Reducing the Cost for Non-Blocking in Atomic Commitment *

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

Abstract

Non-blocking atomic commitment protocols enable a decision (commit or abort) to be reached at every correct participant, despite the failure of others. The cost for non-blocking implies however (1) a high number of messages and communication steps required to reach commit, and (2) a complicated termination protocol needed in the case of failure suspicions. In this paper, we present a non-blocking protocol, called MD3PC (Modular and Decentralized Three Phase Commit), which enables to trade resiliency against efficiency. As conveyed by our performance measures, MD3PC is faster than existing non-blocking protocols, and in the case of a broadcast network and a reasonable resiliency rate (e.g 2 or 3) is almost as efficient as the classical (blocking) 2PC. The termination protocol of MD3PC is encapsulated inside a majority consensus protocol. This modularity leads to a simple structure of MD3PC and enables a precise characterization of its liveness in an asynchronous system with an unreliable failure detector.

1 Introduction

1.1 Atomic commitment

Failure atomicity of transactions, also called *all-or-nothing* property [1], is ensured by an *atomic commitment* protocol. Launched at the end of a transaction, the atomic commitment protocol returns either *commit* or *abort*. The simplest and best known atomic commitment protocol is the so called *Two Phase Commit (2PC)* protocol [7]. The 2PC is considered as a "cheap" protocol as it only needs 3 communication steps and $3n$ messages to commit (where n is the number of participants in the protocol) [1]. The main drawback of 2PC is to be blocking: if the coordinator crashes while the participants are uncertain about the *commit/abort* outcome of the transaction, the participants might have to wait for the recovery of the coordinator. During this blocking period, valuable resources (i.e locked objects) are kept inaccessible to other transactions. An atomic commitment protocol is said to be *non-blocking* if it enables a decision to be reached by correct participants despite the failure of some.

*To appear in the IEEE International Conference on Distributed Computing Systems (ICDCS-96), Hong Kong, May 1996.

1.2 The cost for non-blocking

The *Three Phase Commit* (3PC) protocol [8] is the best known example of a non-blocking atomic commitment protocol. The 3PC has, however, two drawbacks making it rarely used in practice. First, to reach commit, 3PC needs 5 communication steps and $5n$ messages (i.e 2 more communication steps and $2n$ more messages than 2PC) [1]. As a consequence, 3PC has a higher latency than 2PC (i.e. needs more time to commit). Second, in the case of failure suspicions, 3PC has a complicated *termination protocol.* For example, it contains a leader election subprotocol requiring to check several cases depending on the participant states [8]. We have been considering the issue of designing non-blocking, yet efficient and modular, atomic commitment protocols. In [4], we have discussed the efficiency of the decentralized version of the 3PC, called D3PC [8], and we have shown that its termination protocol can be encapsulated inside a *uniform consensus* protocol (as defined in [2]). As it needs only 3 communication steps to commit (like 2PC), D3PC has a lower latency than 3PC. However, whereas 2PC needs $3n$ messages to commit, $n(2n+1)$ messages are necessary in D3PC[1]. Hence, when a large number of participants are involved, the number of messages leads to bad performances with D3PC.

1.3 The MD3PC protocol

In this paper, we present a protocol called *MD3PC (Modular Decentralized Three Phase Commit)* which can be viewed as an optimization of D3PC. Like 2PC and D3PC, MD3PC also requires only 3 communication steps to reach commit. A fundamental characteristic of MD3PC is that the subprotocol required for non-blocking is only performed by a subset of the participants and the cardinality of this subset depends on the number of crash failures to be tolerated (i.e on the resiliency rate). This modularity enables to trade the resiliency against the number of messages needed to reach a commit decision. More precisely, to be resilient to f participant crash failures (given $f < n/2$), MD3PC needs $3n(f+1)$ messages, whereas with a broadcast network, (only) $n+f+2$ messages.
The performance measures, that we present in the paper, show that MD3PC is faster than 3PC and D3PC. Furthermore, in the case of a reasonable resiliency rate (e.g $f = 2$ or 3) and with a broadcast network, MD3PC is almost as efficient as 2PC (whatever the participant number). The termination protocol of MD3PC can be encapsulated inside any protocol that solves the *majority consensus* problem, which we introduce as a simple variation of the well-known uniform consensus problem. This modular structure leads to a simple structure of MD3PC and enables to give a precise characterization of its liveness, using the recent work of Chandra and Toueg on solving consensus problems with unreliable failure detectors [2]. We show that up to $f < n/2$ crash failures MD3PC eventually terminates in an asynchronous system with any unreliable failure detector in class $\diamond\mathcal{S}$ [2].
The rest of the paper is organized as follows. In Section 2, we define the system model and present the non-blocking atomic commitment problem. Then we define the majority consensus problem, used later to characterize the termination of MD3PC. In Section 3, we present the MD3PC protocol. In Section 4, we compare 2PC, 3PC, D3PC and MD3PC, in terms of their latency and number of messages. Then we present some performance figures. In Section 5, we summarize the main contributions of the paper and we discuss some open

[1] With broadcast communication (broadcast network), $n+2$ messages are needed in 2PC, and $2n+1$ are needed in D3PC.

questions. The appendix contains the correctness proof of MD3PC.

2 Model and problems

2.1 System Model

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*. 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]. Processes fail by crashing; we do not consider Byzantine failures.

We assume that our asynchronous system is augmented with a failure detector [2]. A failure detector can be viewed as a distributed oracle. Each process p_i has access to a local failure detector module D_i. This module maintains a list of processes that it currently *suspects* to have crashed. We do not make any assumption on how a failure detector is implemented. It can, for example, simply be based on time-outs, and failure detector modules can consult each other before suspecting a process.

Failure detectors are abstractly characterized by their *completeness* and *accuracy* properties. In the following, we consider failure detectors in class $\Diamond\mathcal{S}$ (*Eventually Strong*) [2]. These failure detectors guarantee (1) *strong completeness* (eventually every process that crashes is permanently suspected by every correct process), and (2) *eventual weak accuracy* (eventually some correct process is never suspected by any correct process). Notice that failure detectors in $\Diamond\mathcal{S}$ are unreliable: they can make mistakes, i.e false failure suspicions. The fundamental characteristic of $\Diamond\mathcal{S}$ is that it is the *weakest* class of failure detectors that allow to solve the consensus (and uniform consensus) problem(s) [2, 3].

In the consensus and uniform consensus problems, every process p_i starts with a binary initial value v_i ($v_i \in \{0, 1\}$) and the processes must agree on some binary decision value v ($v \in \{0, 1\}$). The consensus problem is defined by the three conditions: (1) *agreement*: no two correct processes decide differently; (2) *uniform-validity*: the decision value is the initial value of some process; (3) *termination*: every correct process eventually decides. The uniform consensus problem is defined by the *uniform-validity* and *termination* conditions of consensus, and the following *uniform-agreement* condition: no two processes (correct or not) decide differently.

2.2 The Non-blocking atomic commitment problem

The aim of an atomic commitment protocol is to bring the participants in a transaction to agree on an outcome of the transaction (either *commit* or *abort*). The outcome of the transaction depends on the *votes (yes* or *no)* provided by the participants. Each vote reflects the ability of the participant to ensure that its data object updates can be made permanent. 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 that *commit* must be decided if (i) all votes are *yes* and (ii) *there is no failure* [8]. As we show in [5], this *non-triviality* condition is very strong, since it requires precise knowledge about failures. This knowledge can only be provided by reliable failure detectors. We consider a weaker

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

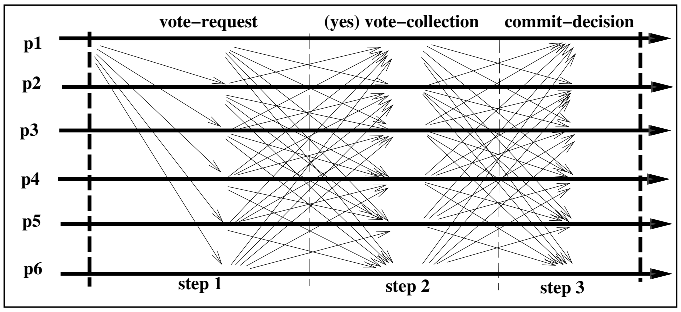

Figure 1: The D3PC protocol: coordinator is p_1

non-triviality condition where *commit* must be decided if (i) all votes are *yes*, and (ii) no process is ever suspected.
As in [5], we thus define the non-blocking atomic commitment problem by the following four conditions: (1) *uniform-agreement:* no two participants decide differently; (2) *uniform-validity:* if the decision is *commit*, then all participants have voted *yes*; (3) *termination:* every correct participant eventually decides; (4) *non-triviality:* if no participant votes *no* and no participant is ever suspected, then the decision is *commit.*[3]

2.3 The majority consensus problem

While the D3PC protocol requires uniform consensus as a termination protocol [4], the protocol we present in this paper, namely MD3PC, requires a variation of uniform consensus which we call *majority consensus.* The majority consensus problem is defined by the *uniform-agreement* and *termination* conditions of uniform consensus, and the following *majority-uniform-validity* condition: (i) the decision value is the initial value of some participant, and (ii) if a majority of initial values are 1, then the decision value is 1. This *majority-uniform-validity* condition makes the problem asymetric, since 1 is privileged over 0. As we will see in Section 3, this asymetry enables to reduce the number of messages required to decide commit in step 3 of the protocol.
We will show in Section 3.4 that like consensus and uniform consensus, the majority consensus problem can also be solved with any failure detector in class $\diamond\mathcal{S}$.

3 The MD3PC Protocol

As the MD3PC protocol can be viewed as an optimization of D3PC, we first recall the basic characteristics of the latter protocol. Then we give an overview of MD3PC, by pointing out the trade off between the number of messages needed to commit and the resiliency rate. Finally, we give a detailed description of MD3PC. The correctness proofs are sketched at the end of this section, and detailed in the appendix.

3.1 Overview of D3PC

[3]Note that if the failure detector is reliable, this definition turns out to be the classical definition given in [8].

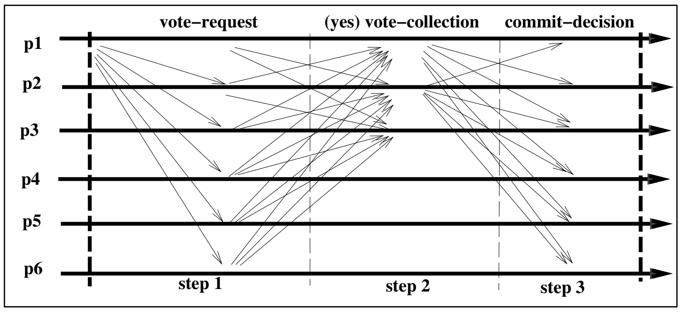

Figure 2: The MD3PC protocol: coordinator is p_1 and $Set_{NB} = \{p_1, p_2, p_3\}$

The D3PC protocol has three steps. Figure 1 illustrates the protocol in the "good" case, where every participant votes *yes* and no participant is suspected to have crashed. The set of participants is $\{p_1, p_2, p_3, p_4, p_5, p_6\}$ and p_1 is the coordinator. In step 1, the coordinator asks all participants to vote. A participant that votes *yes* sends its vote to all the participants. In step 2, when a participant receives a *yes* vote from all, it sends a *preCommit* message to all the participants. In step 3, when a participant receives a *preCommit* message from all, it decides *commit*.
A participant that votes *no* decides *abort* immediately (in step 1), and in a unilateral way[4]. The termination protocol invoked in the case of a failure suspicion can be encapsulated inside a uniform consensus protocol [4].

3.2 Overview of MD3PC

Like D3PC, MD3PC has three communication steps. Figure 2 illustrates the MD3PC protocol in the "good" case, where every participant votes *yes* and no participant is suspected. The steps have the same structure in both protocols but less messages are sent in MD3PC. More precisely, step 2 of MD3PC (which is actually the step needed to ensure non-blocking) is only executed by a subset noted Set_{NB} of the participants.
In step 1 of MD3PC, like in D3PC, the coordinator asks every participant to vote. Votes are, however, only sent to the members of Set_{NB} (whereas they are sent to all in D3PC). If f is the resiliency rate (the number of crash failures to be tolerated), we require Set_{NB} to contain $2f+1$ members. In Figure 2 for example, we assume $f = 1$ and we have $Set_{NB} = \{p_1, p_2, p_3\}$ (i.e $|Set_{NB}| = 3$). In step 2, when a member of Set_{NB} receives a *yes* vote from all it sends a *preCommit* message to all. Finally, in step 3, when a participant receives a *preCommit* message from a *majority* of Set_{NB} (e.g $\{p_1, p_2\}$ in Figure 2), it decides *commit*. This is a subtle difference with D3PC where no participant can decide *commit* until it receives *preCommit* message from all[5].
If, during step 1, a participant p_i either suspects the coordinator or votes *no*, then p_i decides *abort* in a unilateral way. The handling of the other cases remains simple, as they can be reduced to majority consensus where *commit* corresponds to the value 1 and *abort* corresponds to the value 0. The majority consensus protocol is launched by the members of Set_{NB}, in

[4]Without consulting any other participant.

[5]A similar idea was proposed in the Q3PC protocol [9] to deal with false failure suspicions. In Q3PC however, the votes are sent to all participants (rather than only to a subset Set_{NB} in MD3PC) and the majority is required from all the participants (rather than only from a subset Set_{NB} in MD3PC).

the case of failure suspicions. In step 2, a failure suspicion leads a member p_i of Set_{NB} to start majority consensus with *abort* as its initial value. In step 3, a failure suspicion leads p_i to start majority consensus with *commit* as its initial value.

In the following section, we describe the MD3PC protocol in more details. We assume that there is a majority consensus protocol, which we will discuss in Section 3.4.

3.3 Description of MD3PC

The detailed MD3PC protocol is given by the function *atomicCommitment()* in Figure 3. This function is called by every correct participant p_i. The majority consensus problem is solved by the function *majConsensus()* which takes as argument either *commit* or *abort*, and returns either of these values. We assume that all the participants know that p_1 is the coordinator and know the members of Set_{NB}. During the MD3PC protocol, every process p_i is informed by its local failure detector module $\mathcal{D}_i$ of crash suspicions. For example, the notation $coord \in \mathcal{D}_i$, at line 5, indicates that process p_i suspects the coordinator. The function *atomicCommitment()* triggers two concurrent tasks noted *Task 1* (lines 2 to 30) and *Task 2* (lines 31 to 33), and terminates by the execution of a "**return** v" statement, where v is either *commit* or *abort* (lines 7, 10, 19, 25, 30, and 33). When p_i executes **return** v, it decides on the outcome v of the transaction.

Task 2 is a decision task. It waits for a decision message $(p_j, outcome, decide)$ and forwards the *outcome* to the other participants (line 32). This ensures that if one correct participant decides on an outcome then all the correct participants also decide on the outcome.

Task 1 executes the main protocol (i.e step 1, step 2 and step 3).

Step 1 (lines 2-11): At line 3, the coordinator sends the *voteRequest* message to all. From there on, the protocol is decentralized: there is no coordinator. A participant p_i waits until (i) it receives the message *(voteRequest,step1)* from the coordinator, or (ii) it suspects the coordinator (line 4). In case (ii), p_i sends the message $(p_i, abort, decide)$ to all (line 6) and decides *abort* (line 7). In case (i), p_i defines its vote and, if the vote is *no* (line 8), sends the message $(p_i, abort, decide)$ to all (line 9), and decides *abort* (line 10). If p_i votes *yes*, it sends the message $(p_i, yes, step2)$ to the members of Set_{NB} (line 11). Only the members of Set_{NB} proceed to step 2. The others proceed directly to step 3.

Step 2 (lines 12-20): A process $p_i \in Set_{NB}$ waits until (i) it receives *yes* votes from all participants, or (ii) it suspects another participant (line 14). In case (ii), p_i asks all the members of Set_{NB} to start the majority consensus protocol by sending them the message $(p_i, startCons)$ (line 16), then p_i also starts majority consensus with *abort* as its initial value (line 17). The decision of the majority consensus protocol becomes the outcome of p_i, which is forwarded to all participants (line 18). In case (i), p_i sends the message $(p_i, preCommit, step3)$ to all participants (line 20) and proceeds to step 3.

Step 3 (lines 21-30): A participant p_i waits until (i) it receives a *(-,preCommit,step3)* message from a majority of Set_{NB}, or (ii) it receives a *startCons* message (this possibility is only for members of Set_{NB}). In case (ii), p_i starts the majority consensus protocol with *commit* as its initial value (line 23). The decision of the majority consensus protocol becomes the outcome of p_i and is forwarded to all participants (line 24). In case (i), p_i sends the message $(p_i, commit, decide)$ to all participants (line 29) and decides *commit* (line 30).

3.4 Correctness of MD3PC

We present now four propositions that state the correctness of the MD3PC protocol presented in Figure 3. Each proposition corresponds to a condition of the non-blocking atomic commitment problem (as defined in Section 2.2). We only sketch the proofs here (see [6] for the complete proofs).

Proposition 1 (Validity). *If commit is decided then all participants have voted yes.*

PROOF (SKETCH). *Commit* is decided only if some participant p_i has decided *commit* at lines 19, 25 or 30 of Task 1. For p_i to reach line 25 or line 30, all participants must have sent a *yes* vote. By the *majority-uniform-validity* condition of majority consensus, p_i can decide *commit* at line 19, only if some participant p_j in Set_{NB} has started majority consensus with *commit* as its initial value, which implies that all participants have sent a *yes* vote. □

Proposition 2 (Uniform-Agreement). *No two participants decide differently.*

PROOF (SKETCH). By Proposition 1, if some participant does not send a *yes* vote to all Set_{NB} members, then no participant can decide *commit*, and obviously no two participants can decide differently. Assume thus that all participants send a *yes* vote to all the members of Set_{NB}. We consider two cases: (i) at least one participant decides *commit* at line 30, or (ii) no participant decides *commit* at line 30. We show that in case (i), no participant can decide *abort*. In case (ii), we show that every participant that decides, decides through majority consensus. By the *agreement* condition of majority consensus, no two participants decide differently. □

Proposition 3 (Termination). *If $f < |Set_{NB}|/2$ and the failure detector is in $\diamond\mathcal{S}$, then every correct participant eventually decides.*

PROOF (SKETCH). We consider two cases: (i) at least one correct participant p_i does not send a *yes* vote, or (ii) all correct participants send a *yes* vote (line 11). In case (i), we show that by the assumption of reliable channels, and the *strong completeness* property of $\diamond\mathcal{S}$, every correct participant eventually decides *abort*. Consider case (ii). Either some correct participant receives a *preCommit* message from a majority of members of Set_{NB}, in which case all correct participant decides *commit*. Otherwise, by the assumption of reliable channels, the assumption that a majority of Set_{NB} are correct, the *strong completeness* property of $\diamond\mathcal{S}$, every correct participant in Set_{NB} eventually decides through majority consensus, and hence every correct participant eventually decides. □

Proposition 4 (Non-Triviality). *If no participant votes no and no participant is ever suspected, then the decision is commit.*

PROOF (SKETCH). If *abort* is decided, then some participant has decided *abort* in Task 1, either at line 7, 10, 19, or 25. At lines 7, 10, and 19, a participant can decide *abort* only if

```
function atomicCommitment()

    PreCommitNbr := 0; coord := p_1;   /* Initialisation */

cobegin
||  /* Task 1 */
                                                                   /* Step 1 */
    if p_i = coord then   /* Only the coordinator executes: */
        send (voteRequest,step1) to all participants ;

    wait until [received (voteRequest,step1) from coord or coord ∈ D_i] ;
        if coord ∈ D_i then
            send (p_i, abort, decide) to all participants ;
            return abort ;   /* Abort decision */
        if (unable to make updates permanent) then   /* vote = no */
            send (p_i, abort, decide) to all participants ;
            return abort ;   /* Abort decision */
        send (p_i, yes, step2) to all the members of Set_NB ;

                                                                   /* Step 2 */
    if p_i ∈ Set_NB then   /* Only the members of Set_NB execute: */
        for each participant p_j :
            wait until [received (p_j, yes, step2) from p_j or p_j ∈ D_i] ;
                if p_j ∈ D_i then
                    send (p_i, startCons) to all the members of Set_NB ;
                    outcome := majConsensus(abort) ;   /* Termination by majConsensus */
                    send (p_i, outcome, decide) to all participants ;
                    return outcome ;
        send (p_i,preCommit, step3) to all participants ;

                                                                   /* Step 3 */
    loop
        wait until [received ((p_j,preCommit, step3) or (p_j, startCons)) from p_j] ;
            if (p_i ∈ Set_NB and (received (p_j, startCons) from p_j)) then
                outcome := majConsensus(commit) ;   /* Termination by majConsensus */
                send (p_i, outcome, decide) to all participants ;
                return(outcome) ;
            if (received (p_j,preCommit, step3) from p_j) then
                PreCommNbr := PreCommNbr + 1;
                if PreCommNbr > |Set_NB|/2 then;
                    send (p_i, commit, decide) to all participants ;
                    return commit ;   /* Commit decision */
    endloop

||  /* Task 2 */
    wait until [received (p_j, outcome, decide) from p_j] ;
        send (p_i, outcome, decide) to all participants ;
        return outcome ;
coend
```

Figure 3: The MD3PC protocol of a participant p_i

Protocol	Number of steps	Number of messages point–to–point communication	Number of messages broadcast communication
2PC	3	3n	n+2
3PC	5	5n	2n+3
D3PC	3	n(2n+1)	2n+1
MD3PC	3	3n(f+1)	n+f+2

Figure 4: Steps and messages in the "good" case

it has suspected another participant, or if it has voted *no*. For a participant to reach line 25, some participant must have been suspected. □

It is important to remark that, in order to satisfy *uniform-agreement*, the properties of $\Diamond\mathcal{S}$ are not necessary. This means that if these properties are not ensured, the protocol might block but it will never allow two participants to reach different decisions.

4 Evaluation of MD3PC

In this section, we compare the performances of the MD3PC protocol with 2PC, 3PC and D3PC, in the "good" case, i.e. when there is no suspicion and no participant votes *no* [6].
The table, in Figure 4, shows the number of messages and communication steps needed to reach commit with each protocol. As we have already pointed out, 2PC, D3PC, and MD3PC, need 3 communication steps whereas 3PC needs 5 communication steps. When establishing the number of messages needed, we distinguish two cases: (a) using point-to-point communication, and (b) using broadcast communication (broadcast network). For example, in case (a), with 12 participants ($n = 12$) and a resiliency rate of 2 ($f = 2$), 2PC requires 36 messages, 3PC requires 60, D3PC requires 300, and MD3PC requires 108 messages. In case (b), 2PC requires 14 messages, 3PC requires 27, D3PC requires 25, and MD3PC requires 16 messages. It is thus not surprising that, using broadcast communication, MD3PC has almost the same performances as 2PC (see Figure 5 (b)): they both require 3 steps and nearly the same number of messages (14 and 16).
The tables, in Figure 5, compare the performances of the protocols, (a) using point-to-point communication, and (b) using broadcast communication. The measures have been obtained on FDDI (100Mb/s), with SPARC20 machines (running Solaris 2.3), using the UDP transport protocol. The measures convey an average time to commit a transaction. Different participants are located on different machines. The MD3PC protocol has been implemented given a resiliency rate of 2 ($f = 2$, i.e $|Set_{NB}| = 5$). The performances show that MD3PC is significantly faster than 3PC and D3PC, and using broadcast communication, MD3PC is almost as efficient as 2PC.

5 Concluding remarks

The MD3PC (Modular Decentralized Three Phase Commit) protocol, presented in this paper, provides the non-blocking feature while (1) requiring the same number of steps to commit

[6] This corresponds to the most frequent case.

(a) point–to–point communication

Protocol	3 participants	6 participants	9 participants	12 participants
2PC	4.4 ms. ± 0.0	6.5 ms. ± 0.0	10.1 ms. ± 0.1	12 ms. ± 0.1
3PC	7.5 ms. ± 0.0	11.4 ms. ± 0.0	17.7 ms. ± 0.1	21.2 ms. ± 0.2
D3PC	5.4 ms. ± 0.0	10 ms. ± 0.0	17 ms. ± 0.1	22 ms. ± 0.2
MD3PC	5.4 ms. ± 0.0	8.5 ms. ± 0.0	13 ms. ± 0.0	16 ms. ± 0.1

(b) broadcast communication

Protocol	3 participants	6 participants	9 participants	12 participants
2PC	5ms ± 0.0	7.5 ms. ± 0.0	10 ms. ± 0.1	12.3 ms. ± 0.1
3PC	8ms. ± 0.0	11.8 ms ± 0.0	16 ms. ± 0.1	21 ms. ± 0.2
D3PC	5.2 ms. ± 0.0	8 ms. ± 0.0	12 ms. ± 0.1	17.5 ms. ± 0.2
MD3PC	5.2 ms. ± 0.0	8 ms. ± 0.0	11 ms. ± 0.0	13.4 ms. ± 0.1

Figure 5: Time to commit a transaction ($f = 2$, $|Set_{NB}| = 5$).

than the classical (blocking) 2PC, and (2) trading the number of messages required to reach commit for the desired resiliency. This paper shows that MD3PC significantly provides better performances than the existing non-blocking 3PC and D3PC protocols [8]. Given a reasonable resiliency rate and using broadcast communication, the MD3PC is almost as efficient as the 2PC protocol [7].
Failure scenarios are handled using a termination protocol which is specified using the majority consensus problem (a variation of the well-known consensus problem [2]). This modularity leads to a simple structure for the MD3PC protocol and enables a rigorous characterization of its liveness in asynchronous systems with unreliable failure detectors.

References

[1] P.A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison Wesley. 1987.

[2] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. To appear in Journal of the ACM (1996). A preliminary version appeared in *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*, pp 325-340. ACM Press. August 1991.

[3] T. Chandra, V. Hadzilacos and S. Toueg. The weakest failure detector for solving consensus. In *Proceedings of the 11th ACM Symposium on Principles of Distributed Computing*, pp 147-159. ACM Press. August 1992.

[4] R. Guerraoui and A. Schiper. The Decentralized Non-Blocking Atomic Commitment Protocol. In *Proceedings of the 14th IEEE Symposium on Parallel and Distributed Processing*, pp 2-9. San Antonio (Texas), October 1995.

[5] R. Guerraoui. Revisiting the relationship between Non-Blocking Atomic Commitment and Consensus problems. In *Proceedings of the 9th International Workshop on Distributed Algorithms*, pp 87-100. (LNCS 972), Springer Verlag. 1995.

[6] R. Guerraoui and A. Schiper. Reducing the Cost for Non-Blocking in Atomic Commitment. Technical Report, Ecole Polytechnique Fédérale de Lausanne, Computer Science Department, 1996.

[7] J. Gray. Notes on Database Operating Systems. In *Operating Systems: An Advanced Course*, pp 10-17. Lecture Notes in Computer Science (60), Springer Verlag. 1978.

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

[9] D. Skeen. A Quorum-Based Commit Protocol. In *Proceedings of the Berkeley Workshop on Distributed Data Management and Computer Networks* (6), pp 69-80. February 1982.