Fast Non-Blocking Atomic Commit: An Inherent Trade-off

Partha Dutta, Rachid Guerraoui, Bastian Pochon *

Distributed Programming Laboratory, EPFL
CH-1015 Lausanne, Switzerland

Abstract

This paper investigates the time-complexity of the non-blocking atomic commit (NBAC) problem in a synchronous distributed model where t out of n processes may fail by crashing. We exhibit for $t \geq 3$ an inherent trade-off between the *fast abort* property of NBAC, i.e., aborting a transaction as soon as possible if some process votes "no," and the *fast commit* property, i.e., committing a transaction as soon as possible when all processes vote "yes" and no process crashes. We also give two algorithms: the first satisfies fast commit and a weak variant of fast abort, whereas the second satisfies fast abort and a weak variant of fast commit.

Key words: Distributed algorithms, complexity, atomic commit

1 Introduction

The synchronous model. We consider a set $\Pi = \{p_1, p_2, \ldots, p_n\}$ $(n \geq 3)$ of processes in a synchronous crash-stop model [5]. [1] The processes may fail by crashing and do not recover from a crash. Any process that does not crash in a run (any execution of an algorithm) is said to be *correct* in that run; otherwise the process is said to be *faulty*. In any given run, at most $t < n$ processes may crash, and we denote by f the effective number of processes that crash in that run. The processes proceed in rounds. Each round consists of two phases: (a) in the *send phase*, all processes (that did not crash) send messages to all processes; (b) in the *receive phase*, the processes receive the messages sent in the send phase of that round and update their local states. If some process p_i completes the send phase of the round, every process that completes the receive phase of the round receives the message sent by p_i in

* Corresponding author: Bastian.Pochon@epfl.ch

[1] We refer the reader to [5] for details on the model.

the send phase. If p_i crashes during the send phase, then any subset of the processes might not receive the message sent by p_i in that round.

The Non-blocking Atomic Commit Problem. In the *non-blocking atomic commit* problem [1,6] (NBAC), each process is supposed to cast a vote, either 0 or 1, proposing to either abort or commit a distributed transaction. Each process is supposed to eventually decide [2] on either 0 (abort the transaction) or 1 (commit the transaction), such that the following properties are satisfied: (uniform agreement) no two processes decide differently, (termination) every correct process eventually decides, (abort validity) 0 is the only possible decision if some process proposes 0, and (commit validity) 1 is the only possible decision if every process is correct and proposes 1.

The abort validity property of NBAC states that, if any process proposes 0, then 0 is the only possible decision value. This leads to an interesting observation: if a process p_i receives a message from any process that proposes 0, then p_i can immediately decide 0. Clearly, there is an algorithm which ensures a global decision [3] by round 1 in any run in which some process proposes 0 (no matter how many crashes occur in that run). This property, which we call *fast abort*, allows the processes to quickly retry committing a transaction in case of a "logical" abort. [4]

On the other hand we would also like to commit a transaction as fast as possible when all processes propose 1. In [2,4], it is shown that in runs with at most f crashes ($0 \leq f \leq t$), $min(f+2, t+1)$ [5] is a lower bound for a global decision. An algorithm that achieves this bound, for $0 \leq f \leq t$, is said to be *early deciding*. We say that a NBAC algorithm satisfies the *fast commit* property, if it globally decides by round 2 in every run in which all processes propose 1 and no process crashes. Note that early decision implies fast commit.

Contribution. Interestingly, for $t \geq 3$, we show that fast abort is incompatible with fast commit. More precisely, while fast abort and fast commit can both be individually achieved (as we discuss later in the paper), we prove that no single NBAC algorithm can have both properties. We also present two NBAC algorithms, each of these satisfying one of the properties and a weaker form of the other one. We say in this context that a NBAC algorithm satisfies *weak fast abort* if it globally decides by round 2 in every run in which some process proposes 0, and a NBAC algorithm satisfies *weak fast commit* if it globally decides by round 3 in every run in which all processes propose

[2] Throughout this paper, our bounds are for decision events, not halting events.
[3] A run globally decides in round k if every process that decides in that run, decides by round k, and some process decides in round k.
[4] I.e., some process proposes 0. This could occur for instance because of a concurrency control problem.
[5] For the sake of brevity we are being slightly imprecise here; the lower bound really is $f+2$ for $f \leq t-2$, and $f+1$ for $f \geq t-1$. The special case is $f = t-1$.

1, and no process crashes. Our first algorithm satisfies fast commit and weak fast abort, and our second algorithm satisfies fast abort and weak fast commit. Additionally, both algorithms match the bounds of [2,4] for the runs with process crashes, namely, they both globally decide in $min(f+2, t+1)$ rounds in runs with at most f crashes, provided $f \geq 1$.

2 Incompatibility of Fast Commit and Fast Abort

As previously mentioned, it is possible to globally decides by round 1 in every run in which some process proposes 0. However, observe that if a process p_i is required to decide in round 1 in any run in which some process proposes 0, then p_i has to decide 0 in round 1 if p_i does not receive the round 1 message from any other process p_j, because p_i does not know whether p_j proposed 0 or 1.

Proposition 1 *For $3 \leq t \leq n-1$, no NBAC algorithm can satisfy both the fast abort and the fast commit properties.*

Proof. Consider by contradiction a NBAC algorithm A which satisfies both fast abort and fast commit. We exploit indistinguishability between five different runs of A, and derive a contradiction.

1. In run $R1$, process p_1 proposes 0, and all other processes propose 1. Process p_1 crashes before sending any message in round 1. By abort validity, the only possible decision in this run is 0. By fast abort, every process distinct from p_1 decides 0 at the end of round 1, in particular p_2.

2. Run $R2$ starts from the initial configuration in which all processes propose 1 (including p_1). Process p_1 crashes in round 1 after sending a message to all processes but p_2. Clearly, p_2 cannot distinguish $R1$ from $R2$. Thus p_2 decides 0 at the end of round 1 in $R2$.

3. Run $R3$ is identical to $R2$, except that p_2 now crashes at the beginning of round 2, before sending any message in round 2, and p_3 crashes at the beginning of round 3. All remaining processes are correct. Clearly, at the end of round 1, $R2$ and $R3$ are indistinguishable for p_2, and hence, p_2 decides 0 at the end of round 1 in $R3$, and then crashes.

4. Run $R4$ is failure-free, starting from the initial configuration in which all processes propose 1. By fast commit, and commit validity, all processes decide 1 at the end of round 2 in $R4$, in particular p_3.

5. Finally, run $R5$ is similar to $R4$, but processes p_1 and p_2 crash in the send phase of round 2, such that both processes send a message to only p_3 in round

2, and process p_3 crashes at the beginning of round 3. Clearly, $R4$ and $R5$ are indistinguishable for p_3 at the end of round 2. Thus p_3 decides 1 at the end of round 2, and then crashes.

In $R3$, process p_2 decides 0 and crashes. In $R5$, process p_3 decides 1 and crashes. Runs $R3$ and $R5$ are however indistinguishable for all processes distinct from p_1, p_2, and p_3. To see why, observe that $R3$ and $R5$ are different only at p_1 and p_2 at the end of round 1, and $p1$ and p_2 send messages only to p_3 in round 2. None of the three processes send any messages after round 2. This contradicts uniform agreement. $\square$

In the next section, we circumvent this incompatibility by weaking one of the properties when $t \geq 3$. We give two algorithms: the first algorithm satisfies weak fast abort and fast commit, whereas the second algorithm satisfies fast abort and weak fast commit. For $t \leq 2$, it is possible to design an NBAC algorithm that satisfies both fast commit and fast abort: we give that algorithm in [3].

3 Fast NBAC Algorithms

In this section we assume that $t \geq 3$. We first give a NBAC algorithm in Fig. 1, which satisfies fast commit and weak fast abort. The algorithm is called FCWFA. It is a flooding algorithm, optimized for the fast commit and the weak fast abort properties, and the special case where $f = t - 1$. In round 1, the processes exchange their estimate *est*, initialized to their proposal value, and try to adjust their estimate in anticipation of a weak fast abort: if a process does not receive $est = 1$ from *all processes*, it changes its estimate to 0, as it might be the case that some process proposed 0. In round 2, after exchanging their estimate, the processes decide 0 if they are certain that any other process will either decide 0 or continue with a 0 estimate. Otherwise, the processes decide at the end of round 2 if they notice a failure-free run. From round 2 on, each process p_i records, in a set $Halt_i$, the identity of the processes known to have crashed. In the next rounds, processes exchange their estimate with each other, and update their set $Halt_i$ with the identity of the processes from which no message has been received. A process p_i decides in a round $r \geq 2$ whenever its set $Halt_i$ does not contain more than $r - 2$ processes.

Interestingly, FCWFA can easily be adapted to a binary uniform consensus algorithm reaching all known global decision lower bounds, by removing lines 7, 8, 19, and 20. Additionally, for the case $f = t - 1$, the resulting uniform consensus algorithm, can be viewed as a drastic simplification of the $Tree_t$ algorithm of [2].

With FCWFA, every process which decides, decides by round $f + 2$, for $f \leq$

1: At process p_i:
2: $est_i := \bot$; $decided_i := false$; $Halt_i := \emptyset$; $S_r := \emptyset,\ 1 \le r \le t+1$ % S_r is a multiset %

3: **procedure** propose(v_i)
4: $est_i := v_i$
5: send($1, est_i$) to all
6: $S_1 := \{ est_j \mid (1, est_j)$ has been received in round $1\}$
7: **if** $|S_1| < n$ or $\exists est_j \in S_1 : est_j = 0$ **then**
8: $est_i := 0$ $\{$*8': decide(0) ; $decided_i := true$*$\}$

9: **for** $r = 2 \ldots t+1$ **do**
10: **if** $decided_i$ **then** send(r, DEC, est_i) to all ; **return**
11: **else** send(r, EST, est_i) to all
12: $S_r := \{ est_j \mid (r, \text{EST}, est_j)$ has been received in round $r\}$
13: **if** receive any message (r, DEC, est_j) for some est_j **then**
14: $est_i := est_j$; decide(est_i) ; $decided_i := true$
15: **else**
16: $Halt_i := \Pi \backslash \{ p_j \mid est_j \in S_r \}$
17: **if** $\exists est_j \in S_r : est_j = 0$ **then**
18: $est_i := 0$
19: **if** $r = 2$ and $\forall est_j \in S_2 : est_j = 0$ **then** $\{$*19': **if** $r = 2$ and $|S_2| < n$ **then***$\}$
20: decide(0) ; $decided_i := true$ $\{$*20': $est_i := 0$*$\}$
21: **else if** $r \le t-1$ and $|Halt_i| \le r-2$ **then** $\{$*21': **else if** $3 \le r \le t-1$ and $|Halt_i| \le r-2$ **then***$\}$
22: decide(est_i) ; $decided_i := true$
23: **else if** $r = t$ and $|S_t| \ge n-t+1$ **then**
24: decide(est_i) ; $decided_i := true$

25: decide(est_i) ; **return**

Fig. 1. A fast commit, weakly fast abort, early deciding NBAC algorithm (FCWFA). Replacing line 8, 19, 20 and 21 with $8'$, $19'$, $20'$ and $21'$ gives a fast abort, weakly fast commit NBAC algorithm (FAWFC).

$t-2$, or round $f+1$, for $f \ge t-1$, in every run where there are at most f processes that crash (early deciding). For an intuition of why FCWFA is faster when $f = t-1$ (vs. $f \le t-2$), consider a run in which no process has decided by round $t-1$. At the end of round $t-1$, two processes have different estimates only if there remains at most a single process that may crash (that is, $f \ge t-1$). Hence, any process can decide on its estimate at the end of round t, if it receives $n-t+1$ messages in round t. For the sake of clarity, we omit the obvious optimization where any process which proposes 0 can decide 0 before taking any step in the algorithm.

Interestingly, a slight modification of FCWFA results in a second NBAC algorithm that satisfies weak fast commit and fast abort properties. This second algorithm is called FAWFC. The corresponding modifications are shown between brackets directly in Fig. 1.

We prove the correctness of the algorithms and their complexity properties. In both algorithms, variable S_r, for $1 \le r \le t+1$, denotes sets which can hold duplicate values at the same time. In the following proofs, we denote the local copy of a variable *var* at process p_i by var_i, and the value of var_i at the end of round r by var_i^r. We call a message carrying an estimate $est = 1$ a *commit* message, and similarly, a message carrying an estimate $est = 0$ an *abort* message. We denote by $crashed^r$ the set of processes that crash *before* completing round r. We first prove two general claims which hold for both algorithms.

Claim 2 *In FCWFA and FAWFC, if no process has decided by round $r-1 \geq 1$ and at the end of round r two distinct processes p_i and p_j are such that $est_i^r \neq est_j^r$, then $|crashed^r| \geq r$.*

Proof. We prove the claim by induction on the round number. We note that if no process decides by round $r-1$, then processes do not receive any DEC message in round r, and hence update their estimate in round r. For the base case $r = 2$, assume that the conditions of the claim hold, and that, w.l.o.g., $est_i^2 = 1$ and $est_j^2 = 0$. It follows that $est_j^1 = 1$; otherwise, upon receiving the abort message from p_j in round 2, p_i would have changed its *est* to 0. In round 2, since p_j changed its *est* from 1 to 0, p_j received at least one abort message that p_i has not received. Hence some process p_k sent an abort message in round 2 and crashed in the send phase of round 2 before sending the message to p_i. Thus, $est_k^1 = 0$. Furthermore, since $est_i^2 = 1$, est_i^1 is also 1, and it follows that p_i received commit message from all n processes in round 1. Since $est_k^1 = 0$ and all process have sent commit messages in round 1, p_k has received less than n message in round 1. Thus, some process distinct from p_k has crashed in round 1. Hence $|crashed^2| \geq 2$. Assume now the claim for round $r-1$ (induction hypothesis). We prove the claim for round r. Suppose that no process decides by round r and consider two distinct processes p_k and p_l such that $est_k^r = 1$ and $est_l^r = 0$. Clearly, $est_k^{r-1} = 1$. As both processes completed round r, p_k received round r message from p_l, hence $est_l^{r-1} = 1$. Thus there is a process p_x which sent an abort message to p_l in round r, and crashed before sending a round r message to p_k. Thus, $est_x^{r-1} = 0$. Since $est_k^{k-1} = 1$ and $est_x^{r-1} = 0$ and no process has decided by round $r-2$, from induction hypothesis it follows that $|crashed^{r-1}| \geq r-1$. As p_x crashes in round r, $|crashed^r| \geq r$. □

Claim 3 *In FCWFA and FAWFC, for any round $r \geq 2$ and any process p_i that completes round r without receiving a* DEC *message, $crashed^{r-1} \subseteq Halt_i^r$.*

Proof. Since p_i completes round r without receiving a DEC message, it updates $Halt_i$ in line 16. If a process p_j crashes by round $r-1$, then p_i does not receive round r message from p_j, and hence, includes p_j in $Halt_i$. □

The next two propositions assert the correctness and efficiency of FCWFA. (The corresponding proofs for FAWFC can be obtained by straightforward modifications; for space limitation, we give those proofs in [3].)

Proposition 4 *FCWFA solves NBAC.*

Proof. We prove here the termination, commit validity, abort validity, and agreement properties of NBAC in FCWFA.

Termination. All correct processes decide by round $t+1$, and no process blocks in any round.

Abort-Validity. If any process proposes 0 then, every process that completes round 1, either receives less than n messages or receives at least one abort message, and hence, executes line 8. Thus, in round 2, only abort messages are exchanged amongst processes. Every process that completes round 2 executes line 20 and decides 0.

Commit-Validity. Consider a run in which every process proposes 1 and no process fails. At the end of round 1, every process receives commit messages from n processes, and hence, does not executes line 8. Thus, in round 2, only commit messages are exchanged amongst processes. Consequently, processes receive n commit messages in round 2 as well, and for all processes, $Halt^2 = \emptyset$. Thus every process decides 1 at line 22.

Uniform Agreement. We consider the lowest round r in which at least one process decides. Let p_i be one of the processes that decides in round r, say on value v. We show that every process that decides in round r, decides v, and processes that complete round r without deciding, have $est^r = v$. This immediately implies uniform agreement. We consider four cases: (1) $r = 2$, (2) $3 \leq r \leq t-1$, (3) $r = t$, and (4) $r = t+1$. (Notice that no process decides in round 1.)

Case 1. Consider the subcase (1a) where $v = 1$. Since p_i decides 1, it did not receive any abort message. Furthermore, as p_i decides in round 2, $|Halt_i^2| \leq 0$, i.e., p_i received round 2 messages from all processes. In other words, p_i received n commit messages in round 2. Hence, all processes received n commit messages in round 1, and no process crashes before completing round 1. Therefore, only commit messages are sent in round 2. Thus, no process decides 0 in round 2, and every process that completes round 2, has $est^2 = 1$. Consider now the subcase (1b) where $v = 0$. Thus p_i receives only abort messages in round 2, including from itself. Since p_i completes round 2, any process that completes round 2, receives the abort message from p_i. Thus no process can decide 1 in round 2, and every process that completes round 2 without deciding, changes its *est* to 0 on receiving the abort message from p_i.

Case 2. We note that p_i must have decided at line 22. (Process p_i cannot decide at line 14 because r is the lowest round in which some process decides.) Suppose by contradiction that some process p_j decides $1-v$ in round r, or completes round r with $est^r = 1-v$. Since both p_i and p_j complete round r, they receive each other's round r messages. If any of them has $est = 0$ at the end of round $r-1$, then both processes would have $est^r = 0$. Hence, $est_i^{r-1} = est_j^{r-1} = 1$. Thus in round r, some process p_x sent an abort message to one of the processes (p_i or p_j) and not to the other one. Thus $est_x^{r-1} = 0$, and, by Claim 2, $|crashed^{r-1}| \geq r-1$. Thus, at the end of round r, by Claim 3,[6] $|Halt_i^r| \geq r-1$. A contradiction with the fact that p_i decides in line line 22 of round r.

Case 3. No process has decided by round $t-1$. If all processes that complete round $t-1$ have the same *est*, then uniform agreement trivially follows. Suppose two processes have different *est* at the end of round $t-1$. Then by Claim 2, $|crashed^{t-1}| \geq t-1$; i.e., there are at most $n-t+1$ processes that complete round $t-1$. Since p_i decides in round $r=t$, so p_i decides in line 24 and has received at least $n-t+1$ message in round t. Thus exactly $n-t+1$ processes complete round $t-1$. If any other process decides in round t, it receives the same $n-t+1$ messages as p_i, and hence, decides v. If a process p_j completes round t without deciding, then it has received $n-t$ messages in round t, and hence, t processes crash by round t. Then, p_i is a correct processes (as it has completed round t), and p_j receives the DEC message sent by p_i in round $t+1$, and decides v.

Case 4. If no process decides by round t and two processes have distinct *est* at the end of round $t+1$, then from Claim 2, $|crashed^{t+1}| \geq t+1$. A contradiction. □

Proposition 5 *FCWFA satisfies weak fast abort, fast commit, and early decision.*

Proof. For weak fast abort, consider a run that starts from an initial configuration where at least one process p_i proposes 0. Every process p_j which completes round 1 sets its estimate est_j to 0 at the end of round 1 (because either p_j receives p_i's abort message, or p_j does not receive any message from p_i). Thus processes receive only abort messages in round 2. Thus, every process that completes round 2, decides 0 at that round (line 20).

Notice that, early decision for $f=0$, implies fast commit. We now show that the algorithm satisfies early decision. Suppose, $f \leq t-2$ in a run, and some process p_i completes round $f+2$ without deciding. Then p_i has not received any DEC message by round $f+2$. We claim that every process in $Halt_i^{f+2}$ is faulty. Suppose otherwise; if some correct process p_j is in $Halt_i^{f+2}$, then p_j has halted after deciding, and it has sent a DEC message in round $f+2$ or a lower round. Since p_i has not received any DEC message by round $f+2$, no correct process is in $Halt_i^{f+2}$. Thus $|Halt_i^{f+2}| \leq f$. Thus, in round $f+2$, p_i evaluates the condition in line 21 to true, and decides in line 22. For the case where $f=t-1$, observe that, if $f=t-1$ processes crash in a run, and some process does not decides by round $t=f+1$, then at the end of round $t=f+1$, every process that is not crashed, either receives a DEC message or receives at least

[6] Since r is the lowest round in which some process decides, p_i does not receive any DEC message in round r.

$n - t + 1$ messages, and hence, decides on its estimate. If $f = t$, clearly, every process that decides, decides by round $f + 1 = t + 1$. $\square$

4 Concluding Remarks

In the decentralized (non-blocking) three-phase commit (D3PC) algorithm of [6], which is the fastest NBAC algorithm we knew of so far (in terms of number of rounds), all processes decide in round 1 in every failure-free run where some process proposes 0, and in round 2 in the failure-free run where all processes propose 1. In D3PC however, no process decides in round 1 in a run where some process proposes 0 and crashes before sending any message. This means, in our terminology, that D3PC satisfies fast commit but not fast abort, which is consistent with our incompatibility result. Moreover D3PC does not satisfy early decision provided $f \geq 1$.

5 Acknowledgments

We thank the anonymous reviewers for their suggestions that significantly helped us to improve the presentation of the paper and specifically the algorithms.

References

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

[2] Charron-Bost B. and Schiper A., Uniform Consensus Harder than Consensus, *EPFL Technical Report DSC/2000/028* (EPFL, 2000).

[3] Dutta P., Guerraoui R., Pochon B., Fast Non-Blocking Atomic Commit: An Inherent Trade-off, EPFL Technical Report, ID IC/2004/29, School of Computer and Communication Sciences, EPFL, 2003. Available at: `http://ic2.epfl.ch/publications/documents/IC_TECH_REPORT_200429.pdf`

[4] Keidar I. and Rajsbaum S., On the Cost of Fault-Tolerant Consensus when There are No Faults: a Tutorial, *MIT Technical Report MIT-LCS-TR-821* (MIT, 2001). Also Information Processing Letters (IPL), 85(1): 47–52, 2003.

[5] Lynch N., *Distributed Algorithms* (Morgan Kaufmann, San Francisco, 1996).

[6] Skeen D., Nonblocking commit protocols, *ACM SIGMOD International Symposium on Management of Data*, pages 133–142, 1981.