This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.
Contents

1 Introduction
2 Failure detectors
   • Introduction
   • Specification
   • Reductions
3 Reliable Broadcast
4 Consensus
   • Introduction
   • Algorithms
   • Discussion
5 Randomization
   • Ben-Or protocol
6 Hybrid approach
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
Introduction

**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)
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
Introduction to FD

Definition (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**

### Definition (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


Formal definitions (1)

**Definition (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

**Definition (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)

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

Definition (Failure detector history)
A failure detector history is a function \( H : \Pi \times \mathcal{T} \rightarrow 2^\Pi \), 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

Definition (Strong Completeness)
Eventually, every faulty process is permanently suspected by every correct process.

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

Definition (Weak Completeness)
Eventually, every faulty process permanently suspected by some correct process.

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

Motivation behind Weak Completeness
We do not want every process “to ping” all other processes continuously
Accuracy (1)

Definition (Strong Accuracy)
Every correct process is never suspected.

\[ \forall F, \forall H, \forall t \in T, \forall p \in correct(F), \forall q : p \notin H(q, t) \]

Definition (Weak Accuracy)
Some correct process is never suspected.

\[ \forall F, \forall H, \forall t \in T, \exists p \in correct(F), \forall q : p \notin H(q, t) \]
Accuracy (2)

Definition (Eventual Strong Accuracy)
There is a time after which every correct process is not suspected by any correct process.

\[ \forall F, \forall H, \exists t \in T, \forall t \geq t', \forall p \in \text{correct}(F), \forall q \in \text{correct}(F) : p \notin H(q, t') \]

Definition (Eventual Weak Accuracy)
There is a time after which some correct process is never suspected by any correct process.

\[ \forall F, \forall H, \exists t \in T, \forall t \geq t', \exists p \in \text{correct}(F), \forall q \in \text{correct}(F) : p \notin H(q, t') \]
## Failure Detector Classes

<table>
<thead>
<tr>
<th>Completeness</th>
<th>Accuracy</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong</td>
<td>Strong</td>
<td>Ev. Strong</td>
<td>Ev. Weak</td>
<td></td>
</tr>
<tr>
<td>Perfect</td>
<td>Strong</td>
<td>Ev. Perfect</td>
<td>Ev. Strong</td>
<td></td>
</tr>
<tr>
<td>$P$</td>
<td>$S$</td>
<td>$\diamond P$</td>
<td>$\diamond S$</td>
<td></td>
</tr>
<tr>
<td>Weak</td>
<td>Weak</td>
<td>Ev. Weak</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$W$</td>
<td></td>
<td>$\diamond W$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Reductions

Definition (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 $p_i$

\begin{verbatim}
upon initialization do
    suspected$_i^E$ ← ∅

repeat periodically
    B-broadcast(⟨SUSPECT, suspected$_i^D$⟩)

upon B-deliver(⟨SUSPECT, S⟩) from $p_j$ do
    suspected$_i^E$ ← suspected$_i^E$ ∪ $S$ − {$_p_i$, $p_j$}
\end{verbatim}

Using this reduction, we can show that:

- $\diamondsuit W \geq \diamondsuit S$, so $\diamondsuit W \equiv \diamondsuit S$
- $W \geq S$, so $W \equiv S$
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 $\Diamond P$ executed by $p$

**upon** initialization do

\[
\begin{align*}
    \text{Set } \textit{delivered} & \leftarrow \text{Set}\langle \text{Message}\rangle & \text{ % Msgs already delivered} \\
    \text{Map } \textit{from} & \leftarrow \text{new } \text{Map}\langle \text{Process, Set}\rangle() & \text{ % Msgs sent from processes}
\end{align*}
\]

**upon** R-broadcast$(m)$ do

\[
\begin{align*}
    \text{send } \langle m, p \rangle & \text{ to } \Pi
\end{align*}
\]

**upon** $q \in \Diamond P.$suspect() do

\[
\begin{align*}
    \text{foreach } m \in \textit{from}[q] \text{ do} \\
        \text{send } \langle m, q \rangle & \text{ to } \Pi
\end{align*}
\]

**upon** receive$(\langle m, s \rangle)$ do

\[
\begin{align*}
    \textit{from}[s] & \leftarrow \textit{from}[s] \cup \{m\} \\
    \text{if not } m & \in \textit{delivered} \text{ then} \\
        \text{R-deliver}(m) \\
        \textit{delivered} & \leftarrow \textit{delivered} \cup \{m\} \\
    \text{if } s & \in \Diamond P.$suspect() \text{ then} \\
        \text{send } \langle m, s \rangle & \text{ to } \Pi
\end{align*}
\]
Reliable Broadcast with $\diamond P$ – Scenario 1

$p_1 \rightarrow m_1 \rightarrow p_2 \rightarrow m_1 \rightarrow p_3$
Reliable Broadcast with $\Diamond P$ – Scenario 2

Alberto Montresor (UniTN)
Reliable Broadcast with ◇P – Proof

- **Uniform Integrity, Validity:** As before

- **Agreement:**
  - Let \( p \) be a process that R-delivers a message \( m \)
  - Let \( q \) be another process
  - Let \( s = \text{sender}(m) \); there are two cases
    - Case 1: \( s \) is correct – by Validity of Perfect Channels
    - Case 2: \( s \) is faulty – by Eventual Strong Accuracy

**Comment**

If the failure detector is not accurate, more messages will be sent; but not other adverse effect will occur.
Proof of Reliable Broadcast using $\Diamond P$

By contradiction, let $p$ and $q$ two correct processes such that $p$ R-delivers a message $m$ that $q$ never R-delivers.

Let $\text{sender}(m) = s$. There are two cases

- $s$ is correct; by Validity of Perfect Channels, $m$ will eventually be received by $q$, that R-deliver it (a contradiction)

- $s$ is faulty; by Eventual Strong Accuracy, $s$ will be eventually suspected by all correct process, so also by $p$; $p$ will forward the message $m$ to all processes, and by Validity of Perfect Channels $q$ will eventually receive it.
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!
Consensus and Failure Detectors

Problem
Is perfect failure detection necessary for Consensus?

◊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 $f < n/2$, ◊S is necessary and sufficient to solve Consensus

- Note: ◊S ≡ ◊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]
Consensus Algorithm based on $\diamond S$ 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 $c \leftarrow r \mod n$ % Coordinator
$r \leftarrow r + 1$

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

if $i = c$ then
```
B-broadcast($\langle$PHASE1, $r$, $est$, $p_i$)$)
```

1

wait B-deliver($\langle$PHASE1, $r$, $v$, $p_c$)$) or $p_c \in suspected_i^\diamond S$

if $p_c \in suspected_i$ then
```
aux \leftarrow ?
```
else
```
aux \leftarrow v
```
```
Consensus Algorithm based on $\diamondsuit S$ executed by process $p_i$

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

B-broadcast($\langle\text{PHASE2}, r, aux, p_i\rangle$)

Set $rec \leftarrow \emptyset$ % Received values
Set $proc \leftarrow \emptyset$ % Replying processes

while $|proc| \leq \lfloor n/2 \rfloor$ do

2. wait B-deliver($\langle\text{PHASE2}, r, v, p_j\rangle$)

$rec \leftarrow rec \cup \{v\}$

$proc \leftarrow proc \cup \{p_j\}$

if $rec = \{v\}$ then $est \leftarrow v$; B-broadcast($\langle\text{DECIDE}, v\rangle$); $stop \leftarrow true$

if $rec = \{v, ?\}$ then $est \leftarrow v$

if $rec = \{?\}$ then do nothing

upon B-deliver($\langle\text{DECIDE}, v\rangle$) do

if not decided then

B-broadcast($\langle\text{DECIDE}, v\rangle$)

decide($v$)

$decided \leftarrow true$
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

n=7

f=3

p decides v  q changes its estimate to v
• When a process decides $v$, it has received a majority of messages containing $v$
• There is a majority of correct nodes
• The intersection of these majorities is not empty
• Every correct process will receive at least one $v$ from such process
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 $c \leftarrow r \mod n$ % Coordinator
    $r \leftarrow r + 1$
    
    { Phase 1 of round $r$; from $p_c$ to all }
    if $i = c$ then
        B-broadcast($\langle$PHASE1, $r$, $est$, $p_i$\rangle)

    wait B-deliver($\langle$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 $p_i$

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

B-broadcast($\langle$PHASE2, $r$, aux, $p_i$$\rangle$)

SET $\text{rec} \leftarrow \emptyset$  % Received values
SET $\text{proc} \leftarrow \emptyset$  % Replying processes

while $\text{proc} \cup \text{suspected}_i^S \neq \Pi$ do  % Was: $|\text{proc}| < n/2$
  \hspace{1em} wait B-deliver($\langle$PHASE2, $r$, $v$, $p_j$$\rangle$)
  \hspace{1em} $\text{rec} \leftarrow \text{rec} \cup \{v\}$
  \hspace{1em} $\text{proc} \leftarrow \text{proc} \cup \{p_j\}$
  \hspace{2em} if $\text{rec} = \{v\}$ then $\text{est} \leftarrow v$; B-broadcast($\langle$DECIDE, $v$$\rangle$); $\text{stop} \leftarrow \text{true}$
  \hspace{2em} if $\text{rec} = \{v, ?\}$ then $\text{est} \leftarrow v$
  \hspace{2em} if $\text{rec} = \{?\}$ then do nothing

upon B-deliver($\langle$DECIDE, $v$$\rangle$) do
  \hspace{1em} if not decided then
    \hspace{2em} B-broadcast($\langle$DECIDE, $v$$\rangle$)
    \hspace{2em} decide($v$)
    \hspace{2em} decided $\leftarrow \text{true}$
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

Definition (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.
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)

**Bibliography**


When speaking about shared FD services, make a note back to the discussion about implementing Reliable Broadcast with or without a FD.
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).
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($\langle$REPORT, $r$, $est$$\rangle$)
  wait to deliver more than $n - f$ $\langle$REPORT, $r$, $*$ $\rangle$ messages
  if delivered more than $n/2$ $\langle$REPORT, $r$, $v$$\rangle$ messages with the same value $v$ then
    B-broadcast($\langle$PROPOSAL, $r$, $v$$\rangle$)
  else
    B-broadcast($\langle$PROPOSAL, $r$, $?$$\rangle$)
  wait to deliver more than $n - f$ $\langle$PROPOSAL, $r$, $*$ $\rangle$ messages
  if delivered a $\langle$PROPOSAL, $r$, $v$$\rangle$ with $v$ with $v \neq ?$ then
    $est \leftarrow v$
  else
    $est \leftarrow \text{random}(\{0, 1\})$
  if delivered more than $f$ $\langle$PROPOSAL, $r$, $v$$\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

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

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 $\langle$PROPOSAL, 0, $v$$\rangle$
  - all processes decide on the second phase of round 0
Proof of correctness

**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.
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

Bibliography

Reading Material

  http://www.disi.unitn.it/~montresco/ds/papers/CT96-JACM.pdf