INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

PUBLICATION INTERNE

N° 1254

SOLVING CONSENSUS USING CHANDRA-TOUEG'S UNRELIABLE FAILURE DETECTORS: A GENERAL QUORUM-BASED APPROACH

ACHOUR MOSTEFAOUI , MICHEL RAYNAL

ISSN 1166-8687

IRISA

CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES
Campus de Beaulieu – 35042 Rennes Cedex – France
Tél. : (33) 02 99 84 71 00 – Fax : (33) 02 99 84 71 71
http://www.irisa.fr

Solving Consensus Using Chandra-Toueg's Unreliable Failure Detectors: a General Quorum-Based Approach

Achour Mostefaoui , Michel Raynal

Thème 1 — Réseaux et systèmes
Projet Adp

Publication interne n ° 1254 — Juillet 1999 — 14 pages

Abstract: This paper addresses the Consensus problem in asynchronous distributed systems (made of n processes, at most f of them may crash) equipped with unreliable failure detectors. A *generic* Consensus protocol is presented: it is quorum-based and works with any failure detector belonging to the class $\mathcal{S}$ (provided that $f \leq n-1$) or to the class $\diamond\mathcal{S}$ (provided that $f < n/2$). This quorum-based generic approach for solving the Consensus problem is new (to our knowledge). Moreover, the proposed protocol is conceptually simple, allows early decision and uses messages shorter than previous solutions.

The generic dimension and the surprising design simplicity of the proposed protocol provide a better understanding of the basic algorithmic structures and principles that allow to solve the Consensus problem with the help of unreliable failure detectors.

Key-words: Asynchronous Distributed System, Consensus, Crash Failure, Perpetual/Eventual Accuracy, Quorum, Unreliable Failure Detector.

(Résumé : tsvp)

To appear in Proceedings of the 13th International Symposium on Distributed Computing (DISC'99), Bratislava, LNCS Series, September 1999.

email: {mostefaoui,raynal}@irisa.fr

Centre National de la Recherche Scientifique (UPRESSA 6074) Université de Rennes 1 – Insa de Rennes

Institut National de Recherche en Informatique et en Automatique – unité de recherche de Rennes

Une solution au problème de consensus fondée sur les quorums

Résumé : Cet article traite du problème de consensus dans les systèmes répartis (n étant le nombre de processus et f le nombre maximum de défaillances possibles) munis de suspecteurs de défaillances de Chandra et Toueg. Il propose un même protocole pour deux classes de suspecteurs de défaillances : la classe $\mathcal{S}$ (dans ce cas $f \leq n-1$) et la classe $\diamond\mathcal{S}$ (dans ce cas $f < n/2$).

La simplicité du protocole alliée à sa dimension générique permet de mieux comprendre les fondements algorithmiques des protocoles de consensus utilisant des suspecteurs de défaillances.

Mots clés : Consensus, quorum, propriété d'exactitude perpétuelle/inéluctable, suspecteur de défaillances, système réparti asynchrone.

1 Introduction

The *Consensus* problem is now recognized as being one of the most important problems to solve when one has to design or to implement reliable applications on top of an unreliable asynchronous distributed system. Informally, the Consensus problem is defined in the following way. Each process proposes a value, and all non-crashed processes have to agree on a common value which has to be one of the proposed values. The Consensus problem is actually a fundamental problem. This is because the most important practical agreement problems (*e.g.*, Atomic Broadcast, Atomic Multicast, Weak Atomic Commitment) can be reduced to it (see, for example, [2], [5] and [6] for each of the previous problems, respectively). The Consensus problem can be seen as their "*greatest common denominator*".

Solving the Consensus problem in an asynchronous distributed system where processes can crash is far from being a trivial task. More precisely, it has been shown by Fischer, Lynch and Paterson [4] that there is no deterministic solution to the Consensus problem in those systems as soon as processes (even only one) may crash. The intuition that underlies this impossibility result lies in the inherent difficulty of safely distinguishing a crashed process from a "slow" process, or from a process with which communications are "very slow". This result has challenged and motivated researchers to find a set of minimal properties that, when satisfied by the runs of a distributed system, allows to solve Consensus despite process crashes.

The major advance to circumvent the previous impossibility result is due to Chandra and Toueg who have introduced [2] (and studied with Hadzilacos [3]) the *Unreliable Failure Detector* concept. A failure detector can be seen as a set of modules, each associated with a process. The failure detector module attached to a process provides it with a list of processes it suspects to have crashed. A failure detector module can make mistakes by not suspecting a crashed process or by erroneously suspecting a correct process. In their seminal paper [2] Chandra and Toueg have introduced several classes of failure detectors. A class is defined by a *Completeness* property and an *Accuracy* property. A completeness property is on the actual detection of crashes. The aim of an accuracy property is to restrict erroneous suspicions. Moreover an accuracy property is *Perpetual* if it has to be permanently satisfied. It is *Eventual* if it is allowed to be permanently satisfied only after some time.

In this paper, we are interested in solving the Consensus problem in asynchronous distributed systems equipped with a failure detector of the class $\mathcal{S}$ or with a failure detector of the class $\diamond\mathcal{S}$. Both classes are characterized by the same completeness property, namely, "Eventually, every crashed process is suspected by every correct process". They are also characterized by the same basic accuracy property, namely, "There is a correct process that is never suspected". But these two classes differ in the way (modality) their failure detectors satisfy this basic accuracy property. More precisely, the failure detectors of $\mathcal{S}$ perpetually satisfy the basic accuracy property, while the failure detectors of $\diamond\mathcal{S}$ are allowed to satisfy it only eventually.

Several Consensus protocols based on such failure detectors have been designed. Chandra and Toueg have proposed a Consensus protocol that works with any failure detector of the class $\mathcal{S}$ [2]. This protocol tolerates any number of process crashes. Several authors have proposed Consensus protocols based on failure detectors of the class $\diamond\mathcal{S}$: Chandra and Toueg [2], Schiper [9] and Hurfin and Raynal [8]. All these $\diamond\mathcal{S}$-based Consensus protocols require a majority of correct processes. It has been shown that this requirement is necessary [2]. So, these protocols are optimal with respect to the number of crashes they tolerate. Moreover, when we consider the classes of failure detectors that allow to solve the Consensus problem, it has ben shown that $\diamond\mathcal{S}$ is the weakest one [3].

$\mathcal{S}$-based Consensus protocols and $\diamond\mathcal{S}$-based Consensus protocols are usually considered as defining two distinct families of failure detector-based Consensus protocols. This is motivated by (1) the fact that one family assumes perpetual accuracy while the other assumes only eventual accuracy, and (2) the fact that the protocols of each family (and even protocols of a same family) are based on different algorithmic principles. In this paper, we present a generic failure detector-based Consensus protocol which has several interesting characteristics. The most important one is of course its "generic" dimension: it works with any failure detector of the class $\mathcal{S}$ (provided $f < n$) or with any failure detector of the class $\Diamond\mathcal{S}$ (provided $f < n/2$) (where n and f denote the total number of processes and the maximum number of processes that may crash, respectively). This protocol is based on a single algorithmic principle, whatever is the class of the underlying failure detector. Such a generic approach for solving the Consensus problem is new (to our knowledge). It has several advantages. It favors a better understanding of the basic algorithmic structures and principles that are needed to solve the Consensus problem with the help of a failure detector. It also provides a better insight into the "perpetual/eventual" attribute of the accuracy property, when using unreliable failure detectors to solve the Consensus problem. (So, it allows to provide a single proof, where the use of this attribute is perfectly identified.) Moreover, the algorithmic unity of the protocol is not obtained to the detriment of its efficiency. Last but not least, the design simplicity of the protocol is also one of its noteworthy properties.

The paper is composed of seven sections. Section 2 presents the distributed system model and the failure detector concept. Section 3 defines the Consensus problem. The next two sections are devoted to the generic protocol: it is presented in Section 4 and proved in Section 5. Section 6 discusses the protocol and compares it with previous failure detector-based Consensus protocols. Finally Section 7 concludes the paper.

2 Asynchronous Distributed System Model

The system model is patterned after the one described in [2, 4]. A formal introduction to failure detectors is provided in [2, 3].

2.1 Asynchronous Distributed System with Process Crash Failures

We consider a system consisting of a finite set Π of $n > 1$ processes, namely, $\Pi = \{p_1, p_2, \dots, p_n\}$. A process can fail by *crashing*, *i.e.*, by prematurely halting. It behaves correctly (*i.e.*, according to its specification) until it (possibly) crashes. By definition, a *correct* process is a process that does not crash. Let f denote the maximum number of processes that can crash ($f \leq n-1$). Processes communicate and synchronize by sending and receiving messages through channels. Every pair of processes is connected by a channel. Channels are not required to be FIFO, they may also duplicate messages. They are only assumed to be reliable in the following sense: they do not create, alter or lose messages. This means that a message sent by a process p_i to a process p_j is assumed to be eventually received by p_j, if p_j is correct[1]. The multiplicity of processes and the message-passing communication make the system *distributed*. There is no assumption about the relative speed of processes or the message transfer delays. This absence of timing assumptions makes the distributed system *asynchronous*.

2.2 Unreliable Failure Detectors

Informally, a failure detector consists of a set of modules, each one attached to a process: the module attached to p_i maintains a set (named $suspected_i$) of processes it currently suspects to have crashed. Any failure detector module is inherently unreliable: it can make mistakes by not suspecting a crashed process or by erroneously suspecting a correct one. Moreover, suspicions are not necessarily stable: a process p_j can be added to or removed from a set $suspected_i$ according to whether p_i's failure detector

[1] The "no message loss" assumption is required to ensure the Termination property of the protocol. The "no creation and no alteration" assumptions are required to ensure its Validity and Agreement properties (see Sections 3 and 4).

module currently suspects p_j or not. As in papers devoted to failure detectors, we say "process p_i suspects process p_j" at some time, if at that time we have $p_j \in suspected_i$.

As indicated in the introduction, a failure detector class is formally defined by two abstract properties, namely a *Completeness* property and an *Accuracy* property. In this paper, we consider the following completeness property [2]:

- Strong Completeness: Eventually, every process that crashes is permanently suspected by every correct process.

Among the accuracy properties defined by Chandra and Toueg [2] we consider here the two following ones:

- Perpetual Weak Accuracy: Some correct process is never suspected.
- Eventual Weak Accuracy: There is a time after which some correct process is never suspected by correct processes.

Combined with the completeness property, these accuracy properties define the following two classes of failure detectors [2]:

- $\mathcal{S}$: The class of *Strong* failure detectors. This class contains all the failure detectors that satisfy the strong completeness property and the perpetual weak accuracy property.
- $\diamond\mathcal{S}$: The class of *Eventually Strong* failure detectors. This class contains all the failure detectors that satisfy the strong completeness property and the eventual weak accuracy property.

Clearly, $\mathcal{S} \subset \diamond\mathcal{S}$. Moreover, it is important to note that any failure detector that belongs to $\mathcal{S}$ or to $\diamond\mathcal{S}$ can make an arbitrary number of mistakes.

3 The Consensus Problem

3.1 Definition

In the Consensus problem, every correct process p_i *proposes* a value v_i and all correct processes have to *decide* on some value v, in relation to the set of proposed values. More precisely, the *Consensus problem* is defined by the following three properties [2, 4]:

- Termination: Every correct process eventually decides on some value.
- Validity: If a process decides v, then v was proposed by some process.
- Agreement: No two correct processes decide differently.

The agreement property applies only to correct processes. So, it is possible that a process decides on a distinct value just before crashing. *Uniform Consensus* prevents such a possibility. It has the same Termination and Validity properties plus the following agreement property:

- Uniform Agreement: No two processes (correct or not) decide differently.

In the following we are interested in the Uniform Consensus problem.

3.2 Solving Consensus with Unreliable Failure Detectors

The following important results are associated with the Consensus problem when one has to solve it in an asynchronous distributed system, prone to process crash failures, equipped with an unreliable failure detector.

- In any distributed system equipped with a failure detector of the class $\mathcal{S}$, the Consensus problem can be solved whatever the number of crashes is [2].
- In any distributed system equipped with a failure detector of the class $\Diamond\mathcal{S}$, at least a majority of processes has to be correct (*i.e.*, $f < n/2$) for the Consensus problem to be solvable [2].
- When we consider the classes of failure detectors that allow to solve the Consensus problem, $\Diamond\mathcal{S}$ is the weakest one [3]. This means that, as far as failure detection is concerned, the properties defined by $\Diamond\mathcal{S}$ constitute the borderline beyond which the Consensus problem can not be solved[2].
- Any protocol solving the Consensus problem using an unreliable failure detector of the class $\mathcal{S}$ or $\Diamond\mathcal{S}$, solves also the Uniform Consensus problem [6].

4 The General Consensus Protocol

4.1 Underlying Principles

The algorithmic principles that underly the protocol are relatively simple. The protocol shares some of them with other Consensus protocols [2, 8, 9]. Each process p_i manages a local variable est_i which contains its current estimate of the decision value. Initially, est_i is set to v_i, the value proposed by p_i. Processes proceed in consecutive asynchronous rounds. Each round r (initially, for each process p_i, $r_i = 0$) is managed by a predetermined process p_c (*e.g.*, c can be defined according to the round robin order). So, the protocol uses the well-known *rotating coordinator* paradigm[3].

Description of a round A round (r) is basically made of two phases (communication steps).
First phase of a round. The current round coordinator p_c sends its current estimate (est_c) to all processes. This phase terminates, for each process p_i, when p_i has received an estimate from p_c or when it suspects p_c. In addition to est_i, p_i manages a local variable $est\_from\_c_i$ that contains either the value it has received from p_c, or the default value $\perp$. So, $est\_from\_c_i = \perp$ means that p_i has suspected p_c, and $est\_from\_c_i \neq \perp$ means that $est\_from\_c_i = est_c$. If we assumed that all non-crashed processes or none of them have received p_c's estimate and they all have the same perception of crashes, then they would get the same value in their $est\_from\_c_i$ local variables. Consequently, they could all "synchronously" either decide (when $est\_from\_c_i \neq \perp$) or proceed to the next round (when $est\_from\_c_i = \perp$).
Second phase of a round. Unfortunately, due to asynchrony and erroneous failure suspicions, some processes p_j can have $est\_from\_c_j = est_c$, while other processes p_k can have $est\_from\_c_k = \perp$ at the end of the first phase. Actually, the aim of the first phase was to ensure that $\forall\ p_i$: $est\_from\_c_i = est_c$ or $\perp$. The aim of the second phase is to ensure that the Agreement property will never be violated. This prevention is done in the following way: if a process p_i decides $v = est_c$ during r and if a process p_j progresses to $r + 1$, then p_j does it with $est_j = v$. This is implemented by the second phase that

[2]The "weakest class" proof is actually on the class $\Diamond\mathcal{W}$ of failure detectors [3]. But, it has been shown that $\Diamond\mathcal{W}$ and $\Diamond\mathcal{S}$, that differ in the statement of their completeness property, are actually equivalent: the protocol that transforms any failure detector of the class $\Diamond\mathcal{W}$ in a failure detector of the class $\Diamond\mathcal{S}$ is based on a simple gossiping mechanism [2].

[3]Due to the completeness property of the underlying failure detector, this paradigm can be used without compromising the protocol termination. More precisely, the completeness property can be exploited by a process to not indefinitely wait for a message from a crashed coordinator.

requires each process p_i to broadcast the value of its $est\_from\_c_i$ local variable. A process p_i finishes the second phase when it has received est_from_c values from "enough" processes. The meaning of "enough" is captured by a set Q_i, dynamically defined during each round. Let rec_i be the set of est_from_c values received by p_i from the processes of Q_i. We have: $rec_i = \{\bot\}$ or $\{v\}$ or $\{v, \bot\}$ (where v is the estimate of the current coordinator). Let p_j be another process (with its Q_j and rec_j). If $Q_i \cap Q_j \neq \emptyset$, then there is a process $p_x \in Q_i \cap Q_j$ that has broadcast $est\_from\_c_x$ and both p_i and p_j have received it. It follows that rec_i and rec_j are related in the following way:

$$\begin{array}{rcl} rec_i = \{v\} & \Rightarrow & (\forall\, p_j :\ (rec_j = \{v\}) \ \vee \ (rec_j = \{v, \bot\})) \\ rec_i = \{\bot\} & \Rightarrow & (\forall\, p_j :\ (rec_j = \{\bot\}) \ \vee \ (rec_j = \{v, \bot\})) \\ rec_i = \{v, \bot\} & \Rightarrow & (\forall\, p_j :\ (rec_j = \{v\}) \ \vee \ (rec_j = \{\bot\}) \ \vee \ (rec_j = \{v, \bot\})) \end{array}$$

The behavior of p_i is then determined by the content of rec_i:

- When $rec_i = \{v\}$, p_i knows that all non-crashed processes also know v. So, p_i is allowed to decide on v provided that all processes that do not decide consider v as their current estimate.
- When $rec_i = \{\bot\}$, p_i knows that any set rec_j includes $\bot$. In that case, no process p_j is allowed to decide and p_i proceeds to the next round.
- When $rec_i = \{v, \bot\}$, according to the previous items, p_i updates its current estimate (est_i) to v to achieve the Agreement property. Note that if a process p_j decides during this round, any process p_i that proceeds to the next round, does it with $est_i = v$.

Definition of the Q_i set As indicated previously, the definition of the Q_i sets has to ensure that the predicate $Q_i \cap Q_j \neq \emptyset$ holds for every pair (p_i, p_j). The way this condition is realized depends on the class to which the underlying failure detector belongs.

Let us first consider the case where the failure detector belongs to the class $\mathcal{S}$. In that case, there is a correct process that is never suspected. Let p_x be this process. If Q_i contains p_x, then p_i will obtain the value of $est\_from\_c_x$. If follows that if $(\forall\, p_i)$ Q_i is such that $\Pi = Q_i \cup suspected_i$, then, $\forall\, (p_i, p_j)$, we have $p_x \in Q_i \cap Q_j$.

Let us now consider the case where the failure detector belongs to the class $\diamond\mathcal{S}$. In that case, $f < n/2$ and there is a time after which some correct process is no longer suspected. As we do not know the time from which a correct process is no longer suspected, we can only rely on the majority of correct processes assumption. So, by taking $(\forall\, p_i)$ Q_i equal to a majority set, it follows that, $\forall\, (p_i, p_j)$, $\exists p_x$ such that, $p_x \in Q_i \cap Q_j$.

Note that in both cases, Q_i is not statically defined. In each round, its actual value depends on message receptions and additionally, in the case of $\mathcal{S}$, on process suspicions.

On the quorum-based approach The previous principles actually define a quorum-based approach. As usual, (1) each quorum must be *live*: it must include only non-crashed processes (this ensures processes will not block forever during a round). Furthermore, (2) each quorum must be *safe*: it must have a non-empty intersection with any other quorum (this ensures the agreement property cannot be violated). As indicated in the previous paragraph, the quorum safety requirement is guaranteed by the "perpetual" modality of the accuracy property (for $\mathcal{S}$), and by the majority of correct processes assumption (for $\diamond\mathcal{S}$).

Other combinations of eventual weak accuracy (to guarantee eventual termination) and live and safe (possibly non-majority) quorums would work[4]. Bringing a quorum-based formulation to the fore

[4] As an example, let us consider quorums defined from a $\sqrt{n} * \sqrt{n}$ grid (with $n = q^2$). This would allow the protocol to progress despite $n - (2 * \sqrt{n} - 1)$ crashes or erroneous suspicions in the most favorable case. Of course, in the worst case, the use of such quorums could block the protocol in presence of only $\sqrt{n}$ crashes or erroneous suspicions. Details on quorum definition can be found in [1].

is conceptually interesting. Indeed, the protocol presented in the next section works for any failure detector satisfying strong completeness, eventual weak accuracy and the "quorum" conditions.

4.2 The Protocol

The protocol is described in Figure 1. A process p_i starts a Consensus execution by invoking Consensus(v_i). It terminates it when it executes the statement **return** which provides it with the decided value (lines 12 and 16).

It is possible that distinct processes do not decide during the same round. To prevent a process from blocking forever (*i.e.*, waiting for a value from a process that has already decided), a process that decides, uses a reliable broadcast [7] to disseminate its decision value (similarly as protocols described in [2, 8, 9]). To this end the Consensus function is made of two tasks, namely, $T1$ and $T2$. $T1$ implements the previous discussion. Line 12 and $T2$ implement the reliable broadcast.

```
Function Consensus(v_i)
cobegin
 (1) task T1: r_i ← 0; est_i ← v_i; % v_i ≠ ⊥ %
 (2)  while true do
 (3)    c ← (r_i mod n) + 1; est_from_c_i ← ⊥; r_i ← r_i + 1; % round r = r_i %
 (4)    case (i = c) then est_from_c_i ← est_i
 (5)         (i ≠ c) then wait ((EST(r_i, v) is received from p_c)∨(c ∈ suspected_i));
 (6)                      if (EST(r_i, v) has been received) then est_from_c_i ← v
 (7)    endcase;   % est_from_c_i = est_c or ⊥ %
 (8)    ∀j do send EST(r_i, est_from_c_i) to p_j enddo;
 (9)    wait until (∀p_j ∈ Q_i: EST(r_i, est_from_c) has been received from p_j);
                   % Q_i has to be a live and safe quorum %
                   % For  S:   Q_i is such that Q_i ∪ suspected_i = Π %
                   % For ◇S:   Q_i is such that |Q_i| = ⌈(n + 1)/2⌉ %
 (10)   let rec_i = {est_from_c | EST(r_i, est_from_c) is received at line 5 or 9};
                   % est_from_c = ⊥ or v with v = est_c %
                   % rec_i = {⊥} or {v} or {v, ⊥} %
 (11)   case (rec_i = {⊥}) then skip
 (12)        (rec_i = {v}) then ∀j ≠ i do send DECIDE(v) to p_j enddo; return(v)
 (13)        (rec_i = {v, ⊥}) then est_i ← v
 (14)   endcase
 (15) enddo

 (16) task T2: upon reception of DECIDE(v):
                   ∀j ≠ i do send DECIDE(v) to p_j enddo; return(v)
coend
```

Figure 1: The Consensus Protocol

5 Correctness Proof

5.1 Validity

Lemma 1 *Let us consider a round r and a process p_i. The round r is coordinated by p_c. We have:*
(1) If p_c participates in round r, then est_c is equal to an initial value proposed by a process.
(2) If p_i computes rec_i during round r, then: $rec_i = \{\bot\}$ or $rec_i = \{v\}$ or $rec_i = \{v, \bot\}$, where v is equal to est_c. Moreover, if $v \in rec_i$, p_c has participated in round r.
(3) If p_i starts round $r+1$, it does it with an estimate (est_i) whose value is equal to an initial value.

Proof The proof is by induction on the round number.

- Base case. Let us consider the round $r = 1$. It is coordinated by $p_c = p_1$, and est_c is equal to v_c (p_c's proposal, line 1). The local variable $est\_from\_c_j$ of any process p_j that (during this round) executes line 8, is equal either to est_c (if p_j has received an estimate from p_c -line 4- or if $p_j = p_c$ -line 6-) or to $\bot$ (if p_j has suspected p_c, line 5). So, if p_j executes line 8, it broadcasts either the value of est_c ($= v_c$) or $\bot$. It follows that any p_i that computes rec_i during the first round, can only receive v_c or $\bot$ at line 9. Consequently, we have: $rec_i = \{\bot\}$ or $rec_i = \{v_c\}$ or $rec_i = \{v_c, \bot\}$.

  Now, let us first note that, initially, $est_i = v_i$ (line 1). Due to lines 11-13, if p_i starts $r+1$, it does it either with the value of est_i left unchanged (line 11) or with $est_i = v_c$ (line 13). So, the lemma is true for $r = 1$.

- Assume the lemma is true until r, $r \geq 1$. This means that if p_c (the round $r+1$ coordinator) participates in $r+1$, then we had (at the end of r) $rec_c = \{\bot\}$ or $rec_c = \{v\}$ or $rec_c = \{v, \bot\}$, where v is an initial value. Due to the induction assumption and to the **case** statement (lines 11-14) executed by p_c at the end of r, it follows that p_c starts $r+1$ with est_c equal to an initial value proposed by a process. Now, the situation is similar to the one of the base case, and consequently, the same argument applies to the round $r+1$ case, which proves the lemma.

$\square_{Lemma\ 1}$

Theorem 1 *If a process p_i decides v, then v was proposed by some process.*

Proof If a process decides at line 16, it decides the value decided by another process at line 12. So we only consider the case where a value that has been decided at line 12. When a process p_i decides v at line 12, it decides on the value ($\neq \bot$) of the rec_i singleton. Due to the items (1) and (2) of Lemma 1, v is an initial value of a process. $\square_{Theorem\ 1}$

5.2 Termination

Lemma 2 *If no process decides during $r' \leq r$, then all correct processes start $r+1$.*

Proof The proof is by contradiction. Suppose that no process has decided during a round $r' \leq r$, where r is the smallest round number in which a correct process p_i blocks forever. So, p_i is blocked at line 4 or at line 9.

Let us first examine the case where p_i blocks at line 4. Let p_c be the round r coordinator. If $p_i = p_c$, it cannot block at line 4, as it does not execute this line. Moreover, in that case, it executes the broadcast at line 8 or crashes. If $p_i \neq p_c$, then:
- Either p_i suspects p_c. This is due to an erroneous suspicion or to the strong completeness property of the failure detector.

- Or p_i never suspects p_c. Due to the strong completeness property, this means that p_c is correct. From the previous observation, p_c has broadcast its current estimate at line 8, and p_i eventually receives it. It follows that p_i cannot remain blocked at line 4.

Let us now consider the case where p_i blocks at line 9. For this line there are two cases to consider, according to the class of the underlying failure detector.

- The failure detector belongs to $\mathcal{S}$ and $f \leq n-1$. In that case, the set Q_i of processes from which p_i is waiting for messages is such that $Q_i \cup suspected_i = \Pi$. As, no correct process blocks forever at line 4, each of them executes a broadcast at line 8. It follows from these broadcasts and from the strong completeness property that, $\forall\, p_j$, p_i will receive a round r estimate from p_j or will suspect it. Consequently, p_i cannot block at line 9.

- The failure detector belongs to the class $\diamond\mathcal{S}$ and $f < n/2$. In that case Q_i is defined as the first majority set of processes p_j from which p_i has received a $\text{EST}(r, est\_from\_c)$ message. As there is a majority of correct processes, and as (due to the previous observation) they do not block forever at line 4, they broadcast a round r estimate message (line 8). It follows that any correct process receives a message from a majority set of processes. Consequently, p_i cannot block at line 9.

Finally, let us note that, due to the item (2) of Lemma 1, a correct process p_i terminates correctly the execution of the **case** statement (lines 11-14). It follows that if p_i does not decide, it proceeds to the next round. A contradiction. $\square_{Lemma\ 2}$

Theorem 2 *If a process p_i is correct, then it decides.*

Proof If a (correct or not) process decides, then, due to the sending of DECIDE messages at line 12 or at line 14, any correct process will receive such a message and decide accordingly (line 14).

So, suppose that no process decides. The proof is by contradiction. Due to the accuracy property of the underlying failure detector, there is a time t after which there is a correct process that is never suspected. Note that $t = 0$ if the failure detector belongs to $\mathcal{S}$, and $t \geq 0$ if it belongs to $\diamond\mathcal{S}$ (assuming the protocol starts executing at time $t = 0$).

Let p_x be the correct process that is never suspected after t. Moreover, let r be the first round that starts after t and that is coordinated by p_x. As by assumption no process decides, due to Lemma 2, all the correct processes eventually start round r.

The process p_x starts round r by broadcasting its current estimate value (est_x), which, due to Lemma 1, is equal to an initial value. Moreover, during r, p_x is not suspected. Consequently, all processes p_i that participate in round r (this set includes the correct processes) receive est_x at line 4, and adopt it as their $est\_from\_c_i$ value. If follows that no value different from est_x can be broadcast at line 8; consequently, est_x is the only value that can be received at line 9. Hence, for any correct process p_i, we have $rec_i = \{est_x\}$ at line 10. It follows that any correct process executes line 12 and decides accordingly. $\square_{Theorem\ 2}$

The following corollary follows from the proof of the previous theorem.

Corollary 1 *If the underlying failure detector belongs to the class $\mathcal{S}$, the maximum number of rounds is n. Moreover, there is no bound on the round number when the underlying failure detector belongs to the class $\diamond\mathcal{S}$.*

5.3 Uniform Agreement

Lemma 3 *If two processes p_i and p_j decide at line 12 during the same round, they decide the same value.*

Proof If both p_i and p_j decide during the same round r, at line 12, we had $rec_i = \{v\}$ and $rec_j = \{v'\}$ at line 10. Moreover, from item (2) of Lemma 1, we have $v = v' = est_c$ (where est_c is the value broadcast during r by its coordinator). $\Box_{Lemma\ 3}$

Theorem 3 *No two processes decide differently.*

Proof Let r be the first round during which a process p_i decides. It decides at line 12. Let v be the value decided by p_i. Let us assume another process decides v' during a round $r' \geq r$. If $r' = r$, then due to Lemma 3, we have $v = v'$. So, let us consider the situation where $r' > r$. We show that the estimate values (est_j) of all the processes p_j that progress to $r + 1$ are equal to v. This means that no other value can be decided in a future round[5].

Let us consider any process p_k that terminates the round r. Let us first note that there is a process p_x such that $p_x \in Q_i \cap Q_k$. This follows from the following observation:
- If the failure detector belongs to $\mathcal{S}$, then by considering p_x, the correct process that is never suspected, we have $p_x \in Q_i \cap Q_k$.
- If the failure detector belongs to the class $\diamond\mathcal{S}$, as Q_i and Q_k are majority sets, we have $Q_i \cap Q_k \neq \emptyset$, and there is a p_x such that $p_x \in Q_i \cap Q_k$.

As p_i has decided v at line 12 during r, we had during this round $rec_i = \{v\}$. This means that p_i has received v from all the processes of Q_i, and so from p_x. Thus, p_k has also received v from p_x, and consequently, $rec_k = \{v\}$ or $rec_k = \{v, \perp\}$. It follows that if p_k proceeds to the next round, it executes line 13. Consequently, for all processes p_j that progress to $r + 1$, we have $est_j = v$. This means that, from round $r + 1$, all estimate values are equal to v. As no value different from v is present in the system, the only value that can be decided in a round $> r$ is v. $\Box_{Theorem\ 3}$

6 Discussion

6.1 Cost of the Protocol

Time complexity of a round As indicated in Corollary 1, the number of rounds of the protocol is bounded by n, when used with a failure detector of the class $\mathcal{S}$. There is no upper bound when it is used with a failure detector of the class $\diamond\mathcal{S}$. So, to analyze the time complexity of the protocol, we consider the length of the sequence of messages (number of communication steps) exchanged during a round. Moreover, as on one side we do not master the quality of service offered by failure detectors, but as on the other side, in practice failure detectors can be tuned to very seldom make mistakes, we do this analysis considering the underlying failure detector behaves reliably. In such a context, the time complexity of a round is characterized by a pair of integers. Considering the most favorable scenario that allows to decide during the current round, the first integer measures its number of communication steps. The second integer considers the case where a decision can not be obtained during the current round and measures the minimal number of communication steps required to progress to the next round. Let us consider these scenarios.

[5]When we consider the terminology used in $\diamond\mathcal{S}$-based protocols, this means the value v is *locked*. This proof shows that the "*value locking*" principle is not bound to the particular use of $\diamond\mathcal{S}$. With $\mathcal{S}$, a value is locked as soon as it has been forwarded by the (first) correct process that is never suspected. With $\diamond\mathcal{S}$, a value is locked as soon as it has been forwarded by a majority of processes.

- The first scenario is when the current round coordinator is correct and is not suspected. In that case, 2 communication steps are required to decide. During the first step, the current coordinator broadcasts its value (line 8). During the second step, each process forwards that value (line 8), waits for "enough" messages (line 9), and then decides (line 12). So, in the most favorable scenario that allows to decide during the current round, the round is made of two communication steps.

- The second scenario is when the current round coordinator has crashed and is suspected by all processes. In that case, as processes correctly suspect the coordinator (line 5), they actually skip the first communication step. They directly exchange the $\perp$ value (line 8) and proceed to the next round (line 11). So, in the most favorable scenario to proceed to the next round, the round is made of a single communication step.

So, when the underlying failure detector behaves reliably, according to the previous discussion, the time complexity of a round is characterized by the pair $(2, 1)$ of communication steps.

Message complexity of a round During each round, each process sends a message to each process (including itself). Hence, the message complexity of a round is upper bounded by n^2.

Message type and size There are two types of message: EST and DECIDE. A DECIDE message carries only a proposed value. An EST message carries a proposed value (or the default value $\perp$) plus a round number. The size of the round number is bounded by $log_2(n)$ when the underlying failure detector belongs to $\mathcal{S}$ (Corollary 1). It is not bounded in the other case.

6.2 Related Work

Several failure detector-based Consensus protocols have been proposed in the literature. We compare here the proposed protocol (in short MR) with the following protocols:
- The $\mathcal{S}$-based Consensus protocol proposed in [2] (in short, $CT_{\mathcal{S}}$).
- The $\diamond\mathcal{S}$-based Consensus protocol proposed in [2] (in short, $CT_{\diamond\mathcal{S}}$).
- The $\diamond\mathcal{S}$-based Consensus protocol proposed in [9] (in short, $SC_{\diamond\mathcal{S}}$).
- The $\diamond\mathcal{S}$-based Consensus protocol proposed in [8] (in short, $HR_{\diamond\mathcal{S}}$).

As MR, all these protocols proceed in consecutive asynchronous rounds. Moreover, all, but $CT_{\mathcal{S}}$, are based on the rotating coordinator paradigm. It is important to note that each of these protocols has been specifically designed for a special class of failure detectors (either $\mathcal{S}$ or $\diamond\mathcal{S}$). Differently from MR, none of them has a generic dimension. Let us also note that only MR and both CT protocols cope naturally with message duplications (*i.e.*, they do not require additional statements to discard duplicate messages).

Let V={ initial values proposed by processes } $\cup$ $\{\perp\}$. Table 1 compares $CT_{\mathcal{S}}$ and MR (when used with $\mathcal{S}$). Both protocols use n^2 messages during each round. A round is made of one or two communication steps in MR, and of a single communication step in $CT_{\mathcal{S}}$. The first column indicates the total number (k) of communication steps needed to reach a decision. For MR, this number depends on the parameter f. As indicated, $CT_{\mathcal{S}}$ does not allow early decision, while MR does. The second column indicates the size of messages used by each protocol. As the current round number is carried by messages of both protocols, it is not indicated.

Table 2 compares MR (when used with $\diamond\mathcal{S}$) with $CT_{\diamond\mathcal{S}}$, $SC_{\diamond\mathcal{S}}$ and $HR_{\diamond\mathcal{S}}$. In all cases, there is no bound on the round number and all protocols allow early decision. So, the first column compares the time complexity of a round, according to the previous discussion (Section 6.1). The second column is devoted to the message size. As each protocol uses messages of different size, we only consider their

	# communication steps	Message size
$\mathrm{CT}_{\mathcal{S}}$	$k = n$	An array of n values $\in V$
MR with $\mathcal{S}$	$2 \leq k \leq 2(f+1)$	A single value $\in V$

Table 1: Comparing MR with $\mathrm{CT}_{\mathcal{S}}$

biggest messages. Moreover, as in all protocols, each of those messages carries its identity (sender id, round number) and an estimate value, the second column indicates only their additional fields. Let us additionally note that, differently from $\mathrm{SC}_{\diamond\mathcal{S}}$ and $\mathrm{HR}_{\diamond\mathcal{S}}$, MR does not require special statements to prevent deadlock situations.

	Time complexity of a round	Message size
$\mathrm{CT}_{\diamond\mathcal{S}}$	$(3, 0)$	An integer timestamp
$\mathrm{SC}_{\diamond\mathcal{S}}$	$(2, 2)$	A boolean and a process id
$\mathrm{HR}_{\diamond\mathcal{S}}$	$(2, 1)$	A boolean
MR with $\diamond\mathcal{S}$	$(2, 1)$	No additional value

Table 2: Comparing MR with $\mathrm{CT}_{\diamond\mathcal{S}}$, $\mathrm{SC}_{\diamond\mathcal{S}}$ and $\mathrm{HR}_{\diamond\mathcal{S}}$

Finally, let us note that MR provides a (factorized) proof, that is shorter and simpler to understand than the proofs designed for the other protocols.

7 Conclusion

This paper has presented a *generic* Consensus protocol that works with any failure detector belonging to the class $\mathcal{S}$ (provided that $f \leq n-1$) or to the class $\diamond\mathcal{S}$ (provided that $f < n/2$).

The proposed protocol is conceptually simple, allows early decision and uses messages shorter than previous solutions. It has been compared to other Consensus protocols designed for specific classes of unreliable failure detectors. Among its advantages, the design simplicity of the proposed protocol has allowed the design of a simple (and generic) proof. The most noteworthy of its properties lie in its quorum-based approach and in its generic dimension.

It is important to note that a Consensus protocol initially designed to work with a failure detector of the class $\mathcal{S}$ will not work when $\mathcal{S}$ is replaced by $\diamond\mathcal{S}$. Moreover, a Consensus protocol initially designed to work with a failure detector of $\diamond\mathcal{S}$ requires $f < n/2$; if $\diamond\mathcal{S}$ is replaced by $\mathcal{S}$, the protocol will continue to work, but will still require $f < n/2$ which is not a necessary requirement in that context. Actually, modifying a $\diamond\mathcal{S}$-based Consensus protocol to work with $\mathcal{S}$ and $f < n-1$ amounts to design a new protocol. The generic dimension of the proposed protocol prevents this drawback. In that sense, the proposed protocol is the first failure detector-based Consensus protocol that is not bound to a particular class of failure detectors.

Last but not least, the design of this generic protocol is a result of our effort to understand the relation linking $\mathcal{S}$ on one side, and $\diamond\mathcal{S}$ plus the majority requirement on the other side, when solving the Consensus problem with unreliable failure detectors.

Acknowledgments

The authors are grateful to Jean-Michel Hélary who made insightful comments on a first draft of this paper.

References

[1] Agrawal D. and El Abbadi A., Exploiting Logical Structures in Replicated Databases. *Information Processing Letters*, 33:255-260, 1990.

[2] Chandra T. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2):225-267, March 1996.

[3] Chandra T., Hadzilacos V. and Toueg S., The Weakest Failure Detector for Solving Consensus. *Journal of the ACM*, 43(4):685–722, July 1996.

[4] Fischer M.J., Lynch N. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374–382, April 1985.

[5] Fritzke U., Ingels P., Mostefaoui A. and Raynal M., Fault-Tolerant Total Order Multicast to Asynchronous Groups. *Proc. 17th IEEE Symposium on Reliable Distributed Systems*, Purdue University, pp. 228-235, October 1998.

[6] Guerraoui R., Revisiting the Relationship between Non-Blocking Atomic Commitment and Consensus. *Proc. 9th Int. Workshop on Distributed Algorithms (WDAG95)*, Springer-Verlag LNCS 972 (J.M. Hélary and M. Raynal Eds), pp. 87-100, September 1995.

[7] Hadzilacos V. and Toueg S., Reliable Broadcast and Related Problems. In *Distributed Systems*, ACM Press (S. Mullender Ed.), New-York, pp. 97-145, 1993.

[8] Hurfin M. and Raynal M., A Simple and Fast Asynchronous Consensus Protocol Based on a Weak Failure Detector. *Distributed Computing*, 12(4), 1999.

[9] Schiper A., Early Consensus in an Asynchronous System with a Weak Failure Detector. *Distributed Computing*, 10:149-157, 1997.