On the Hardness of Failure-Sensitive Agreement Problems

Rachid Guerraoui
Swiss Federal Institute of Technology,
CH-1015 Lausanne

Abstract

In [1],[3] and [6], respectively, it was stated that the weakest failure detector for any of non-blocking atomic commit, terminating reliable broadcast and leader election, is the *Perfect* failure detector $\mathcal{P}$. This paper presents a counter example of those results. We exhibit a failure detector that is incomparable to $\mathcal{P}$, and yet solves those problems.

Keywords: Distributed computing, fault-tolerance, failure detector, agreement, commit, election, broadcast.

Approximate word number : 3000.

1 Introduction

Non-blocking atomic commit, terminating reliable broadcast and *leader election* are three fundamental *agreement* problems in reliable distributed computing. This paper discusses the solvability of these problems in distributed systems where channels are reliable, processes can fail by crashing, and process failures can be detected using failure detectors.

Failure-sensitive agreement. In non-blocking atomic commit, the processes must *agree* on the outcome of distributed transactions: *commit* or *abort*. The outcome depends on the votes of the processes: *yes* or *no* [5]. In terminating reliable broadcast, the processes need to *agree* on whether to deliver a message broadcast by some specific *sender* process, or to deliver a default message [4]. In leader election, the processes must elect a leader and make sure to avoid any *disagreement* about which process is leader at any given time [6]. Besides the fact that those three problems are all *agreement* problems, they also have a common *"failure-sensitive"* flavor: in each of those problems, the decision value depends somehow on the failure pattern (i.e., on the fact that some processes have crashed or not): [1]

- In atomic commit, the processes must decide *commit* if all processes vote *yes* and *no process crashes.*
- In terminating reliable broadcast, the processes must deliver the message broadcast by the specific sender if *the sender does not crash.*
- In leader election, a new leader must be elected if *the current leader crashes.*

Background. In [1],[3] and [6], respectively, it was stated that the weakest failure detector for any of non-blocking atomic commit, terminating reliable broadcast and leader election, is the *Perfect* failure detector $\mathcal{P}$ [1]. Failure detector $\mathcal{P}$ ensures that (a) eventually, every correct process permanently suspects every crashed process, and (b) no process is suspected before it crashes. Hence, according to [1], [3] and [6], to solve any of those "failure-sensitive" agreement problems, perfect knowledge about failures is *sufficient* and *necessary.* More precisely: (1) one can devise non-blocking atomic commit, terminating reliable broadcast and leader election protocols, using the failure detector $\mathcal{P}$, and (2) if any failure detector $\mathcal{D}$ solves any of those problems, then there is an algorithm that transforms $\mathcal{D}$ into $\mathcal{P}$, i.e., $\mathcal{D}$ is at least as *strong* as $\mathcal{P}$ ($\mathcal{D} \succeq \mathcal{P}$).

A counter example. This paper contradicts those results through a simple counter example. We show that $\mathcal{P}$ is not the weakest failure detector to solve any of non-blocking atomic commit, terminating reliable broadcast or leader election. We exhibit a simple failure detector, denoted by $\mathcal{M}$ (the *Marabout* failure detector), and we show that (1) $\mathcal{M}$ cannot be transformed to $\mathcal{P}$, and (2) $\mathcal{M}$ is sufficient to solve those problems. Intuitively, $\mathcal{M}$ is accurate about the future whereas $\mathcal{P}$ is accurate about the past. The two failure detectors are actually incomparable.

It is important to notice that the aim here is not introduce any meaningful failure detector nor any useful agreement protocol. The objective is rather to point out the difficuly of identifying the weakest failure detector for "failure-sensitive" problems like non-blocking atomic commit, terminating reliable broadcast and leader election. It is also worth noticing that we do not actually contradict the proofs of [6]. We rather point out the fact that [6] shows that $\mathcal{P}$ is the weakest failure detector to solve leader election, among a *subset* of the possible failure detectors (in the original sense of [1]).

Roadmap. We consider an asynchronous computation model augmented with the failure detector abstraction [1, 2]. Basically, we assume a distributed system composed of a finite set of n processes $\Omega = \{p_1, p_2, \ldots, p_n\}$ ($|\Omega| = n > 1$). Every pair of processes is connected by a reliable communication channel. Processes execute deterministic algorithms and can fail by crashing. A discrete global

[1] Note that in other agreement problems like *consensus*, the decision value must simply be a value proposed by some process [1] (no matter what the failure pattern is).

clock is assumed, and Φ, the range of the clock's ticks, is the set of natural numbers. The global clock is used for presentation simplicity and is not accessible to the processes. The reader interested in specific details about the original failure detector model should consult [2]. The rest of the paper is organized as follows. Section 2 recalls the definition of the perfect failure detector $\mathcal{P}$ and introduces the *Marabout* failure detector $\mathcal{M}$. We show here that $\mathcal{P}$ and $\mathcal{M}$ are incomparable. Section 3 shows that $\mathcal{M}$ solves terminating reliable broadcast, non-blocking atomic commit and leader election. Section 4 concludes the paper with some general remarks.

2 Perfect detection vs. perfect prediction

Among the failure detectors introduced in [1], the strongest is the *Perfect* failure detector $\mathcal{P}$.[2] Each module of $\mathcal{P}$ outputs a subset of the processes in Ω that are *suspected* (to have crashed), i.e., $R_{\mathcal{P}} = 2^{\Omega}$. For every failure pattern F, $\mathcal{P}(F)$ is the set of histories H such that the following two properties are satisfied:

1. *Strong Completenes:* Eventually, every faulty process is permanently suspected by every correct process. More precisely:
   - $\exists t \in \Phi, \forall p_i \in crashed(F), \forall p_j \in correct(F), \forall t' \geq t : p_i \in H(p_j, t')$.
2. *Strong Accuracy:* No process is suspected before it crashes. More precisely:
   - $\forall t \in \Phi, \forall p_i, p_j \in \Omega - F(t) : p_i \notin H(p_j, t)$.

We introduce here the *Marabout* failure detector, denoted by $\mathcal{M}$. Roughly speaking, $\mathcal{M}$ *predicts* the crashes of the processes in an accurate manner, but does not say when the crashes will actually occur. Each module of $\mathcal{M}$ outputs a subset of the processes in Ω that are suspected to (*eventually*) crash, i.e., $R_{\mathcal{M}} = 2^{\Omega}$. For every failure pattern F, $\mathcal{M}(F)$ is the set of histories H such that the following two properties are satisfied:

1. *Perpetual Completenes:* Every faulty process is permanently suspected by every correct process. More precisely:
   - $\forall t \in \Phi, \forall p_i \in crashed(F), \forall p_j \in correct(F) : p_i \in H(q, t)$.
2. *Perpetual Accuracy:* No process is suspected unless it crashes. More precisely:
   - $\forall t \in \Phi, \forall p_i, p_j \in correct(F) : p_i \notin H(p_j, t)$.

[2] For presentation simplicity, but without loss of generality, we do not distinguish here the *Perfect* failure detector $\mathcal{P}$ from the *class* of *Perfect* failure detectors $\mathcal{P}$ [1].

In the following, we show that failure detectors $\mathcal{P}$ and $\mathcal{M}$ are incomparable. Intuitively, $\mathcal{P}$ provides perfect failure detection: it outputs accurate information about past crashes. In contrast, $\mathcal{M}$ provides perfect failure prediction: it outputs accurate information about future crashes. These are incomparable kinds of knowledge about failures.

Lemma 2.1 **($\mathcal{P} \not\succeq \mathcal{M}$)** *No algorithm can transform $\mathcal{P}$ into $\mathcal{M}$.*

PROOF: The proof is by contradiction. We assume that there is an algorithm $A_{\mathcal{P}\to\mathcal{M}}$ that transforms failure detector $\mathcal{P}$ into $\mathcal{M}$. We then show that if $A_{\mathcal{P}\to\mathcal{M}}$ can transform that failure detector into some failure detector that satisfies *Perpetual Completeness*, then this failure detector cannot satisfy *Perpetual Accuracy*.

We denote by *output*$(\mathcal{M})$ the variable that $A_{\mathcal{P}\to\mathcal{M}}$ uses to emulate failure detector $\mathcal{M}$; *output*$(\mathcal{M}, t)_{p_i}$ denotes the value of that variable at a given time t and process p_i. Let p_1 and p_2 be any two processes in Ω (remember that we assume $|\Omega| > 1$). Let F be the failure pattern where all processes are correct, except p_1 which crashes at time $t = 2000$. Let $R_1 =< F, H, C, S, T >$ be any partial run of $A_{\mathcal{P}\to\mathcal{M}}$ where $T[|T|] < 2000$. Since $\mathcal{M}$ (i.e., *output*$(\mathcal{M})$) satisfies *Perpetual Completeness*, we have: *output*$(\mathcal{M}, T[|T|])_{p_2} = \{p_1\}$. Let F' be the failure-free pattern. Partial run $R_2 =< F', H, C, S, T >$ is also a partial run of $A_{\mathcal{P}\to\mathcal{M}}$ because: (1) $|S| = |T|$, (2) S is applicable to C, and (3) for all $k \leq |S|$ where $S[k] = (p_i, m, d, A)$, since $p_i \notin F(T[k])$ and $d = H(p_i, T[k])$, we have $p_i \notin F'(T[k])$ and $d = H(p_i, T[k])$ (no process crashes in F'). Since R_1 and R_2 have the same schedule S, and we assume deterministic algorithms, we also have *output*$(\mathcal{M}, T[|T|])_{p_2} = \{p_1\}$ in R_2, in contradiction with the *Perpetual Accuracy* property (since no process crashes in F', we should have had *output*$(\mathcal{M}, T[|T|])_{p_2} = \emptyset$). $\Box$

Lemma 2.2 **($\mathcal{M} \not\succeq \mathcal{P}$)** *No algorithm can transform $\mathcal{M}$ into $\mathcal{P}$.*

PROOF: The proof is similar to the proof of Lemma 2.1 above. It is also by contradiction. Assume that there is an algorithm $A_{\mathcal{M}\to\mathcal{P}}$ that transforms $\mathcal{M}$ into $\mathcal{P}$. We denote by *output*$(\mathcal{P})$ the variable used to emulate failure detector $\mathcal{P}$; *output*$(\mathcal{P}, t)_{p_i}$ denotes the value of that variable at a given time t and process p_i. Let p_1 and p_2 be any two processes in Ω. Let F be the failure pattern where p_1 crashes at time 0 and all other processes are correct. At any time t, failure detector $\mathcal{M}$ outputs $\{p_1\}$ at process p_2.

Assume that *output*$(\mathcal{P})$ satisfies *Strong Completeness*. There is a partial run of A, $R_1 =< F, H, C, S, T >$ such that *output*$(\mathcal{P}, T[|T|])_{p_2} = \{p_1\}$. Consider now failure pattern F' where p_1 crashes at time $T[|T|] + 1$ and all other processes are correct. Since $\mathcal{M}$ outputs exactly the same values in F and in F', i.e., $\{p_1\}$, then $R_2 =< F', H, C, S, T >$ is also a partial run of A: (1) $|S| = |T|$, (2) S is applicable to C, and (3) for all $k \leq |S|$ where $S[k] = (p_i, m, d, A)$, since

$p_i \notin F(T[k])$ and $d = H(p_i, T[k])$, we have $p_i \notin F'(T[k])$ and $d = H(p_i, T[k])$ (every process that crashes in F' crashes in F). Since we assume deterministic algorithms, in R_2 we have: $output(\mathcal{P}, T[|T|])_{p_2} = \{p_1\}$. In other words, p_2 suspects p_1 before it crashes and hence violates *Strong Accuracy.* $\square$

From Lemma 2.1 and Lemma 2.2 we have the following proposition.

Proposition 2.3 **($\mathcal{M} \not\sim \mathcal{P}$)** *$\mathcal{P}$ and $\mathcal{M}$ are incomparable.*

3 On the use of Marabouts

We show below that failure detector $\mathcal{M}$ can solve non-blocking atomic commit, terminating reliable broadcast and leader election. The combination of these results and Lemma 2.2 contradicts the results of [1],[3] and [6]. In other words, $\mathcal{P}$ is actually not the weakest failure detector for any of non-blocking atomic commit, terminating reliable broadcast and leader election.

3.1 Non-blocking atomic commit

The atomic commit problem consists for the processes to *decide* on an outcome value *commit* or *abort*. The outcome value depends on *votes (yes* or *no) proposed* by the processes. Every process proposes exactly one value. We consider the *non-blocking* version of the problem, in which every correct process eventually decides even if some processes have crashed [5]. The non-blocking atomic commit problem is specified by the following properties:

- **Agreement:** No two processes decide differently.
- **Termination:** Every correct process eventually decides.
- **Abort-validity:** *Abort* is the only possible decision if some process votes *no.*
- **Commit-validity:** *Commit* is the only possible decision if every process votes *yes* and no process crashes.

In [3], it is stated that $\mathcal{P}$ is the weakest failure detector for non-blocking atomic commit. We show below that the statement is actually wrong because $\mathcal{M}$ and $\mathcal{P}$ are incomparable, yet $\mathcal{M}$ solves non-blocking atomic commit.

Proposition 3.1 *$\mathcal{M}$ solves non-blocking atomic commit.*

PROOF (SKETCH). We give here a brief description of an algorithm that solves non-blocking atomic commit with $\mathcal{M}$. The basic idea of the algorithm is the following. Every process p_i first consults its failure detector module $\mathcal{M}_{p_i}$. If p_i predicts the crash of any process (i.e., the output of $\mathcal{M}_{p_i}$ is not empty), then p_i

immediately decides *abort*. Process p_i can safely do so thanks to the *Perpetual Accuracy* property of $\mathcal{M}$: no process predicts a crash unless some process is faulty. Otherwise, if p_i does not predict the crash of any process (i.e., $\mathcal{M}_{p_i}$ outputs $\emptyset$), p_i sends its vote to all processes (including itself) and waits for the votes of all processes. Process p_i can safely wait (i.e., p_i does not risk to indefinitly block) thanks to the *Perpetual Completeness* property of $\mathcal{M}$. If some process crashes, $\mathcal{M}_{p_i}$ would have already output a value that is different from $\emptyset$ and p_i would have decided *abort*. By the assumption of reliable channels, if p_i is correct, p_i will eventually receive all votes from all processes. If p_i receives a *no* vote, it decides *abort*. Otherwise, if p_i receives *yes* votes from all, p_i decides *commit*.

This protocol satisfies all properties of non-blocking atomic commit, thanks to *Perpetual Accuracy* and *Perpetual Completeness* properties of $\mathcal{M}$, together with the assumption of reliable channels. □

3.2 Terminating reliable broadcast

In the *terminating reliable broadcast (TRB)* problem, a distinguished *sender* process, noted p_i, is supposed to broadcast a message m from a set M of possible messages: we note $sender(m) = p_i$. All processes are supposed to deliver, either that message m, or a message $F_i \notin M$ (F_i states that the sender p_i is faulty) [4]. Terminating reliable broadcast is similar to *reliable broadcast*, except that TRB requires that every correct process always deliver a message (even if the sender crashes before broadcasting a message).[3] More precisely, given a sender process p_i, and a message m, TRB$_i$ is defined by the following properties:

- **Agreement:** No two correct processes deliver two different messages.
- **Validity:** If p_i is correct and p_i broadcasts a message m, then p_i delivers m.
- **Termination:** Every correct process eventually delivers exactly one message.
- **Integrity:** If a correct process delivers a message m then $sender(m) = p_i$, and if $m \neq F_i$, then m was previously broadcast by p_i.

We call terminating reliable broadcast here the problem that gathers multiple instances of TRB_i: one for every process $p_i \in \Omega$. It was stated in [1] that the weakest failure detector to solve terminating reliable broadcast is $\mathcal{P}$. We show below that the statement of [1] is not accurate because failure detector $\mathcal{M}$ solves terminating reliable broadcast.

[3]The problem is also similar to the well-known "Byzantine Generals' Problem". The main difference has to do with the model within which the problems are defined. Terminating reliable broadcast is defined in a system model where processes fail (only) by crashing, whereas the Byzantine Generals' problem is defined in a system model where processes can fail by behaving malicously.

Proposition 3.2 *$\mathcal{M}$ solves terminating reliable broadcast.*

PROOF (SKETCH): We give here a brief description of an algorithm that solves terminating reliable broadcast with $\mathcal{M}$. The algorithm is very similar to the non-blocking atomic commit algorithm we sketched above. Every process p_i consults its failure detector module. If p_i predicts the crash of any process p_j, then p_i delivers F_j. If p_i does not predict the crash of some process p_k, then p_i simply waits for p_k's message. By the *Perpetual Accuracy* property of $\mathcal{M}$, p_i only delivers F_j if p_j indeed crashes: hence the validity property of TRB_j. Let p_k be any other process that delivers some message for TRB_j. By the *Perpetual Completeness* property of $\mathcal{M}$, p_k does necessarily predict the crash of p_j and delivers F_j. This implies the agreement property of TRB_j. The termination property follows from the assumption of reliable channels and the *Perpetual Completeness* property of $\mathcal{M}$. Finally, the integrity property follows from the assumption of reliable channels. □

3.3 Leader election

In the leader election problem (in the sense of [6]), at any time, at most one process considers itself the *leader* and if a leader crashes, a new leader must eventually. We show that *leader election* does not require a *Perfect* failure detector.

To precisely capture the notion of leadership, we assume that every process has a local copy of a distributed variable, denoted by *leader*. The copy of *leader* at a process p_i is denoted by $leader_{p_i}$ and for any process p_i, $leader_{p_i} \in \{true, false\}$.

We say that a process p_i is leader at a time t if p_i has not crashed by time t and $leader_{p_i} = true$. We define the *leader election* problem with the two following properties:

- **Agreement:** No two processes can be leader at the same time.
- **Termination:** At any time, there is eventually a leader.

Proposition 3.3 *$\mathcal{M}$ solves leader election.*

PROOF (SKETCH): We give here a brief description of a protocol that solves leader election with $\mathcal{M}$. Initially, $leader_{p_i}$ is assigned to *false* at every process p_i. Every process p_i consults its failure detector module and considers the set E of processes that p_i does not predict (to crash). If p_i is in E and p_i is the process with the lowest i within the processes of E, then p_i assigns $leader_{p_i}$ to *true* (i.e., p_i elects itself the leader). By the *Perpetual Completeness* and the *Perpetual Accuracy* properties of $\mathcal{M}$, for any given failure pattern, every failure detector outputs exactly the same set of processes at all times and all processes.

Hence the agreement property of leader election. By the *Perpetual Accuracy* property, a process does not elect itself leader unless it makes sure it will never crash, which ensures the termination property of leader election. □

4 Remarks

On proofs and assumptions. This paper contradicts the results of [1], [3] and [6]. It is thus legitimate to wonder whether we actually contradict the corresponding proofs or the assumptions underlying those proofs.

In [1], it is stated but not proved that the weakest failure detector for terminating reliable broadcast is $\mathcal{P}$. In [3], the authors present a proof to show that the weakest failure detector to solve terminating reliable broadcast is $\mathcal{P}$. In particular, the authors describe how to emulate $\mathcal{P}$ using a sequence of executions of terminating reliable broadcast. A closer look at the proof reveals however that what is emulated is not actually $\mathcal{P}$, but some failure detector which ensures the *Strong Completeness* property of $\mathcal{P}$, and the following accuracy property: *no process is suspected unless it is faulty.* This accuracy property is different from the actual *Strong Accuracy* property of $\mathcal{P}$ (no process is suspected *before* it crashes). The authors of [3] also describe how to emulate $\mathcal{P}$ using a sequence of executions of non-blocking atomic commit. Besides the fact that what is emulated is not precisely $\mathcal{P}$, the authors assume here that the problem is solvable among every pair of two processes - which is different from assuming (only) that the problem is solvable among a set of arbitrary n processes. Finally, in [6], the authors prove that the weakest failure detector for leader election is $\mathcal{P}$. In fact, the authors made a set of assumptions that restrict the space of possible failure detectors. Hence, $\mathcal{P}$ is shown to be the weakest failure detector for leader election among a *subset* of "uniform" failure detectors, and not among all failure detectors in the original sense of [1]. It is easy to see that our failure detector $\mathcal{M}$ does not belong to that subset.

On boggus failure detectors. To prove our results, we introduced the *Marabout* failure detector $\mathcal{M}$. Obviously, $\mathcal{M}$ is a "boggus" failure detector. It cannot be implemented even in a completely synchronous system, i.e., $\mathcal{M}$ does not actually encapsulate the synchrony of the system as "good" failure detectors should do. As we pointed out in the introduction, our aim was not to introduce meaningful failure detectors and useful agreement algorithms, but rather to describe a simple counter example for [1], [3] and [6].

One might claim that by excluding failure detectors that predict the future,[4] we could circumvent the counter example of this paper and indeed state that $\mathcal{P}$ is the weakest failure detector to solve non-blocking atomic commit, terminating reliable broadcast and leader election. This is not straightforward neither. Other counter examples might be considered. Imagine for instance a failure detector that outputs lists of *trusted* processes, such that, for every failure pattern, exactly

[4]This can be done by restricting the actual definition of the notion of failure detector in [1].

one correct process permanently *trusts* itself. This failure detector cannot be transformed into the *Perfect* failure detector $\mathcal{P}$, it does not predict the future, yet it solves leader election (every process that trusts itself elects itself the leader).

References

[1] T. Chandra and S. Toueg. *Unreliable Failure Detectors for Reliable Distributed Systems.* Journal of the ACM, 43(2), March 1996.

[2] T. Chandra, V. Hadzilacos and S. Toueg. *The Weakest Failure Detector for Solving Consensus.* Journal of the ACM, 43(4), July 1996.

[3] E. Fromentin, M. Raynal, and F. Tronel. *About Classes of Problems in Asynchronous Distributed Systems with Process Crashes*. Technical Report IRISA 1178. Also in proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS), 1999.

[4] V. Hadzilacos and S. Toueg. *Fault-Tolerant Broadcasts and Related Problems*. Cornell University, Technical Report (TR 94-1425), 1994. Also in *Distributed Systems*, S. Mullender (ed), Addison-Wesley, 1993.

[5] D. Skeen. *NonBlocking Commit Protocols*. In proceedings of the ACM SIGMOD International Conference on Management of Data, pages 133-142, ACM Press, 1981.

[6] L. Sabel and K. Marzullo. *Election Vs. Consensus in Asynchronous Systems*. Technical Report TR95-1488, Cornell Univ, 1995. Also, Technical Report CS95-411, UCSD, 1995.