Distrib. Comput. (1997) 10: 149–157

DISTRIBUTED COMPUTING

Early consensus in an asynchronous system with a weak failure detector*

André Schiper

Ecole Polytechnique Fédérale, Département d'Informatique, CH-1015 Lausanne, Switzerland

Received: April 1995 / Accepted: October 1996

Summary. Consensus is one of the most fundamental problems in the context of fault-tolerant distributed computing. The problem consists, given a set Ω of processes having each an initial value v_i, in deciding among Ω on a common value v. In 1985, Fischer, Lynch and Paterson proved that the consensus problem is not solvable in an asynchronous system subject to a single process crash. In 1991, Chandra and Toueg showed that, by augmenting the asynchronous system model with a well defined *unreliable* failure detector, consensus becomes solvable. They also give an algorithm that solves consensus using the $\diamond\mathscr{S}$ failure detector. In this paper we propose a new consensus algorithm, also using the $\diamond\mathscr{S}$ failure detector, that is more efficient than the Chandra-Toueg consensus algorithm. We measure efficiency by introducing the notion of *latency degree*, which defines the minimal number of communication steps needed to solve consensus. The Chandra-Toueg algorithm has a latency degree of 3 (it requires at least three communication steps), whereas our early consensus algorithm requires only two communication steps (latency degree of 2). We believe that this is an interesting result, which adds to our current understanding of the cost of consensus algorithms based on $\diamond\mathscr{S}$.

Key words: Asynchronous system – Unreliable failure detector – Consensus – Latency

1 Introduction

Consensus is one of the most fundamental problems in the context of fault-tolerant distributed computing. The problem is defined on a set Ω of processes: each process $p_i \in \Omega$ starts with an initial value v_i, and the processes in Ω have to agree on a common outcome value v, such that v is the initial value of one of the processes in Ω. The consensus problem is commonly classified as an "agreement" problem. Other well known agreement problems are the *total order broadcast* problem (also called *atomic broadcast*), in which the processes have to agree on the delivery order of messages [9], and the *atomic commitment* problem, in which the processes have to agree on the outcome *commit*/*abort* of a transaction [1].

The consensus problem is considered to be a difficult problem. The difficulty has been pointed out by Fischer, Lynch and Paterson, who showed that consensus is not solvable in an asynchronous system[1] subject to even a single process crash [7]. The Fischer-Lynch-Paterson impossibility result has led to the introduction of randomization techniques in order to solve the consensus problem [4]; it has also led to consider other system models such as *partial synchrony* [5, 6]. A summary of these approaches can be found in [3].

A significant step towards a general solution of the consensus problem was accomplished in 1991 by the introduction of the notion of *failure detector* [3]. Chandra and Toueg showed that, by augmenting the asynchronous system model with a well defined *unreliable* failure detector, consensus becomes solvable. A failure detector can be seen as a set of (failure detector) modules FD_i, one module FD_i being attached to every process p_i in the system. Each failure detector module FD_i maintains a list of processes that it currently suspects to have crashed. Chandra and Toueg characterize the failure detectors by two properties: *completeness* and *accuracy*. Informally, *completeness* requires that the failure detector eventually suspects every crashed process, while *accuracy* restricts the false suspicions that a failure detector can make. Chandra and Toueg introduce various failure detectors, among others the *Eventually Weak* failure detector, denoted $\diamond\mathscr{W}$, and the *Eventually Strong* failure detector, denoted $\diamond\mathscr{S}$. We come back to the properties that characterize these failure detectors in the next section. It is, however, important to mention that $\diamond\mathscr{W}$ is the weakest failure detector for solving consensus in asynchronous systems [2].

*Research supported by the "Fonds national suisse" under contract number 21-43196.95

[1] An asynchronous system sets no bound on the transmission delays of messages, which makes it impossible to distinguish a crashed process from a process that is just slow, or connected through a slow channel

In this paper we give a new consensus algorithm that is more efficient than the Chandra-Toueg consensus algorithm based on $\diamond\mathscr{S}$. We measure efficiency by introducing the notion of *latency degree*, which defines the minimal number of communication steps needed by an algorithm $\mathscr{A}$ to solve a problem $\mathscr{P}$[2]. In the absence of failure, and assuming that the failure detector makes very few mistakes, the latency degree gives an indication of the minimal delay incurred by $\mathscr{A}$ to solve $\mathscr{P}$. To illustrate the latency degree measure, consider the atomic commitment problem [1]: the classical two phase commit protocol (or 2PC) has a latency degree of 3 (the protocol requires at least 3 communication steps), whereas the three phase commit protocol (or 3PC) [11] has a latency degree of 5 (the protocol requires at least 5 communication steps).

The Chandra-Toueg algorithm for solving consensus with the $\diamond\mathscr{S}$ failure detector[3] has a latency degree of 4. A trivial optimization may however reduce the latency degree of the algorithm from 4 to 3 (see Sect. 4.3). Thus we consider that the Chandra-Toueg consensus algorithm using $\diamond\mathscr{S}$ has a latency degree of 3. The early consensus algorithm given in the paper, which also uses the $\diamond\mathscr{S}$ failure detector, has a latency degree of 2. We believe that this is an interesting result, which adds to our current understanding of the cost of consensus algorithms based on $\diamond\mathscr{S}$.

The rest of the paper is organized as follows. Section 2 presents the system model, defines the latency degree measure, the consensus problem, and the properties of the $\diamond\mathscr{S}$ and $\diamond\mathscr{W}$ failure detectors. Section 3 gives an overview of the Chandra-Toueg consensus algorithm, and introduces our early consensus algorithm. The complete algorithm and its proof are given in Sect. 4. Section 4 also analyses the cost of both algorithms, in terms of latency degree and number of messages. We conclude the paper with Sect. 5.

2 System model and definitions

2.1 System model

We consider a set of n processes $\Omega = \{p_1, p_2, \ldots, p_n\}$ completely connected through a set of channels. Processes fail by crashing (we do not consider Byzantine failures). A correct process is a process that does not crash in an infinite run. Communication is by message passing, asynchronous and reliable:

- "asynchrony" means that there is no bound on communication delays, nor on relative process speeds;
- "reliability" means that a message sent by a process p_i to a process p_j is eventually received by p_j, if p_j is correct (i.e. does not crash)[4].

A process $p_i \in \Omega$ may (1) send a message to another process, (2) receive a message sent by another process, (3) perform some local computation, or (4) crash. We note f the maximum number of processes of Ω that can be subject to crash failures, and we assume that there is always a majority of correct processes in Ω, i.e. $f < |\Omega|/2$ (or $f < n/2$)[5].

2.2 Consensus

In the consensus problem, every process $p_i \in \Omega$ initially *proposes* a value v_i taken from a set of possible values, and the processes in Ω have to *decide* on some value v such that the following properties hold [3]:

Termination. Each correct process eventually decides.

Validity. If a process decides v, then v was proposed by some process.

Agreement. No two correct processes decide differently.

The Agreement condition allows incorrect processes to decide differently from correct processes. In this paper we consider *uniform* consensus, defined by the *Uniform Agreement* property:

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

It has been shown that any algorithm that solves the consensus problem using the failure detector $\diamond\mathscr{S}$ (see Sect. 2.3), also solves the *uniform* consensus problem [8]. In particular, the Chandra-Toueg consensus algorithm based on the $\diamond\mathscr{S}$ failure detector solves uniform consensus.

2.3 Failure detectors

We recall briefly some definitions taken from [3]. A failure detector is a set of n local modules FD_1 to FD_n, where FD_i is attached to process $p_i \in \Omega$. Each failure detector module FD_i maintains a list of processes that it currently suspects to have crashed. "*Process* p_i *suspects process* p_j" at some local instant t, means that at local time t, process p_j is in the list of suspected processes maintained by FD_i. A failure detector module can make mistakes by incorrectly suspecting a process. Suspicions are not necessarily stable: if at a given instant p_i suspects p_j, it can later learn that the suspicion was incorrect. Process p_j is then removed by FD_i from its list of suspected processes.

Chandra and Toueg define various failure detectors ordered by *reducibility*. Let FD and FD' be two failure detectors. FD' is said to be *reducible* to FD if there exists an algorithm $\mathscr{A}_{FD \to FD'}$ that transforms FD into FD'. FD' is also said to be weaker than FD, written $FD' \preceq FD$. If $FD \preceq FD'$ and $FD' \preceq FD$ hold, then FD and FD' are said to be *equivalent*, which is written $FD \cong FD'$. From the failure detectors in [3], we consider only $\diamond\mathscr{W}$ and $\diamond\mathscr{S}$:

Eventually Weak. $\diamond\mathscr{W}$. The $\diamond\mathscr{W}$ failure detector satisfies the two following properties:

[2] The concept of *latency degree* allows us to avoid the ambiguous notion of *number of phases* of an algorithm

[3] The $\diamond\mathscr{S}$ failure detector is equivalent to the $\diamond\mathscr{W}$ failure detector, see [3]. The consensus algorithm becomes simpler using $\diamond\mathscr{S}$ rather than $\diamond\mathscr{W}$

[4] The early consensus algorithm is proven with a weaker channel assumption: a message sent by a process p_i to a process p_j is eventually received by p_j, if p_j and p_i are both correct

[5] In [3] it is shown that without this assumption consensus cannot be solved using $\diamond\mathscr{S}$ in asynchronous systems

(1) *weak completeness*: eventually every crashed process is permanently suspected by some correct process, and

(2) *eventual weak accuracy*: there is a time after which some correct process is not suspected by any correct process.

Eventually Strong. $\diamond\mathscr{S}$. The $\diamond\mathscr{S}$ failure detector satisfies the two following properties:

(1) *strong completeness*: eventually every crashed process is permanently suspected by every correct process, and

(2) *eventual weak accuracy*: there is a time after which some correct process is not suspected by any correct process.

$\diamond\mathscr{W}$ is the weakest failure detector that makes it possible to solve consensus in an asynchronous system [2]. Moreover, as $\diamond\mathscr{W} \cong \diamond\mathscr{S}$ [3], the same result holds for $\diamond\mathscr{S}$. In the following, we consider the $\diamond\mathscr{S}$ failure detector.

2.4 Latency degree

The cost of a distributed algorithm is sometimes expressed in terms of *number of phases*. Instead of this cost measure, we introduce the notion of *latency degree* that we believe is less ambiguous. The latency degree can easily be defined using a slight variation of Lamport's logical clock [10]. Consider Lamport's logical clock, with the following rules:

- a *send* event and a *local* event on a process p_i do not modify p_i's logical clock value;
- let $ts(send(m))$ be the time-stamp of the $send(m)$ event, and $ts(m)$ the time-stamp carried by message m. We define: $ts(m) \stackrel{\text{def}}{=} ts(send(m)) + 1$;
- the time-stamp of an event $receive(m)$ on a process p_i is the maximum between $ts(m)$, and the time-stamp of the event at p_i immediately preceding the $receive(m)$ event.

We are specifically interested here in algorithms that solve *agreement problems*. Consensus and atomic commitment are typical examples of agreement problems. The events by which processes *decide* play a key role in the runs of agreement algorithms, and we are interested in the time-stamp of the *decision* events, i.e. the time-stamp of the events by which processes decide.

Given a run $\mathscr{R}_{\mathscr{A}}$ generated by an agreement algorithm $\mathscr{A}$, we define the *latency* of $\mathscr{A}$ as the largest time-stamp of all *decide* events (at most one per process) in the run $\mathscr{R}_{\mathscr{A}}$. As an example, consider the following algorithm: (1) process $p_1 \in \Omega$ sends initially a message m to Ω, and (2) every process $p_j \neq p_1$, upon reception of m, sends the message $ack(m)$ to p_1. Process p_1 decides as soon as it receives $ack(m)$ from a majority of processes, and no other process decides. This algorithm is usually called a *one phase algorithm*. It has a latency of 2.

An agreement algorithm $\mathscr{A}$ can generate runs with different latencies. Consider for example the Chandra-Toueg consensus algorithm using $\diamond\mathscr{S}$: depending on the failures, and on the suspicions, the number of rounds needed to complete the algorithm can vary, and thus the latency can also vary from one run to another. We define the *latency degree* of an agreement algorithm $\mathscr{A}$ as the *minimal* latency of $\mathscr{A}$ over all possible runs $\mathscr{R}_{\mathscr{A}}$ that can be produced by $\mathscr{A}$. The minimal latency is typically obtained in a run in which no suspicions are generated, which is the most frequent case.

With this measure, the Chandra-Toueg consensus algorithm using $\diamond\mathscr{S}$ has a latency degree of 4 (which can actually be reduced to a latency degree of 3 by a trivial optimization, see Sect. 4.3), and our early consensus algorithm has a latency degree of 2 (see Sect. 4.3). Moreover, the two phase commit protocol (or 2PC) [1] has a latency degree of 3, and the three phase commit protocol [11] has a latency degree of 5.

3 The Chandra-Toueg vs the early consensus algorithm

In this section we briefly outline the Chandra-Toueg consensus algorithm using the $\diamond\mathscr{S}$ failure detector (or CT algorithm to abbreviate), and sketch our early consensus algorithm. The complete consensus algorithm, its proof, the analysis of its latency degree and of its cost in number of messages, are given in Sect. 4.

3.1 Overview of the CT algorithm

The CT algorithm is based on the rotating coordinator paradigm [3]. The computation proceeds in asynchronous rounds. Every process in Ω knows that, during round r, the coordinator p_c is process number $(r \bmod n) + 1$. In round r, all the messages are sent to, and received from, the current coordinator p_c. Every process p_i maintains a variable $estimate_i$ which denotes p_i's estimate of the decision value: $estimate_i$ is initially set to p_i's initial value v_i, and is updated during the execution of the consensus protocol, until a decision is reached. In every round r, the algorithm is as follows:

1. at the beginning of round r, every process p_i sends $estimate_i$ to the current coordinator p_c;
2. the coordinator p_c waits to receive the estimate from a majority of processes, updates its estimate according to the estimates received (the coordinator chooses the estimate that has been updated in the most recent round), and broadcasts its new estimate;
3. a process p_i waits either (1) to receive the new estimate from the coordinator, or (2) to suspect the coordinator. In the first case, p_i adopts the estimate received, sends *ack* to the coordinator, and proceeds to the next round. In the second case, p_i does not change its estimate, sends a negative acknowledgement *nack* to the coordinator, and also proceeds to the next round;
4. the coordinator waits to receive either *ack* or *nack* from a majority of processes. If the coordinator has received the *acks* from a majority of processes, its current estimate becomes the decision value, and the coordinator reliably broadcasts the decision to all. Otherwise, the coordinator proceeds to the next round.

The algorithm satisfies the following property, which ensures the *Uniform Agreement* property of the uniform consensus problem. Once a majority of processes have adopted $estimate_c$ proposed by the coordinator p_c of round r (i.e. once a majority of processes have sent *ack* to the

coordinator), then the value $estimate_c$ is "locked": no other value can become the decision value.

We discuss the latency degree of the CT algorithm, and its cost in number of messages in Sect. 4, when comparing it with our algorithm.

3.2 Overview of the early consensus algorithm

The early consensus algorithm is also based on the rotating coordinator paradigm, and similarly, every process p_i manages a variable $estimate_i$. In round r, the coordinator p_c tries to impose its estimate as the decision value. However no *acks* are used to reach a decision. Instead, when a process p_i receives $estimate_c$, it forwards $estimate_c$ to all. A process decides on $estimate_c$ as soon as it has received $estimate_c$ from a majority of processes. In other words, the decision can be taken in round r as follows:

1. at the beginning of round r, the coordinator p_c broadcsts its $estimate_c$ value to all;
2. a process p_i receiving $estimate_c$, reissues it to all;
3. as soon as p_i has received $estimate_c$ from a majority of processes, it decides on $estimate_c$.

The protocol indeed terminates in the first round if the first coordinator p_1 is correct and not suspected. If p_1 crashes while in the first round, or p_1 is suspected by a majority of processes, then the processes proceed to a second round. Before proceeding to the next round, the estimates of the processes are updated, so as to satisfy the following property:

> *If some process has decided on* $estimate_c$ *in round* r, *then any process that proceeds to round* $r+1$, *starts round* $r+1$ *with* $estimate_c$ *as its current estimate.*

This property ensures the *Uniform Agreement* property of the uniform consensus problem: if some process has decided $estimate_c$ in round r, then in any round $r' > r$ the decision can be on no other value than $estimate_c$.

4 The early consensus algorithm

4.1 The algorithm

The complete consensus algorithm is given in Fig. 1, as a function *early-consensus*. In order to solve consensus, each process $p_i \in \Omega$ calls *early-consensus* with its initial value v_i as parameter. The call of the function terminates when p_i executes the instruction **return** (either at line 13, or at line 30): we say that p_i decides on a value v exactly when p_i executes **return** v. The function *early-consensus* consists of two concurrent tasks: one task executes lines 11–13, the other task executes lines 14–49.

The first task handles the reception of the decision message (p_j, v_j, *decide*) (line 10), and reissues the message to all (line 11). This ensures that, if a correct process decides, then every correct process eventually also decides.

The second task (lines 14–49) is the central part of the algorithm. The variable r_i indicates the current round of process p_i. The round number r_i is included in every message sent by p_i (see for example line 19); moreover, a process p_i only receives messages carrying a round number equal to its current round number r_i (see for example line 22). Each round of the *early consensus* algorithm is divided into two phases, numbered 1 and 2:

- in phase 1 of every round ρ, the *early consensus* algorithm tries to decide on the estimate value of the coordinator p_c of round ρ;
- if the coordinator of round ρ is suspected by a majority of processes, then phase 2 of round ρ is used to define a new consensus problem, to be solved in round $\rho + 1$. The initial value of process p_i for the consensus problem of round $\rho + 1$ is the estimate of p_i at the end of phase 2 of round ρ.

Phase 1 (*lines 15–30*). At line 19, the coordinator sends its current estimate to all the processes (message (p_i, r_i, 1, $estimate_i$)). The estimate is a pair (*process number*, *initial value*) (see line 3). This representation allows us to attach, to an initial value v_i, the process number i of the process that has proposed v_i. The third field in the message (p_i, r_i, 1, $estimate_i$) indicates that the message is sent during phase 1. The message (p_i, r_i, 1, $estimate_i$) is received by a process p_j at line 22[6]: the condition "**when** $phase_i = 1$" at line 22 states that the message (p_j, r_i, 1, $estimate_j$) can be received by p_i if and only if p_i is in phase 1. At line 25, process p_i adopts the estimate received from p_j, and at line 26 process p_i forwards this estimate to all. The lines 25, 26 are not performed by the coordinator, because (1) the coordinator does not need to adopt its own estimate, and (2) at line 19 the coordinator has already sent its estimate to all.

Process p_i decides at line 30 on $estimate_i.second$ as soon as it has received (p_j, r_i, 1, $estimate_i$) from a majority of processes. At line 29, process p_i sends its decision to all. This ensures that, if p_i is correct, then every correct process eventually also decides.

From phase 1 to phase 2 (*lines 31–37*). If no process suspects the coordinator in phase 1, then the decision value is the estimate of the coordinator. If a process p_i suspects the coordinator at line 31 (notation: $coord_i \in \Diamond\mathscr{S}_i$), then p_i sends (p_i, r_i, *suspicion*) to all (line 32), indicating that p_i suspects the coordinator of round r_i. Once a process p_i knows that the coordinator is suspected by a majority of processes, then p_i proceeds to phase 2 (line 36). Moreover, upon proceeding to phase 2, p_i sends (p_i, r_i, 2, $estimate_i$) to all (line 37). The reception of this message at line 38 forces a process to phase 2 (line 40); upon proceeding to phase 2, every process p_i similarly sends (p_i, r_i, 2, $estimate_i$) to all (line 41). The condition "$phase_i = 1$" at line 40, prevents a process that has already sent (p_i, r_i, 2, $estimate_i$) to all at line 37, from sending the same message twice.

Phase 2 (*lines 38–48*). In phase 2, process p_i receives messages (p_j, r_i, 2, $estimate_j$) (line 38). Upon each reception of such a message, p_i adopts the $estimate_j$ value, if and only if $estimate_j.first = coord_i$ (line 43), i.e. if and only if the estimate is that of the coordinator of the current round. Once p_i has received the message (p_j, r_i, 2, $estimate_j$) from a majority of processes, p_i can switch to phase 1 of the next round (lines 45, 46).

[6] A message sent by p_i to all is also received by p_i

```
1  function early-consensus(v_i) ;      /* algorithm for a process p_i */

2    r_i ← 0                          /* current round */
3    estimate_i ← (i, v_i)            /* the two fields are written "estimate_i.first", respt "estimate_i.second" */
4    phase_i                          /* each round has two phases, numbered 1 and 2 */
5    coord_i                          /* coordinator for round r_i */
6    currentRoundTerminated_i         /* boolean variable */
7    msgCounter_i                     /* integer variable */
8    coordSuspected_i                 /* "true" if and only if coord_i is suspected */
9    nbSuspicions_i                   /* integer variable */

10 cobegin
11 || upon reception of (p_j, r_j, v_j, decide) from p_j :
12      send (p_i, r_j, v_j, decide) to all ;
13      return v_j ;

14 || loop
15      phase_i ← 1 ; currentRoundTerminated_i ← false ;
16      coordSuspected_i ← false ; nbSuspicions_i ← 0 ;
17      coord_i ← (r_i mod n) + 1 ;

18      if i = coord_i then     /* p_i is the coordinator for the current round */
19        send (p_i, r_i, 1, estimate_i) to all ;

20      while not currentRoundTerminated_i
21        select      /* select one of the branches starting at line 22, 31, 34, 38 */

22          upon reception of (p_j, r_i, 1, estimate_j) from p_j when phase_i = 1 :
23            /* estimate_j is the estimate of the coordinator of round r_i */
24            first reception: msgCounter_i ← 1 ;
25                             if i ≠ coord_i then estimate_i ← estimate_j ;
26                                                 send (p_i, r_i, 1, estimate_i) to all ;
27            other receptions: msgCounter_i ← msgCounter_i + 1 ;
28            if msgCounter_i > n/2 then
29               send (p_i, r_i, estimate_i.second, decide) to all ;
30               return estimate_i.second ;

31          upon coord_i ∈ ◇S_i when not coordSuspected_i :
32            send(p_i, r_i, suspicion) to all ;
33            coordSuspected_i ← true ;

34          upon reception of (p_j, r_i, suspicion) from process p_j :
35            nbSuspicions_i ← nbSuspicions_i + 1 ;
36            if nbSuspicions_i > n/2 then phase_i ← 2 ;
37                                         send (p_i, r_i, 2, estimate_i) to all ;

38          upon reception of (p_j, r_i, 2, estimate_j) from p_j :
39            first reception: msgCounter_i ← 1 ;
40                             if phase_i = 1 then phase_i ← 2 ;
41                                                 send (p_i, r_i, 2, estimate_i) to all ;
42            other receptions: msgCounter ← msgCounter_i + 1 ;
43            if estimate_j.first = coord_i then estimate_i ← estimate_j ;
44            if msgCounter_i > n/2 then
45               currentRoundTerminated_i ← true ;
46               r_i ← r_i + 1 ;

47        end select
48      end while
49    end loop
50 coend
```

Fig. 1. Early consensus algorithm: code for a process p_i

4.2 The proofs

4.2.1 Preliminary lemmas

We start by proving five lemmas (Lemmas 4.1 to 4.5) that will help in proving the correctness of the early consensus algorithm of Fig. 1. In these lemmas, we say that *process* p_i *decides* v *in round* ρ of the early consensus algorithm, if either (1) p_i decides at line 30 of Fig. 1 when $r_i = \rho$, or (2) p_i decides at line 13 because at line 11 it received a message $(p_j, \rho, v, decide)$.

Lemma 4.1. *If one correct process decides, then each correct process eventually decides.*

Proof. Let p_i be a correct process that decides, either at line 30 or at line 13. In both cases, p_i sends the decision to all (message $(p_i, r_j, v_j, decide)$ sent at line 12, message $(p_i, r_i, estimate_i.second, decide)$ sent at line 29). By the reliable channel assumption, each correct process that has not yet decided, eventually receives $(p_j, r_j, v_j, decide)$ (line 11), and also decides. □

Lemma 4.2. *Let* $f < n/2$ *and assume the failure detector* $\diamond\mathscr{S}$. *If no correct process decides in round* $r \leqq \rho$, *then each correct process eventually proceeds to round* $\rho + 1$.

Proof. The proof is by induction on the round number ρ.

i) Base step: $\rho = 0$.

Assume that no correct process decides in round 0. We prove the following successive results:

i1) At least one correct process eventually proceeds from phase 1 to phase 2 of round 0.

i2) Each correct process eventually proceeds from phase 1 to phase 2 of round 0.

i3) Each correct process eventually proceeds from phase 2 of round 0 to round 1.

*Proof of i*1). We prove the result by showing that "*no correct process decides in round* 0" and "*no correct process proceeds to phase* 2 *of round* 0" lead to a contradiction.

Assume that no correct process decides in round 0, and that no correct process proceeds to phase 2 of round 0: we first prove that (*a*) there is at least one correct process that never receives any "estimate" message at line 22.

Proof of (*a*). The proof is by contradiction. Assume that every correct process receives $(p_j, 0, 1, estimate_j)$ at line 22: in this case each correct process sends $(p_i, 0, 1, estimate_i)$ to all (line 26). Because $phase_i = 1$ remains true for all correct processes, and no correct process decides in round 0, all correct processes eventually receive more than $n/2$ messages $(p_j, 0, 1, estimate_j)$ at line 22 (recall $f < n/2$). Thus the condition $msgCounter_i > n/2$ (line 28) becomes eventually true for each correct process, i.e. each correct process eventually decides in round 0. A contradiction. Thus (a) holds: there is at least one correct process that never receives any "estimate" message at line 22.

Property (a) can only hold if p_1, the coordinator of round 0, is incorrect (otherwise, the message $(p_i, 0, 1, estimate_i)$ sent at line 19 by p_1 is eventually received by all correct processes at line 22, as by hypothesis no correct process proceeds to phase 2, i.e. $phase_i = 1$ remains true for all correct processes). If p_1 is incorrect, by the *eventual strong completeness* property of $\diamond\mathscr{S}$, process p_1 is eventually permanently suspected by every correct process. Thus every correct process sends $(p_i, r_i, suspicion)$ to all (line 32). As $f < n/2$, the condition $nbSuspicions_i > n/2$ (line 36) eventually becomes true for every correct process, and every correct process eventually proceeds to phase 2 (line 36). A contradiction.

*Proof of i*2). By *i*1), if no correct process decides in round 0, then at least one correct process p_k proceeds to phase 2 of round 0. In this case, p_k sends $(p_k, 0, 2, estimate_k)$ to all (line 37). Because p_k is correct, each correct process that does not move to phase 2 at line 36, eventually receives some "estimate" message at line 38, and proceeds to phase 2 of round 0 (line 40).

*Proof of i*3). By *i*2), if no correct process decides in round 0, then each correct process eventually proceeds to phase 2 of round 0 (line 36 or 40). At line 37 or 41, each correct process sends $(p_i, 0, 2, estimate_i)$ to all. By hypothesis, there are more than $n/2$ correct processes. Thus each correct process receives more than $n/2$ messages $(p_j, 0, 2, estimate_j)$ (line 38), and the condition $msgCounter_i > n/2$ (line 44) eventually becomes true. Thus each correct process eventually proceeds to round 1. This completes the proof of the case $\rho = 0$.

ii) Induction step

Assume that no correct process decides in round $r \leqq \rho + 1$. We have to prove that, in this case, each correct process eventually proceeds to round $\rho + 2$.

If no correct process decides in round $r \leqq \rho + 1$, then trivially no correct process decides in round $r \leqq \rho$. By the induction hypothesis, each correct process eventually proceeds to round $\rho + 1$. We have to prove that, if no correct process decides in round $\rho + 1$, then all the correct processes eventually proceed to round $\rho + 2$.

The proof is identical to the proof of the base case $\rho = 0$ (with $\rho + 1$ instead of 0, and $\rho + 2$ instead of 1), and will thus not be reproduced. The proof consists similarly of the following steps:

ii1) At least one correct process eventually proceeds from phase 1 to phase 2 of round $\rho + 1$.

ii2) Each correct process eventually proceeds from phase 1 to phase 2 of round $\rho + 1$.

ii3) Each correct process eventually proceeds from phase 2 of round $\rho + 1$ to round $\rho + 2$. □

Lemma 4.3. *All messages* $(p_i, \rho, 1, estimate_i)$ *sent during phase* 1 *of round* ρ *carry the* $estimate_c$ *value of the coordinator* p_c *of round* ρ.

Proof. The proof is by induction on the length of the *send-receive* chain of messages $(p_i, \rho, 1, estimate_i)$. The messages $(p_i, \rho, 1, estimate_i)$ are numbered as follows:

– the message $(p_i, \rho, 1, estimate_i)$ sent by the coordinator at line 19 is numbered 0;
– if the message $(p_j, \rho, 1, estimate_j)$ received by p_i at line 22 is numbered k, then the message $(p_i, \rho, 1, estimate_i)$ sent by p_i at line 26 is numbered $k + 1$.

Base step. Trivially, for the message $(p_i, \rho, 1, estimate_i)$ number 0, $estimate_i$ is the estimate of the coordinator of round ρ.

Induction step. Consider a message $(p_i, \rho, 1, estimate_i)$ number $k+1$. This message is sent by some process p_i at line 26, after having received, at line 22, the message $(p_j, \rho, 1, estimate_j)$ number k. By induction hypothesis, this message carries the $estimate_c$ value of the coordinator of round ρ. Therefore, because of line 25, the message $(p_i, \rho, 1, estimate_i)$ also carries the $estimate_c$ value of the coordinator of round ρ. □

Lemma 4.4. *If a process (correct or not) decides v in round ρ, then v is the "$estimate_c.second$" value of the coordinator p_c of round ρ.*

Proof. A process p_j can decide in round ρ either at line 13, or at line 30. However, process p_j can decide at line 13 of round ρ if and only if there is a process p_i that has decided at line 30, as it is not possible for all processes that decide in round ρ to do so at line 13. Consider thus the decision of p_i at line 30. If p_i is the coordinator of round ρ, then $estimate_i.second$ is trivially the $estimate_c.second$ value of the coordinator of round ρ. Otherwise, by line 25, the decision is on the first estimate value received by p_i at line 22. By Lemma 4.3, the value received is the estimate of the coordinator of round ρ. □

Lemma 4.5. *If a process (correct or not) decides v in round ρ, then every process p_i that begins round $\rho+1$ does so with $estimate_i.second = v$.*

Proof. By Lemma 4.4, if v is decided in round ρ, then v is the $estimate_c.second$ value of the coordinator p_c of round ρ.

A process p_j can decide in round ρ either at line 13, or at line 30. However, process p_j can decide at line 13 of round ρ if and only if there is a process p_i that has decided at line 30, as it is not possible for all processes that decide in round ρ to do so at line 13. Consider thus the decision at line 30.

Let p_i be a process that decides v in round ρ at line 30. By Lemma 4.4, v is the $estimate_c.second$ value of the coordinator p_c of round ρ. Moreover, because of line 28, when p_i decides at line 30, a majority of processes in phase 1 have sent $(p_j, \rho, 1, estimate_j)$ to all. By Lemma 4.3, every estimate sent in phase 1 is the estimate of the coordinator p_c of round ρ. Thus when p_i decides at line 30, a majority of processes (including p_i itself) have their estimate equal to the estimate of p_c. Let us call this set $\mathsf{CoordEstimateSet}_\rho$: we have $|\mathsf{CoordEstimateSet}_\rho| > n/2$.

Consider now a process p_k that proceeds to round $\rho+1$. This is only possible after p_k has received the message $(p_j, \rho, 2, estimate_j)$ from a majority of processes, including from itself (line 44). Let us call this set $\mathsf{AuthorizationSet}_k$: we have $|\mathsf{AuthorizationSet}_k| > n/2$.

Altogether we have $|\mathsf{AuthorizationSet}_k| > n/2$ and $|\mathsf{CoordEstimateSet}_\rho| > n/2$, therefore $\mathsf{AuthorizationSet}_k \cap \mathsf{CoordEstimateSet}_\rho \neq \emptyset$. This means that p_k receives the message $(p_j, \rho, 2, estimate_j)$ at line 38 from at least one process in $\mathsf{CoordEstimate}_\rho$, and at line 43 process p_k sets $estimate_k.second$ to v. Thus, when p_k proceeds to round $\rho+1$, we have $estimate_k.second = v$. □

4.2.2 Correctness proof of the early consensus algorithm

We now prove, based on the lemmas of the previous section, that the early consensus algorithm satisfies the Validity, Termination and Uniform Agreement properties.

Proposition 4.6 (Validity). *The early consensus algorithm of Fig. 1 satisfies the Validity property.*

Proof. Suppose, for contradiction, that Validity does not hold. Then some process p_i sets $estimate_i.second$ to a value that is not the proposal of any process. Let ρ be the earliest round in which this happens.

Case 1. Assume that this happens in phase 1 of round ρ, i.e. at line 25. By Lemma 4.3 every estimate sent in phase 1 of round ρ is the estimate of the coordinator p_c of round ρ. If $\rho = 0$, $estimate_c.second$ is the proposal of p_c. A contradiction.

If $\rho > 0$, then $estimate_c.second$ is the estimate of p_c at the end of round $\rho - 1$. As by hypothesis ρ is the earliest round in which some process p_i sets $estimate_i.second$ to a value that is not the proposal of some process, the value $estimate_c.second$ is the proposal of some process. A contradiction.

Case 2. Assume that this happens in phase 2 of round ρ: some process sets for the first time $estimate_i.second$ to a value that is not the proposal of some process in phase 2 of round ρ. This can only occur at line 43, where $estimate_j$ is the estimate received at line 38. Any estimate received at line 38 is sent by some process either at line 37, or at line 41. In both cases, the estimate sent is the estimate of some process in phase 1 of round ρ. Thus some process must have set, in phase 1, its estimate to a value that is not the proposal of some process, in contradiction with the result established by Case 1. □

Proposition 4.7 (Termination). *Let $f < n/2$ and assume the failure detector $\diamond\mathscr{S}$. Then, the early consensus algorithm of Fig. 1 satisfies the Termination property.*

Proof. By the eventual weak accuracy property of the $\diamond\mathscr{S}$ failure detector, there is a time t after which some correct process p_k is not suspected by any correct process. Let ρ be a round such that (i) p_k is the coordinator of ρ, and (ii) every correct process enters round ρ after t (if such a round does not exist, then by Lemma 4.2 one correct process has decided in a round $\rho' < \rho$, and so, by Lemma 4.1, every correct process decides, and the Termination property holds). As $f < n/2$ and no correct process suspects p_k in round ρ, the condition $nbSuspicions_i < n/2$ holds forever for every process p_i in round ρ. Thus no process (correct or not) ever proceeds to phase 2 of round ρ. In round ρ, process p_k sends $(p_k, \rho, 1, estimate_k)$ to all (line 19). As p_k is correct, each correct process eventually receives $(p_k, \rho, 1, estimate_k)$ (line 22). Moreover, at line 26, each correct process sends $(p_i, \rho, 1, estimate_i)$ to all. As $f < n/2$, any correct process that does not decide by receiving a "decide" message (line 11), will receive enough estimates to decide at line 30. □

Proposition 4.8 (Uniform Agreement). *The early consensus algorithm of Fig. 1 satisfies the Uniform Agreement property.*

Proof. Assume that a process p_i (correct or not) decides v in round ρ. We prove that another process p_j cannot decide on a different value.

Assume that p_j decides in round ρ'. If $\rho = \rho'$, by Lemma 4.4, p_i and p_j both decide on the $estimate_c.second$ value of the coordinator p_c of round ρ, i.e. on the same value.

Consider the case $\rho' > \rho$ (if $\rho' < \rho$, rename p_i to p_j, and p_j to p_i). By Lemma 4.5, every process p_i that begins round $\rho' > \rho$, does so with $estimate_i.second = v$. By Lemma 4.4, the value decided by p_j in round ρ' is also v. □

4.3 Latency degree

In the best case (no suspicion, the first coordinator p_1 correct), every correct process p_i decides at line 30 on p_1's proposal, after having received $(p_j, 1, 1, estimate_j)$ at line 22 from a majority of processes:

– the message $(p_1, 1, 1, estimate_1)$ sent initially by p_1 at line 19 carries a time-stamp equal to 1;
– assume that every process $p_i \neq p_1$ receives first, at line 22, the message $(p_1, 1, 1, estimate_1)$ from p_1. Thus every process $p_i \neq p_1$ sends at line 26 the message $(p_i, 1, 1, estimate_i)$, carrying a time-stamp equal to 2;
– every process decides at line 30 after having received the message $(-, 1, 1, estimate_j)$ from a majority of processes, where the messages received carry either a time-stamp equal to 1 (message received from the coordinator p_1), or a time-stamp equal to 2 (message received from another process $p_j \neq p_1$). The early consensus algorithm has thus a latency degree of 2.

In comparison, the latency degree of the CT algorithm is 4, which can however be trivially reduced to 3. Consider the first round of the CT algorithm (Sect. 3.1):

1. the initial estimate sent by every process p_i to the first coordinator p_1 carries a time-stamp equal to 1;
2. the updated estimate of p_1, sent to all, carries a time-stamp equal to 2;
3. the *act* or *nack* sent by every process p_i to the coordinator carries a time-stamp equal to 3;
4. finally, the decision broadcast by the coordinator carries a time-stamp equal to 4.

This gives a latency degree equal to 4. The reader might however notice that, in the first round of the CT algorithm, step 1 can be omitted. The first round can start in step 2 by having p_1 send its proposal to all. This trivial optimization leads to a consensus algorithm with a latency degree of 3.

4.4 Cost analysis

We compare now the number of messages issued by the early consensus algorithm with the number of messages issued by the CT algorithm. In both cases, we consider the best scenario. However, there are two ways to count the number of messages: (1) only the messages needed to reach the decision are counted, or (2) all the messages sent by the algorithm are counted. To understand the difference, consider Fig. 1. In the best case scenario, every correct process decides at line 30. Nevertheless, every process deciding at line 30 has to send, at line 29, the decision message $(p_i, r_i, estimate_i.second, decide)$ to all. In the absence of failures, this last message is not necessary for deciding, but nevertheless has to be sent. Moreover, we distinguish the case of a broadcast network from the case of a point-to-point network: with a network of the first kind, sending a message to all costs only 1 message, whereas with one of the second kind, sending a message to all (i.e. to $n - 1$ other processes) costs $n - 1$ messages.

Broadcast network: *number of messages needed to reach a decision.* In the best case scenario, the early consensus algorithm issues only n messages to reach the decision (one message sent by p_1 to all at line 19, one message sent to all by each of the processes $p_2, \ldots, p_n$, at line 26). In comparison, the optimized CT algorithm issues $n + 1$ messages (one *estimate* sent by p_1 to all, $n - 1$ acknowledgments sent by $p_2, \ldots, p_n$ to p_1, and one *decide* message sent by p_1 to all). To summarize, both algorithms require $O(n)$ messages in the best case scenario.

Point-to-point network: *number of messages needed to reach a decision.* In the case of a point-to-point network, the early consensus algorithm requires $n(n - 1)$ messages (n "send to all"), whereas the optimized CT algorithm requires 3 $(n - 1)$ messages ($(n - 1)$ *estimates* sent by p_1 to all, $(n - 1)$ acknowledgments sent to p_1 by $p_2, \ldots, p_n$, and $(n - 1)$ *decide* messages sent by p_1 to all).

To summarize, in a point-to-point network the early consensus algorithm costs $O(n^2)$ messages in the best case scenario, whereas the CT algorithm costs $O(n)$ messages.

Broadcast network: *total number of messages sent.* When counting the total number of messages sent by both algorithms, the "decision" messages have to be added to the previous numbers:

– in the early consensus algorithm, the "decision" message is sent by every process p_i to all, at line 29. This adds n messages in a broadcast network, leading to an overall cost of $2n - 1$ messages.
– in the CT algorithm, the "decision" message is sent by p_1 to all using a reliable broadcast. Reliable broadcast is implemented by having every process that receives the decision, reissue the decision to all [3]. This is identical to the additional cost of the early consensus algorithm, i.e. n messages, leading to an overall cost of $2n + 1$ messages.

Thus both algorithms still require $O(n)$ messages in the best case scenario.

Point-to-point network: *total number of messages sent.* In a point-to-point network, the additional "decision" messages, sent by every process to all, lead to an additional cost of $n(n - 1)$ messages for both algorithms. Thus, in the best case scenario, we have an overall cost of $O(n^2)$ messages for both algorithms (more precisely, $2n(n - 1)$ messages for the early consensus algorithm, and $(3 + n)(n - 1)$ for the CT algorithm).

5 Conclusion

The paper has presented a new algorithm for solving consensus in an asynchronous system using the failure

detector $\diamond\mathscr{S}$. In the best case scenario, the algorithm solves consensus in two communication steps, which is called a "*latency degree* of 2". In comparison, the Chandra-Toueg consensus algorithm has a latency degree of 4, which can be improved through a trivial optimization to a latency degree of 3. Thus our early consensus algorithm requires one less communication step than the Chandra-Toueg consensus algorithm.

This result is not only of theoretical interest. It is also of practical consequence for consensus related problems, e.g. *atomic broadcast*, also called *total order broadcast* (see for example [9]). Consider the atomic broadcast of message m to Ω, initiated by $p_i \in \Omega$. The reduction of atomic broadcast to consensus requires 1 communication step, needed to broadcast m to Ω [3]. In other words, when using the early consensus algorithm, atomic broadcast has a latency degree of 3: 1 communication step for the reduction of atomic broadcast to consensus, and two communication steps for consensus.

To conclude, we think that the existence of a consensus algorithm with a low latency degree should contribute to demystify consensus. Hopefully, this will lead to consider consensus as it should be, i.e. as a basic building block for implementing fault-tolerant distributed systems, rather than just an interesting problem for theoreticians.

Acknowledgements. I would like to thank Rachid Guerraoui, Julie Vachon and the anonymous reviewers for their comments and suggestions that helped improve the paper.

References

1. Bernstein PA, Hadzilacos V, Goodman N: Concurrency control and recovery in distributed database systems. Addison-Wesley, New York 1987
2. Chandra TD, Hadzilacos V, Toueg S: The weakest failure detector for solving consensus. J ACM 43(4): 685–722 (1996)
3. Chandra TD, Toueg S: Unreliable failure detectors for reliable distributed systems. J ACM 43(2): 225–267 (1996). A preliminary version appeared in the Proceedings of the Tenth ACM Symposium on Principles of Distributed Computing, pp 325–340. ACM Press, August 1991
4. Chor B, Dwork C: Randomization in byzantine agreement. In: Micali S (ed) Advances in computing research, randomness in computation, vol 5, pp 443–497. JAI Press, 1989
5. Dolev D, Dwork C, Stockmeyer L: On the minimal synchrony needed for distributed consensus. J ACM 34(1): 77–97 (1987)
6. Dwork C, Lynch N, Stockmeyer L: Consensus in the presence of partial synchrony. J ACM 35(2): 288–323 (1988)
7. Fischer M, Lynch N, Paterson M: Impossibility of distributed consensus with one faulty process. J ACM 32: 374–382 (1985)
8. Guerraoui R: Revisiting the relationship between non-blocking atomic commitment and consensus. In: 9th International Workshop on Distributed Algorithms (WDAG-9) LNCS 972, pp 87–100. Springer, Berlin Heidelberg New York 1995
9. Hadzilacos V, Toueg S: Fault-tolerant broadcasts and related problems. In: Mullender S (ed) Distributed systems, pp 97–145. ACM Press, 1993
10. Lamport L: Time, clocks, and the ordering of events in a distributed system. Commun ACM 21(7): 558–565 (1978)
11. Skeen D: Nonblocking commit protocols. In: ACM SIGMOD International Conference on management of data, pp 133–142. ACM, 1981

André Schiper has been a professor of Computer Science at the EPFL (Federal Institute of Technology in Lausanne) since 1985, leading the Operating Systems laboratory. During the academic year 1992–93 he was on sabbatical leave at Cornell University, Ithaca (NY). He was the program chair of the 1993 International Workshop on Distributed Algorithm (WDAG-7), and co-organizer of the International Workshop "Unifying Theory and Practice in Distributed Systems" (Schloss Dagstuhl, Germany, September 1994). He is a member of the ESPRIT Basic Research Network of Excellence in Distributed Computing Systems Architectures (CaberNet). His current research interests are in the areas of fault-tolerant distributed systems and group communication, which has led to the development of the *Phoenix* group communication middleware.