Revisiting the relationship between non-blocking atomic commitment and consensus*

Rachid Guerraoui

Département d'Informatique
Ecole Polytechnique Fédérale de Lausanne
1015 Lausanne, Switzerland
e-mail: guerraoui@di.epfl.ch

Abstract. This paper discusses the relationship between the *Non-Blocking Atomic Commitment problem (NB-AC)* and the Consensus problem in asynchronous systems with *unreliable* failure detectors. We first confirm that NB-AC is harder than Consensus. In contrast to Consensus, NB-AC is impossible to solve with unreliable failure detectors even with a single crash failure. We define a weaker problem than NB-AC, called *Non-Blocking Weak Atomic Commitment (NB-WAC)*, which is sufficient to solve for most practical situations. A fundamental characteristic of NB-WAC is its *reducibility* to Consensus. The previous results on solving Consensus with unreliable failure detectors apply therefore to NB-WAC. An interesting intermediate result of this reducibility is that Uniform Consensus and Consensus are equivalent problems. We show actually that any algorithm that solves Consensus with unreliable failure detectors also solves Uniform Consensus.

1 Introduction

To ensure transaction failure atomicity in a distributed system, an agreement problem must be solved among a set of participating processes. This problem, called the Atomic Commitment problem (AC), requires the participants to agree on an outcome for the transaction: *commit* or *abort*. When it is required that every correct participant eventually reach an outcome despite the failure of other participants, the problem is called *Non-Blocking Atomic Commitment (NB-AC)*. Solving this problem enables correct participants to relinquish resources (e.g locks) without waiting for crashed participants to recover. The *Two Phase Commit (2PC)* algorithm, for example, solves AC but not NB-AC [2], whereas the *Three Phase Commit* algorithm of [15] solves NB-AC in synchronous systems (when communication delays and process relative speeds are bounded). In this paper we compare the NB-AC problem and the Consensus problem in

* Appears in the proceedings of the International Workshop on Distributed Algorithms, Springer Verlag (LNCS), 1995.

asynchronous systems with crash failures and reliable channels, augmented with (possibly unreliable) failure detectors [4].

Consensus and NB-AC are similar problems in that they are both non-blocking agreement problems. The so-called *FLP impossibility result*, which states that it is impossible to solve any non-trivial agreement in an asynchronous system even with a single crash failure, applies to both problems [7]. The starting point of this paper is the fundamental result of Chandra and Toueg [4], which states that Consensus is solvable in asynchronous systems with unreliable failure detectors. An interesting question is then whether NB-AC can also be solved in asynchronous systems with unreliable failure detectors.

The answer to this question is "No", and this is not surprising because the NB-AC problem has been considered harder than Consensus [6, 12]. However, in contrast to initial intuition, the reason NB-AC is harder than Consensus is not its *Uniform Agreement* condition[2]. We show that Uniform Consensus (Consensus + *Uniform Agreement*) and Consensus are equivalent problems with respect to unreliable failure detectors. The difficulty in solving NB-AC is actually its *Non-Triviality* condition (*commit* must be decided if all participants vote *yes*, and *there is no failure*). This condition, usually only intended to avoid trivial solutions to the problem, requires precise knowledge about failures which unreliable failure detectors cannot provide.

Nevertheless, with a weaker non-triviality condition (*commit* must be decided if all participants vote *yes*, and *no participant is ever suspected*), we define a problem weaker than NB-AC, called *NB-WAC (Non-Blocking Weak Atomic Commitment)*. This problem is in fact adequate in real-world transactional systems. A fundamental characteristic of NB-WAC is that it is *reducible* to Consensus in asynchronous systems with unreliable failure detectors, i.e. whenever Consensus is solvable, NB-WAC is also solvable. The results of Chandra and Toueg on solving Consensus with unreliable failure detectors [4] therefore apply to NB-WAC: (1) NB-WAC is solvable with failure detector class $\mathcal{S}$ if at least one participant is correct, and (2) NB-WAC is solvable with failure detector class $\diamond\mathcal{S}$ if there is a majority of correct participants.

The rest of the paper is organized as follows. In Section 2 we describe our system model. In Section 3 we define NB-AC and show that it is harder than Consensus. In Section 4 we show that Consensus and Uniform Consensus are equivalent with respect to unreliable failure detectors. In Section 5 we define NB-WAC and show that it is reducible to Consensus. Finally, Section 6 summarizes the main contributions of this paper and discusses related and future work.

2 Model

Our model of asynchronous computation with failure detection is the one described in [4]. In the following, we only recall some informal definitions and

[2] The *Uniform Agreement* condition forbids any two participants (correct or not) to decide differently. NB-AC requires *Uniform Agreement* whereas Consensus requires only *Agreement* (two *correct* participants cannot decide differently).

results that are needed in this paper.

2.1 Processes

We consider a distributed system composed of a finite set of processes $\Omega = \{p_1, p_2, \ldots, p_n\}$ completely connected through a set of channels. Communication is by message passing, *asynchronous* and *reliable*. Processes fail by crashing; Byzantine failures are not considered. Asynchrony means that there is no bound on communication delays or process relative speeds. A reliable channel ensures that a message, sent by a process p_i to a process p_j, is eventually received by p_j, if p_i and p_j are correct (i.e. do not crash). To simplify the presentation of the model, it is convenient to assume the existence of a discrete global clock. This is merely a fictional device inaccessible to processes. The range of clock ticks is the set of natural numbers. A history of a process $p_i \in \Omega$ is a sequence of events $h_i = e_i^0 \cdot e_i^1 \cdots e_i^k$, where e_i^k denotes an event of process p_i occured at time k. Histories of correct processes are infinite. If not infinite, the process history of p_i terminates with the event $crash_i^k$ (process p_i crashes at time k). Processes can fail at any time, and we use f to denote the number of processes that may crash. We consider systems where at least one process is correct (i.e. $f < |\Omega|$).

A failure detector is a distributed oracle which gives hints on failed processes. We consider algorithms that use failure detectors. An algorithm defines a set of *runs*, and a run of algorithm A using a failure detector $\mathcal{D}$ is a tuple $R =< F, H_{\mathcal{D}}, I, S, T >$: I is an initial configuration of A; S is an infinite sequence of events of A (made of process histories); T is a list of increasing time values indicating when each event in S occured; F is a failure pattern that denotes the set $F(t)$ of processes that have crashed at any time t; H is a failure detector history, which gives to each process p and at any time t, a (possibly false) view $H(p,t)$ of the failure pattern: $H(p,t)$ denotes a set of processes, and $q \in H(p,t)$ means that process p *suspects* process q at time t.

2.2 Failure detector classes

Failure detectors are abstractly characterized by *completeness* and *accuracy* properties [4]. Completeness characterizes the degree to which crashed processes are permanently suspected by correct processes. Accuracy restricts the false suspicions that a process can make. Two completeness properties have been identified. *Strong Completeness*, i.e. there is a time after which every process that crashes is permanently suspected by *every* correct process, and *Weak Completeness*, i.e. there is a time after which every process that crashes is permanently suspected by *some* correct process. Four accuracy properties have been identified. *Strong Accuracy*, i.e. no process is suspected before it crashes; *Weak Accuracy*, i.e. some correct process is never suspected; *Eventual Strong Accuracy*, i.e. there is a time after which correct processes are not suspected by any correct process; and *Eventual Weak Accuracy*, i.e. there is a time after which some correct process is never suspected by any correct process.

Completeness	Accuracy			
	Strong	**Weak**	**◇ Strong**	**◇ Weak**
Strong	P	S	$\diamond P$	$\diamond S$
Weak	Q	W	$\diamond Q$	$\diamond W$

Fig. 1. Failure detector classes

A failure detector class is a set of failure detectors characterized by the same completeness and the same accuracy properties (Figure 1). For example, the failure detector class $\mathcal{P}$ is the set of failure detectors characterized by *Strong Completeness* and *Strong Accuracy*. Failure detectors characterized by *Strong Accuracy* are *reliable*: no false suspicions are made. Otherwise, they are *unreliable*. For example, failure detectors of $\mathcal{S}$ are *unreliable*, whereas failure detectors of $\mathcal{P}$ are *reliable*.

2.3 Reducibility and transformation

An algorithm A *solves* a problem B if every run of A satisfies the specification of B. A problem B is said to be *solvable with* a class $\mathcal{C}$ if there is an algorithm which solves B using any failure detector of $\mathcal{C}$. A problem B^1 is said to be *reducible to* a problem B^2 with class $\mathcal{C}$, if any algorithm that solves B^2 with $\mathcal{C}$ can be transformed to solve B^1 with $\mathcal{C}$. If B^1 is not reducible to B^2, we say that B^1 is *harder than* B^2.

A failure detector class $\mathcal{C}^1$ is said to be *stronger than* a class $\mathcal{C}^2$, (written $\mathcal{C}^1 \succeq \mathcal{C}^2$), if there is an algorithm which, using any failure detector of $\mathcal{C}^1$, can emulate a failure detector of $\mathcal{C}^2$. Hence if $\mathcal{C}^1$ is stronger than $\mathcal{C}^2$ and a problem B is solvable with $\mathcal{C}^2$, then B is solvable with $\mathcal{C}^1$. The following relations are obvious: $\mathcal{P} \succeq \mathcal{Q}$, $\mathcal{P} \succeq \mathcal{S}$, $\diamond\mathcal{P} \succeq \diamond\mathcal{Q}$, $\diamond\mathcal{P} \succeq \diamond\mathcal{S}$, $\mathcal{S} \succeq \mathcal{W}$, $\diamond\mathcal{S} \succeq \diamond\mathcal{W}$, $\mathcal{Q} \succeq \mathcal{W}$, and $\diamond\mathcal{Q} \succeq \diamond\mathcal{W}$. As it has been shown that any failure detector with *Weak Completeness* can be transformed into a failure detector with *Strong Completeness* [4], we also have the following relations: $\mathcal{Q} \succeq \mathcal{P}$, $\diamond\mathcal{Q} \succeq \diamond\mathcal{P}$, $\mathcal{W} \succeq \mathcal{S}$, and $\diamond\mathcal{W} \succeq \diamond\mathcal{S}$. Classes $\mathcal{S}$ and $\diamond\mathcal{P}$ are incomparable.

2.4 Consensus

In the Consensus problem (or simply Consensus), every participant *proposes* an input value, and correct participants must eventually *decide* on some common output value. Consensus is specified by the following conditions. *Agreement:* no two correct participants decide different values; *Uniform-Validity:* if a participant decides v, then v must have been proposed by some participant; *Termination:* every correct participant eventually decides. Chandra and Toueg have stated the following two fundamental results [4] :

1. If $f < |\Omega|$, Consensus is solvable with $\mathcal{S}$.
2. If $f < \lceil|\Omega|/2\rceil$, Consensus is solvable with $\diamond\mathcal{S}$.

3 NB-AC is harder than consensus

In this Section, we show that the Non-Blocking Atomic Commitment problem (or simply NB-AC) is not solvable in asynchronous systems with unreliable failure detectors. This impossibility result holds even with the assumption that at most one process may crash. Hence NB-AC is harder than Consensus.

3.1 The Non-Blocking Atomic Commitment problem

Atomic commitment problems are at the heart of distributed transactional systems. A transaction originates at a process called the Transaction Manager (abbreviated TM), which accesses data by interacting with various processes called Data Managers (abbreviated DM). The TM initially performs a *begin-transaction* operation, then various *write* and *read* operations (by translating writes and reads into messages sent to the DMs), and finally an *end-transaction* operation. To ensure the so-called *failure atomicity* property of the transaction, all DMs on which write operations have been performed, must resolve an *Atomic Commitment* problem (as part of the *end-transaction* operation). These DMs are called participants in the problem. In this paper we assume that the participants know each other, and know about the transaction [1].

The atomic commitment problem requires the participants to *reach* a common outcome for the transaction among two possible values: *commit* and *abort*. We will say that a participant *AC-decides commit* (respectively *AC-decides abort*). The write operations performed by the DMs become permanent if and only if participants AC-decide *commit*. The outcome AC-decided by a participant depends on *votes (yes* or *no)* provided by the participants. We will say that a participant *votes yes* (respectively *votes no*). Each vote reflects the ability of the participant to ensure that its data updates can be made permanent. We do not make any assumption on how votes are defined, except that they are not predetermined. For example, a participant votes *yes* if and only if no concurrency control conflict has been locally detected, and the updates have been written to stable storage. Otherwise the participant votes *no*. A participant can AC-decide *commit* only if all participants vote *yes*. In order to exclude trivial situations where participants always AC-decide *abort*, it is generally required that *commit* must be decided if all votes are *yes* and *no participant crashes* [2].

We consider the *Non-Blocking Atomic Commitment problem (NB-AC)* in which a correct participant AC-decides even if some participants have crashed. NB-AC is specified by the following conditions:

- **Uniform-Agreement:** No two participants AC-decide different outcomes.
- **Uniform-Validity:** If a participant AC-decides *commit*, then all participants have voted *yes*.

- **Termination:** Every correct participant eventually AC-decides.
- **NonTriviality:** If all participants vote *yes*, and there is no failure, then every correct participant eventually AC-decides *commit*.

Uniform-Agreement and *Uniform-Validity* are safety conditions. They ensure the *failure atomicity* property of transactions. *Termination* is a liveness condition which guarantees non-blocking. *NonTriviality* excludes trivial solutions to the problem where participants always AC-decide *abort*. This condition can be viewed as a liveness condition from the application point of view since it ensures progress (i.e. transaction commit) under reasonable expectations: when no crash and no participant votes *no*.

3.2 Impossibility of solving NB-AC

We show that NB-AC is harder than Consensus since even when assuming a single crash, unreliable failure detectors are not strong enough to solve NB-AC. We state this result for classes $\mathcal{S}$ and $\diamond\mathcal{P}$ (Sect 2.3). Hence the result holds for $\diamond\mathcal{S}$, $\diamond\mathcal{Q}$, $\mathcal{W}$, and $\diamond\mathcal{W}$.

Theorem 1. *If $f > 0$, NB-AC cannot be solved with either $\diamond\mathcal{P}$ or $\mathcal{S}$.*

PROOF. (By contradiction[3]). Consider an algorithm A which solves NB-AC using any failure detector of $\diamond\mathcal{P}$ (respectively of $\mathcal{S}$). Consider a failure detector $\mathcal{D}$ of $\diamond\mathcal{P}$ (respectively of $\mathcal{S}$) and a run $R =< F, H_\mathcal{D}, I, S, T >$ of A. In R, all participants vote *yes*. One participant p_1 crashes immediately without sending any message, and all other participants are correct. Consider a correct participant p_2. If p_2 does not AC-decide, then the *Termination* condition of NB-AC is violated in run R: a contradiction. Assume thus a time t at which p_2 AC-decides either (1) *commit* or (2) *abort*. Consider both cases:

1. *p_2 AC-decides commit at time t.* Consider a run $R^1 =< F, H_\mathcal{D}, I^1, S, T >$ of A, identical to R, except that p_1 votes *no* (instead of *yes*). Participant p_2 executes exactly the same events in R^1 as in R, and AC-decides *commit* at time t (R^1 is indistinguishable from R to p_2). As one participant (p_1) has voted *no*, the *Uniform-Validity* condition of NB-AC is violated in run R^1 of A: a contradiction.
2. *p_2 AC-decides abort at time t.* Consider a run $R^2 =< F^2, H^2_\mathcal{D}, I^2, S^2, T^2 >$ of A. In R^2, all participants (including p_1) are correct, and all messages from p_1 are delayed until after $t' > t$. Assume that $H^2_\mathcal{D}$ is identical to $H_\mathcal{D}$, except that after $t' > t$, $p1$ is never suspected in $H^2_\mathcal{D}$. As no participant crashes in R_3, then $H^2_\mathcal{D}$ satisfies *Strong Completeness*. Consider accuracy.
   - If $\mathcal{D}$ is of class $\diamond\mathcal{P}$, $H_\mathcal{D}$ satisfies *Eventual Strong Accuracy*, i.e. there is a time after which correct participants are never suspected by any correct participant. As $H^2_\mathcal{D}$ is identical to $H_\mathcal{D}$, except that p_1 is never suspected by any participant after time t', then $H^2_\mathcal{D}$ satisfies *Eventual Strong Accuracy*.

[3] The intuitive idea of this proof was given in [5].

– If $\mathcal{D}$ is of class $\mathcal{S}$, $H_{\mathcal{D}}$ satisfies *Weak Accuracy*, i.e. some correct participant p_k is never suspected in $H_{\mathcal{D}}$. As $p_k \neq p_1$ (p_1 crashes in R), then p_k is never suspected in $H^2_{\mathcal{D}}$. Hence $H^2_{\mathcal{D}}$ satisfies *Weak Accuracy.*

Until time t, participant p_2 executes exactly the same events as in R and AC-decides *abort* at t (until time t, R^2 is indistinguishable from R to p_2). As all participants are correct and all have voted *yes*, the *NonTriviality* condition of NB-AC is violated in run R^2 of A: a contradiction. □

By the relations between failure detector classes (Sect. 2.3), we have the following Corollary.

Corollary 1. *If $f > 0$, NB-AC is not solvable with either $\diamond\mathcal{Q}$, $\mathcal{S}$, $\diamond\mathcal{S}$, or $\diamond\mathcal{W}$.*

Intuitively, the reason why NB-AC is not solvable with unreliable failure detectors is that NB-AC requires precise knowledge about failures. Assume a participant p which neither knows that all participants have voted *yes*, nor that some participant has voted *no*. Participant p cannot wait indefinitely for the votes of all participants (some may have crashed), and p cannot AC-decide *abort* unless it knows that some participant has crashed. An unreliable failure detector (which can make false failure suspicions) does not give p such knowledge and is therefore not strong enough to solve NB-AC. The need for precise knowledge about failures is contained in the *NonTriviality* condition of NB-AC. It is surprising that a condition which is intended to eliminate trivial solutions, introduces a significant difficulty in the problem. In Section 5, we weaken the *NonTriviality* condition, still precluding trivial solutions, so that the new weaker problem has solutions with unreliable failure detectors.

4 Uniform Consensus equivalent to Consensus

Generally, Atomic Commitment problems have been considered harder than Consensus because of their *Uniform Agreement* condition (not because of their *NonTriviality* condition) [6, 12]. Broadly speaking, Consensus enables two participants to decide differently as long as at least one of them crashes, whereas Atomic Commitment problems forbid two participants from ever AC-deciding differently (whether they crash or not).

In what follows, we show that in asynchronous systems with unreliable failure detectors, *Uniform Consensus* (Consensus + *Uniform Agreement*) is *reducible* to Consensus, i.e whenever Consensus is solvable, Uniform Consensus is also solvable.

4.1 Uniform Consensus reducible to Consensus with unreliable failure detectors

The Uniform Consensus problem is specified by the *Uniform-Validity* and *Termination* conditions of Consensus (Sect 2.4), and the following *Uniform-Agreement* condition:

– **Uniform-Agreement:** No two participants (correct or not) decide different values.

First, we consider unreliable failure detector classes characterized by *Strong Completeness*. These are $\diamond\mathcal{P}$, $\mathcal{S}$, and $\diamond\mathcal{S}$ (Figure 1). We will come back in Corollary 2 to classes characterized by *Weak Completeness*.

To show that Uniform Consensus is reducible to Consensus with $\diamond\mathcal{P}$ (respectively $\mathcal{S}$, $\diamond\mathcal{S}$), it suffices to show that, if there is an algorithm A that solves Consensus with $\diamond\mathcal{P}$ (respectively $\mathcal{S}$, $\diamond\mathcal{S}$), then we can construct an algorithm A' that solves Uniform Consensus with $\diamond\mathcal{P}$ (respectively $\mathcal{S}$, $\diamond\mathcal{S}$). Theorem 2 below is even stronger as it claims that A' is A itself.

Theorem 2. *Any algorithm that solves Consensus with $\diamond\mathcal{P}$ (respectively $\mathcal{S}$, $\diamond\mathcal{S}$), also solves Uniform Consensus with $\diamond\mathcal{P}$ (respectively $\mathcal{S}$, $\diamond\mathcal{S}$).*

PROOF. We show that there is a failure detector $\mathcal{D}$ of $\diamond\mathcal{P}$ (respectively of $\mathcal{S}$, $\diamond\mathcal{S}$) such that, if an algorithm A using $\mathcal{D}$ has a run R where Uniform Consensus is not solved, A has also a run R^1 where Consensus is not solved.

Consider a run $R =< F, H_{\mathcal{D}}, I, S, T >$ of A such that the *Uniform Agreement* condition is not satisfied in R but the specification of Consensus is satisfied (otherwise it is obvious that R^1 is R itself). In run R, two participants p_i and p_j decide different values and at least one of them crashes, say p_i. Assume p_i decides v_i at time t_i, and p_j decides v_j at time t_j ($v_i \neq v_j$).

Consider a run $R^1 =< F^1, H^1_{\mathcal{D}}, I^1, S^1, T^1 >$ with the same failure pattern as in R, except that p_i and p_j are correct in R^1. Delay in R^1 the reception of all messages from p_i and p_j, not received in R before $max(t_i, t_j)$, until $t' > max(t_i, t_j)$. Assume that $H^1_{\mathcal{D}}$ is identical to $H_{\mathcal{D}}$ until t', and after t' no correct participant is ever suspected and every participant that crashes is permanently suspected. It is thus clear that $H^1_{\mathcal{D}}$ satisfies *Strong Completeness*, *Eventual Strong Accuracy* and *Eventual Weak Accuracy*. As $H^1_{\mathcal{D}}$ does not contain any suspicion other than those in $H^1_{\mathcal{D}}$, if $H_{\mathcal{D}}$ satisfies *Weak Accuracy* (i.e. if $\mathcal{D}$ is of $\mathcal{S}$), $H^1_{\mathcal{D}}$ also satisfies *Weak Accuracy*.

In run R^1, participant p_i executes the same events as in R until time t_i, and decides v_i (until t_i, run R^1 is indistinguishable from R to p_i). Similarly, participant p_j executes the same events as in R until t_j, and decides v_j (until time t_j, R^1 is indistinguishable from R to p_j). Hence in run R^1, two correct participants decide differently. □

It is worthwhile to note that the algorithms described in [4], which was initially designed to solve Consensus with $\mathcal{S}$ and $\diamond\mathcal{S}$, also solve Uniform Consensus.

By the relations between failure detector classes (Sect. 2.3), we have Corollary 2 below.

Corollary 2. *Uniform Consensus is reducible to Consensus with $\diamond\mathcal{P}$ (respectively $\diamond\mathcal{Q}$, $\mathcal{S}$, $\mathcal{W}$, $\diamond\mathcal{S}$, $\diamond\mathcal{W}$).*

Corollary 3 follows from the previous results on solving Consensus (Sect 2.4) and the relations between failure detector classes (Sect. 2.3).

Corollary 3. *If* $f < |\Omega|$, *Uniform Consensus is solvable with either* $\mathcal{S}$ *or* $\mathcal{W}$, *and if* $f < \lceil |\Omega|/2 \rceil$, *Uniform Consensus is solvable with either* $\diamond\mathcal{S}$, $\diamond\mathcal{P}$, $\diamond\mathcal{Q}$, $\mathcal{W}$, *or* $\diamond\mathcal{W}$.

4.2 Uniform Consensus versus Consensus with reliable failure detectors

In this section we show that Theorem 2 does not hold with $\mathcal{P}$ (hence it does not hold with $\mathcal{Q}$).[4] We give an algorithm A that solves Consensus with any failure detector of $\mathcal{P}$, but there is a failure detector $\mathcal{D}$ of $\mathcal{P}$, such that A does not solve Uniform Consensus using $\mathcal{D}$. The algorithm is described by the function *consensus(v_i)* in Figure 2, called by every participant p_i, where v_i represents the input value *proposed* by p_i. Function *consensus()* terminates by the execution of a "**return** *outcome*" statement, where *outcome* is the decision value (line 8): when p_i executes **return** *outcome*, p_i decides *outcome*. Participant p_i is informed by its local failure detector module, $\mathcal{D}_i$, of failure suspicions: the notation $p_j \in \mathcal{D}_i$ (line 3) indicates that p_i suspects p_j.

```
  function consensus(v_i)
1     j:=1 ;
2     while j < i
3         wait until [received (p_j, v_j, decide) or p_j ∈ D_i] ;
4             if received (p_j, v_j, decide) then
5                 v_i:=v_j ;
6         j:=j + 1 ;
7     send (p_i, v_i, decide) to all ;
8     return v_i ;
```

Fig. 2. An algorithm which solves Consensus but not Uniform Consensus

The basic idea of the algorithm is the following. Participant p_1 immediately sends the decision message $(p_1, v_1, decide)$ (line 7) (bypassing lines 2-6), and decides v_1 (line 8). If p_2 does not suspect p_1 before it receives $(p_1, v_1, decide)$, then p_2 adopts v_1 (line 5). Then p_2 sends $(p_2, v_2, decide)$ $(v_1 = v_2)$ to all (line 7), and decides v_1 (as did p_1) (line 8). Participant p_3 waits until it receives $(p_1, v_1, decide)$ or it suspects p_1, and it receives $(p_2, v_2, decide)$ or it suspects p_2. In the case where p_3 receives both messages and $v_1 \neq v_2$, p_3 adopts v_2. More generally, participants decide on the input value proposed by the participant p_k, such that k is the smallest index among participants that are never suspected.

[4] Note that this does not mean that Uniform Consensus is not reducible to Consensus.

Theorem 3.1. *The algorithm in Figure 2 solves Consensus with $\mathcal{P}$.*

PROOF. We show that the three conditions of Consensus are satisfied.

1. *Agreement.* Assume that a correct participant p_i decides v. As p_i is correct, then by the *Strong Accuracy* property of $\mathcal{D}$, no participant suspects p_i. Since p_i must have sent its decision message $(p_i, v, decide)$ (line 7) before deciding (line 8), then by the reliable channels assumption, every correct participant receives $(p_i, v, decide)$. Hence every participant p_k such that $i < k$ must have received $(p_i, v, decide)$ (line 4) before deciding. Thus p_k decides v (line 5).
2. *Validity.* Every outcome decided by some participant is, by construction, an input proposed by some participant (line 1).
3. *Termination.* We show by induction on i that every correct participant p_i eventually decides. If p_1 is correct then it sends a decision message (line 7) and decides (line 8). Assume that for every $k > 1$, if p_k is correct then it eventually decides, and consider p_{k+1}. As every correct participant must have sent a decision message (line 7) before deciding (line 8), then by the *Strong Completeness* property of $\mathcal{D}$ and the reliable channels assumption, p_{k+1} cannot remain blocked undefinitely at the while statement of line 2. For every $j < (k+1)$, either p_{k+1} receives the decision message $(p_j, v_j, decide)$, or p_{k+1} suspects p_j. Consequently, if p_{k+1} is correct, then p_{k+1} eventually decides. □

Theorem 3.2. *The algorithm in Figure 2 does not solve Uniform Consensus with $\mathcal{P}$.*

PROOF. We show that there is a failure detector $\mathcal{D}$ of $\mathcal{P}$, and a run of the algorithm where the *Uniform Agreement* condition is violated. Consider the run $R =< F, H_{\mathcal{D}}, I, S, T >$ such that p_1 and p_2 have different input values. Assume that in $H_{\mathcal{D}}$, every process that crashes is permanently suspected and no correct participant is ever suspected. Hence $H_{\mathcal{D}}$ satisfies *Strong Completeness* and *Strong Accuracy*. Assume that p_1 crashes immediatly after deciding v_1 (line 8), and its decision message $(p_1, v_1, decide)$ (line 7) never arrives at p_2. Hence p_2 suspects p_1 and decides v_2 ($v_2 \neq v_1$). Thus p_1 and p_2 decide differently in run R. □

By the relations between failure detector classes $\mathcal{P}$ and $\mathcal{Q}$, Theorem 3.1 and Theorem 3.2 apply also to $\mathcal{Q}$. Clearly, the algorithm in Figure 2 does not solve Consensus with unreliable failure detectors. Indeed, if p_1 and p_2 are correct and p_2 falsely suspect p_1, both may decide different values.

5 NB-WAC reducible to Consensus

In this Section, we define the Non-Blocking Weak Atomic Commitment problem (or simply NB-WAC) by weakening the *NonTriviality* condition of NB-AC. Then we show that in asynchronous systems with unreliable failure detectors, NB-WAC is reducible to Uniform Consensus. This implies, by Theorem 2, that NB-WAC is also reducible to Consensus.

5.1 The Non Blocking Weak Atomic Commitment problem

NB-WAC is specified by the *Uniform-Agreement*, *Uniform-Validity* and *Termination* conditions of NB-AC (Sect 3.1) and by the following *NonTriviality* condition:

- **NonTriviality:** If all participants vote *yes*, and no participant is ever suspected, then every correct participant eventually AC-decides *commit*.

As all failure detector classes we consider ensure *Weak Completeness* (every participant that crashes is eventually suspected), the *NonTriviality* condition of NB-WAC is weaker than the *NonTriviality* condition of NB-AC. Nevertheless, the *NonTriviality* condition of NB-WAC still eliminates trivial solutions to the problem where participants always AC-decide *abort*.[5] Note that an algorithm that solves NB-WAC may lead to always abort transactions with an underlying failure detector that always suspects some process. In practice however, failure detectors do not behave this way. Failure detectors are usually implemented using time-outs and suspect processes only after time-out expirations. The expiration of a time-out (either correct or false suspicion) is generally considered, in real-world transactional systems, a sufficient reason to *abort* a transaction [2].

5.2 NB-WAC reducible to Uniform Consensus

The reduction algorithm described by the function *atomicCommitment()* in Figure 3, transforms any algorithm that solves Uniform Consensus with $\diamond\mathcal{P}$ (respectively $\mathcal{S}$, $\diamond\mathcal{S}$), into an algorithm that solves NB-WAC with $\diamond\mathcal{P}$ (respectively $\mathcal{S}$, $\diamond\mathcal{S}$).

We assume that participants know each others, and every participant p, either crashes, or calls the function *atomicCommitment()*. The vote of participant p_i is denoted $vote_i$, and we represent a Uniform Consensus algorithm by the function *uniformConsensus()* (called at line 5 and 7). The values proposed and returned by *uniformConsensus()* are *commit* and *abort*. Function *atomicCommitment()* terminates by the execution of a "**return** *outcome*" statement, where *outcome* is either *commit* or *abort* (lines 6 and 8): when p_i executes **return** *outcome*, p_i AC-decides *outcome*. Participant p_i is informed by its local failure detector of crash suspicions: the notation $p_j \in \mathcal{D}_i$ (line 3) ($\mathcal{D}_i \in \{\diamond\mathcal{P}_i, \mathcal{S}_i, \diamond\mathcal{S}_i\}$) indicates that p_i suspects p_j.

The basic idea of the algorithm is the following. Every participant sends its vote to all participants (including itself). A participant that either receives a vote *no* or suspects another participant, starts Uniform Consensus by proposing *abort* (line 5), and AC-decides the outcome returned by Uniform Consensus (line 6). Every participant that receives *yes* votes from all participants, starts Uniform

[5] Coan and Welch have discussed in [5] the benefits of defining a weak *NonTriviality* condition in order to develop randomized Non Blocking Atomic Commitment protocols.

```
  function atomicCommitment(vote_i)
1     send (p_i, vote_i) to all
2     for j = 1 to n
3         wait until [received (p_j, vote_j) or p_j ∈ D_i] ;
4             if p_j ∈ D_i or vote_j = abort then
5                 outcome_i := uniformConsensus(abort) ;
6                 return outcome_i ;
7     outcome_i := uniformConsensus(commit) ;
8     return outcome_i ;
```

Fig. 3. An algorithm that reduces NB-WAC to Uniform Consensus

Consensus by proposing *commit* (line 7), and AC-decides the outcome returned by Uniform Consensus (line 8).

Theorem 4. *The algorithm in Figure 3 reduces NB-WAC to Uniform Consensus with either $\Diamond\mathcal{P}$, $\mathcal{S}$, or $\Diamond\mathcal{S}$.*

Proof. We show that the four conditions of NB-WAC are satisfied.

1. *Uniform Agreement.* Any participant that AC-decides *outcome* (lines 6 and 8), must have decided *outcome* through Uniform Consensus (lines 5 and 7). By the *Uniform Agreement* condition of Uniform Consensus, no two participants can decide differently.
2. *Uniform Validity.* A participant AC-decides *outcome*, only if it decides

   outcome through Uniform Consensus. By the *Validity* condition of Uniform Consensus, a participant decides *commit* only if some participant p has proposed *commit* (in line 7). To reach line 7, p must have received *yes* votes from all.
3. *Termination.* There are two cases to consider for any correct participant p: (3.1) p receives *yes* votes from all, and (3.2) p does not. In case (3.1), p starts Uniform Consensus (line 7). In case (3.2), if p receives any vote *no*, p starts Uniform Consensus (line 5). Otherwise, as every correct participant sends its vote, then by the assumption of reliable channels and the *Eventual Strong Completeness* property of $\Diamond\mathcal{P}$ (respectively $\mathcal{S}$, $\Diamond\mathcal{S}$), p eventually suspects some participant and starts Uniform Consensus (line 5). Hence every correct participant starts Uniform Consensus. By the *Termination* condition of Uniform Consensus, every correct participant eventually decides and thus AC-decides.
4. *NonTriviality.* If there are no suspicions and all votes are *yes*, then every participant which starts Uniform Consensus proposes *commit* (line 7). By the *Uniform-Validity* condition of Uniform Consensus, every correct participant decides *commit* and thus AC-decides *commit* (line 8). $\square$

By Corollary 3 and the relations between failure detector classes (Sect 2.3), we have Corollary 4.

Corollary 4. *If $f < |\Omega|$, NB-WAC is solvable with either $\mathcal{S}$ or $\mathcal{W}$, and if $f < \lceil |\Omega|/2 \rceil$, NB-WAC is solvable with either $\Diamond\mathcal{S}$, $\Diamond\mathcal{P}$, $\Diamond\mathcal{Q}$, or $\Diamond\mathcal{W}$.*

NB-WAC algorithms can be obtained by combining the reduction algorithm of Figure 3 and the algorithms solving Consensus with $\mathcal{S}$ and $\Diamond\mathcal{S}$ [4]. In comparison, the algorithms described in [1, 15] for example can be seen as algorithms that use $\mathcal{P}$. Elsewhere, we have described centralized and decentralized *Three Phase Commit* algorithms using $\Diamond\mathcal{S}$ [9, 10].

6 Concluding Remarks

The importance of this work is in extending the applicability field of the results of Chandra and Toueg [4] on solving problems in asynchronous systems (with crash failures and reliable channels) augmented with unreliable failure detectors. The applicability of these results to problems other than Consensus has been discussed in [4, 11, 13, 14]. To our knowledge, it is however the first time that (non-blocking) atomic commitment problems are discussed in asynchronous systems with unreliable failure detectors.

By weakening the *NonTriviality* condition of atomic commitment, we have defined a problem, called Non-Blocking Weak Atomic Commitment (NB-WAC), which is adequate in practical transactional systems. We have shown that (1) Uniform Consensus is reducible to Consensus, and (2) NB-WAC is reducible to Uniform Consensus. As a consequence, the results of Chandra and Toueg on solving Consensus with unreliable failure detectors apply to NB-WAC.

We would like to define, in terms of failure detector characteristics, lower bounds on fault-tolerance for NB-WAC. It has been stated that $\Diamond\mathcal{W}$ is the weakest failure detector class that can solve Consensus [3], and $\Diamond\mathcal{P}$ cannot solve Consensus if more than a majority of participants can fail [4]. An interesting question is whether these lower bounds are relevant for NB-WAC. Furthermore, one may wonder if the *NonTriviality* condition defined in this paper is the strongest one that makes the problem solvable with unreliable failure detectors. Finally, we have not considered unreliable failure detectors with a known bounded number of false suspicions. Whether NB-WAC is reducible to Consensus with these failure detectors is an open question.

Acknowledgement

I am deeply grateful to André Schiper for his crucial help. The presentation of the paper was greatly improved by the suggestions of Aleta Ricciardi and the referees. I would also like to thank Tushar Chandra, Vassos Hadzilacos, Mikel Larrea and Sam Toueg for interesting discussions.

References

1. O. Babaoglu and S. Toueg. Non-Blocking Atomic Commitment. In *Distributed Systems*, pages 147-166. Sape Mullender ed, ACM Press, 1993.
2. P.A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison Wesley, 1987.
3. T. Chandra, V. Hadzilacos and S. Toueg. The Weakest Failure Detector for Solving Consensus. *Proceedings of the 11th ACM Symposium on Principles of Distributed Computing*, pages 147-158. ACM Press, 1992.
4. T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. Technical Report, Department of Computer Science, Cornell Univ, 1994. A preliminary version appeared in the *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*, pages 325-340. ACM Press, 1991.
5. B. Coan and J. Welch. Transaction commit in a realistic timing model. *Distributed Computing*, pages 87-103. 4(2), 1990.
6. D. Dolev and R. Strong. A Simple Model For Agreement in Distributed Systems. In *Fault-Tolerant Distributed Computing*, pages 42-50. B. Simons and A. Spector ed, Springer Verlag (LNCS 448), 1987.
7. M. Fischer, N. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, pages 374-382. (32) 1985.
8. J. Gray. A Comparison of the Byzantine Agreement Problem and the Transaction Commit Problem. In *Fault-Tolerant Distributed Computing*, pages 10-17. B. Simons and A. Spector ed, Springer Verlag (LNCS 448), 1987.
9. R. Guerraoui, M. Larrea and A. Schiper. Non-Blocking Atomic Commitment with an Unreliable Failure Detector. To appear in *Proceedings of the 14th IEEE Symposium on Reliable Distributed Systems*, 1995.
10. R. Guerraoui and A. Schiper. The Decentralized Non-Blocking Atomic Commitment Protocol. To appear in *Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing*, 1995.
11. R. Guerraoui and A. Schiper. Transaction model vs Virtual Synchrony model: bridging the gap. In *Distributed Systems: From Theory to Practice*, pages 121-132. K. Birman, F. Mattern and A. Schiper ed, Springer Verlag (LNCS 938), 1995.
12. V. Hadzilacos. On the relationship between the atomic commitment and consensus problems. In *Fault-Tolerant Distributed Computing*, pages 201-208. B. Simons and A. Spector ed, Springer Verlag (LNCS 448), 1987.
13. L. Sabel and K. Marzullo. Election Vs. Consensus in Asynchronous Systems. Technical Report TR95-1488, Cornell Univ, 1995.
14. A. Schiper and A. Sandoz. Primary Partition "Virtually-synchronous Communication" Harder than Consensus. *Proceedings of the 8th International Workshop on Distributed Algorithms*, pages 39-52. Springer Verlag (LNCS 857), 1994.
15. D. Skeen. NonBlocking Commit Protocols. In Proceedings of the *ACM SIGMOD International Conference on Management of Data*, pages 133-142. ACM Press, 1981.

This article was processed using the LaTeX macro package with LLNCS style