The weakest failure detectors to solve certain fundamental problems in distributed computing

Carole Delporte-Gallet  Hugues Fauconnier
Vassos Hadzilacos  Rachid Guerraoui
Petr Kouznetsov  Sam Toueg
Contribution

The weakest failure detectors for:

- Implementing an atomic register
- Solving consensus
- Solving *quittable* consensus (QC)
- Solving non-blocking atomic commit (NBAC)

in distributed message-passing systems, for all environments!
Some related work

- Implementing registers with a majority of correct processes [ABD95]
- The weakest failure detector for consensus with a majority of correct processes [CHT96]
- Implementing registers and solving consensus in other environments [DFG02]
- NBAC with failure detectors [FRT99, Gue02, GK02]
Roadmap

1. Model: asynchronous system with failure detectors
2. Implementing a register
3. Solving consensus
4. Solving QC
5. Solving NBAC
Asynchronous message-passing system

- Communication by message-passing through reliable channels
- Processes can fail only by crashing
  Correct processes never crash

- In such a system:
  ✓ Register can be implemented if and only if a majority of processes are correct [ABD95]
  ✓ (Weak) consensus is not solvable if at least one process can crash [FLP85]
Environments

An environment $E$ specifies *when* and *where* failures might occur

Examples:
- Majority of processes are correct
- At most one process crash
Failure detectors [CT96, CHT96]

Each process has a failure detector module that provides some (maybe incomplete and inaccurate) information about failures

*Failure signal* failure detector FS: at each process, FS outputs green or red.

- If red is output, then a failure previously occurred.
- If a failure occurs, then eventually red is output at all correct processes.
The weakest failure detector

D is the weakest failure detector to solve problem P in an environment E if and only if:

- D is sufficient for P in E: D can be used to solve P in E
- D is necessary for P in E: D can be extracted from any failure detector D′ that can be used to solve P in E
Roadmap

1. Model: asynchronous system with failure detectors
2. **Implementing a register**
3. Solving consensus
4. Solving QC
5. Solving NBAC
Problem: implementing a register

- An atomic register is an object accessed through *reads* and *writes*
- The \texttt{write}(v) stores \(v\) at the register and returns \texttt{ok}
- The \texttt{read} returns the last value written at the register
**Quorum** failure detector $\Sigma$

At each process, $\Sigma$ outputs a set of processes

- Any two sets (output at any times and at any processes) intersect.
- Eventually every set contains only correct processes.
Σ is sufficient to implement registers

- Adapt the “correct majority-based” algorithm of [ABD95] to implement (1 reader, 1 writer) atomic register using Σ:
  Substitute
  « process p waits until a majority of processes reply »
  with
  « process p waits until all processes in Σ reply »
Σ is necessary to implement registers

Let A be any implementation of registers that uses some failure detector D.
Must show that we can extract Σ from D.

☐ Each write operation involves a set of “participants”: the processes that help the operation take effect (w.r.t. A and D)

Fact: the set of participants includes at least one correct process
Extraction algorithm

Every process p periodically:

- writes in its register the participant sets of its previous writes
- reads participant sets of other processes
- outputs
  - the participant set of its previous write, and
  - for every known participant set S, one live process in S

All output sets intersect and eventually contain only correct processes
Registers: the weakest failure detector

Σ is the weakest failure detector to implement atomic registers, in any environment
Roadmap

1. Model: asynchronous system with failure detectors
2. Implementing a register
3. *Solving consensus*
4. Solving QC
5. Solving NBAC
Leader failure detector $\Omega$ [CHT96]

Outputs the id of a process. Eventually, the id of the same correct process is output at all correct processes.
Consensus $\Leftrightarrow$ registers + $\Omega$

- $\Omega$ can be used to solve consensus with registers, in any environment [LH94]

- Consensus $\Rightarrow$ Registers: any consensus algorithm can be used to implement registers, in any environment [Lam86, Sch90]

- Consensus $\Rightarrow$ $\Omega$: $\Omega$ can be extracted from any failure detector $D$ that solves consensus, in any environment [CHT96]
Consensus: the weakest failure detector

- Consensus $\Leftrightarrow$ registers $+$ $\Omega$ (in any environment)
- $\Sigma$ is the weakest FD to implement registers (in any environment)

Thus,

$(\Omega, \Sigma)$ is the weakest failure detector to solve consensus, in any environment
Roadmap

1. Model: asynchronous system with failure detectors
2. Implementing a register
3. Solving consensus
4. *Solving QC*
5. Solving NBAC
Quittable consensus (QC)

QC is like consensus except that

*if a failure occurs*, then processes can agree

- on the special value Q (« Quit »), *or*
- on one of the proposed values (as in consensus)
Failure detector \( \Psi \)

- For some initial period of time \( \Psi \) outputs some predefined value \( T \)
- Eventually,
  - \( \checkmark \) \( \Psi \) behaves like \( (\Omega,\Sigma) \), or
  - \( (\text{only if a failure occurs}) \) \( \Psi \) behaves like FS (outputs red)

**NB:** If a failure occurs, \( \Psi \) can choose to behave like \( (\Omega,\Sigma) \) or like FS (the choice is the same at all processes)
Ψ is sufficient to solve QC

\[
\text{Propose}(v) \quad \text{// } v \text{ in } \{0,1\} \\
\quad \text{wait until } \Psi \neq \top \\
\quad \text{if } \Psi = \text{red then return } Q \quad \text{// If } \Psi \text{ behaves like FS} \\
\quad d := \text{ConsPropose}(v) \quad \text{// If } \Psi \text{ behaves like } (\Omega, \Sigma) \\
\quad \text{// run a consensus algorithm} \\
\quad \text{return } d
\]
Ψ is necessary to solve QC

Let A be a QC algorithm that uses a failure detector D.

Must show that we can extract Ψ from A and D
Simulating runs of A

Every process periodically samples D and exchanges its FD samples with other processes
=> using these FD samples, the process locally simulates runs of A [CHT96]
Extracting $\Psi$

If there are “enough” simulated runs of A in which non-$Q$ values are decided, then it is possible to extract $(\Omega, \Sigma)$.

Otherwise, it is possible to extract FS.

Processes use the QC algorithm A to agree on which failure detector to extract.
QC: the weakest failure detector

Ψ is the weakest failure detector to solve QC, in any environment
Roadmap

1. Model: asynchronous system with failure detectors
2. Implementing a register
3. Solving consensus
4. Solving QC
5. Solving NBAC
NBAC

A set of processes need to agree on whether to commit or to abort a transaction.

Initially, each process votes Yes ("I want to commit") or No ("We must abort")

Eventually, processes must reach a common decision (Commit or Abort):
- Commit is decided => all processes voted Yes
- Abort is decided => some process voted No or a failure previously occurred
NBAC $\leftrightarrow$ QC + FS

- QC+FS $\Rightarrow$ NBAC:
  given (a) any algorithm for QC and (b) FS, we can solve NBAC

- NBAC $\Rightarrow$ QC:
  Any algorithm for NBAC can be used to solve QC

- NBAC $\Rightarrow$ FS:
  Any algorithm for NBAC can be used to extract FS
NBAC: the weakest failure detector

- NBAC ⇔ QC + FS (in any environment)
- $\Psi$ is the weakest FD to solve QC (in any environment)

Thus,

$(\Psi, FS)$ is the weakest failure detector to solve NBAC, in any environment
The original results

- C. Delporte-Gallet, H. Fauconnier and R. Guerraoui
  Shared memory vs. message-passing

- R. Guerraoui, V. Hadzilacos, P. Kouznetsov and S. Toueg
  The weakest failure detectors for quittable consensus and non-blocking atomic commit
  Technical report, LPD, EPFL, 2004
Thank you!
Quittable consensus (QC)

propose(v) (v in \{0,1\}) returns a value in \{0,1,Q\}
(Q stands for « quit »)

- Agreement: no two processes return different values
- Termination: every correct process eventually returns a value
- Validity: only a value v in \{0,1,Q\} can be returned
  - If v in \{0,1\}, then some process previously proposed v
  - If v=Q, then a failure previously occurred
Emulating $\Sigma$: the reduction algorithm

Periodically (round $k$):

1. $\Pi(k) := \text{set of participants of write $k$ by process } i$
2. $E_i := \{\Pi(j)\} \ j \leq k$
3. write($E_i$) to register $R_i$
4. $E_i := E_i \cup \Pi(k)$
5. send ($k,?$) to all
6. wait until, for every $j$, received ($k,\text{ack}$) from every $X$
7. read in register $R_j$
8. current output of $\Sigma := \text{set of all processes sent}$
   (ack,$k$) $\cup \Pi(k-1)$
Emulating $\Sigma$: the proof intuition

- For any round $k$, process $i$ stores all $P_i(k')$ ($k'<k$) in $R_i$ and includes $P_i(k-1)$ to its emulated set $\Sigma_i$
  
  $\Rightarrow$
  
  Any process $j$ that reads $R_i$ afterwards will include at least one process from $P_i(k-1)$ to its emulated set $\Sigma_j$
  
  $\Rightarrow$
  
  Every two emulated sets intersect

- Eventually, only correct processes send acks
  
  $\Rightarrow$
  
  Eventually, the emulation set includes only correct processes
NBAC

Propose(v) (v in {Yes,No}) returns a value in {Commit,Abort}

- Agreement: no two processes return different values
- Termination: every correct process eventually returns a value
- Validity: a value in {Commit,Abort} is returned
  - If Commit is returned, then every process voted Yes
  - If Abort is returned, then some process voted no or a failure previously occurred
NBAC using QC and FS

send v to all
wait until received all votes or FS outputs red
    \ wait until all votes received or
    \ a failure occurs
if all votes are received and are Yes then
    proposal := 1  \ propose to commit
else
    proposal := 0  \ propose to abort
if QC.Propose(proposal) returns 1 then
    return Commit
else
    return Abort