Two-Phase Atomic Commitment Protocol in Asynchronous Distributed Systems with Crash Failure

Yong-Hwan Cho, Sung-Hoon Park and Seon-Hyong Lee
School of Electrical and Computer Engineering, Chungbuk National Unvi. Cheongju
ChungBuk 361-763
E-mail: [yhcho,spark]@chungbuk.ac.kr

Abstract

This paper defines the Non-Blocking Atomic Commitment problem in a message-passing asynchronous system and determines a failure detector to solve the problem. This failure detector, which we call the modal failure detector star, and which we denote by $M^*$, is strictly weaker than the perfect failure detector $P$ but strictly stronger than the eventually perfect failure detector $\square P$. The paper shows that at any environment, the problem is solvable with $M^*$.

1. Introduction

1.1 Background

We address the fault-tolerant Non-Blocking Atomic Commitment problem, simply NB-AC, in an asynchronous distributed system where the communication between a pair of processes is by a message-passing primitive, channels are reliable and processes can fail by crashing. In distributed systems, to ensure transaction 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 [5,11,12,17]. 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) [2,6].

The problem of Non-Blocking Atomic Commitment becomes much more complex in distributed systems (as compared to single-computer systems) due to the lack of both a shared memory and a common physical clock and because of unpredictable message delays. Evidently, the problem cannot be solved deterministically in a crash-prone asynchronous system without any information about failures. There is no way to determine that a process is crashed or just slow. Clearly, no deterministic algorithm can guarantee Non-Blocking Atomic Commitment simultaneously. In this sense, the problem stems from the famous impossibility result that consensus cannot be solved deterministically in an asynchronous system that is subject to even a single crash failure [7].

1.2 Failure Detectors

In this paper, we introduced a modal failure detector $M^*$ and showed that the Non-Blocking Atomic Commitment problem is solvable with it in the environment with majority correct processes. The concept of (unreliable) failure detectors was introduced by Chandra and Toueg [3,4], and they characterized failure detectors by two properties: completeness and accuracy. Based on the properties, they defined several failure detector classes: perfect failure detectors $P$, weak failure detectors $W$, eventually weak failure detectors $\square W$ and so on. In [3] and [4] they studied what is the "weakest" failure detector to solve Consensus. They showed that the weakest failure detector to solve Consensus with any number of faulty processes is $\Omega^+\Sigma$ and the one with faulty processes bounded by $\lceil n/2 \rceil$ (i.e., less than $\lceil n/2 \rceil$ faulty processes) is $\Omega W$. After the work of [8], several studies followed. For example, the weakest failure detector for stable leader election is the perfect failure detector $P$ [4], and the one for Terminating Reliable Broadcast is also $P$ [1,3].

Recently, as the closest one from our work, Guerraoui and Kouznetsov showed a failure detector class for mutual exclusion problems that is different from the above weakest failure detectors. The failure detector, called the Trusting failure detector, satisfies the three properties, i.e., strong completeness, eventual strong accuracy and trusting accuracy so that it can solve the mutual exclusion problem in asynchronous distributed systems with crash failure. And they used
the bakery algorithm to solve the mutual exclusion problem with the trusting failure detector.

1.3 Contributions

How about the Non-Blocking Atomic Commitment problem? More precisely, what is the weakest failure detector to solve the Non-Blocking Atomic Commitment problem? The mutual exclusion algorithm is completely different from the NB-AC in which the order of getting the critical section is decided based on a ticket order. In contrast to the mutual exclusion algorithm, the NB-AC algorithm should receive the messages from all members of a group to make a decision.

In general, Non-Blocking Atomic Commitment algorithms assume that the system is either a failure-free model [13,14,16] or a synchronous model in which (1) if a process crash, it is eventually detected by every correct process and (2) no correct process is suspected before crash [13,16]: with the conjunction of (1) and (2), the system is assumed to be equipped with the capability of the perfect failure detector \( P \) [3]. In other words, the perfect failure detector \( P \) is sufficient to solve the Non-Blocking Atomic Commitment problem. But is \( P \) necessary? For the answer to the question, we present a modal failure detector star \( M^* \), that is a new failure detector we introduce here, which is strictly weaker than \( P \) (but strictly stronger than \( \Diamond P \), the eventually perfect failure detector of [3]). We show that the answer is “no” and we can solve the problem using the modal failure detector star \( M^* \).

Roughly speaking, failure detector \( M^* \) satisfies (1) eventual strong accuracy and (2) strong completeness together with (3) modal accuracy, i.e., initially, every process is suspected, after that, any process that is once confirmed to be correct is not suspected before crash. If \( M^* \) suspects the confirmed process again, then the process has crashed. However, \( M^* \) might suspect temporarily every correct process before confirming it’s alive as well as might not suspect temporarily a crashed process before confirming it’s crash. Intuitively, \( M^* \) can thus make at least one mistake per every correct process and algorithms using \( M^* \) are, in terms of a practical distributed system view, more useful than those using \( P \).

We here present the algorithm to show that \( M^* \) is sufficient to solve Non-Blocking Atomic Commitment and it is inspired by the well-known Non Blocking Atomic Commit Protocols of D. Skeen [4,7].

1.4 Road Map

The rest of the paper is organized as follows. Section 2 addresses motivations and related works and Section 3 overviews the system model. Section 4 introduces the Modal failure detector star \( M^* \). Section 5 shows that \( M^* \) is sufficient to solve the problem, respectively. Section 6 concludes the paper with some practical remarks.

2. Motivations and Related Works

Actually, the main difficulty in solving the Non-Blocking Atomic Commitment problem in presence of process crashes lies in the detection of crashes. As a way of getting around the impossibility of Consensus, Chandra and Toueg extended the asynchronous model of computation with unreliable failure detectors and showed in [4] that the FLP impossibility can be circumvented using failure detectors. More precisely, they have shown that Consensus can be solved (deterministically) in an asynchronous system augmented with the failure detector \( \Diamond S \) (Eventually Strong) and the assumption of a majority of correct processes. Failure detector \( \Diamond S \) guarantees Strong Completeness, i.e., eventually, every process that crashes is permanently suspected by every process, and Eventual Weak Accuracy, i.e., eventually, some correct process is never suspected. Failure detector \( \Diamond S \) can however make an arbitrary number of mistakes, i.e., false suspicions.

A Non-Blocking Atomic Commitment problem, simply NB-AC, is an agreement problem so that it is impossible to solve in asynchronous distributed systems with crash failures. This stems from the FLP result which mentioning the consensus problem can’t be solved in asynchronous systems. Can we also circumvent the impossibility of solving NB-AC using some failure detector? The answer is of course “yes”. The NB-AC algorithm of D. Skeen [16] solves the NB-AC problem with assuming that it has the capability of the failure detector \( P \) (Perfect) in asynchronous distributed systems. This failure detector ensures Strong Completeness (recalled above) and Strong Accuracy, i.e., no process is suspected before it crashes [2]. Failure detector \( P \) does never make any mistake and obviously provides more knowledge about failures than \( \Diamond S \).

But it is stated in [7] that Failure detector \( \Diamond S \) cannot solve the NB-AC problem, even if only one process may crash. This means that NB-AC is strictly harder than Consensus, i.e., NB-AC requires more knowledge about failures than Consensus. An interesting question is then “What is the weakest failure detector for
solving the NB-AC problem in asynchronous systems with unreliable failure detectors?" In this paper, as the answer to this question, we show that there is a failure detector that solves NB-AC weaker than the Perfect Failure Detector. This means that the weakest failure detector for NB-AC is not a Perfect Failure Detector \( P \).

3. Model

We consider in this paper a crash-prone asynchronous message passing system model augmented with the failure detector abstraction [3].

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 DM and initially 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 transactions.

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. 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.
- Non-Triviality: 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. Non-Triviality 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.

4. The modal failure detector star \( M* \)

Each module of failure detector \( M* \) outputs a subset of the range \( 2^\Omega \). Initially, every process is suspected. However, if any process is once confirmed to be correct by any correct process, then the confirmed process id is removed from the failure detector list of \( M* \). If the confirmed process is suspected again, the suspected process id is inserted into the failure detector list of \( M* \). The most important property of \( M* \), denoted by Modal Accuracy, is that a process that was once confirmed to be correct is not suspected before crash. Let \( H_M \) be any history of such a failure detector \( M* \). Then \( H_M(i,t) \) represents the set of processes that process \( i \) suspects at time \( t \). For each failure pattern \( F, M(F) \) is defined by the set of all failure detector histories \( H_M \) that satisfy the following properties:

- **Strong Completeness**: There is a time after which every process that crashes is permanently suspected by every correct process:
  - \( \forall i,j \in \Omega, \forall i\in correct(F), \exists j\in F(t), \exists t': t'>t', j\in H_M(i,t') \).

- **Eventual Strong Accuracy**: There is a time after which every correct process is never suspected by any correct process. More precisely:
- \( \forall i, j \in \Omega \forall i \in \text{correct}(F), \exists t: \forall t' > t, \forall j \in \text{correct}(F), j \notin H(i, t') \).

**Modal Accuracy**: Initially, every process is suspected. After that, any process that is once confirmed to be correct is not suspected before crash. More precisely:
\(- \forall i, j \in \Omega: j \in H(i, t_0), t_0 < t < t', j \notin H(i, t) \land j \in \Omega \cdot F(t') \Rightarrow j \notin H(i, t') \)

Note that Modal Accuracy does not require that failure detector \( M^* \) keeps the Strong Accuracy property over every process all the time. However, it only requires that failure detector \( M^* \) never makes a mistake before crash about the process that was confirmed at least once to be correct.

If process \( M^* \) outputs some crashed processes, then \( M^* \) accurately knows that they have crashed, since they had already been confirmed to be correct before crash. However, concerning those processes that had never been confirmed, \( M^* \) does not necessarily know whether they crashed (or which processes crashed).

5. **Solving NB-AC Problem with \( M^* \)**

We give in Figure 1 an algorithm solving NB-AC using \( M^* \) in any environment of group where at least one node is available. The algorithm uses the fact that eventual strong accuracy property of \( M^* \). More precisely, with such a property of \( M^* \) and the assumption of at least one node being available, we can implement our algorithm of Figure 1.

```
Var status: \{rem, try, ready\} initially rem
Var coordinator: initially NULL
Var token: initially empty list
Var group: set of processes

Periodically(\( \tau \)) do
request \( M^* \) for \( H_M \)

1. **Upon received** (trying, upper_layer)
2. if not (status = try) then
3. wait until \( \forall j \in \text{group}_i \land j \notin H_M \)
4. status := try
5. **send** (ready, \( i \)) to \( \forall j \in \text{group}_i \)

6. **Upon received** (ok, \( j \))
7. token := token \cup \{j\}
8. **If** group = token **then**
9. **send** (commit, \( i \)) to \( \forall j \in Q_k \)
10. status := rem

11. **Upon received** (ready, \( j \))
12. if status = rem then **send** (ok, \( i \)) to \( j \)
13. coordinator := \( i \)
14. status := ready
15. else **send** (no, \( i \)) to \( j \)

16. **Upon received** (no, \( j \))
17. if status = try then **send** (abort, \( i \)) to \( \forall j \in \text{group}_i \)
18. status := rem

19. **Upon received** (abort, \( j \))
20. if status = ready then **do abort**()
21. status := rem

22. **Upon received** (commit, \( j \))
23. if status = ready then commit-transaction()
24. status := rem

25. **Upon received** \( H_M \) from \( M_i \)
26. if (status = try \& \exists j \in \text{my_group} \text{and} H_M) then **send** (abort, \( i \)) to \( \forall j \in \text{my_group} \text{abort-transaction}() \)
27. status := rem
28. if (status = ready \& coordinator \in H_M) then coordinator := NULL abort-transaction()
29. status := rem
```

Figure 1: NB-AC algorithm using \( M^* \) : process \( i \).

We give in Figure 1 an algorithm solving NB-AC using \( M^* \) in any environment \( E \) of a group with any number of correct processes (\( f < n \)). Our algorithm of Figure 1 assumes:
- Each process \( i \) has access to the output of its modal failure detector module \( M^*_i \);
- At least one process is available;

In our algorithm of Figure 1, each process \( i \) has the following variables:
1. A variable status, initially rem, represents one of the following states \{rem, try, ready\};
2. A variable coordinator, initially NULL, which denotes the coordinator when \( i \) sends its ok message to other node;
3. A list token, initially empty, keeping the ok messages that \( i \) has received from each member of the group.

Description of [Line 1-5] in Figure 1; the idea of our algorithm is inspired by the well-known NB-AC algorithm of D. Skeen[4,7]. That is, the processes that wish to try their Atomic Commitment first wait for the group whose members are all alive based on the information \( H_M \) from its failure detector \( M^* \). Those
processes eventually know the group by the eventual strong accuracy property of $M^*$ in line 3 of Figure 1 and then sets its status to “try”, meaning that it is try to commit. It sets the variable group with all members and send the message “(ready, i)” to all nodes in the group.

Description of [Line 6-10] in Figure 1: the coordinator asking for a ready to proceed an atomic commitment from every process of the group does not take steps until the all “ok messages” are received from the group. But it eventually received ok or no messages from the group, and it will commits or aborts the transaction.

Description of [11-15] in Figure 1: On received “ready message from the coordinator, the node sends “ok” to the coordinator and it set its status with “ready” meaning that it is in ready state to wait a decision that is “commit” or “abort”.

Description of [16-18] in Figure 1: If the coordinator received the message “no” from a node of group, it sends the “abort” message to every member of the group and after that it remains in “rem” state again.

Description of [19-21] in Figure 1: The node i, received “abort” from coordinator j, if it is in ready state, aborts the transaction.

Description of [22-24] in Figure 1: The node i, received “commit” from coordinator j, if it is in ready state, commits the transaction.

Description of [25-27] in Figure 1: When the node i received the failure detector history $H_m$ from $M^*$, if it is a coordinator and knows that a node of group died, it sends the abort message to all members of group.

Description of [28-29] in Figure 1: Upon received the failure detector history $H_m$ from $M^*$. If it is a node waiting a decision from the coordinator and it knows that the coordinator died, it aborts the transaction.

Now we prove the correctness of the algorithm of Figure 1 in terms of two properties: Uniform-Agreement and Uniform-Validity. Let $R$ be an arbitrary run of the algorithm for some failure pattern $F \in E$ ($f < n$). Therefore we prove Lemma 1 and 2 for $R$ respectively.

**Lemma 1.** (Uniform-Agreement) No two participants atomic-commit decide different outcomes.

**Proof:** By contradiction, assume that $i$ and $j$ ($i \neq j$) have made a different decision, one is commit and other is abort at time $t'$. According to the line 7-9 of the algorithm 1, the process $i$ sends “ok” message and $j$ sends “no” message to the coordinator. Without loss of generality, one of the following events occurred before $t''$ at every member of a group:

1. Assume the event that $i$ received “commit” message from the coordinator. Then all participants of group eventually received the “commit” message from the coordinator: a contradiction.
2. Assume the event that $j$ received “abort” message from the coordinator. Then all participants of group eventually received the “abort” message from the coordinator: a contradiction.

Hence, Uniform-Agreement is guaranteed.

**Lemma 2.** (Uniform-Validity) If a participant atomic decides commit, then all participants have voted yes.

**Proof:** Assume that a correct process $i$ sends “no” message but commits the transaction at time $t'$, and all correct processes except $i$ send “ok” message to the coordinator after $t'$. According to the algorithm, after $t'$, the coordinator eventually receives the messages from the group including process $i$ and make a decision: commit or abort. But the coordinator received at least one “no” message from the participant of group. It would send “abort” message to all member of group. So it is contradiction.

**Theorem 1** The algorithm of Figure 1 solves NB-AC using $M^*$, in any environment $E$ of a group with $f < n$, combining with two lemmas 1 and 2.

**6. Concluding remarks**

Is it beneficial in practice to use a Non-Blocking Atomic Commitment algorithm based on $M^*$, instead of a traditional algorithm assuming $P$? The answer is “yes”. Indeed, if we translate the very fact of not trusting a correct process into a mistake, then $M^*$ clearly tolerates mistakes whereas $P$ does not. More precisely, $M^*$ is allowed to make up to $n$ mistakes (up to $n$ mistakes for each module $M_i$, $i \in \Pi$). As a result, $M^*$’s implementation has certain advantages comparing to $P$’s (given synchrony assumptions).

For example, in a possible implementation of $M^*$, every process $i$ can gradually increase the timeout corresponding to a heart-beat message sent to a process $j$ until a response from $j$ is received. Thus, every such timeout can be flexibly adapted to the current network conditions. In contrast, $P$ does not allow this kind of “fine-tuning” of timeout: there exists a maximal possible timeout, such that $i$ starts suspecting $j$ as soon as timeout exceeds. In order to minimize the probability of mistakes, it is normally chosen sufficiently large, and the choice is based on some a priori assumptions about current network conditions.

This might exclude some remote sites from the group and violate the properties of the failure detector.
Thus, we can implement $M^*$ in a more effective manner, and an algorithm that solves NB-AC using $M^*$ exhibits a smaller probability to violate the requirements of the problem, than one using $P$, i.e., the use of $M^*$ provides more resilience.

7. References