accuracy of failure detection
- Algorithms must work correctly in spite of inaccurate suspicions
  - IO-accuracy implies any inaccurate suspicion will be detected in finite time

Fault-Tolerant Servers

- Need a continuously running service
- For simplicity assume that the service is stateless
  - e.g. web server for documents
- Use N servers for $N-1$ fault-tolerance
- Requirement:
  - (At least) one server responds to the request
  - Preferably only one server responds

Fault-Tolerant Server: Requirements

- Token: Whoever has the token is the current leader
- Availability:
  - There is always at least one token (modulo timeout period).
- Efficiency:
  - There are never two or more tokens (modulo message arrival period).

- Under partial synchrony:
  - Exactly one token under partial synchrony.

Algorithm for Availability and Efficiency

- process P_i is assumed to have a token if
  - all processes with smaller indices than i are suspected
  - P_i is not suspected.

$$s.token(i) \equiv \forall j : j < i : s.suspects[j] \wedge \neg s.suspects[i].$$

- Note
  - Efficiency requires Infinitely often accuracy
  - Large timeouts not desirable
  - Algorithm does not satisfy “exactly one token” under partial synchrony.

Solution: Main ideas

- Process P_i has a token
  - if all processes that are currently not suspected by P_i have ticket times that are greater than that of P_i.
- Ticket time of P_k: logical time when
  - it was suspected by some process P_i such that (according to P_i)
  - P_k had a token before the suspicion and
  - P_i has a token after the suspicion.
- If a process with a token is suspected its ticket time will become greater than all other ticket times.
  - Movement of a token from a slow process

Algorithm for P_i

ticket:array[1..N] of (int,int) initially $\forall i : ticket[i] = (0, i)$
suspected: array[1..N] of boolean; /* set by IO detector */

$$token(k) \equiv (\forall j \neq k : suspected[j] \vee (ticket[j] > ticket[k])) \wedge \neg suspected[k]$$

(R1) Upon suspicion of P_k with $token(k)$
if $token(i)$ then
$ticket[k] := Lamport's\_logical\_clock;$
send "slow", k, ticket[k] to all processes

(R2) Upon receiving "slow",k,t
$ticket[k] := max(ticket[k], t);$

Properties of the Algorithm

- Availability
  - Under false suspicion
  - Under failures
- Efficiency
  - Any inaccurate suspicion is detected due to IO accuracy
- Exactly one token under partial synchrony
  - All slow processes lose tokens

Conclusions

- Fault Tolerance crucial in distributed systems
- Be careful about the properties of your failure detector
- IO detectors give best of both the worlds
  - asynchronous behavior: mistakes will be discovered.
  - (partially) synchronous behavior: agreement, exactly one token etc.