Improving Fast Paxos: being optimistic with no overhead *

Bernadette Charron-Bost
Ecole polytechnique
91128 Palaiseau, France

André Schiper †
EPFL
1015 Lausanne, Switzerland

Abstract

The paper addresses the cost of consensus algorithms. It has been shown that in the best case, consensus can be solved in two communication steps with $f < n/2$, and in one communication step with $f < n/3$ (f is the maximum number of faulty processes). This leads to a dilemma when choosing a consensus algorithm: greater efficiency or higher resiliency degree. Recently Lamport has proposed a solution called Fast Paxos*, for partly escaping from this dilemma. The idea is to combine two types of rounds in a single consensus algorithm: fast rounds and rounds of the ordinary* Paxos *algorithm. In the best case,* Fast Paxos *solves consensus in one fast round, that is it requires only one communication step. Unfortunately, the combination induces some time overhead, and so* Fast Paxos *becomes more expensive than ordinary* Paxos *when fast rounds do not succeed. In this paper we go one step further: we show that it is possible to tentatively execute a fast round before a classical round without any time overhead if the fast round does not succeed.*

1 Introduction

Consensus is one of the most fundamental problems in fault tolerant distributed computing, a problem related to state machine replication [19]. This importance explains why consensus has attracted so much attention. Solving consensus goes back to the early eighties with the FLP impossibility result, stating that the problem is not solvable by a deterministic algorithm in an asynchronous system if one single process may crash [8]. Since this result, many progresses have been accomplished. In the context of system models, the major results are the definition of the partially synchronous system in which consensus is solvable [6, 7], and the definition of failure detectors as an augmentation of the asynchronous model that makes consensus solvable [4]. In the context of algorithms, two major contributions were the DLS algorithm in [7] and the *Paxos* algorithm [10, 11], which both have the property that the agreement property of consensus is never violated even if messages are lost and the system is asynchronous. The latter feature (no agreement violation despite asynchronism) holds for the Chandra-Toueg (*CT*) consensus algorithm based on the failure detector $\diamond\mathcal{S}$ [4]. As shown in [5], the *Paxos* and the *CT* algorithms have strong similarities (both are based on the same *last voting* scheme), but also have significant differences: (1) *Paxos* tolerates link failures while *CT* requires reliable links, (2) *CT* requires a majority of correct processes not to block, but *Paxos* only requires that at some point in the computation, a majority of processes behave correctly, and (3) *CT* is based on the static rotating coordinator paradigm, while *Paxos* allows the leader to be determined dynamically.

Once solving consensus became well understood, the cost of consensus algorithms started to become a hot topic. One of the cost criteria that received a lot of attention is the number of communication steps (also called *time complexity*) to reach a decision in the best case, which corresponds to *nice* runs of the algorithm. *Paxos* and *CT* have a time complexity of 3 in the best case. This value can be reduced to 2, as shown initially by the early consensus algorithm in [18]. Later Brasileiro *et al.* [3] and Pedone *et al.* [15] [1] have shown that this value can be reduced to 1 under two conditions: (1) 2/3 correct processes and (2) all consensus initial values identical. This leads to the following dilemma when choosing a consensus algorithm: is it better to choose (1) an algorithm with a resiliency degree f less than $n/2$ and two communication steps, or (2) an algorithm with $f < n/3$ and one communication steps if all initial values are identical. Note that identical initial values are typically obtained

† Research funded by the Swiss National Science Foundation under grant number 200021-111701.

[1] In the context of atomic broadcast, which adds one communication step wrt. consensus.

when, using consensus to solve atomic broadcast, messages that are to be broadcast are spontaneously ordered (e.g., on a LAN). So it is a quite realistic assumption in some contexts such as atomic broadcast.

Recently Lamport has proposed a solution for partly escaping from this dilemma. The algorithm, called *Fast Paxos* [13], combines rounds of two algorithms: *classic* rounds and *fast* rounds. Classic rounds are similar to rounds in the Paxos algorithm, while fast rounds allow processes to make a decision in one communication step if all initial values are equal. If a fast round does not succeed, the algorithm switches to classical rounds. The algorithm depends on various parameters, and we can adjust the latter so that (1) if termination in a classical round requires $f < n/2$, then termination in a fast round requires $f \leq \lfloor n/4 \rfloor$, and (2) if termination in a classical round requires $f < n/3$, then termination in a fast round also requires $f < n/3$. Note that Fast Paxos achieves the time complexity and resilience bounds given in [12]. However, there is no free lunch with *Fast Paxos*: switching from a fast round to a classical round has a cost. So, if fast rounds do not succeed often enough in a sequence of consensus, then it is more efficient to use only classical rounds. In other words, *Paxos* may sometimes be better than *Fast Paxos*, as recognized by Lamport in [13]: "*If collisions are too frequent, then classic* Paxos *might be better than* Fast Paxos." (a "collision" corresponds to the case where the initial values of consensus are not all equal, which prevents fast rounds to be successful).

In this paper we go one step further. We show that contrary to *Fast Paxos*, it is possible to combine rounds of two consensus algorithms without any overhead (and without contradicting [13]). In other words, we show that tentatively executing a fast round before a classical round (in case the fast round does not succeed) is not more costly than only executing a classical round.

The rest of the paper is structured as follows. Section 2 introduces the model that we use to express our consensus algorithms. Section 3 gives the two algorithms that we later combine: one is basically the *Paxos* algorithm, and the other one consists in some derandomization of the Rabin consensus algorithm [16]. Section 4 presents our contribution, namely a consensus algorithm in which the execution of unsuccessful fast rounds does not penalize the time complexity of the overall algorithm. Finally, Section 5 concludes the paper.

2 Model for expressing algorithms and consensus

We express below our algorithms in the new HO model (*HO = Heard Of*) that we have defined in [5]. It is inspired by the asynchronous round model defined by Dwork, Lynch and Stockmeyer [7], extended by Gafni [9], and by the work of Santoro and Widmayer [17]. In the HO model, computation consist of *asynchronous* communication-closed rounds (a message sent but not received in round r is lost). Consider a set Π of processes. At each round, any process first sends a message to all (send phase), then receives a subset of the messages sent (receive phase), and finally does some local computation (transition phase). We denote by $HO(p, r)$ the set of processes that p *hears of* at round r, i.e., the processes (including itself) from which p receives a message at round r. There can be various reasons for not receiving a message: the message may have been lost by the channel (link failure), the sender might not have sent the message (send omission), the receiver might not have received the message (receive omission). The key point is that the model describes just transmission faults at each round without attributing these faults to some components (process, channel).

For any round r, its kernel is defined as the set of processes

$$K(r) = \bigcap_{p \in \Pi} HO(p, r).$$

The kernel $K(\phi)$ of a set ϕ of rounds is defined as

$$K(\phi) = \bigcap_{\forall r \in \phi} K(r).$$

An *HO model* is defined by the predicate — over the collection of sets $(HO(p, r))_{p \in \Pi, r > 0}$ — that it guarantees for all computations. For example, we shall consider the HO model in which at least one round is *uniform*, that is the HO model defined by the predicate:

$$\exists r_0 > 0,\ \forall p, q \in \Pi^2\ :\ HO(p, r_0) = HO(q, r_0).$$

A problem is solvable in an HO model defined by predicate $\mathcal{P}$ if there exists an (round-based) algorithm $\mathcal{A}$, such that all runs of $\mathcal{A}$ satisfying $\mathcal{P}$ meet the problem specification. In this paper, we focus on the *Consensus* problem, specified in our approach by the following properties:

- *Integrity:* Any decision value is the initial value of some process.
- *Agreement:* No two processes decide differently.
- *Termination:* All processes eventually decide.

Since there is no notion of faulty process in an HO model, a process is never exempted from making a decision (see Termination). Such a strong liveness requirement may seem unreasonable in two basic respects. First, it may make Consensus needlessly unsolvable in the sense that the resulting Consensus specification might not be solvable in the HO counterpart of a system in which the classical Consensus problem is solvable (termination requirement holds only for correct processes). In [5] we show that this objection does

not hold for all the classical types of systems where Consensus is solvable. The second question is the applicability of algorithms in which *all* processes decide, for systems with real crash failures. The fundamental point here is that a process that has crashed can take no step, and so is no more heard by any process. Consequently, what actually happens on this process has no impact on the rest of the computation. This is why there is no problem to implement an HO algorithm solving the Consensus specification given above, in a system with possible crash failures: the capability of making a decision provided by the HO algorithm is just not implemented by processes that have crashed.

Besides, the HO approach has many advantages. First, it leads to very concise and simple algorithms. Second, the high abstraction level provided by HO models allows us to interpret predicates on the HO's in multiple ways, including link failures: two different types of system may have the same HO counterpart. Third, predicates allow us to identify synthetic conditions under which consensus algorithms are correct. In particular, [5] rigorously establishes some weak conditions that are sufficient to ensure termination of the *Paxos* algorithm. More generally, correctness proofs in HO models are much more direct and elegant since they are no more smothered by the analysis of the causes of transmission faults.

Round *vs.* phase: In several papers (e.g., [10, 4]) consensus algorithms are structured into *rounds*, where a round consists of several *phases*. This terminology conflicts with the notion of rounds in HO models. We swap the words *round* and *phase* to use classical terminology [14]: in the paper, a consensus algorithm is structured into *phases*, where each phase consists of one or more consecutive *rounds*. With this terminology, time complexity simply corresponds to the number of rounds.

Fast decision from some initial configuration *vs.* global fast decision: In the paper we are interested in the *fast decision* of consensus algorithms. Given some initial configuration C,[2] *fast decision from* C corresponds to the "best case" for C, that is, the minimum number of rounds required for all processes to decide from C. *Global fast decision* of a consensus algorithm corresponds to the "global best case", that is the minimum number of rounds required for all processes to decide over the set of all the runs of the algorithm.

3 The two consensus algorithms

Now we describe the two consensus algorithms that we want to combine. In the first algorithm fast decision requires two rounds, whereas the second algorithm allows fast decision in just one round.

3.1 Algorithm *Pa*: consensus algorithm à la *Paxos*

The first algorithm (see Algorithm 1), which we denote *Pa*, is a direct derivation of the *Paxos* algorithm [11] for HO models, which includes two optimizations already described in the literature [2], allowing us to reduce the number of communication steps in "nice" runs.

The algorithm is decomposed into phases, where each phase ϕ consists of three rounds, namely rounds $3\phi - 2$, $3\phi - 1$, and 3ϕ. Each round r starts with the *send* part denoted by S^r (see line 7). Each process p then receives messages from every process in $HO(p, r)$. Finally, processes execute the *state transition* part denoted by T^r (see line 10). Note that the conditions at lines 11, 20 and 27 should not be misinterpreted. *These are not conditions that define when the state transition part T^r starts: the start of the T^r part is defined by the predicates over the HOs.* If the conditions at lines 11, 20 and 27 are false in some round r for process p, then p skips the corresponding T^r part; *process p is not blocked*!

The notation $coord_p(\phi)$ in Algorithm 1 denotes the process that p considers to be the coordinator in phase ϕ. As in *Paxos*, the procedure for selecting coordinators is outside of the algorithm. As in *Paxos*, we can have multiple coordinators in the same phase, i.e., for two processes p, q and phase ϕ, we can have $coord_p(\phi) \neq coord_q(\phi)$. Note that if two coordinators coexist in phase ϕ, because of line 9, the condition of line 11 can be true for at most one coordinator, i.e., at most one coordinator can send a proposal in phase ϕ at line 18.

Two optimizations, allowing fast decision in two rounds, are included in *Pa*. The first optimization consists in modifying S^r so that each process sends a $\langle ack \rangle$ message at round 3ϕ to all processes, rather than only to its coordinator (see line 25). The second optimization consists in skipping the first round of every phase whenever the round is not needed, that is whenever the coordinator is the same as in the previous phase. To keep the algorithm simple, only the first optimization is implemented here.

Table 1 gives the conditions under which the algorithm is correct: safety is always guaranteed and liveness requires the existence of some phase ϕ_0 in which (1) all the HO's contain more than $n/2$ elements, (2) all processes agree on the same coordinator denoted $coord(\phi_0)$, and (3) all processes hear of $coord(\phi_0)$, i.e., $coord(\phi_0) \in K(\phi_0)$.

Fast decision: No decision is possible in less than 2 rounds. Moreover, fast decision does not depend on some

[2] Recall that an initial configuration is a collection of initial values, one per process.

Algorithm 1 Algorithm *Pa*: the consensus algorithm *à la Paxos*.

1: **Initialization:**
2: $x_p \in V$, initially v_p $\{v_p$ *is the initial value of* $p\}$
3: $vote_p \in V \cup \{?\}$, initially v_p
4: $voteToSend_p$ a Boolean,
initially `true` **if** $\forall q\colon Coord(q,1) = p$ **else** `false`
5: $ts_p \in \mathbb{N}$, initially 0

6: **Round** $r = 3\phi - 2$ **:**
7: S^r :
8: **if** $\phi > 1$ **then**
9: send $\langle x_p\ ,\ ts_p \rangle$ to $coord_p(\phi)$

10: T^r :
11: **if** $p = coord_p(\phi)$ **and** $(\phi > 1)$ **and** $(\#\langle x\ ,\ ts\rangle$ received $> n/2)$ **then**
12: let $\overline{\theta}$ be the largest θ from $\langle -,\ \theta\rangle$ received
13: $vote_p :=$ one $\overline{x}$ such that $\langle \overline{x}\ ,\ \overline{\theta}\rangle$ is received
14: $voteToSend_p := \texttt{true}$

15: **Round** $r = 3\phi - 1$ **:**
16: S^r :
17: **if** $p = coord_p(\phi)$ **and** $voteToSend_p$ **then**
18: send $\langle vote_p\rangle$ to all processes

19: T^r :
20: **if** received $\langle v\rangle$ from $coord_p(\phi)$ **then**
21: $x_p := v$; $ts_p := \phi$

22: **Round** $r = 3\phi$ **:**
23: S^r :
24: **if** $ts_p = \phi$ **then**
25: send $\langle ack,\ x_p\rangle$ to all processes

26: T^r :
27: **if** $\exists v$ such that $\#\langle ack,\ v\rangle$ received $> n/2$ **then**
28: DECIDE(v)
29: $voteToSend_p := \texttt{false}$

particular initial configurations: it depends on the cardinality of the sets HO and the coordinators in the first phase (see Table 3).

3.2 Algorithm *Ra(R)*: consensus algorithm *à la* Rabin

The second algorithm (see Algorithm 2), which we denote *Ra*, can be viewed as a deterministic version of the Rabin consensus algorithm [16, 15]. A similar scheme is used in [3] and in the fast rounds of *Fast Paxos* [13].

Each phase of the *Ra* algorithm, parameterized with a constant R, consists of one single round. The interesting feature of the algorithm is that making a decision is possible in one round if all the initial values are identical (the practical relevance of this case is discussed in Section 1). Table 2 gives precise conditions under which the algorithm is correct. Safety requires $n > 3R$: the latter condition ensures that if some process decides v at line 13 of round r, then in any round $r' \geq r$, only v can be assigned to any x_p. To ensure liveness, we proceed in two steps. First, we require that there exists some round ϕ_0[3] such that in ϕ_0 all processes hear of the same set HO and $|HO| \geq n - R$. This makes the system "space uniform" in the sense that at the end of phase ϕ_0, all processes have the same value for x_p. Secondly, if there exists a round ϕ_p greater than ϕ_0 and such that $|HO(p,\phi_p)| \geq n - R$, then p makes a decision at the end of round ϕ_p. This discussion is summarized in Table 2. Note that we obtain the weakest correctness condition (safety and liveness) for algorithm *Ra(R)* for the value $R = \lfloor \frac{n-1}{3} \rfloor$.

[3] Since a phase consists here of one single round, *phase* and *round* are equivalent.

COND. FOR SAFETY	CONDITION FOR LIVENESS
none	$\exists \phi_0 > 0\,,\ \begin{cases} \forall p \in \Pi : \\ \quad \lvert HO(p,\phi_0)\rvert > n/2 \\ \forall p,q \in \Pi^2\ : \\ \quad coord_p(\phi_0) = coord_q(\phi_0) \\ \forall p \in \Pi : \\ \quad coord_p(\phi_0) \in K(\phi_0) \end{cases}$

Table 1. Conditions for the correctness of algorithm *Pa*.

Algorithm 2 The *Ra(R)* algorithm: the consensus algorithm *à la Rabin*.

1: **Initialization:**
2: $x_p := v_p$ $\{\ v_p$ *is the initial value of* $p\ \}$

3: **Round** r**:**
4: S^r :
5: send $\langle\ x_p\ \rangle$ to all processes

6: T^r :
7: **if** $|HO(p,r)| \geq n - R$ **then**
8: **if** the values received, except at most R, are equal to $\overline{x}$ **then**
9: $x_p := \overline{x}$
10: **else**
11: $x_p :=$ smallest x received
12: **if** $n - R$ values received are equal to $\overline{x}$ **then**
13: DECIDE$(\overline{x})$

Fast decision: If all initial values are identical, a decision is possible in one round. Besides, a decision is possible in two rounds from other initial configurations (see Table 3). The above discussion about Ra's correctness shows that fast decision requires that the sets HO contain at least $n - R$ elements with $n > 3R$, which corresponds to the weakest requirement $|HO| > 2n/3$ for parameter $R = \lfloor \frac{n-1}{3} \rfloor$. Note this necessary condition for fast decision is also sufficient from initial configurations with identical values, i.e., for global fast decision.

COND. FOR SAFETY	CONDITION FOR LIVENESS
$n > 3R$	$\exists \phi_0 > 0, \exists HO, \lvert HO \rvert \geq n-R :$ $\begin{cases} \forall p : HO(p, \phi_0) = HO \\ \wedge \\ \forall p, \exists \phi_p > \phi_0 : \lvert HO(p, \phi_p) \rvert \geq n-R \end{cases}$

Table 2. Conditions for the *Ra(R)* algorithm.

3.3 Summary

Table 3 summarizes the features of the two algorithms *Pa* and *Ra(R)*. For fast decision, the best algorithm is *Ra*. However the algorithm has a stronger requirement on the cardinalities of the HO's.

ALG.	COND. ON $\lvert HO \rvert$ FOR FAST DECISION	GLOBAL FAST DECISION	FAST DECISION FROM OTHER INIT CONFIG
Pa	$> n/2$	2	2
Ra ($\lfloor \frac{n-1}{3} \rfloor$)	$> 2n/3$	1	2

Table 3. Fast decision of the two consensus algorithms.

4 Combining the *Ra(R)* and *Pa* consensus algorithms

When combining the two consensus algorithms Pa and $Ra(R)$, our aim is to obtain an algorithm that inherits the best features of Pa and of $Ra(R)$. The three criteria that we consider appear in Table 3: (1) condition on $\lvert HO \rvert$, (2) global fast decision, (3) fast decision from other initial configurations.

A simple way to combine two consensus algorithms A and A' is just to juxtapose A and A', that is to execute A and A' in parallel (with the variables of A being distinct from the variables of A'). Unfortunately the solution does not work: since the decision value is not entirely determined by the set of initial values, one process may decide v by algorithm A, while another process decides $v' \neq v$ by algorithm A'. So the two algorithms need to be "semantically merged". To guarantee agreement, the merging procedure must ensure that a configuration C is v-valent[4] for one of the algorithms iff it is also v-valent for the other algorithm.

We now describe another combination of *Ra(R)* and *Pa* that has the *best features of each* of the two algorithms *without any overhead*. We shall proceed in two steps: we propose a first combination, and then improve it to get the final algorithm.

4.1 *Hybrid-1(R)*: first combination of *Ra(R)* and *Pa*

The first combination of *Ra(R)* and *Pa* is called *Hybrid-1(R)*. When the initial values are identical, similarly to *Ra(R)*, we want *Hybrid-1(R)* to be able to decide at the end of the first round. Similarly to *Pa*, we want *Hybrid-1(R)* to be able to decide with only $\lvert HO \rvert > n/2$. The solution is given by Algorithm 3:

- Round $3\phi - 2$ of *Hybrid-1(R)* is obtained by merging round ϕ of *Ra(R)* and round $3\phi - 2$ of *Pa*.
- Rounds $3\phi - 1$ and 3ϕ of *Hybrid-1(R)* are identical to the corresponding rounds of *Pa*.

For agreement, we must ensure that if some process decides $\overline{x}$ as in *Ra(R)*, then a coordinator of *Pa* selects $\overline{x}$ as the vote to send to the participants. The key idea is the following. A process can decide $\overline{x}$ according to *Ra(R)* if $n-R$ values received are equal to $\overline{x}$ (see lines 11, 12). For another process p to detect that $\overline{x}$ *might have been decided*, p needs to receive at least $2R$ messages: indeed, in this case any majority among these $2R$ messages consists of messages equal to $\overline{x}$. This is expressed in line 13 by the condition "$> max(-, 2R)$", and in lines 14, 15. The condition "$> max(n/2, -)$" in line 13 is the condition of *Pa* for agreement: a coordinator can select the value with the largest time-stamp only if it has received more than $n/2$ messages.

However, we must avoid that (i) the selection rule of *Ra* based on the reception of $2R$ messages and (ii) the selection rule of *Pa* based on the time-stamps conflict. A conflict exists if the following occurs at the beginning of some phase ϕ:

1. $\lceil \frac{n+1}{2} \rceil - R$ values received are identical to $\overline{x}$ (i.e., $\overline{x}$ should be chosen according to *Ra(R)*, lines 14, 15).
2. The largest time-stamp is equal to $\overline{\theta} > 0$ and for $\lceil \frac{n+1}{2} \rceil$ processes q, we have $x_q = \overline{\overline{x}}$ and $ts_q = \overline{\theta}$, with $\overline{\overline{x}} \neq \overline{x}$ (i.e., $\overline{\overline{x}}$ should be chosen according to *Pa*, lines 17, 18).

If 1 and 2 both hold, the coordinator cannot choose both $\overline{x}$ and $\overline{\overline{x}}$! To prevent this from occurring, we must have $\lceil \frac{n+1}{2} \rceil - R + \lceil \frac{n+1}{2} \rceil > n$ (there are only n processes). For n odd this leads to $R = 0$; for n even this leads to $R \leq 1$.

[4] A configuration C of an algorithm is v-valent if, from C, the only possible decision value is v.

To avoid these constraints on R, observe that that the conflict cannot occur in the first phase $\phi = 1$ (in the first phase the coordinator does not need to select a value with the largest time-stamp). So if lines 10 to 12 are only executed in phase $\phi = 1$, then no conflict between 1 and 2 is possible. This is expressed by condition $\Phi = 1$ at line 10 and by $\langle \overline{x}\,,\, 0\rangle$ at line 14.

The correctness condition for algorithms *Hybrid-1(R)* is given in Table 4 (compare with Tables 1 and 2). We obtain the weakest constraint on HO when $n/2 \geq 2R$, i.e., for the value $R \leq n/4$ (see the lower bounds in [12]).

Algorithm 3 The *Hybrid-1 (R)* algorithm.

```
 1: Initialization:
 2:    x_p ∈ V, initially v_p   {v_p is the initial value of p}
 3:    vote_p ∈ V ∪ {?}, initially ?
 4:    voteToSend_p a Boolean, initially false
 5:    ts_p ∈ ℕ, initially 0

 6: Round r = 3φ − 2 :
 7:    S^r :
 8:       send ⟨x_p , ts_p, coord_p(φ)⟩ to all processes

 9:    T^r :
10:       if (φ = 1) and #⟨−, −, −⟩ received ≥ n − R then
11:          if n − R messages received are equal to ⟨x̄, −, −⟩ then
12:             DECIDE(x̄)
13:       if p = coord_p(φ) and #⟨−, −, p⟩ received > max(n/2, 2R) then
14:          if the messages received, except at most R, are equal to ⟨x̄ , 0, p⟩ then
15:             vote_p := x̄
16:          else
17:             let θ̄ be the largest θ from ⟨−, θ, p⟩ received
18:             vote_p := x̄ such that ⟨x̄, θ̄, p⟩ is received
19:          voteToSend_p := true

20: Round r = 3φ − 1 :
21:    S^r :
22:       if p = coord_p(φ) and voteToSend_p then
23:          send ⟨vote_p⟩ to all processes

24:    T^r :
25:       if received ⟨v⟩ from coord_p(φ) then
26:          x_p := v ; ts_p := φ

27: Round r = 3φ :
28:    S^r :
29:       if ts_p = φ then
30:          send ⟨ack, x_p⟩ to all processes

31:    T^r :
32:       if ∃v s.t. #⟨ack, v⟩ received > n/2 then
33:          DECIDE(v)
34:       voteToSend_p := false
```

Fast decision of *Hybrid-1(R)*: If all initial values are identical, a decision is possible in one round. It requires that the set HO contains at least $n - R$ elements (line 10). With the condition $|HO| > max(n/2,\, 2R)$, the weakest requirement on the size of the set HO is when $n/2 = 2R$ (i.e., $R = \lfloor n/4 \rfloor$), which leads to $|HO| \geq n - n/4 = 3n/4$ (see Table 5). Besides, a decision is possible in three rounds from other initial configurations; it only requires $|HO| > n/2$.

COND. FOR SAFETY	CONDITION FOR LIVENESS
$n > 3R$	$\exists \phi_0 > 0\,,\ \begin{cases} \forall p \in \Pi : \\ \quad \lvert HO(p, \phi_0)\rvert > max(n/2,\, 2R) \\ \forall p, q \in \Pi^2 : \\ \quad coord_p(\phi_0) = coord_q(\phi_0) \\ \forall p \in \Pi : \\ \quad coord_p(\phi_0) \in K(\phi_0) \end{cases}$

Table 4. Conditions for the *Hybrid-1 (R)* algorithm.

Algorithm *Hybrid-1(R)* is worse than *Ra* and *Pa* in terms of fast decision "from other initial configurations" (see Table 5). This is because the round 1 of *Hybrid-1(R)* serves only for making a fast decision when all the initial values are identical. If this is not the case, then round 1 is useless. In Section 4.3 we modify round 1 of *Hybrid-1(R)* to make it useful even if the initial values are not identical.

ALGORITHM	COND. ON HO FOR CORRECTNESS	GLOBAL FAST DECISION	FAST DECISION FROM OTHER INIT CONFIG
Ra ($\lfloor \frac{n-1}{3} \rfloor$)	$> 2n/3$	1	2
Pa	$> n/2$	2	2
Hybrid-1 ($\lfloor n/4 \rfloor$)	$\geq$ **3n/4**	**1**	**3**
Hybrid-1 ($\lfloor n/4 \rfloor$)	$>$ **n/2**	**3**	**3**

Table 5. Fast decision of algorithm *Hybrid-1 (R)* (fast decision depends on the size of HO).

4.2 *Hybrid-1(R)*: proof of correctness

We now proof the correctness of algorithm *Hybrid-1(R)*. Validity is obvious. We show that agreement and termination hold.

Proposition 4.1 *If the condition for safety of Table 4 holds, algorithm* Hybrid-1(R) *satisfies agreement for any $R < n$.*

Proof: We have two cases to consider: (1) the first decision at line 12 in phase $\phi = 1$, and (2) the first decision

decision at line 33 in any phase.

Case (1): If two processes decide at line 12, the condition $n > 3R$ ensures that they both decide the same value. Consider now the case where some process decides at line 12, and the other processes at line 33. Let process p decide $\overline{x}$ at line 12 in phase $\phi = 1$. So, for $n - R$ processes q we have $x_q = \overline{x}$. A decision at line 33 is only possible after some coordinator has executed lines 13 to 19. Consider the smallest phase in which some coordinator c executes these lines. Since $|HO(c, r)| > max(n/2,\, 2R)$ (line 13), we have $|HO(c, r)| > 2R$. This ensures that the condition of line 14 evaluates to `true` for the value $\overline{x}$, i.e., c sets $vote_c$ to the value $\overline{x}$. From here on, it is easy to see that any process q that updates ts_q at line 26, assigns $\overline{x}$ to x_q. So only $\overline{x}$ can be decided.

Case(2): Let ϕ_0 be the smallest phase in which some process p decides $\overline{x}$ at line 33. So, for $n/2$ processes q we have $x_q = \overline{x}$, $ts_q = \phi_0$ and for the other processes q' we have $ts_{q'} < \phi_0$. From here on a coordinator c in some phase larger than ϕ_0 can only assign $\overline{x}$ to $vote_c$. So only $\overline{x}$ can be decided. □

Proposition 4.2 *If the condition for liveness of Table 4 holds, algorithm Hybrid-1(R) satisfies termination.*

Proof: Let ϕ_0 be a phase that satisfies the conditions for liveness of Table 4. Let $c = coord(\phi_0)$ be the unique coordinator of phase ϕ_0. Since $|HO(c, \phi_0)| > max(n/2,\, 2R)$ (Table 4), for process c the condition at line 13 evaluates to `true`, $voteToSend_c$ is set to `true` (line 19), and $vote_c$ is sent to all (line 23). Since $c \in K(\phi_0)$ (Table 4) every process receives the vote of c at line 25, and sends $\langle ack,\, x_p \rangle$ to all processes (line 30). Since for all p, $|HO(p, \phi_0)| > n/2$ (Table 4), every process receives more than $n/2$ of these messages (line 32) and decides at line 33. □

4.3 *Hybrid-2(R)*: an improved combination of *Ra(R)* and *Pa*

The first round of *Hybrid-1(R)* is useless when initial values are not identical. In our second combination of *Pa* and *Ra(R)* that we call *Hybrid-2(R)*, the first round is used for two purposes: (1) to decide at the first round if possible (as in *Hybrid-1(R)* and *Hybrid-1(R)*), and (2) to have a coordinator trying to impose its initial value. However (1) and (2) must be consistent. This means that some value v proposed by the coordinator in round 1 can be adopted by a participant if and only if, according to *Ra(R)*, the initial configuration is not v'-valent with $v' \neq v$.

Algorithm 4 shows the first phase of *Hybrid-2(R)*. The other phases of *Hybrid-2(R)* are identical to *Hybrid-1(R)*. The key idea of *Hybrid-2(R)* is in lines 12 to 19. Contrary to *Hybrid-1(R)*, where the corresponding lines are executed only by the coordinator, these lines are executed by *all* processes. If the condition of line 13 holds, then the initial configuration may be $\overline{x}$-valent with respect to *Ra(R)*. In this case, the value received from the coordinator is ignored, *unless $\overline{x}$ is also received from the coordinator* (line 15). If the value received from the coordinator is ignored, then the first phase is useful only for fast decision, similarly to *Hybrid-1(R)*. Since our aim is to make the first round always useful, the value received from the coordinator at phase 1 must never be ignored, which requires $R = 0$.

Algorithm 4 First phase of *Hybrid-2(R)*. The other phases are identical to *Hybrid-1(R)*.

```
 1: Initialization:
 2:   x_p ∈ V, initially v_p   {v_p is the initial value of p}
 3:   vote_p ∈ V ∪ {?}, initially ?
 4:   ts_p ∈ ℕ, initially 0

 5: Round r = 1 :
 6:   S^r :
 7:     send ⟨x_p , ts_p , coord_p(φ)⟩ to all processes

 8:   T^r :
 9:     if |HO(p, r)| ≥ n − R then
10:       if n − R messages received are equal to some ⟨x̄, −, −⟩ then
11:         DECIDE(x̄)
12:     if |HO(p, r)| > max(n/2, 2R) then
13:       if the messages received, except at most R, are equal to ⟨x̄, −, −⟩ then
14:         x_p := x̄
15:         if received ⟨x̄, −, −⟩ from coord_p(φ) and
                          #⟨−, −, coord_p(φ)⟩ received > n/2 then
16:           ts_p := φ
17:       else
18:         if received ⟨x̄, −, −⟩ from coord_p(φ) and
                          #⟨−, −, coord_p(φ)⟩ received > n/2 then
19:           x_p := x̄;  ts_p := φ

20: Round r = 2 :
21:   S^r :
22:     if ts_p = φ then
23:       send ⟨ack , x_p⟩ to all processes

24:   T^r :
25:     if ∃v s.t. #⟨ack , v⟩ received > n/2 then
26:       DECIDE(v)
```

The correctness conditions of *Hybrid-2(R)* are the same as those of *Hybrid-1(R)* (Table 4). Note that correctness does not require $R = 0$. Remember also that the condition at lines 9, 12 and 25 are not conditions that define when the state transition part T^r starts: the start of the T^r part is defined by the predicates over the HOs. Specifically, the condition at line 9 in *Hybrid-2(0)* does not mean that the algorithm blocks if one process has crashed.

Fast decision: Similarly to *Hybrid-1(R)*, *Hybrid-2(R)* allows fast decision in one round if initial values are all identical. If this is not the case, a decision in two rounds is possible if $R = 0$. Table 6 compares the characteristics of algorithm *Hybrid-2(0)* with *Ra($\lfloor \frac{n-1}{3} \rfloor$)* and *Pa*. The table shows that *Hybrid-2(0)*, similarly to *Ra($\lfloor \frac{n-1}{3} \rfloor$)*, can achieve fast decision in one round. This requires $|HO| = n$, while *Ra* only requires $|HO| > 2n/3$. However, being optimistic

(i.e., assuming HO equal to n and all initial values identical) *does not lead to any overhead if these conditions do not hold* (compare line 2 (alg. *Pa*) and line 4 (alg. *Hybrid-2(0)*) of Table 6).

ALGORITHM	COND. ON HO FOR CORRECTNESS	GLOBAL FAST DECISION	FAST DECISION FROM OTHER INIT CONFIG
Ra ($\lfloor \frac{n-1}{3} \rfloor$)	$> 2n/3$	1	2
Pa	$> n/2$	2	2
Hybrid-2 (0)	$=$ **n**	**1**	**2**
Hybrid-2 (0)	$>$ **n/2**	**2**	**2**

Table 6. Fast decision of algorithm *Hybrid-2(0)* **(fast decision depends on the size of** HO**).**

Discussion: $R = 0$ in *Hybrid-2(0)* does not mean that *Hybrid-2(0)* is not fault tolerant. It only means that fast decision in one round is no more possible if one process has crashed. In this case, the fastest decision requires two rounds. However, by excluding crashed processes using a group membership service, *fast decision in one round is again possible*. We explain now the idea.

The role of a group membership service is to add and remove processes from a group of processes. Consider a group G of size n. As long as no process in G has crashed, we can have $|HO| = |G| = n$, i.e., fast decision in one round is possible with *Hybrid-2(0)*. As soon as one process p in G crashes, then fast decision requires two rounds, even if all the initial values are identical. By excluding p from G, we have a new membership for G of size $n - 1$. The case $|HO| = |G| = n - 1$ allows again a fast decision in one round. Thus, *Hybrid-2(0)* shows the benefit of excluding crashed processes from a group.

4.4 *Hybrid-2(R)*: Proof of correctness

Validity of *Hybrid-2(R)* is obvious. If no process decides in phase $\phi_0 = 1$, then agreement holds with exactly the same arguments as for *Hybrid-1(R)*. The case $\phi_0 = 1$ is discussed below. If the liveness conditions of Table 5 hold in some phase $\phi_0 > 1$, then termination is guaranteed with exactly the same arguments as for *Hybrid-1(R)*. The case $\phi_0 = 1$ is discussed below.

Proposition 4.3 *If the condition for safety of Table 4 holds, and if the first process that decides does so in phase* $\phi_0 = 1$*, then the algorithm* Hybrid-2(R) *satisfies agreement.*

Proof: We have two cases to consider: (1) decision at line 11, and (2) decision at line 26.

Case (1): Let process p decide $\overline{x}$ at line 11. So, for $n-R$ processes q we have $x_q = \overline{x}$. We have three cases to consider: (i) process q also decides at line 11 of Algorithm 4 (i.e., in phase $\phi = 1$), (ii) process q decides at line 26 of Algorithm 4 (i.e., in phase $\phi = 1$), or (iii) process q decides in some phase $\phi > 1$.
Case (i): Since $n > 3R$, q necessarily decides $\overline{x}$.
Case (ii): If q decides at line 26, then at least one process must have executed line 16 or line 19, i.e., the condition of line 12 has evaluated to *true* for at least one process. Consider a process p' such that the condition of line 12 evaluates to *true*. Since (a) $|HO(p', r)| > 2R$ and (b) for $n - R$ processes q we have $x_q = \overline{x}$, consequently for process p' the condition of line 13 necessarily evaluates to true for $\overline{x}$, i.e, p' sets $x_{p'}$ to $\overline{x}$ (line 14). So $\overline{x}$ is the only possible decision value at line 26.
Case (iii): By the argument of case (ii), if some process p' updates $x_{p'}$ in phase $\phi = 1$, it sets $x_{p'}$ to $\overline{x}$. So the number of processes p' with $x_{p'} = \overline{x}$ does not decrease. So if some process p decide $\overline{x}$ at line 11, then at the end of phase ϕ for $n - R$ processes q we have $x_q = \overline{x}$. Agreement follows by repeating the arguments of case (1) in Proposition 4.1.

Case (2): Similar to the proof of case (2) in Lemma 4.1. □

Proposition 4.4 *If the liveness conditions of Table 4 hold in phase* $\phi_0 = 1$*, then the algorithm* Hybrid-2(0) *terminates.*

Proof: Let p be the unique coordinator of phase $\phi_0 = 1$. Since $|HO(p, 1)| > n/2$ (see Table 4) and $R = 0$, we have $|HO(p, 1)| > 2R$. So the condition of line 12 evaluates to `true`. Since $coord(\phi_0) \in K(\phi_0)$, every process q receives the message from $coord(\phi_0)$ at line 15 or 18, every process q assign 1 to ts_q (line 16 or 19), and every process q sends $\langle ack, x_q \rangle$ to all processes (line 23). For all p, $|HO(p, 1)| > n/2$ (see Table 4), so every process receives more than $n/2$ messages $\langle ack, x_q \rangle$ (line 25) and decides at line 26. □

5 Conclusion

The algorithm *Hybrid-2(0)*, obtained by combining two consensus algorithms has the nice feature of being optimistic without incurring any overhead in terms of time complexity. The optimistic assumptions are: (1) n correct processes, and (2) all initial values identical. If these two assumptions hold, *Hybrid-2(0)* solves consensus in one round. If none of these assumptions hold, *Hybrid-2(0)* requires 2 rounds, similarly to *Pa*. So optimism does not lead to an overhead, a rather surprising result.

As mentioned in Section 1, the idea of combining two consensus algorithm appears in *Fast Paxos* (an idea that previously appeared in [1] in a different context). However optimism has a cost in *Fast Paxos*, whenever the optimistic assumptions do not hold. *Hybrid-2(0)* has shown that it is possible to be optimistic with no overhead.

Another interesting result of *Hybrid-2(0)* is that it shows an algorithmic justification for excluding crashed processes from a group: excluding crashed processes allows again a decision in one round, while as long as crashed processes are kept in the group, the fastest decision requires two rounds.

Acknowledgements. We would like to thank Leslie Lamport for discussions related to Paxos and Fast Paxos, and Tastuhiro Tsuchiya for correcting errors of a previous version of the paper.

References

[1] M. Aguilera and S. Toueg. Failure detection and randomization: A hybrid approach to solve consensus. *SIAM J. Comput.*, 28(3):890–903, 1998.

[2] R. Boichat, P. Dutta, S. Frolund, and R. Guerraoui. Reconstructing Paxos. *ACM SIGACT News*, 34(1):47–67, 2003.

[3] F. Brasileiro, F. Greve, A. Mostéfaoui, and M. Raynal. Consensus in one communication step. In *6th International Conference Parallel Computing Technologies (PaCT)*, pages 42–50. Springer Verlag, LNCS 2127, 2001.

[4] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of ACM*, 43(2):225–267, 1996.

[5] B. Charron-Bost and A. Schiper. The Heard-Of model: Unifying all benign failures. Technical report, EPFL, July 2006.

[6] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchrony needed for distributed consensus. *Journal of ACM*, 34(1):77–97, January 1987.

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

[8] M. Fischer, N. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of ACM*, 32:374–382, April 1985.

[9] Eli Gafni. Round-by-round fault detectors: Unifying synchrony and asynchrony. In *Proc of the 17th ACM Symp. Principles of Distributed Computing (PODC)*, pages 143–152, Puerto Vallarta, Mexico, June-July 1998.

[10] L. Lamport. The Part-Time Parliament. TR 49, Digital SRC, September 1989.

[11] L. Lamport. The Part-Time Parliament. *ACM Trans. on Computer Systems*, 16(2):133–169, May 1998.

[12] L. Lamport. Lower bounds for asynchronous consensus. Technical Report MSR-TR-2004-71, Microsoft, 2004.

[13] L. Lamport. Fast Paxos. Technical Report MSR-TR-2005-12, Microsoft, 2005.

[14] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[15] F. Pedone, A. Schiper, P. Urban, and D. Cavin. Solving Agreement Problems with Weak Ordering Oracles. In *Proceedings of the 4th European Dependable Computing Conference (EDCC-4)*, LNCS-2485, pages 44–61, Toulouse, France, October 2002. Springer-Verlag.

[16] M. Rabin. Randomized Byzantine Generals. In *Proc. 24th Annual ACM Symposium on Foundations of Computer Science*, pages 403–409, 1983.

[17] N. Santoro and P. Widmayer. Time is not a healer. In *Proceedings of the 6th Symposium on Theor. Aspects of Computer Science*, pages 304–313, Paderborn, Germany, 1989.

[18] A. Schiper. Early consensus in an asynchronous system with a weak failure detector. *Distributed Computing*, 10(3):149–157, April 1997.

[19] F. B. Schneider. Implementing Fault Tolerant Services Using the State Machine Approach: A Tutorial. *Computing Surveys*, 22(4):299–319, December 1990.