Non Blocking Atomic Commitment with an Unreliable Failure Detector [*]

Rachid Guerraoui   Mikel Larrea   André Schiper
Département d'Informatique
Ecole Polytechnique Fédérale de Lausanne
1015 Lausanne, Switzerland

Abstract

In a transactional system, an *atomic commitment* protocol ensures that for any transaction, all *Data Manager* processes agree on the same outcome *(commit* or *abort)*. A *non-blocking atomic commitment* protocol enables an outcome to be decided at every correct process despite the failure of others. In this paper we apply, for the first time, the fundamental result of Chandra and Toueg on solving the abstract consensus problem, to non-blocking atomic commitment. More precisely, we present a non-blocking atomic commitment protocol in an asynchronous system augmented with an unreliable failure detector that can make an infinity of false failure suspicions. If no process is suspected to have failed, then our protocol is similar to a *three phase commit* protocol. In the case where processes are suspected, our protocol does not require any additionnal termination protocol: failure scenarios are handled within our regular protocol and are thus much simpler to manage.

1 Introduction

A transaction can be viewed as an atomic group of invocations on data objects. Transaction atomicity implies *concurrency atomicity* and *failure atomicity* [2]. Concurrency atomicity (also called *isolation* or *serializability*) means that the intermediate effects of a transaction on shared data objects are invisible to other transactions. Failure atomicity (also called *all-or-nothing* property) means that either a transaction completes successfully and is *committed*, or is *aborted* and has no effect on data objects. To ensure the failure atomicity property, termination of a transaction that updates distributed data objects has to be coordinated among *Data Manager* processes according to some *atomic commitment* protocol. The aim of this protocol is to bring the processes to agree on an outcome for the transaction: *commit* or *abort*.

Several atomic commitment protocols have been proposed in the literature. The simplest and the best known protocol is the *two phase commit protocol (2PC)* [10]. This protocol is easy to implement but can be *blocking*. For example, if the coordinator fails while all *Data*

[*]Paper appears in the *14th IEEE Symposium on Reliable Distributed Systems*, 13-15 Sept. 95, Bad Neuenahr, Germany. Research supported by the "Fonds national suisse" and OFES under contract number 21-32210.91, as part of the ESPRIT Basic Research Project BROADCAST (number 6360), and by PPR-IF under contract number 5003-34344.

Manager processes are waiting for a decision message, then none of these processes can terminate the transaction. A process must wait for the coordinator to recover before deciding on an outcome. During this period, the process will be prevented from relinquishing valuable resources that it may have acquired for exclusive use on behalf of the transaction, e.g locks on shared data objects. Assuming a *synchronous* system (where there is a known upper bound on process relative speeds and communication delays), many authors have proposed *non-blocking* atomic commitment protocols [13, 14, 1, 12]. Assuming a *semi-asynchronous* system (where messages are usually delivered within some known time bound, but sometimes come late), Coan and Welch have presented in [4] a randomized non-blocking atomic commitment protocol. A non-blocking protocol enables an outcome to be decided at every correct process despite the failure of others. Non-blocking protocols are desirable since they limit the time intervals during which transactions may be holding valuable resources.

In this paper, we are interested in deterministic non-blocking atomic commitment protocols in *asynchronous* systems (where no assumption is made about process relative speeds or communication delays). In such systems, the *three phase commit (3PC)* protocol of [13] may lead processes to decide on different outcomes for a transaction and thus may violate the failure atomicity property. The protocols in [14, 1, 12] guarantee that, even in an asynchronous system, two processes can never decide on different outcomes. However, there is no rigorous characterization of the conditions under which these protocols are non-blocking in asynchronous systems.

Difficulty in designing non-blocking atomic commitment protocols in asynchronous systems is not surprising because of the so called *FLP* impossibility result [7]. This result states that, in an asynchronous system, no agreement among processes can be solved in a deterministic and fault-tolerant way (i.e non blocking). As atomic commitment is a typical agreement problem, the FLP impossibility result also applies for non-blocking atomic commitment [11]. Recently, Chandra and Toueg have shown that by adding a failure detector to an asynchronous system, the *consensus* problem (an abstract form of agreement) becomes solvable if a majority of processes are correct. They have given in [3] a protocol that solves the consensus problem based on an unreliable failure detector, named *Eventually Strong*, and noted $\diamond\mathcal{S}$. Roughly speaking, the failure detector $\diamond\mathcal{S}$ ensures that (1) eventually every process that fails is permanently suspected (to have failed) by every correct (that do not fail) process, and (2) eventually some correct process is never suspected by any correct process. The failure detector $\diamond\mathcal{S}$ can make an infinity of false failure suspicions.

We present an adaptation of the Chandra-Toueg consensus protocol to the non-blocking atomic commitment problem. Our definition of the non-blocking atomic commitment problem is slightly different from previous definitions given in the literature (in the context of synchronous systems). Our protocol solves the non-blocking atomic commitment problem, assuming the failure detector $\diamond\mathcal{S}$ and a majority of correct processes [1]. If these assumptions are not satisfied, our protocol will not terminate, but it guarantees that no two processes can decide on different outcomes. If no process is suspected, then our protocol is similar to existing *three phase commit protocols (3PC)* [13, 14, 12]. In the case where failures occur,

[1]As shown in [3], a protocol that is correct with $\diamond\mathcal{S}$ can easily be transformed to be correct with a failure detector $\diamond\mathcal{W}$. We will briefly discuss this issue in Section 4 of this paper.

the termination protocols generally associated to 3PC protocols become extremely complicated [2]. In comparison, all possible failure cases are handled within our regular protocol. Hence, besides a rigorous characterization of its liveness (i.e properties of $\Diamond\mathcal{S}$ and a majority of processes are correct), our protocol provides a simpler way to handle failure scenarios.

The rest of the paper is organized as follows. The next Section describes our model of an asynchronous system augmented with a failure detector. In Section 3, we present an abstract definition of the non-blocking atomic commitment problem. Then we point out the main differences between non-blocking atomic commitment and consensus problems. Section 4 first presents our protocol in an intuitive way, and then in a formal way. Finally, Section 5 summarizes the main contributions of this paper.

2 Model

2.1 System Model

We consider a distributed system composed of a finite set of processes $S = \{p_1, p_2, \ldots, p_n\}$ completely connected through a set of channels. Communication is by message passing, *asynchronous* and *reliable*. Asynchrony means that there is no bound on communication delays. 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 fail)[2]. A reliable channel can be implemented by retransmitting lost or corrupted messages. Processes fail by crashing; we do not consider Byzantine failures. A process $p_i \in S$ may (1) send a message to another process, (2) receive a message sent by another process, (3) perform some local computation, or (4) fail. Failure is modeled by the local event $crash_i$. The process history of $p_i \in S$ is a sequence of events $h_i = e_i^0 \cdot e_i^1 \cdots e_i^k$. Histories of correct processes are infinite. If not infinite, the process history of p_i terminates with event $crash_i$.

The absence of a bound on communication delays makes it difficult to handle crash failures. Chandra and Toueg [3] propose to augment an asynchronous system, with a failure detector which gives hints on crashed processes. We follow their approach and assume that our asynchronous system is augmented with a failure detector. A failure detector can be viewed as a distributed oracle. Each process has access to a local failure detector module. This module maintains a list of processes that it currently suspects to have failed. We do not make any assumption on how a failure detector is implemented. For example, the local failure detector module of a process could query other processes, and suspect those that do not respond in a timely fashion. In order to reduce the number of false suspicions, the local failure detector module can also consult other failure detector modules before suspecting a process.

Failure detectors can be abstractly characterized by their *completeness* and *accuracy* properties. Completeness requires that the failure detector eventually suspects every process that actually fails, while accuracy restricts the mistakes that a failure detector can make. In the following, we will consider the *Eventually Strong* failure detector, noted $\Diamond\mathcal{S}$. The failure detector $\Diamond\mathcal{S}$ guarantees (1) *strong completeness*, i.e. eventually every process that fails is permanently suspected by every correct process, and (2) *eventual weak accuracy*, i.e. there is

[2]This assumption does not exclude link failures, if we require that any link failure is eventually repaired.

```
   /* The transaction manager executes: */
1      send (start,transaction,operations,participants) to all participants ;

   /* All participants (including the transaction manager) execute: */
2      when receive(start,transaction,operations,participants)
3          /* Perform operations requested by transaction */
4          if (able to make updates permanent) then
5              vote := yes ;
6          else
7              vote := no ;
           /* Start the atomic commitment protocol */
8          atomicCommitment(transaction,participants)        /* see Figure 3 */
```

Figure 1: Transaction Execution Model

a time after which some correct process is never suspected by any correct process. The failure detector $\diamond\mathcal{S}$ can thus make an infinity of mistakes (false failure suspicions).

2.2 Distributed Transactions

For each transaction, the processes that perform updates on its behalf are called *participants*. Each participant updates data objects that are local to it. We consider the transaction execution model explained in [1], and described in Figure 1. The transaction originates from a single participant called the *Transaction Manager*. The transaction manager distributes the transaction to the participants by sending them *start* messages containing a description of the transaction operations and the full list of participants. After a participant performs the operations requested by the transaction, the participant uses a variable *vote* which denotes its ability to locally install the updates. A *yes* vote indicates that the participant is able to make the updates permanent. A *no* vote indicates that the participant is unable to make the updates permanent. We do not make any assumption on how the vote is determined by a participant. For example :

- In a *pessimistic* approach, where a lock has been acquired by the transaction manager [6], a participant votes *yes* if and only if the participant has written the updates on stable storage. Otherwise, the participant votes *no*, e.g if the disk is full.
- In an *optimistic* approach (without locking) [2], a participant votes *yes* if and only if no concurrency control conflict has been locally detected, and the participant has written the updates on stable storage. Otherwise, the participant votes *no*.

Finally, the participants start an atomic commitment protocol (i.e by executing the procedure *atomicCommitment(transaction, participants)* at line 8 of Figure 1). The aim of this protocol is to bring the participants to agree on an outcome for the transaction (either *commit* or *abort*). The transaction failure atomicity property lies on this agreement.

3 The Non-Blocking Atomic Commitment Problem

In this section we define the non-blocking atomic commitment problem in our distributed system model.

3.1 Overview

Each participant must decide on an outcome for the transaction among two possible values: *commit* or *abort*. The participants must agree on the same outcome despite failures. The outcome can be *commit* only if the votes of all participants are *yes*. In order to exclude trivial situations where participants always decide *abort*, it is generally required [1] that *commit* must be decided if (1) all votes are *yes* and (2) *no process fails*. This *strong Non-Triviality* condition is not adequate in an asynchronous system, since it requires precise knowledge about failures. A characteristic of an asynchronous system is that a failed process cannot be distinguished from a process that is just very slow. We consider a *weaker* Non-Triviality condition where *commit* must be decided if (1) all votes are *yes* and (2) *no process has ever been suspected* [3].

3.2 Definition

We define the non-blocking atomic commitment problem by the following conditions:

- **AC-Uniform-Agreement:** If two participants decide, they decide on the same outcome.
- **AC-Uniform-Validity:** The outcome is *commit* only if all participants have voted *yes*.
- **AC-Termination:** Every correct participant eventually decides.
- **AC-Non-Triviality:** If all participants vote *yes*, and no participant has ever been suspected, the outcome must be *commit*.

The *AC-Agreement* and the *AC-Validity* conditions are safety conditions. They ensure the *failure atomicity* property of transactions. The *AC-Termination* condition is a liveness condition which guarantees the non-blocking property. The *AC-Non-Triviality* condition excludes trivial solutions to the problem where participants always decide to abort transactions. This condition can be viewed as a liveness condition from the application point of view since it ensures progress (i.e transaction commit) whenever possible (i.e when no suspicion and no *no* vote).

3.3 Atomic Commitment vs Consensus

As it has been pointed out in [11], the non-blocking atomic commitment problem is very similar to the abstract consensus problem. In the latter problem, every participant starts with an initial value, and the participants must agree on a common final value. The consensus problem is defined by the following three conditions[4]: *C-Uniform-Agreement*: if two participants decide, they decide on the same final value; *C-Uniform-Validity*: the final value must be the initial value of some participant; *C-Termination*: every correct participant eventually decides. The so called *FLP impossibility result* [7] states that in an asynchronous system, there is no deterministic and fault-tolerant protocol that solves the consensus problem even if only one participant can fail. Hadzilacos has proved in [11] that this impossibility result

[3] We discuss this issue in more details in [8].

[4] We consider actually here the *uniform* consensus problem. However, we show in [8] that in asynchronous systems with unreliable failure detectors, consensus and uniform consensus are equivalent.

applies also to the non-blocking atomic commitment problem.

However, Chandra-Toueg have shown that, by adding a failure detector to an asynchronous system, consensus becomes solvable [3]. They have given a protocol that solves the consensus problem based on the failure detector $\diamond\mathcal{S}$. In Section 4, we present an adaptation of their protocol to the non-blocking atomic commitment problem. It is worthwhile, at this stage, to point out the main differences between the non-blocking atomic commitment problem, as defined in Section 3.2, and the consensus problem:

- In the non-blocking atomic commitment problem, a participant can decide *abort* as soon as it learns that at least one participant has voted *no*. Hence, a participant that votes *no* can unilateraly decide *abort*. No such unilateral decision is possible in a consensus protocol.
- In the non-blocking atomic commitment problem, a participant can decide *commit* only if it knows that *all* participants have voted *yes*. No decision in the consensus problem requires full knowledge about all initial values.

4 A Non-Blocking Atomic Commitment Protocol

In this section, we present our non-blocking atomic commitment protocol. We first give an intuitive idea of the protocol, then we present it in a more precise way and prove its correctness. We consider n participant processes, $p_1, p_2, .., p_n$, and we note f the maximum number of participants that may fail. Furthermore, we consider the commitment of *one single* transaction[5].

4.1 Overview of the protocol

Any participant p which votes *no* immediatly decides *abort* and sends this decision to all participants. If p is correct, all participants will learn the decision. If p fails, it will eventually be suspected by $\diamond\mathcal{S}$. This is because $\diamond\mathcal{S}$ satisfies *strong completeness:* eventually every process that fails is permanently suspected by every correct process.

The participants that do not perform unilateral aborts (i.e the participants that vote *yes*) take part in the main protocol with multiple *asynchronous rounds*. This protocol is based on the *rotating coordinator* paradigm [5]. In this protocol, all participants execute the same sequence of rounds. At any given round, one single participant is the *coordinator*. All participants have a priori knowledge that during round r, the coordinator is the participant p_c, where $c = (r \ mod \ n) + 1$. For example, at round 0 the coordinator is p_1. At a given time, two participants may be in two different rounds and thus may have different coordinators. A coordinator tries to decide on an outcome by executing an algorithm similar to a three phase commit [13] (Figure 2). The coordinator first waits for information of votes, or notification of failure suspicions, about all participants. Depending on the votes or on the suspicions, the coordinator *suggests* an outcome by sending an *estimate (pre-commit* or *pre-abort)* to all participants (step C.1 in Figure 2). If enough (actually $n - f$) participants agree with

[5]Messages concerning different transactions are assumed to be clearly distinguished.

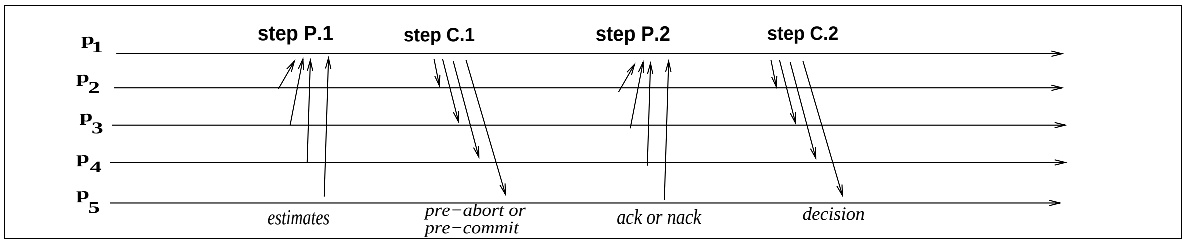

Figure 2: The steps in a round where p_1 is the coordinator

the estimate, the coordinator succeeds in deciding an outcome and sends it to all participants (step C.2 in Figure 2). A coordinator that is suspected may not reach a decision, and moves to another round.

We will show that if we assume $f < n/2$, our protocol guarantees that no two participants can decide differently. The assumption that less than f participants may fail and the *strong completeness* property of the failure detector $\diamond\mathcal{S}$, ensure that no participant remains blocked forever waiting for messages from other participants. The *eventual weak accuracy* property of $\diamond\mathcal{S}$ (i.e eventually some correct process is never suspected by any correct process), ensures that at least one coordinator eventually decides, and sends its decision to all participants.

Our protocol is an adaptation of the Chandra-Toueg consensus protocol with $\diamond\mathcal{S}$ [3]. The main differences between the protocols reflect the asymmetry in the *commit-abort* decision (Section 3.3):

1. In our protocol, any participant which votes *no* immediatly decides *abort* and sends this decision to all participants. In the Chandra-Toueg consensus protocol, only a coordinator can decide.
2. In our protocol, a coordinator can decide *commit* only if it knows that *all* participants have voted *yes*. In the Chandra-Toueg consensus protocol, a coordinator never has to get information from all participants.

4.2 The Protocol

Each participant p maintains a variable est_p *(estimate)* denoting its estimate of the decision, and a variable ts_p *(timestamp)* representing the round where the estimate was last modified. The procedure executed by participant p is described in Figure 3. If p's vote is *no*, p decides *abort* (line 7 in Figure 3) and sends the decision to all participants (line 8 in Figure 3). In this case, either p is correct and all participants will learn the decision, or p fails and will eventually be suspected by $\diamond\mathcal{S}$.

If p's vote is *yes*, p executes two concurrent tasks (line 10 in Figure 3). The first task *(Task 1)* is described in Figure 4. It consists in waiting for a decision message, deciding upon the reception of the message (line 3 of Task 1), and transmitting the decision to the other participants (line 4 of Task 1).

```
procedure atomicCommitment(transaction,participants):

    outcome : { commit, abort } ;                                   /* Data structure */
    $state_p$ : { decided, undecided } ;
    $est_p$ : { pre-abort, pre-commit } ;   /* Estimate */
    $r_p$ : integer;   /* Round */
    $ts_p$ : integer;   /* Timestamp */

1   $state_p$ := undecided ;                                        /* Initialisation */
2   $est_p$ := pre-abort ;
3   $r_p := -1$ ;
4   $ts_p := 0$ ;

5   if vote = no then                                             /* Unilateral abort */
6       $state_p$ := decided ;
7       decide($transaction, abort$) ;
8       send($p, r_p, abort, decide$) to all participants ;
9   else
10      cobegin    Task 1 || Task 2    coend ;                      /* Concurrent Tasks */
```

Figure 3: The atomic commitment procedure

The second task *(Task 2)* is described in Figure 5. In contains the main protocol with asynchronous rounds based on the rotating coordinator paradigm. In each round (denoted r_p for participant p), a participant p, which is not the coordinator, executes sequentially the steps P.1, P.2. The coordinator executes sequentially the steps P.1, C.1, P.2, C.2. During steps P.1, C.1, P.2, or C.2, p either (1) sends a message, (2) waits for a message or for an information about a suspicion (i.e by consulting its failure detector module, noted $\diamond\mathcal{S}_p$), (3) receives a message, or (4) performs some local computation.

The steps P.1, C.1, P.2, C.2. of Task 2 are described in Figure 5, and detailed below.

Step P.1: The participant sends its estimate to the coordinator (line 4 of Task 2). Recall that a participant performs Task 2 (and thus sends its estimate in step P.1), only if it has voted *yes*.

Step C.1: The coordinator waits to receive estimates from $n - f$ participants (line 6 of Task 2), and either to suspect or to receive an estimate from each of the rest of the f participants (line 7 of Task 2). If n estimates have been received (which means that all participants have voted *yes*), the coordinator adopts *pre-commit* as its estimate (line 10 of Task 2), and proposes it to all participants (line 14 of Task 2)[6]. If the coordinator does not receive n estimates, it adopts the estimate with the largest timestamp (line 13 of Task 2), and proposes it to all participants (line 14 of Task 2).

Step P.2: The participant waits either to receive an estimate from the coordinator or to suspect the coordinator (line 15 of Task 2). If the participant receives an estimate, it adopts the coordinator's estimate and sends an *ack* to the coordinator (line 19 of Task 2). Otherwise,

[6]This is a subtle difference with the Chandra-Toueg protocol.

the participant sends a *nack* to the coordinator (line 21 of Task 2). Then the participant moves to the next round.

Step C.2: The coordinator waits to receive answers (*ack* or *nack*) from $n - f$ participants (line 23 of Task 2). If $n - f$ answers are *ack*, the coordinator decides on its estimate, and sends it to all the participants. Otherwise, the coordinator moves to the next round.

```
  Task 1:
1    wait until receive($-, -, outcome, decide$) ;
2        $state_p := decided$ ;
3        decide($transaction, outcome$) ;
4        send($-, -, outcome, decide$) to all participants ;
```

Figure 4: Task 1

4.3 Correctness

We show in the following that our protocol solves the non-blocking atomic commitment problem assuming the failure detector $\Diamond\mathcal{S}$ and $f < n/2$. We first prove *AC-Agreement* (Lemma 1) and *AC-Termination* (Lemma 2).

Lemma 1. *If $f < n/2$, no two participants decide differently.*
PROOF.
Case 1: Unilateral abort (at least one participant votes no).
We show by contradiction that no participant can decide *commit*. For a process to decide *commit*, at least one coordinator must have adopted the estimate *pre-commit* in step C.1 (line 10 of Task 2) because initially all estimates are *pre-abort*. This means that all participants have sent their estimate to the coordinator, which implies that all participants have voted *yes*: a contradiction.

Case 2: No unilateral abort (all participants vote yes).
Assume that one participant p decides *(commit* or *abort)*. This means that p has received a message $(-, -, outcome, decide)$ from a coordinator (line 1 of Task 1). Let r be the smallest round number in which a coordinator p_c (where $c = (r \ mod \ n) + 1$) has sent the message $(-, -, outcome, decide)$ to the participants. We show by induction that, for all rounds $r' \geq r$, any coordinator $p_{c'}$ (where $c' = (r' \ mod \ n) + 1$) which sends a decision message, sends the same outcome. This trivially holds for $r' = r$. Consider $r' = r + 1$.
(1) Assume the outcome is *commit*: in round r, p_c has sent the message $(p_c, r, commit, decide)$ in step C.2 (line 27 of Task 2). This implies that in round r, at least $n - f$ participants have adopted the estimate *pre-commit* in step P.2 (line 17 of Task 2) and no participant has adopted the estimate *pre-abort*. In round $r + 1$, if the coordinator $p_{c'}$ adopts an estimate (line 13 of Task 2), it must adopt *pre-commit* since $f < n/2$ and $p_{c'}$ must have received at least one estimate *pre-commit* with timestamp r (the largest one in round $r + 1$).
(2) Assume the outcome is *abort*: in round r, p_c has sent the message $(p_c, r, abort, decide)$ in step C.2. This implies that in round r, at least $n - f$ participants have adopted the estimate *pre-abort* in step P.2 (line 17 of Task 2) and no participant has adopted the estimate *pre-*

```
Task 2:
while $state_p = undecided$
    $r_p := r_p + 1$ ;
    $coord := p_{(r_p \ mod \ n)+1}$ ;
    send $(p, r_p, est_p, ts_p)$ to coord ;                                /* Step P.1 */
    if $p = coord$ then                                                    /* Step C.1 */
        wait until [(for $n - f$ participants $q$: received $(q, r_p, est_q, ts_q)$ from $q$)
                and (for $n$ participants $q$: received $(q, r_p, est_q, ts_q)$ from $q$ or $q \in \Diamond\mathcal{S}_p$)] ;
            $msgs_p[r_p] = \{(q, r_p, est_q, ts_q)$ such that $p$ received $(q, r_p, est_q, ts_q)$ from $q\}$ ;
            if $|msgs_p[r_p]| = n$ then
                $est_p := pre\text{-}commit$ ;
            else
                $t :=$ largest $ts_q$ such that $(q, r_p, est_q, ts_q) \in msgs_p[r_p]$ ;
                $est_p :=$ select one $est_q$ such that $(q, r_p, est_q, t) \in msgs_p[r_p]$ ;
            send $(p, r_p, est_p)$ to all participants ;
    wait until [received $(coord, r_p, est_{coord})$ from $coord$ or $coord \in \Diamond\mathcal{S}_p$] ;   /* Step P.2 */
        if received $(coord, r_p, est_{coord})$ then
            $est_p := est_{coord}$ ;
            $ts_p := r_p$ ;
            send $(p, r_p, ack)$ to $coord$ ;
        else
            send $(p, r_p, nack)$ to $coord$ ;
    if $p = coord$ then                                                    /* Step C.2 */
        wait until [for $n - f$ participants $q$: received $(q, r_p, ack)$ or $(q, r_p, nack)$] ;
            if [for $n - f$ participants q: (received $(q, r_p, ack)$] then
                $state_p := decided$ ;
                if $outcome = pre\text{-}commit$ then
                    send $(p, r_p, commit, decide)$ to all participants ;
                else
                    send $(p, r_p, abort, decide)$ to all participants ;
```

Figure 5: Task 2

commit. In round $r + 1$, if the coordinator $p_{c'}$ adopts an estimate, it must adopt *pre-abort* since $f < n/2$ and $p_{c'}$ must have received at least one estimate *pre-abort* with timestamp r (the largest one in round $r + 1$). □

Lemma 2. *If $f < n/2$ and assuming the failure detector $\Diamond\mathcal{S}$, every correct participant eventually decides.*
PROOF.
Case 1: Unilateral abort for a correct participant (at least one correct participant votes no).
Consider p a correct participant that votes *no*. Participant p decides *abort* (in a unilateral way) and sends message $(p, r_p, abort, decide)$ to all participants (line 8 in Figure 3). Since p is correct and channels are reliable, every correct participant eventually receives message $(p, r_p, abort, decide)$ and decides.

Case 2: No unilateral abort for a correct participant (all correct participants vote yes).
Every correct participant starts Task 1 and Task 2 (line 10 in Figure 3). If some correct participant decides in Task 1 (line 3 of Task 1), it retransmits the decision message to all participants (line 4 of Task 1). Every correct participant eventually receives the message and decides.
Assume now that every correct participant starts Task 1 and Task 2, and no correct participant decides. We first show (1) that no correct participant remains blocked forever at one of the *wait* statement of Task 2 (lines 6 or 7 in step C.1, line 15 in step P.2, or line 23 in step C.2). Then we show (2) that no correct participant remains blocked forever at the *wait* statement of Task 1 (i.e every correct participant eventually decides).
1. No correct participant remains blocked forever in Task 2.
The proof is by contradiction. Let r be the smallest round number in which some correct participant blocks forever at one of the *wait* statement of Task 2. As all correct participants (they are at least $n - f$) have started Task 2 and none has already decided, they all have reached step P.1 (line 4 of Task 2) in round r: they all have sent a message $(-, r, estimate, -)$ to the coordinator. We note p_c this coordinator (where $c = (r \ mod \ n) + 1$). If p_c crashes, every correct participant eventually suspects p_c (line 15 of Task 2), thanks to *strong completeness*[7] property of $\Diamond\mathcal{S}$. Hence, no correct participant remains blocked undefinitely in step P.2 (line 15 of Task 2). Assume p_c is correct. Coordinator p_c eventually receives $n - f$ messages $(-, r, estimate, -)$ and cannot remain blocked forever in step C.1 (line 6 of Task 2). In addition, the *strong completeness* property of $\Diamond\mathcal{S}$ ensures that p_c eventually receives a message $(-, r, estimate, -)$ from each of the rest of the f participants or suspects it. Hence, p_c cannot remain blocked forever in step C.1 (line 7 of Task 2). A correct participant that does not suspect p_c, eventually receives message (p_c, r, est_{p_c}) and cannot remain blocked forever in step P.2 (line 15 of Task 2). Therefore, every correct participant eventually sends to p_c either a message $(-, r, ack)$ (line 19 of Task 2), or a message $(-, r, nack)$ (line 21 of Task 2). This implies that p_c eventually receives $n - f$ $(-, r, ack)$ or $(-, r, nack)$ messages and cannot remain blocked forever in step C.2 (line 23 of Task 2). As a consequence, all correct participants complete round r: a contradiction.
2. No correct participant remains blocked forever in Task 1.
Since $\Diamond\mathcal{S}$ satisfies *weak accuracy*[8], eventually some correct participant p is not suspected by at least $n - f$ participants. Assume p becomes the coordinator of some round r. From the above (1), at least $n - f$ participants reach step P.2 of round r, adopt the estimate of p (line 17 of Task 2), and send $(-, r, ack)$ to p (line 19 of Task 2). The coordinator p eventually receives $n - f$ messages $(-, r, ack)$, and sends a decision message to all participants. As p is correct, its decision is eventually received by every correct participant that also decides. □

Proposition 1. *With the failure detector $\Diamond\mathcal{S}$ and if $f < n/2$, our protocol solves the non-blocking atomic commitment problem.*
Proof.
AC-Agreement: by Lemma 1.
AC-Validity: As initially all estimates are *pre-abort*, a participant decides *commit* only if some coordinator has changed its estimate from *pre-abort* to *pre-commit* in step C.1 (line 10 of Task 2). In order to do so, this coordinator must have received n estimates, which means

[7] Every process that crashes is eventually suspected by every correct process.
[8] Eventually some correct process is never suspected by any correct process.

that all participants have voted *yes*.
AC-Termination: by Lemma 2.
AC-Non-Triviality: If no process is ever suspected and all participants vote *yes*, then the first coordinator (in round 0) will receive n estimates. The coordinator will propose *pre-commit* to the participants (in step C.1) and later will decide *commit* and will send its decision after receiving $n - f$ messages $(-, -, ack)$ from the participants (in step C.2). Every participant will thus receive this decision message, and decides *commit*. $\square$

4.4 Discussion

Blocking vs. inconsistency: It is important to notice that the *AC-Agreement* condition (Lemma 1) only requires to assume $f < n/2$. If more than a majority of participants fail (i.e more than f participants fail) or if the properties of $\diamond\mathcal{S}$ *(strong completeness* or *eventual weak accuracy)* are not satisfied, our protocol will not terminate, but never allows different participants to reach different decisions.

Solving non-blocking atomic commitment with $\diamond\mathcal{W}$: Chandra and Toueg have also shown in [3] that a failure detector, noted $\diamond\mathcal{W}$ and named *Eventually Weak*, can be transformed into $\diamond\mathcal{S}$ and can thus solve consensus. The failure detector $\diamond\mathcal{W}$ satisfies (1) *weak completeness:* eventually every process that fails is permanently suspected by *some* correct process, and (2) *eventual weak accuracy*(As $\diamond\mathcal{S}$). It is obvious that the transformation of $\diamond\mathcal{W}$ to $\diamond\mathcal{S}$ implies that the non-blocking atomic commitment problem (as defined in our paper) can be solved with $\diamond\mathcal{W}$.

5 Concluding Remarks

The aim of an atomic commitment protocol is to ensure agreement among a set of processes (*Data Managers*) on an outcome for a transaction. A non-blocking atomic commitment protocol enables an outcome to be decided by correct processes despite the failure of others. The difficulty in designing such protocols in asynchronous systems is not surprising, considering the *FLP* impossibility result [7]. Chandra and Toueg have shown in [3] that consensus can be solved in an asynchronous system augmented with an unreliable failure detector. They have proposed a protocol that solves the consensus problem, based on the failure detector $\diamond\mathcal{S}$ which can make an infinity of false failure suspicions.

In this paper, we have given a definition of the non-blocking atomic commitment protocol that is adequate in an asynchronous system augmented with a failure detector. Then we have described a non-blocking atomic commitment protocol inspired by the Chandra-Toueg protocol. To our knowldge, it is the first time that the fundamental result of Chandra and Toueg, on solving consensus in asynchronous systems, is adapted to non-blocking atomic commitment.

Our protocol can be viewed as an adaptation of the Chandra-Toueg protocol to the characteristics of the non-blocking atomic commitment problem. These characteristics concern mainly the asymmetry in the *commit-abort* decision. Moreover, unlike atomic commitment protocols presented in [13, 14, 12], our protocol does not require any additional termination protocol to handle failure scenarios. Also, the assumptions of a failure detector $\diamond\mathcal{S}$ and a majority of processes are correct, provide a rigorous characterization of the liveness of our protocol. If these assumptions are not satisfied, our protocol does not terminate, but never leads two processes to decide differently.

Acknowledgements

Thanks to Tushar Deepak Chandra for interesting discussion about unreliable failure detectors.

References

[1] O. Babaoglu and S. Toueg. Non-Blocking Atomic Commitment. In Sape Mullender, editor, *Distributed Systems*, pages 147-166. 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 and S. Toueg. Unreliable failure detectors for reliable distributed systems. Technical Report 93-1374, Department of Computer Science, Cornell Univ, 1993. A preliminary version appeared in the *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*, pages 325-340. ACM Press, 1991.

[4] B. Coan and J. Welch. Transaction commit in a realistic timing model. *Distributed Computing*, 4(2), pages 87-103. 1990.

[5] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35 (2), pages 288-323. 1988.

[6] K. Eswaren, J. Gray, R. Lorie and I. Traiger. The Notion of Consistency and Predicate Locks in a Database System. *Communications of the ACM*, 19 (11), pages 624-633. 1976.

[7] M. Fischer, N. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM* (32), pages 374-382. 1985.

[8] R. Guerraoui. Revisiting the relationship between Non Blocking Atomic Commitment and Consensus problems. In Proceedings of the *9th International Workshop on Distributed Algorithms.* J.M Helary and M. Raynal editors, LNCS, Springer Verlag. 1995.

[9] R. Guerraoui and A. Schiper. Transaction model vs Virtual Synchrony model: bridging the gap. In *Distributed Systems: from Theory to Practice.* K. Birman, F. Mattern and A. Schiper editors, LNCS 938, Springer Verlag, pages 121-132. 1995.

[10] J. Gray. Notes on Database Operating Systems. In *Operating Systems: An Advanced Course.* R. Bayer, R.M. Graham and G Seegmuller editors, LNCS 60, Springer Verlag, pages 393-481. 1978.

[11] V. Hadzilacos. On the relationship between the atomic commitment and consensus problems. In *Fault-Tolerant Distributed Computing.* B. Simons and A. Spector editors, LNCS 448, Springer Verlag, pages 201-209. 1990.

[12] I. Keidar and D. Dolev. Increasing the Resilience of Atomic Commit at No Additional Cost. *Technical Report CS94-18.* Institute of Computer Science, The Hebrew University of Jerusalem. 1994.

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

[14] D. Skeen. A Quorum-Based Commit Protocol. In Proceedings of the *6th Berkeley Workshop on Distributed Data Management and Computer Networks*, pages 69-80. 1982.