Institute of Computer Science, University of Bern
Distributed Algorithms, Spring 2020

Prof. Christian Cachin
Jovana Mićić

Solution for exercise 3

3.1 Relations among failure detectors (4pt)

A perfect failure detector abstraction is specified in Module 2.6 [CGR11]. Consider a *non-perfect* failure detector notion with the same interface as in Module 2.6, which only satisfies the following completeness notion:

> **NPFD:** *Weak completeness:* Eventually every process that crashes is permanently detected by *some* correct process.

But it still satisfies the *strong accuracy* property.

Implement a perfect failure detector abstraction from this non-perfect variant. (You should describe an emulation that satisfies all properties of the perfect failure detector; it may use the output from the non-perfect failure detector and send point-to-point messages.)

Solution. The given algorithm transforms any given failure detector that satisfies weak completeness into a failure detector that satisfies strong completeness. Informally it works as follows: upon a crash c detected by the non-perfect failure detector, every process p adds c to *suspects* and sends it to all other processes. Set *suspects* is the set of processes that p suspects according to its local failure detector. When p receives a message from p' with the set *suspects*$'$ of p', then it triggers crashes and adds *suspects*$'$ to *suspects* set.

Implements:
PerfectFD, **instance** *P*.
Uses:
NonPerfectFD, **instance** *NP*.
PerfectP2PLinks, **instance** *pl*.

upon event $\langle$ *P-Init* $\rangle$ **do**
suspects $:= \emptyset$

upon event $\langle$ *NP-crash* $\mid c \rangle$
suspects $:=$ *suspects* $\cup \{c\}$
forall $p \in \pi$ **do**
trigger $\langle$ *pl-Send* $\mid p$, *suspects* $\rangle$

upon event $\langle$ *pl-deliver* $\mid p'$, *suspects*$' \rangle$ **do**
forall $c \in$ (*suspects*$' \setminus$ *suspects*) **do**
trigger $\langle$ *P-crash* $\mid c \rangle$
suspects $:=$ *suspects* $\cup$ *suspects*$'$

Correctness. Consider the *strong completeness* property of a perfect failure detector. If a process c crashes it will be detected by some correct process p according to the non-perfect failure detector (NPFD). Then p will send a message to all processes using perfect links. Remember that perfect links ensure that no message is delivered unless it was sent. Every correct process will thus detect the crash of c.

Consider now the *strong accuracy* property of a perfect failure detector. If a process c is detected by any process, then c has crashed. Because NPFD satisfies *strong accuracy* we are sure that process reported by NPFD really crashed.

3.2 Perfect Failure Detector (4pt)

Assume that a bound Δ on message delay is known and that all local processing after delivering a point-to-point message takes at most Φ time. Algorithm 2.5 [CGR11, p. 51] implements a *perfect failure detector* with request/reply messages. Your task is to:

a) Design an alternative implementation that uses only unidirectional messages (that is, only periodic heartbeats from every process to all others instead of request/reply pairs between every pair of processes).

b) Discuss your algorithm in relation to Algorithm 2.5.

Solution.

Implements:
PerfectFD, **instance** *P*.

Uses:
PerfectP2PLinks, **instance** *pl*.

upon event $\langle$ *P-Init* $\rangle$ **do**
alive $:= \pi$
detected $:= \emptyset$
startTimer1(0)
startTimer2(γ) // γ: interval for checking if all heartbeats have been received
// $\gamma \geq \epsilon + \Delta + \Phi$

upon event $\langle$*Timeout1*$\rangle$ **do**
forall $p \in \pi$ **do**
trigger $\langle$ *pl-Send* $\mid p$, *HB* $\rangle$
startTimer1(ϵ) // ϵ: arbitrary long interval for sending heartbeats

upon event $\langle$*Timeout2*$\rangle$ **do**
forall $p \in \pi$ **do**
if $(p \notin$ *alive*$) \wedge (p \notin$ *detected*$)$ **then**
detected $:=$ *detected* $\cup \{p\}$
trigger $\langle$ *P-Crash* $\mid p \rangle$
alive $:= \emptyset$
startTimer2(γ)

upon event $\langle$ *pl-deliver* $\mid p$, *HB* $\rangle$ **do**
alive $:=$ *alive* $\cup \{p\}$

Discussion:

- Detections by one process in the request/reply-style depend only on the clock of this process; detections in heartbeat failure detector depend on accuracy of the local and the remote clocks.
- If hardware broadcast is available, then the heartbeat failure detector takes only n messages per detection interval; this seems not possible to achieve with the request/reply style, which uses n^2 messages.

3.3 Quorum systems (2pt)

Quorum systems represent a fundamental abstraction for coordination among the nodes of a distributed system.

> **Quorum system:** Let $\Pi = \{p, q, \dots\}$ be a universe with n elements (e.g., processes). A *quorum system* $\mathcal{Q} \subset 2^{\Pi}$ is a set of subsets of Π such that every two subsets intersect. Each $Q \in \mathcal{Q}$ is called a *quorum*.

W.l.o.g., the quorum systems considered here are minimal, i.e., for any $Q, Q' \in \mathcal{Q} : Q \not\subseteq Q'$. Three example quorum systems are:

Singleton: $\mathcal{Q} = \big\{\{p\}\big\}$ for $n = 1$.

Majority: $\mathcal{Q} = \big\{Q \subset \Pi \;\big|\; |Q| = \lceil \frac{n+1}{2} \rceil\big\}$.

Grid: Suppose $n = k \cdot k$ and arrange the processes in a square. Every subset of Π is a quorum if and only if it consists of one full row and one element from each row below the full row.

Suppose a protocol run by processes in Π needs some quorum of *correct* processes to operate, the remaining ones may *fail*. What are the minimum and maximum numbers of faulty processes tolerated by each of the three quorum systems?

Bonus question (+2pt): Suppose a client chooses randomly and with uniform distribution a quorum $Q \in \mathcal{Q}$ and sends a request to all processes in Q. Every $p \in \Pi$ therefore receives a request with a certain probability; this probability has been called the *load* of p under the quorum system. What is the maximal load on any process for each of the three quorum systems?

Solution. The table below shows the minimum and maximum number of faulty processes and the maximum load of any process for each of the three quorum systems.

Quorum system	min faulty	max faulty	max load
Singleton	0	0	1
Majority	0	$\lfloor \frac{n-1}{2} \rfloor$	$\frac{\lceil \frac{n+1}{2} \rceil}{n}$
Grid	0	$n - k$	See below

In the grid system, the number of quorum is

$$|\mathcal{Q}| = k^{k-1} + k^{k-2} + \ldots + k^1 + k^0 = \sum_{i=0}^{k-1} k^i.$$

The maximal load for the grid quorum system therefore becomes

$$load = \frac{\frac{1}{k}(|\mathcal{Q}| - 1) + 1}{|\mathcal{Q}|} = \frac{|\mathcal{Q}| - 1 + k}{k|\mathcal{Q}|}.$$

References

[CGR11] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to reliable and secure distributed programming (Second Edition)*, Springer, 2011.