On the Weakest Failure Detector For Read/Write-Based Mutual Exclusion

Carole DELPORTE[†], Hugues FAUCONNIER[†]
Michel RAYNAL[⋆,◇]

[†]IRIF, Université Paris 7 Diderot, Paris, France
[⋆]IRISA, Université de Rennes, France
[◇]Dept of Computing, Hong Kong Polytechnic Univ

Summary

- Global view of the paper
  - ⋆ Crash failures and failure detectors
  - ⋆ Mutual exclusion
- Technical content
  - ⋆ Basic read/write computing model
  - ⋆ The failure detector QP
  - ⋆ QP-based mutual exclusion
  - ⋆ QP is the weakest FD for mutual exclusion
- Conclusion

Global view

Global view

Computability issue

Mutual exclusion in the presence of process crashes

- Not "Which information on failures allows us to solve mutual exclusion despite process crash failures?"
- But "Which is the weakest information on failures needed to solve read/write mutex despite process crash failures?"

Most famous example: The case of Consensus (1)

- Each process proposes a value and all processes (that do not crash) have to agree on the same value which has to be one of the proposed values
- Impossible to solve in the presence of asynchrony and even a single process crash

Fischer M.J., Lynch N.A. and Paterson M.S., Impossibility of distributed consensus with one faulty process *Journal of the ACM*, 32(2):374-382 (1985)

Loui M. and Abu-Amara H., Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research*, 4:163-183, JAI Press (1987)

Example: The case of Consensus (2)

- The weakest information on failures to solve consensus is the failure detector denoted Ω
  - ⋆ Each process p_i is equipped with a read-only local variable $leader_i$
  - ⋆ There is a finite time after which all processes that do not crash have the same id in $leader_i$, and this id is the one of a non-crashed process

Chandra T., Hadzilacos V. and Toueg S. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685-722 (1996)

Fernández A., Jiménez E., Raynal M., and Trédan G., A timing assumption and two t-resilient protocols for implementing an eventual leader service in asynchronous shared-memory systems. *Algorithmica*, 56(4):550-576 (2010)

A few failure detectors

Crash-prone model	Atomic register	Consensus	Starvation-free mutex
Shared memory	given for free	Ω (1)	Γ^1 (2)
msg-passing with $t < n/2$	$\exists$ algorithms	Ω (1)	T (Trusting) (3)
msg-passing with $t < n$	Quorums Σ (4)	$\Sigma + \Omega$ (4)	$T + \Sigma$ (5)

(1) Chandra T., Hadzilacos V. and Toueg S. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685-722 (1996)

(2) Bhatt V., Christman N., Jayanti P., Extracting quorum failure detectors. *Proc. 28th ACM Symposium on Principles of Distributed Computing (PODC'09)*, ACM Press, pp. 73-82 (2009)

(3) Delporte-Gallet C., Fauconnier H., Guerraoui R., and Kouznetsov P., Mutual exclusion in asynchronous systems with failure detectors. *Journal od Parallel and Distributed Computing*, 65:492-505 (2005)

(4) Delporte-Gallet C., Fauconnier H. and Guerraoui R., Tight failure detection bounds on atomic object implementations. *Journal of the ACM*, 57(4), Article 22, 32 pages (2010)

(5) Bhatt V. and Jayanti P., On the existence of weakest failure detectors for mutual exclusion and k-exclusion. *23rd Int'l Symposium on Distributed Computing (DISC'09)*, Springer LNCS 5805, pp. 325-339 (2009)

Technical content

- Model
  - ⋆ Communication: atomic read/write registers
  - ⋆ Crash-prone asynchronous processes
- Result
  - ⋆ The failure detector QP
  - ⋆ Crash-tolerant mutual exclusion from QP
  - ⋆ Optimality of QP

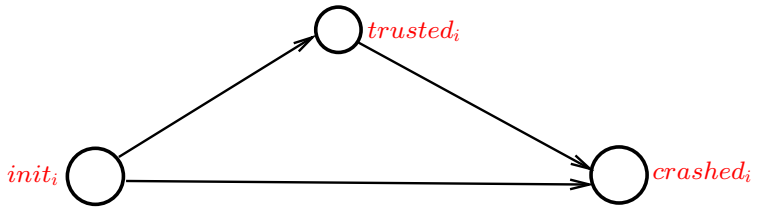
Computing entities

- n asynchronous sequential processes p_1, ..., p_n
- Asynchrony = each process proceeds at its own speed, which can be arbitrary and remains always unknown to the other processes
- Any number of processes may crash (premature halt)
- Terminology: given a run
  a process that crashes is *faulty*, otherwise it is *correct*
- $\mathcal{F}(\tau)$: set of processes crashed at time τ
- $\mathcal{C}$: set of process that do not crash

The failure detector QP

The failure detector QP: automaton

- Three sets: $trusted_i$, $crashed_i$, and $init_i$
- Initially: $trusted_i = crashed_i = \emptyset$, $init_i = \{1, ...n\}$



The failure detector QP: properties (1)

- $\forall\ i, \forall\ \tau$: $trusted_i(\tau) \cap crashed_i(\tau) = \emptyset$

  A process cannot be trusted and crashed at the same time

- $\forall\ i$: $j \in trusted_i(\tau) \Rightarrow$
  $\big(\forall\ \tau' \geq \tau$: $j \in trusted_i(\tau') \cup crashed_i(\tau')\big) \wedge$
  $\big(\forall\ k \in \mathcal{C}$: $\exists \tau'$: $j \in trusted_k(\tau') \cup crashed_k(\tau')\big)$

  A trusted process has to be eventually observed by all correct processes

- $j \in \text{crashed}_i(\tau) \Rightarrow j \in \mathcal{F}(\tau)$

  $crashed_i$ contains only crashed processes

- $j \in crashed_i(\tau) \Rightarrow \big(\forall\ \tau' \geq \tau$: $j \in crashed_i(\tau')\big)$

  Crashes are stable

The failure detector QP: properties (2)

If p_i is correct:

- $j \in \mathcal{F}(\tau) \Rightarrow \big(\exists \tau' \geq \tau\colon\ j \notin trusted_i(\tau')\big)$

  Eventually, no faulty process $\in trusted_i$
- $j \in \mathcal{C} \Rightarrow \big(\exists \tau\colon\ j \in trusted_i(\tau)\big)$

  Eventually, every correct process $\in trusted_i$

QP with respect to P and $\diamond P$

- The perfect failure detector P provides each process p_i with a set $suspected_i$ such that
  - ⋆ no process belongs to $suspected_i$ before it crashes, and
  - ⋆ eventually every process that crashes belongs forever to $suspected_i$
- $\diamond P$ is "eventually P"
- $\prec$: order relation the Computability power of FDs
- $\diamond P \prec QP \prec P$

Crash-tolerant read/write-based mutual exclusion

from QP

Communication and notations

Read/write register model

- Communication: MWMR atomic registers
- Notations
  - ⋆ Capital letters: shared objects
  - ⋆ Small letters: local variables

Crash-tolerant deadlock-free mutual exclusion

- Operations entry() and exit()
- Properties:
  - ⋆ Mutual exclusion: No two processes are simultaneously in their critical section
  - ⋆ Deadlock-freedom: If a correct process p_i has a pending entry() operation and no process is in the critical section, eventually some process p_j (possibly $p_j \neq p_i$) returns from its entry() operation
  - ⋆ Wait-free exit: If a correct process invokes exit(), it returns from its invocation
- If a process crashes while it is in the critical section, it implicitly releases of the critical section

Crash-tolerant mutex algorithm (1)

- Very simple adaptation of Lamport's bakery mutual exclusion algorithm

  Lamport L., A new solution of Dijkstra's concurrent programming problem. *Communications of the ACM*, 17(8):453-455, (1974)

  Taubenfeld G., *Synchronization algorithms and concurrent programming*. Pearson Education/Prentice Hall, 423 pages, ISBN 0-131-97259-6 (2006)

  Raynal M., *Concurrent programming: algorithms, principles, and foundations*. Springer, 515 pages, ISBN 978-3-642-32027-9 (2013)
- Satisfies starvation-freedom

Crash-tolerant mutex algorithm (2)

init $\forall j \in \{1,\ldots,n\} : FLAG[j] \in \{\texttt{down},\texttt{up}\}$, init down;
$\forall j \in \{1,\ldots,n\} : LABEL[j] \in \mathbb{N}$, init 0.

operation entry() **is**
wait $(i \in trusted_i)$;
$FLAG[i] \leftarrow$ up;
$LABEL[i] \leftarrow \max(LABEL[1],\ldots,LABEL[n]) + 1$;
$FLAG[i] \leftarrow$ down;
for all $k \neq i$ **do**
wait$\big((FLAG[k] = \texttt{down}) \vee (k \in crashed_i)\big)$;
wait$\big((LABEL[k] = 0) \vee (LABEL[i], i) < (LABEL[k], k) \vee (k \in crashed_i)\big)$;
end for.

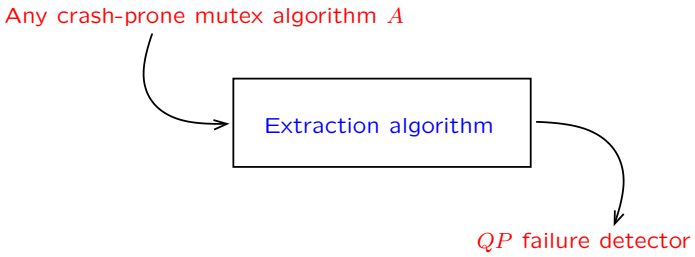
operation exit() **is** $LABEL[i] \leftarrow 0$.

Optimality of QP

weakest FD for read/write-based mutual exclusion

What does mean "weakest FD for mutex"?

Notion of an extraction algorithm



Extraction algorithm E

- Extract means that, at any time, the algorithm E outputs the sets $trusted_i$ and $crashed_i$ at every process p_i, and these sets satisfy the properties defining the failure detector QP
- Extraction algorithm E: uses a set of n mutex (locks) objects $MUTEX[1..n]$, one associated with each p_k
- $MT[k]$ is used to detect the crash of p_k
- Boolean shared $STARTED[1..n]$ init [false, ..., false]
- $STARTED[i]$ is used to capture the progress of p_i
- Outputs of the FD (which is built):
  - $trusted_i$ init $\emptyset$
  - $crashed_i$ init $\emptyset$

Extraction algorithm E at process p_i

run in parallel the tasks $t_1, \ldots, t_n$ defined as follows:
task t_i is
  $MUTEX[i]$.entry();
  $STARTED[i] \leftarrow$ true;
  $trusted_i \leftarrow trusted_i \cup \{i\}$;
  wait(false); % wait forever
  $MUTEX[i]$.exit().

task t_k, $1 \leq k \leq n \wedge k \neq i$, is
  wait($STARTED[k]$);
  $trusted_i \leftarrow trusted_i \cup \{k\}$;
  $MUTEX[k]$.entry(); no-op; $MUTEX[k]$.exit();
  $\langle trusted_i, crashed_i \rangle \leftarrow \langle trusted_i \setminus \{k\}, crashed_i \cup \{k\} \rangle$.

Optimality theorems

- Algorithm E extracts a failure detector QP from any algorithm solving deadlock-free mutual exclusion in any read/write n-process system where any number of processes may crash
- QP is the weakest failure detector for both deadlock-free and starvation-free mutual exclusion in read/write systems where any number of processes may crash
- $\Omega \prec \Diamond P \prec QP \prec P$

Conclusion

- Notion of a failure detector
  weakest information on failures to solve an otherwise impossible problem
- QP is the weakest FD to solve both deadlock-free and starvation-free mutual exclusion in read/write-based system prone to any number of process crashes
- In practice, we can use P because
  - $\star$ QP is very close to P (perfect failure detector)
  - $\star$ P is realistic