Distributed Algorithms
Consensus: Beyond Impossibility Results

Alberto Montresor

Università di Trento

2019/10/09

Table of contents

The usual system model

- System is asynchronous
  - No bounds on messages and process execution delays
  - No bounds on clock drift
- Processes fail by crashing
  - Stop executing actions after the crash
  - We do not consider Byzantine failures
  - At most f processes fail
- Communication is reliable
  - Perfect Links

(Uniform) Consensus

Termination

Every correct process eventually decide on some value

Uniform Integrity

Each process decides at most once

Uniform Validity

If a process decides v, then v was proposed by some process

(Uniform) Agreement

No two correct (any) processes decide differently.

Consensus

Consensus in such systems

- Impossible [FLP85], even if:
  - at most one process may crash ($f = 1$), and
  - all links are reliable

Solving Consensus "in practice"

- Changing the model
- Changing the specification

Remember

- Better safe than sorry! (i.e.: look for safety, not for liveness)

Consensus

Consensus in such systems

- Impossible [FLP85], even if:
  - at most one process may crash ($f = 1$), and
  - all links are reliable

Solving Consensus "in practice"

- Changing the model
- Changing the specification

Remember

- Better safe than sorry! (i.e.: look for safety, not for liveness)

Solving Consensus

- Failure Detectors
  - Move the problem of failure detection to separate modules
  - Solve the problem even with unreliable FD
- Randomized algorithms
  - Processes are equipped with coin-flip oracles that return a random value according to some specific distribution
  - Termination is guaranteed with probability 1
- Hybrid
  - Randomized + failure detectors

Table of contents

Introduction to FD

Failure detector

A distributed oracle whose task is to provide processes with hints about which other processes are *up* (operational) or *down* (crashed)

- A fundamental building block in distributed systems
  - Reliable Broadcast
  - Consensus
  - Group membership & communication
  - ...
- Reality Check:
  - ISIS, used in the 90s for Air Traffic Control Systems

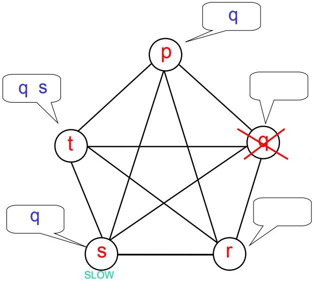
Introduction to FD

Failure detector

A distributed oracle whose task is to provide processes with hints about which other processes are *up* (operational) or *down* (crashed)

However

- Hints may be incorrect
- FD may give different hints to different processes
- FD may change its mind about the operational status of a process



Failure detectors

If they are unreliable, why using failure detectors?

- Defined by abstract properties
  - Not defined in term of a specific implementation
- Modular decomposition
  - We show correctness assuming only abstract properties
  - Any FD implementation can be used!
  - Protocols are not expressed in term of low-level parameters

Failure detectors

Problem

Which is the "weakest" failure detector $Fd_{min}(P)$ that can be used to solve problem P in an asynchronous system?

From a theoretical point of view

- Necessary and sufficient conditions

Practical considerations

- To solve P we need a system where $Fd_{min}(P)$ can be implemented
- It allows us to determine if problem P_1 is "more difficult" than P_2

History

Bibliography

T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems.
Journal of the ACM, 43(2):225–267, 1996.
http://www.disi.unitn.it/~montreso/ds/papers/CT96-JACM.pdf

V. Hadzilacos, S. Toueg, and T. D. Chandra. The weakest failure detector for solving consensus.
Journal of the ACM, 43(4):685–722, 1996.
http://www.disi.unitn.it/~montreso/ds/papers/chandra96weakest.pdf

A. Mostéfaoui and M. Raynal. Solving Consensus using Chandra-Toueg's unreliable failure detectors: A general quorum-based approach.
In *Proc. of the 13th International Symposium on Distributed Computing (DISC'00)*, pages 49–63, Bratislava, Slovak Republic, 1999.
http://www.disi.unitn.it/~montreso/ds/papers/PI-1254.pdf

Formal definitions (1)

Time

- To simplify the presentation, we assume the existence of a discrete global clock (not accessible by processes)
- Let $\mathcal{T} = \mathbb{N}$ be the set of clock ticks

Failure pattern

A failure pattern is a function $F : \mathcal{T} \Rightarrow 2^{\Pi}$, where $F(t)$ denotes the set of processes that have crashed through time t

- $\forall t \in \mathcal{T} : F(t) \subseteq F(t+1)$ (no recovery)

Formal definitions (2)

Correct, crashed set

- $crashed(F) = \bigcup_{t \in \mathcal{T}} F(t)$ (Crashed set)
- $correct(F) = \Pi - crashed(F)$ (Correct set)
- A process is correct if belongs to $correct(F)$, otherwise is faulty

Failure detector history

A failure detector history is a function $H : \Pi \times \mathcal{T} \to 2^{\Pi}$, where $H(p,t)$ is the output of the failure detector of process p at time t

- If $q \in H(p,t)$, we say that p suspects q at time t in H

Completeness

Strong Completeness

Eventually, every faulty process is permanently suspected by every correct process.

$\forall F, \forall H, \exists t \in \mathcal{T}, \forall p \in crashed(F), \forall q \in correct(F), \forall t' \geq t : p \in H(q, t')$

Weak Completeness

Eventually, every faulty process permanently suspected by some correct process.

$\forall F, \forall H, \exists t \in \mathcal{T}, \forall p \in crashed(F), \exists q \in correct(F), \forall t' \geq t : p \in H(q, t')$

Motivation behind Weak Completeness

We do not want every process "to ping" all other processes continuously

Accuracy (1)

Strong Accuracy

Every correct process is never suspected.

$$\forall F, \forall H, \forall t \in \mathcal{T}, \forall p \in correct(F), \forall q : p \notin H(q, t)$$

Weak Accuracy

Some correct process is never suspected.

$$\forall F, \forall H, \forall t \in \mathcal{T}, \exists p \in correct(F), \forall q : p \notin H(q, t)$$

Accuracy (2)

Eventual Strong Accuracy

There is a time after which every correct process is not suspected by any correct process.

$\forall F, \forall H, \exists t \in \mathcal{T}, \forall t \geq t', \forall p \in \mathit{correct}(F), \forall q \in \mathit{correct}(F) : p \notin H(q, t')$

Eventual Weak Accuracy

There is a time after which some correct process is never suspected by any correct process.

$\forall F, \forall H, \exists t \in \mathcal{T}, \forall t \geq t', \exists p \in \mathit{correct}(F), \forall q \in \mathit{correct}(F) : p \notin H(q, t')$

Failure Detector Classes

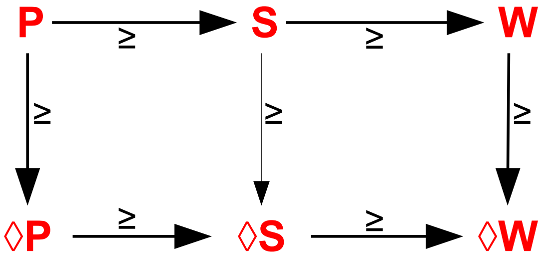
	Accuracy			
	Strong	Weak	Ev. Strong	Ev. Weak
Strong Completeness	Perfect P	Strong S	Ev. Perfect $\diamond P$	Ev. Strong $\diamond S$
Weak Completeness		Weak W		Ev. Weak $\diamond W$

Reductions

Reduction

We say that an algorithm $T_{D \rightarrow E}$ is a reduction from D to E if it transforms a failure detector of class D into a failure detector of class E, and we write $D \geq E$.

Some easy reductions



From Weak Completeness to Strong Completeness

Reduction from class D to class E, executed by process p_i

upon initialization **do**
- $suspected_i^E \leftarrow \emptyset$

repeat periodically
- B-broadcast($\langle \textsc{suspect}, suspected_i^D \rangle$)

upon B-deliver($\langle \textsc{suspect}, S \rangle$) **from** p_j **do**
- $suspected_i^E \leftarrow suspected_i^E \cup S - \{p_i, p_j\}$

Using this reduction, we can show that

- $\diamond W \geq \diamond S$, so $\diamond W \equiv \diamond S$
- $W \geq S$, so $W \equiv S$

Table of contents

Reliable Broadcast Recap

Reliable Broadcast

- Implementable with process failures and message omissions
- Proposed implementation: flooding, $O(n^2)$ messages

Uniform Reliable Broadcast

- Implementable with process failures and no message omissions
- Same implementation (different assumptions)

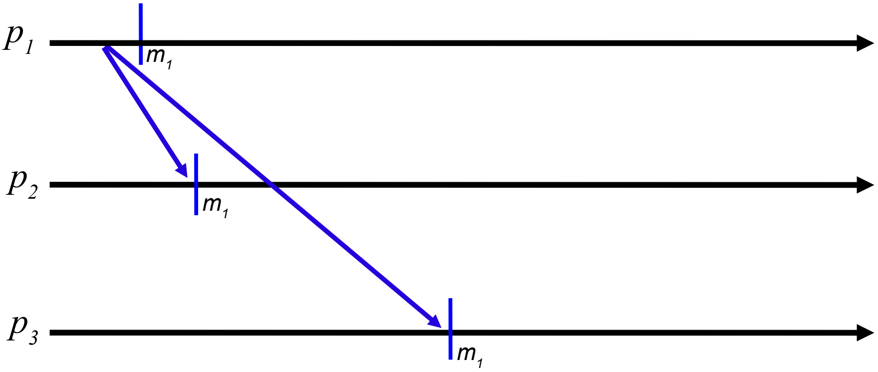
Message complexity

- Conservative protocol: many messages in the absence of failures
- Can we do better than that?
- We apply failure detectors

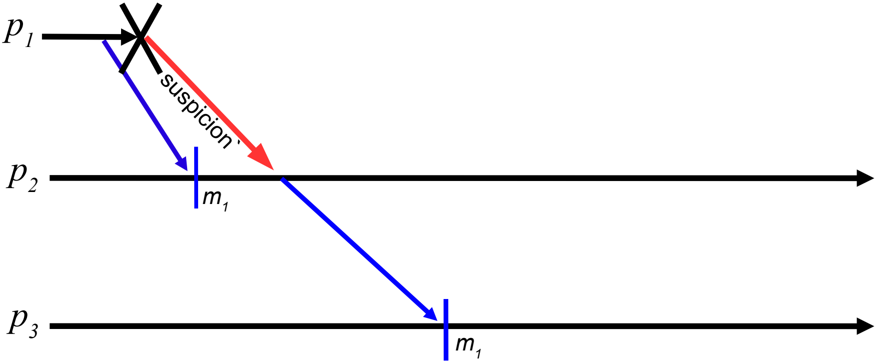
Reliable broadcast protocol based on $\diamond S$ executed by p

```
upon initialization do
    Set delivered ← Set⟨Message⟩                % Msgs already delivered
    Map from ← new Map⟨Process, Set⟩()          % Msgs sent from processes

upon R-broadcast(m) do
    send ⟨m, p⟩ to Π

upon q ∈ ◇P.suspect() do
    foreach m ∈ from[q] do
        send ⟨m, q⟩ to Π

upon receive(⟨m, s⟩) do
    from[s] ← from[s] ∪ {m}
    if not m ∈ delivered then
        R-deliver(m)
        delivered ← delivered ∪ {m}
        if s ∈ ◇P.suspect() then
            send ⟨m, s⟩ to Π
```

Reliable Broadcast with $\diamond S$ – Scenario 1



Reliable Broadcast with $\diamond S$ – Scenario 2



Reliable Broadcast with $\diamond S$ – Proof

- Uniform Integrity, Validity: As before
- Agreement:
  Let p be a correct process that R-delivers a message m
  Let q be another correct process
  Let $s = \mathsf{sender}(m)$; there are two cases
  - Case 1: s is correct – by Validity of Perfect Channels, q will receive m sent by s
  - Case 2: s is faulty – by Strong Completeness, s will be suspected by p, p will send m, q will receive it

Comment

If the failure detector is not accurate, more messages will be sent; but not other adverse effect will occur

Reliable Broadcast through FD

Reliable Broadcast

- Can be implemented using a linear number of messages in the absence of failures
- An Eventually Perfect FD as accurate as possible is required to reduce the number of messages

But...

- Think what is needed to implement a failure detector!

Table of contents

Consensus and Failure Detectors

Problem

Is perfect failure detection necessary for Consensus?

$\diamond S$ versus Consensus

- Initially, it can output arbitrary information
- But there is a time after which:
  - Every process that crashes is suspected (completeness)
  - Some process that does not crash is not suspected (accuracy)
- When $f < n/2$, $\diamond S$ is necessary and sufficient to solve Consensus
- Note: $\diamond S \equiv \diamond W$

Consensus and Failure Detectors

Problem

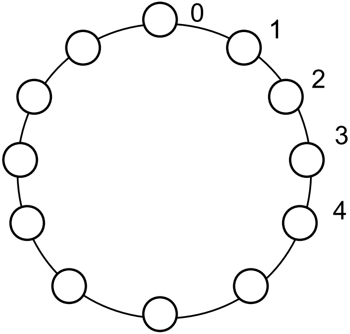
Is perfect failure detection necessary for Consensus?

S versus Consensus

- It can output arbitrary information about most of the processes
- But there is at least one correct process which is never suspected
- When $f < n$, S is necessary and sufficient to solve Consensus
- Note: $S \equiv W$

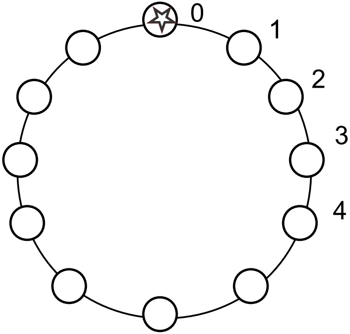
Rotating coordinators

- Processes are numbered $0, 1, \ldots, n-1$
- They execute asynchronous rounds
- In round r, the coordinator
  - corresponds to process (r mod n)
  - tries to impose its estimate as the consensus value
  - succeeds if does not crash and it is not suspected by $\diamond S$
- The protocol described here is based on [Mostéfaoui and Raynal, 1999]



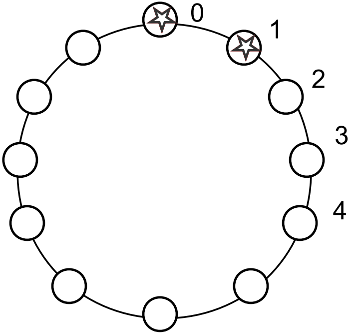
Rotating coordinators

- Processes are numbered $0, 1, \ldots, n-1$
- They execute asynchronous rounds
- In round r, the coordinator
  - corresponds to process (r mod n)
  - tries to impose its estimate as the consensus value
  - succeeds if does not crash and it is not suspected by $\diamond S$
- The protocol described here is based on [Mostéfaoui and Raynal, 1999]



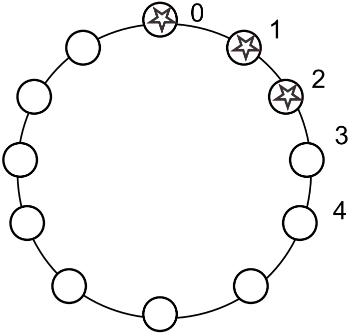
Rotating coordinators

- Processes are numbered $0, 1, \ldots, n-1$
- They execute asynchronous rounds
- In round r, the coordinator
  - corresponds to process ($r \bmod n$)
  - tries to impose its estimate as the consensus value
  - succeeds if does not crash and it is not suspected by $\diamond S$
- The protocol described here is based on [Mostéfaoui and Raynal, 1999]



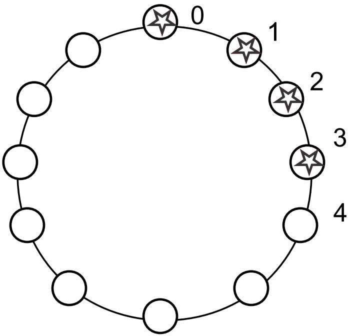
Rotating coordinators

- Processes are numbered $0, 1, \ldots, n-1$
- They execute asynchronous rounds
- In round r, the coordinator
  - corresponds to process ($r \bmod n$)
  - tries to impose its estimate as the consensus value
  - succeeds if does not crash and it is not suspected by $\diamond S$
- The protocol described here is based on [Mostéfaoui and Raynal, 1999]



Rotating coordinators

- Processes are numbered $0, 1, \ldots, n-1$
- They execute asynchronous rounds
- In round r, the coordinator
  - corresponds to process ($r \bmod n$)
  - tries to impose its estimate as the consensus value
  - succeeds if does not crash and it is not suspected by $\diamond S$
- The protocol described here is based on [Mostéfaoui and Raynal, 1999]



Consensus Algorithm based on $\diamond S$ executed by process p_i

```
upon propose(v_i) do
    integer r ← 0                                   % Round
    integer est ← v_i                               % Estimate
    boolean decided ← false
    boolean stop ← false
    while not stop do
        integer c ← r mod n                         % Coordinator
        r ← r + 1
        { Phase 1 of round r; from p_c to all }
        if i = c then
            B-broadcast(⟨PHASE1, r, est, p_i⟩)
1       wait B-deliver(⟨PHASE1, r, v, p_c⟩) or p_c ∈ suspected_i^◇S
        if p_c ∈ suspected_i then
            aux ← ?
        else
            aux ← v
```

Consensus Algorithm based on $\diamond S$ executed by process p_i

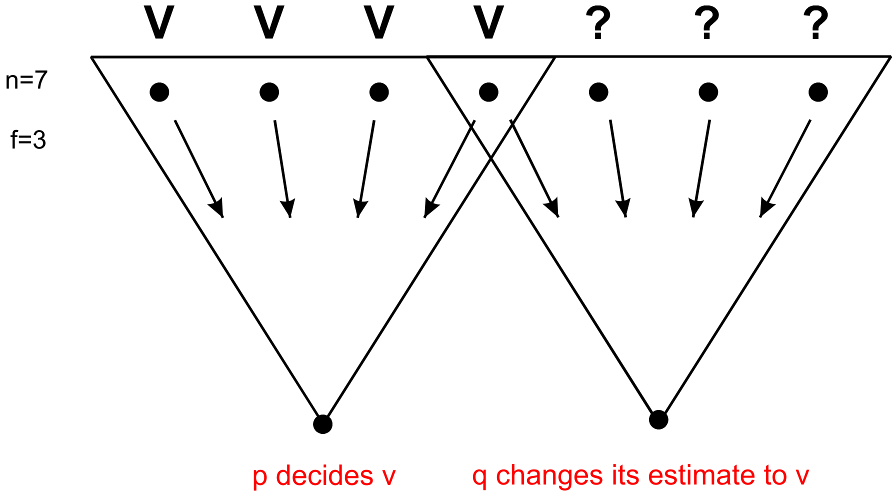
```
    { Phase 2 of round r; from all to all }
    B-broadcast(⟨PHASE2, r, aux, p_i⟩)
    SET rec ← ∅                          % Received values
    SET proc ← ∅                         % Replying processes
    while |proc| ≤ ⌊n/2⌋ do
        wait B-deliver(⟨PHASE2, r, v, p_j⟩)
        rec ← rec ∪ {v}
        proc ← proc ∪ {p_j}
    if rec =  {v}   then est ← v; B-broadcast(⟨DECIDE, v⟩); stop ← true
    if rec = {v, ?} then est ← v
    if rec =  {?}   then do nothing

upon B-deliver(⟨DECIDE, v⟩) do
    if not decided then
        B-broadcast(⟨DECIDE, v⟩)
        decide(v)
        decided ← true
```

Proof of correctness – Termination

Proof: Termination

- **wait** #1: With $\diamond S$, no process blocks forever waiting for a message from a dead coordinator
- **wait** #2: Given that $f < n/2$, eventually every node will receive more than $\lfloor n/2 \rfloor$ messages and will exit from Phase 2
- Thanks to $\diamond S$, eventually some correct process p_c is not falsely suspected. When p_c becomes the coordinator, every correct process receives c's estimate and decides.

Proof of correctness – Agreement



Consensus Algorithm based on S executed by process p_i

upon propose(v_i) **do**
  integer $r \leftarrow 0$ % Round
  integer $est \leftarrow v_i$ % Estimate
  boolean $decided \leftarrow$ **false**
  boolean $stop \leftarrow$ **false**

  while not $stop$ **do**
    integer $c \leftarrow r \bmod n$ % Coordinator
    $r \leftarrow r + 1$

    { Phase 1 of round r; from p_c to all }

    if $i = c$ **then**
      B-broadcast($\langle \text{PHASE1}, r, est, p_i \rangle$)
1   **wait** B-deliver($\langle \text{PHASE1}, r, v, p_c \rangle$) **or** $p_c \in suspected_i^S$
    if $p_c \in suspected_i$ **then**
      $aux \leftarrow ?$
    else
      $aux \leftarrow v$

Consensus Algorithm based on S executed by process p_i

2

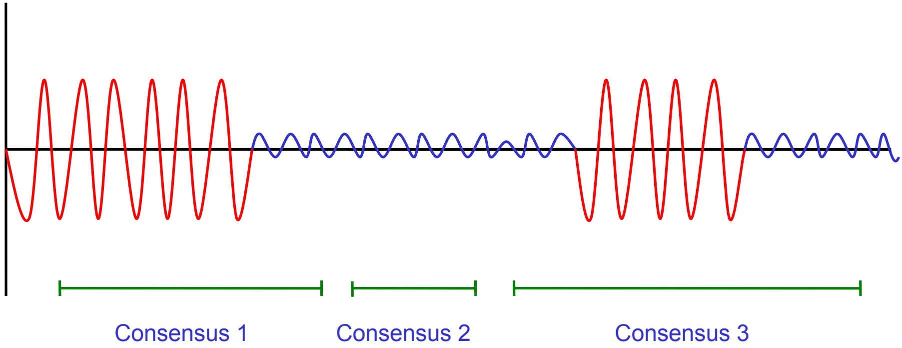
{ Phase 2 of round r; from all to all }

B-broadcast($\langle$PHASE2, r, aux, $p_i\rangle$)
SET $rec \leftarrow \emptyset$ % Received values
SET $proc \leftarrow \emptyset$ % Replying processes
while $proc \cup suspected_i^S \neq \Pi$ **do** % Was: $|proc| < n/2$
  wait B-deliver($\langle$PHASE2, r, v, $p_j\rangle$)
  $rec \leftarrow rec \cup \{v\}$
  $proc \leftarrow proc \cup \{p_j\}$
if $rec = \{v\}$ **then** $est \leftarrow v$; B-broadcast($\langle$DECIDE, $v\rangle$); $stop \leftarrow$ **true**
if $rec = \{v, ?\}$ **then** $est \leftarrow v$
if $rec = \{?\}$ **then** do nothing

upon B-deliver($\langle$DECIDE, $v\rangle$) **do**
  if not $decided$ **then**
    B-broadcast($\langle$DECIDE, $v\rangle$)
    decide(v)
    $decided \leftarrow$ **true**

What if the FD misbehaves

- Accuracy can be not satisfied
  - Consensus algorithm remains always safe
  - It is also live – during “good” FD periods
- Completeness is always satisfied



Indulgent algorithms

Indulgent algorithms

- Never violate the safety property
- If the FD is not accurate, they do not terminate
- Require "stable" periods in order to terminate

The protocol just shown is an indulgent algorithm

Bibliography

R. Guerraoui. Indulgent algorithms.
In *Proc. of the 19th annual ACM Symposium on Principles of Distributed Computing Systems (PODC'00)*, pages 49–63, Portland, OR, 2000.
http://www.disi.unitn.it/~montreso/ds/papers/p289-guerraoui.pdf

Failure detectors as an abstraction

Some advantages

- Increases the modularity and portability of algorithms
- Suggests why Consensus is not so difficult in practice
- Determines minimal info about failures to solve consensus
- Encapsulates various models of partial synchrony

Broadening the applicability of FDs

Other models

- Crashes + Link failures (fair links)
- Network partitioning
- Crash/Recovery
- Byzantine (arbitrary) failures
- FDs + Randomization

Other problems

- Atomic Commitment
- Group Membership
- Leader Election
- Reliable Broadcast ✓

From theory to practice

- FD implementation needs to be message-efficient:
  - FDs with linear message complexity (ring, hierarchical, gossip)
- "Eventual" guarantees are not sufficient:
  - FDs with Quality-of-Service guarantees
- Failure detection should be easily available:
  - Shared FD service (with QoS guarantees)

From theory to practice

Bibliography

R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service.
In *Proc. of Middleware '98*, pages 55–70, The Lake District, United Kingdom, 1998. Springer-Verlag.
http://www.disi.unitn.it/~montreso/ds/papers/fd-gossip.pdf

W. Chen, S. Toueg, and M. Aguilera. On the quality of service of failure detectors.
IEEE Transactions on Computers, 51:561–580, 2002.
http://www.disi.unitn.it/~montreso/ds/papers/fd-qos.pdf

Table of contents

Another approach: randomization

- First protocol to achieve binary Consensus with probabilistic termination in an asynchronous model
- The protocol is f-correct - tolerates up to f crash failures, with $f < n/2$
- Expected time: $O(2^{2n})$ phases to converge

Bibliography

M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract).
In *Proc. of the* 2^{nd} *annual ACM Symposium on Principles of Distributed Computing Systems (PODC'83)*, pages 27–30, 1983.
http://www.disi.unitn.it/~montreso/ds/papers/p27-ben-or.pdf

Ben-Or's Algorithm

- Operates in rounds, each round has two phases:
  - Report phase – each process transmits its value, and waits to hear from other processes
  - Decision phase – if majority found, take its value; otherwise, flip a coin to change the local value
- The idea:
  - If enough processes detected the majority, decide
  - If I know that someone detected majority, switch to the majority's value
  - Otherwise, flip a coin; eventually, a majority of correct processes will flip in the same way

Ben-Or's Algorithm executed by process p_i

upon propose(v_i) **do**
- **integer** $r \leftarrow 0$ % Round
- **integer** $est \leftarrow v_i$ % Estimate

while true do
- $r \leftarrow r + 1$
- B-broadcast($\langle \text{REPORT}, r, est \rangle$)
- **wait** to deliver more than $n - f$ $\langle \text{REPORT}, r, * \rangle$ messages
- **if** delivered more than $n/2$ $\langle \text{REPORT}, r, v \rangle$ messages with the same value v **then**
  - B-broadcast($\langle \text{PROPOSAL}, r, v \rangle$)
- **else**
  - B-broadcast($\langle \text{PROPOSAL}, r, ? \rangle$)
- **wait** to deliver more than $n - f$ $\langle \text{PROPOSAL}, r, * \rangle$ messages
- **if** delivered a $\langle \text{PROPOSAL}, r, v \rangle$ with v with $v \neq ?$ **then**
  - $est \leftarrow v$
- **else**
  - $est \leftarrow$ random($\{0, 1\}$)
- **if** delivered more than f $\langle \text{PROPOSAL}, r, v \rangle$ with v with $v \neq ?$ **then**
  - decide(v)

The algorithm

- Based on the original version of Ben-Or
- It never stops; once decided, it keeps deciding the same value
- It is easy to transform it in an algorithm that stops one round after the one in which the decision has been taken

Proof of correctness

Proof: Uniform Agreement

- At most one value can receive majority in the first phase of a round
- If some process sees $f+1$ $\langle \text{PROPOSAL}, r, v \neq ? \rangle$, then:
  - every process sees at least one $\langle \text{PROPOSAL}, r, v \neq ? \rangle$ message
- if every process sees at least one $\langle \text{PROPOSAL}, r, v \neq ? \rangle$ message, then
  - every process changes its estimate to v
  - every process reports v in the first phase of round $r+1$
- If every process reports v in the first phase of round $r+1$,
  - every process decides v in the second phase of round $r+1$

Proof of correctness

Proof: Validity

- If there are two distinct values at the beginning, one of them will be chosen
- Otherwise, if all processes report their common value v at round 0, then:
  - all processes send $\langle \text{PROPOSAL}, 0, v \rangle$
  - all processes decide on the second phase of round 0

Proof of correctness

Proof: Termination

- If no process sees the majority value, then they all will flip coins, and start everything again
- Eventually a majority among the correct processes flips the same random value
  - The correct processes will observe the majority value.
  - The correct processes will propagate PROPOSAL messages, containing the majority value
- Correct processes will receive the PROPOSAL messages, and the protocol finishes

Table of contents

Hybrid approach

- We can combine
  - Failure Detectors
  - Randomized approach
- Advantages:
  - Deterministic termination if FD is accurate ("good periods")
  - Probabilistic termination otherwise ("bad periods")
- Oracles available at each process
  - FD-oracle: Failure detector $\diamond S$
  - R-oracle: Random coin-flip

Reading material

T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems.
Journal of the ACM, 43(2):225–267, 1996.
http://www.disi.unitn.it/~montreso/ds/papers/CT96-JACM.pdf

A. Mostéfaoui and M. Raynal. Solving Consensus using Chandra-Toueg's unreliable failure detectors: A general quorum-based approach.
In *Proc. of the 13th International Symposium on Distributed Computing (DISC'00)*, pages 49–63, Bratislava, Slovak Republic, 1999.
http://www.disi.unitn.it/~montreso/ds/papers/PI-1254.pdf

M. K. Aguilera and S. Toueg. Failure detection and randomization: A hybrid approach to solve consensus.
SIAM Journal of Computing, 1998.
http://www.disi.unitn.it/~montreso/ds/papers/hybrid-siam1998.pdf