The Weakest Failure Detector for Set-Agreement in Message-Passing Networks

Carole Delporte-Gallet, Hugues Fauconnier, Andreas Tielmann

The Weakest Failure Detector for Set-Agreement in Message-Passing Networks

Carole Delporte-Gallet, Hugues Fauconnier, Andreas Tielmann*

LIAFA University of Paris 7 - Denis Diderot

Technical Report

February 28, 2008

Abstract

Reaching agreement is one of the most fundamental problems in distributed computing. In the set-agreement problem, n processes try to agree on at most $n-1$ different values. This paper determines the weakest failure detector for set-agreement in message-passing networks where processes may fail by crashing. The failure detector is called *weak*$-\mathcal{FS}$ and it returns at every invocation *"go"* or *"wait"*. It ensures that (1) there is at least one process where the output is always *"wait"*, and (2) if there is only one correct process, then the output at this process is eventually *"go"*.

Keywords: set-agreement, failure detectors

1 Introduction

In the set-agreement problem [4], n processes try to agree on at most $n-1$ different values. It has been shown that set-agreement is impossible to be implemented wait-free in purely asynchronous systems where processes can fail by crashing [12, 1, 10]. This has lead to many attempts to find the weakest failure detector[1] for set-agreement [11, 9, 5]. Recently, Zieliński proved that anti-Ω – a failure detector that outputs id's of processes and

*Work was supported by grants from Région Ile-de-France.

[1] i.e. a primitive that enriches the system and provides the processes with information about failures that occure during an execution [3].

the id of at least one correct process only finitely many times – is the weakest failure detector for set-agreement in systems with hardware registers [13]. Furthermore, Zieliński conjectured that failure detector Σ [6] – the weakest failure detector to simulate registers – is both, sufficient and necessary to implement set-agreement. However, Delporte et al. have shown that while Σ is sufficient, it is not necessary [7]. They present a failure detector σ that is strictly weaker than Σ and still sufficient to implement set-agreement.

In this paper, we present failure detector *weak*–$\mathcal{FS}$ and show that it is the weakest failure detector for set-agreement in message passing systems with any number of faults. It returns at every invocation at every process *"go"* or *"wait"* and it ensures that (1) there is at least one process where the output is always *"wait"*, and (2) if there is only one correct process, then the output at this process is eventually *"go"* forever.

To prove our result, we first define our model in Section 2 and then follow the approach of Chandra et al. [2] and first show that *weak*–$\mathcal{FS}$ is sufficient for set-agreement in Section 3 and then show that it is also necessary in Section 4. One remarkable point about our proof is its simplicity. Especially compared to the rather large and involved proof of Zieliński [13] in shared memory systems, it shows that – contrary to a wide belief – results in message passing systems are sometimes easier to prove.

2 Model and definitions

2.1 Processes and failure detectors

Our system consists of a set $\Pi = \{p_1, \ldots, p_n\}$ of $n \geq 2$ processes. These processes communicate totally asynchronously by message passing over a fully connected network with reliable links and any number of processes may fail by prematurely halting, i.e. they crash. It is the system of Chandra et al. [2] which we shortly recall here. For easier reasoning about processes, we assume that there is a global clock $\mathcal{T}$. Nevertheless, this clock cannot be accessed by the processes.

We model the crash failures by the concept of failure patterns which we denote by $\mathcal{F}$. A failure pattern is a function from time $\mathcal{T}$ to 2^{Π} that specifies for every time t which processes have crashed until time t. A process p_i that does not crash in a failure pattern $\mathcal{F}$ is said to be correct ($p_i \in \mathit{correct}(\mathcal{F})$). Processes that are not correct are called faulty. An environment $\mathcal{E}$ is a set of possible failure patterns. We allow every environment, i.e. any number of processes may crash.

A failure detector $\mathcal{D}$ is a distributed oracle that provides the processes with information about failures. For every failure pattern $\mathcal{F} \in \mathcal{E}$, $\mathcal{D}(\mathcal{F})$ is a set of failure detector histories that are allowed for $\mathcal{F}$. A failure detector history H is a function from $\Pi \times \mathcal{T}$ to $\mathcal{R}_{\mathcal{D}}$, the failure detector range of $\mathcal{D}$, i.e. the set of possible outputs.

Following Chandra et al. [2], we define a weakest failure detector for a certain problem in a given environment to be a failure detector that is sufficient to implement the problem in this environment and that is also necessary to implement the problem, i.e. any other failure detector that is sufficient can simulate it in this environment.

We model an algorithm A as a set of n deterministic automata, one for every process in the system. A run of A proceeds in steps and at every time t at most one process executes a step. We assume only fair runs, i.e. every correct process executes infinitely many steps. A step consists of receiving a (possibly empty) message, reading a value of a failure detector, changing the state accordingly, and outputting a (possibly empty) message.

2.2 Set-Agreement

The problem of set-agreement consists for every process p_i with some proposal value v_i to decide a value and to satisfy the following three properties:

Agreement: At most $n - 1$ different values are decided.

Validity: Every value that has been decided must have been a proposal value of some process.

Termination: Eventually, every correct process decides a value.

2.3 Failure detector *weak*–$\mathcal{FS}$

We now define failure detector *weak*–$\mathcal{FS}$ (see [8] for a defintion of failure detector $\mathcal{FS}$). The failure detector outputs one of the two values *"wait"* and *"go"*. The intuition behind this is that if the output at some process is *"wait"*, then there is another process alive, i.e. it makes sense to wait for messages of other processes. To be usefull for set-agreement, we demand that

- at least one process has always output *"wait"* (nevertheless, it might crash), and
- if only one process is correct, then its failure detector output should eventually be *"go"* forever.

By convention, we assume that if a process has crashed, its failure detector output is *"wait"* forever. More formally:

Definition 1. *The range of weak–$\mathcal{FS}$ is {"wait", "go"}. For every environment $\mathcal{E}$, for every failure pattern $\mathcal{F} \in \mathcal{E}$, and every history $H \in$ weak–$\mathcal{FS}(\mathcal{F})$:*

$$\exists p_i \in \Pi, \forall t, H(p_i, t) \neq \text{“go”} \quad (1)$$

$$\wedge \; \mathit{correct}(\mathcal{F}) = \{p_i\} \Rightarrow \exists t, \forall t' \geq t, H(p_i, t') = \text{“go”} \quad (2)$$

3 The sufficient part

To show that failure detector *weak*–$\mathcal{FS}$ is sufficient to solve set-agreement in our model, we give an algorithm that implements set-agreement with *weak*–$\mathcal{FS}$ in Figure 1. For simplicity of the presentation, we assume that a process does not react on interrupts (of a new message or a failure detector change) while it processes another interrupt.

To ensure that at most $n - 1$ proposal values are decided, every process tries to agree with another process on one value. To achieve this, initially some processes send their values. To prevent a circular value exchange, i.e. a situation where the proposal values are simply permuted, the values are only sent to processes with a higher id. This means, that process p_1 sends its value to everybody, process p_i to all processes from p_{i+1} to p_n, and process p_n to nobody.

If some process receives one of these values, it sends a special message *decided* with its decision value and decides. In this way, as long as there is another correct process, every correct process decides either due to one of the messages that was initially sent or, if it does not receive such a message (e.g., because it has a lower id than the other correct processes), it decides due to a *decided* messages of one of this other processes. Note that it may be possible that a process receives its initial value back in a *decided* message, but if so, the sender of the *decided* message does not decide its own proposal value.

To deal with crashes, we only execute these steps if the output of our failure detector is *"wait"*. But in the case of only one correct process in the system, we do not want to wait for messages of other processes forever. Therefore, if the output of the failure detector changes to *"go"* – and by its property (2) it will in the case of only one correct process eventually do so – we simply decide our own proposal value. We can do this without violating agreement, because by property (1) there will always be one process that does not decide due to a *"go"* output, and as we have argued before, processes that decide due to a message exchange eliminate at least one value.

Algorithm for process p_i:

to propose(v)**:**

  initially:

    send $\langle v \rangle$ to all p_j with $j > i$;

  on receive $\langle v' \rangle$ or $\langle \mathit{decided}, v' \rangle$ **do:**

    send $\langle \mathit{decided}, v' \rangle$ to all;

    decide v'; halt; (∗ decision D1 ∗)

  on $\mathit{weak}{-}\mathcal{FS}$ = *"go"* **do:**

    send $\langle \mathit{decided}, v \rangle$ to all;

    decide v; halt; (∗ decision D2 ∗)

Figure 1: Implementing set-agreement with $\mathit{weak}{-}\mathcal{FS}$.

Theorem 1. *The algorithm in Figure 1 implements set-agreement in every environment* $\mathcal{E}$*.*

Proof. We first prove the agreement property of set-agreement. We assume a run where all processes decide and every processes p_i has a distinct initial value v_i. Without this assumption, agreement is trivially met.

By property (1) of the definition of $\mathit{weak}{-}\mathcal{FS}$, not all processes can have decided by decision D2. This means, that it is sufficient to show that if at least one process decides by D1, then at most $n-1$ values are decided.

If some process p_i decides by D1, then it either decides due to a message $\langle \mathit{decided}, v' \rangle$ of another process or due to a message $\langle v' \rangle$ sent initially by another process. We distinguish between the two cases where p_i decides v_i, and where it does not.

Case 1: The only possibility that the decided value v' is equal to p_i's value v_i is that a process p_j with $j > i$ has received p_i's initial message and decided v_i. Therefore, p_i and p_j decide the same value and at most $n-1$ values are decided.

Case 2: If v' is not equal to v_i, then the only possibility that v_i is decided is if a process p_k with $k > i$ has received v_i from p_i. If so, then in an analogous manner, the only possibility that v_k is decided is if another process with a higher id has received it. Since process p_n does not send its value to anybody, this recursion eventually stops and at least one value is never decided.

Validity is trivially satisfied, since only proposal values are sent.

To show termination, we again distinguish two cases: the case when there exist at least two correct processes in a run with a failure pattern $\mathcal{F} \in \mathcal{E}$, and when this is not the case.

Case 1: If there are at least 2 correct processes, then eventually, the one with the higher id receives the message of the other one, sends the *decided* message and decides. All processes that have not yet decided eventually receive this *decided* message and also decide.

Case 2: If there is only 1 correct process, then by property (2) of *weak*–$\mathcal{FS}$, this process eventually decides by decision D2.

□

4 The necessary part

Following the approach of Chandra et al. [2], we show that failure detector *weak*–$\mathcal{FS}$ is necessary to solve set-agreement in our model by providing an algorithm that emulates the output of *weak*–$\mathcal{FS}$ given any algorithm A and failure detector $\mathcal{D}$, such that A using $\mathcal{D}$ solves set-agreement. Figure 2 presents such an algorithm.

The idea for the emulation of *weak*–$\mathcal{FS}$ is that if all messages that are sent by algorithm A get delayed for a very long time, the safety properties of set-agreement still have to hold, while for the case that only one process is correct, even the liveness properties have to hold, i.e. the algorithm has to terminate. Therefore, every process executes A with $\mathcal{D}$, omits to send any messages to other processes that are generated by algorithm A, and outputs *"wait"* until A terminates. In this way, property (1) of *weak*–$\mathcal{FS}$ is always fulfilled, because otherwise the executions at all processes would have terminated without ever receiving a message and therefore agreement could not have been guaranteed. But nevertheless, if there is only one correct process p_i, the algorithm A executed at p_i has to terminate and property (2) of *weak*–$\mathcal{FS}$ is also guaranteed.

The output of our emulation of *weak*–$\mathcal{FS}$ is provided through a special variable *output*.

Algorithm for process p_i:

output := *"wait"*;
execute A using $\mathcal{D}$ with value i, but omit sending messages to others;
if A has terminated, then *output* := *"go"*;

Figure 2: Implementing *weak–*$\mathcal{FS}$ with an algorithm A and a failure detector $\mathcal{D}$ that solve set-agreement.

Theorem 2. *The algorithm in Figure 2 implements weak–*$\mathcal{FS}$ *in every environment* $\mathcal{E}$.

Proof. Assume there exists a run r, where the algorithm in Figure 2 does not fulfill property (1) of *weak–*$\mathcal{FS}$ with a failure pattern $\mathcal{F} \in \mathcal{E}$. This means, that in run r, for every process, there exists a time when *output* = *"go"*, i.e. the execution of algorithm A has terminated at all processes without receiving any message from other processes at all.

Let t be the time when A has terminated at all processes in run r. Then, it is possible to construct a valid run r' of A with the same failure pattern $\mathcal{F}$, where all messages to other processes get delayed to a time after t, and all processes have terminated A at time t. Since A fulfills the validity property of set-agreement and failure detector $\mathcal{D}$ is not allowed to output information about the state of other processes, the decision value at every process p_i can only be its proposal value i. A contradiction to the agreement property of set-agreement! Therefore, property (1) of *weak–*$\mathcal{FS}$ is always satisfied.

If *correct*$(\mathcal{F}) = \{p_i\}$ for a failure pattern $\mathcal{F} \in \mathcal{E}$, then it is possible to construct a run where no faulty process is able to send a message and therefore eventually, by the termination property of set-agreement, algorithm A has to terminate at p_i and the output changes to *"go"*. Therefore, property (2) is also satisfied. □

Corollary 1. *weak–*$\mathcal{FS}$ *is the weakest failure detector for set-agreement in message passing systems in all environments.*

Proof. We have shown in Theorem 1 that *weak–*$\mathcal{FS}$ is sufficient and in Theorem 2 that it is necessary for set-agreement in all environments. □

5 Summary

We have found the weakest failure detector for set-agreement in message-passing networks where processes may fail by crashing. The failure detector is called *weak*–$\mathcal{FS}$ and it returns at every invocation *"go"* or *"wait"*. It ensures that (1) there is at least one process where the output is always *"wait"*, and (2) if there is only one correct process, then the output at this process is eventually *"go"* forever.

References

[1] Elizabeth Borowsky and Eli Gafni. Generalized flp impossibility result for t-resilient asynchronous computations. In *STOC*, pages 91–100, 1993.

[2] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722, 1996.

[3] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[4] Soma Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Inf. Comput.*, 105(1):132–158, July 1993.

[5] Wei Chen, Jialin Zhang, Yu Chen, and Xuezheng Liu. Weakening failure detectors for k -set agreement via the partition approach. In *DISC*, pages 123–138, 2007.

[6] C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui. Shared memory vs message passing, 2003.

[7] Carole Delporte-Gallet, Hugues Fauconnier, and Rachid Guerraoui. Sharing information is harder than agreeing. Technical report, LIAFA Paris 7 and EPFL, 2008.

[8] Rachid Guerraoui. Non-blocking atomic commit in asynchronous distributed systems with failure detectors. *Distributed Computing*, 15(1):17–25, 2002.

[9] Rachid Guerraoui, Maurice Herlihy, Petr Kouznetsov, Nancy Lynch, and Calvin Newport. On the weakest failure detector ever. In *PODC '07: Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 235–243, New York, NY, USA, 2007. ACM.

[10] Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *Journal of the ACM*, 46(6):858–923, 1999.

[11] Michel Raynal and Corentin Travers. In search of the holy grail: Looking for the weakest failure detector for wait-free set agreement. In *OPODIS*, pages 3–19, 2006.

[12] Michael Saks and Fotios Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. *SIAM J. Comput.*, 29(5):1449–1483, 2000.

[13] Piotr Zieliński. Anti-Ω: the weakest failure detector for set-agreement. Technical report, UCAM-CL-TR-694, 2007.