CONSENSUS IN ONE COMMUNICATION STEP IS POSSIBLE

FRANCISCO BRASILEIRO, FABIOLA GREVE, ACHOUR MOSTEFAOUI, MICHEL RAYNAL
Consensus in One Communication Step is Possible

Francisco Brasileiro, Fabiola Greve, Achour Mostefaoui, Michel Raynal

Abstract: This paper presents a novel yet very simple consensus protocol that converges in a single communication step in favorable circumstances. Those situations occur when “enough” processes propose the same value. (“Enough” means “at least (n - f)” where f is the maximum number of processes that can crash in a set of n processes.) The protocol requires f < n/3. It is shown that this condition is necessary. Moreover, if all the processes that propose a value propose the same value, the protocol always terminates in one communication step. It is also shown that additional assumptions can help weaken the f < n/3 requirement to f < n/2.

Key-words: Asynchronous Distributed System, Consensus, Crash Failure.

email: {brasile,greve,mostefaoui,raynal}@irisa.fr
Quand est-il possible de faire un consensus en une étape ?

Résumé : Ce rapport présente un nouveau protocole de consensus qui permet aux processus de décider en une étape de communication dans des circonstances favorables. Celles-ci apparaissent lorsque \((n - f)\) processus proposent la même valeur (où \(n\) est le nombre de processus et \(f\) le nombre maximal d’entre eux qui peuvent crasher). The protocole requiert la condition \(f < n/3\). Il est montré que, dans le cas général, cette condition est nécessaire. Il est également montré que cette condition peut être affaiblie à \(f < n/2\) lorsque le système satisfait des hypothèses supplémentaires.

Mots clés : consensus, panne franche, protocole probabiliste, système réparti asynchrone, tolérance aux défaillances.
1 Introduction

The Consensus problem is now recognized as being one of the most important problems to solve when one has to design or to implement reliable applications on top of an unreliable asynchronous distributed system. Informally, the Consensus problem is defined in the following way. Each process proposes a value, and all non-crashed processes have to agree on a common value which has to be one of the proposed values. The most important practical agreement problems (such as Atomic Broadcast, View Synchrony, Weak Atomic Commitment, Atomic Multicast, etc.) can be reduced to Consensus, which can be seen as their “greatest common sub-problem”. Consequently, a distributed module implementing Consensus constitutes a basic building block on top of which solutions to practical agreement problems can be built. This explains why the Consensus problem is a fundamental problem, and justifies the large interest the literature has brought to it.

Solving the Consensus problem in asynchronous distributed systems is far from being a trivial task. In fact, it has been shown by Fischer, Lynch and Paterson [4] that there is no (deterministic) solution to this problem as soon as processes (even only one) may crash. Two major approaches have been proposed to circumvent this impossibility result. One lies in the use of randomized protocols [2]. The other lies in the unreliable failure detector concept, proposed and investigated by Chandra and Toueg [3]. Several failure detector-based consensus protocols have been designed ([6] presents a general approach to solve the consensus problem in asynchronous systems equipped with Chandra-Toueg’s failure detectors). Interestingly, a Hybrid approach combining failure detectors and random number generators has also been investigated [1, 7].

To converge towards a single decided value, a consensus protocol makes the processes exchange proposed values. Each exchange constitutes a communication step. So, an interesting measure of the efficiency of a protocol is the number of communication steps it requires. In the best scenario, the consensus protocols proposed so far require that processes execute at least two communication steps.

This paper presents a novel and surprisingly simple consensus protocol that allows processes to decide in a single communication step when “enough” processes propose the same value. “Enough” means at least \((n - f)\), where \(n\) is the number of processes and \(f\) is the maximum number of them that can be faulty. This protocol requires \(f < n/3\). Although failures do occur, they are rare in practice. This observation shows that the requirement \(f < n/3\) is not really constraining. It is shown that the requirement \(f < n/3\) is actually a necessary requirement for the family of consensus protocols that allow the processes to decide in one communication step, when all the processes that propose a value propose the same value. The paper finally shows that additional assumptions can help weaken the \(f < n/3\) requirement to \(f < n/2\).

2 System Model and Consensus

2.1 Asynchronous System

The system model is patterned after the one described in [3, 4]. It consists of a finite set \(\Pi\) of \(n > 1\) processes, namely, \(\Pi = \{p_1, \ldots, p_n\}\). A process can fail by crashing, i.e., by prematurely halting; a crashed process does not recover. A process behaves correctly (i.e., according to its specification) until it (possibly) crashes. By definition, a correct process is a process that does not crash. A faulty process is a process that is not correct. As indicated in the Introduction, \(f\) denotes the maximum number of processes that may crash.
Processes communicate and synchronize by sending and receiving messages through channels. Every pair of processes is connected through a channel. Channels are assumed to be reliable: they do not create, alter, duplicate or lose messages. There are assumptions neither on the relative speed of processes nor on message transfer delays.

2.2 The Uniform Consensus Problem

In the Consensus problem, every process $p_i$ proposes a value $v_i$ and all correct processes have to decide on some value $v$, in relation to the set of proposed values. More precisely, the Consensus problem is defined by the following four properties [3, 4]:

- **Termination**: Every correct process eventually decides some value.
- **Validity**: If a process decides $v$, then $v$ was proposed by some process.
- **Integrity**: A process decides at most once.
- **Uniform Agreement**: No two processes (correct or not) decide differently.

2.3 Underlying Consensus

As our aim is to provide a consensus protocol that terminates in one communication step in good scenarios, we adopt the following approach. We consider that the asynchronous distributed system is equipped with a black box solving the consensus problem, and we provide a protocol that decides in one communication step in good scenarios and uses the underlying consensus protocol in the other cases. The underlying consensus black box can be any consensus protocol (e.g., [1, 2, 3, 6, 7]). A process $p_i$ locally invokes it by calling $\text{UnderlyingConsensus}(v_i)$ where $v_i$ is its proposed value.

It is important to notice that, when considered alone, the underlying consensus protocol solves in no way the problem we address, namely, to allow processes to decide in one communication step in "good" circumstances.

3 The Protocol

3.1 Underlying Principle

The idea that underlies the design of the protocol is very simple. It comes from the following observation: if all the processes initially propose the same value, then this value is necessarily the decided value, whatever the protocol and the system behavior. Hence, the proposed protocol executes a first communication step during which the processes exchange the values they propose. Then, each process checks whether all the processes have the same initial value (actually, $(n - f)$ identical values are sufficient). If it is the case, then this value is decided. If it is not, the underlying protocol is used.

3.2 The Protocol

The protocol is described in Figure 1. A process $p_i$ starts a Consensus execution by invoking $\text{Consensus}(v_i)$. It terminates it when it executes the statement $\text{return}$ which provides it with the decided value (at line 4, 7 or 9). To prevent a process from blocking forever (i.e., waiting for a value from a process that has already decided), a process that decides, uses a reliable broadcast [3] to
disseminate its decision value. To this end the Consensus function is made of two tasks, namely, $T1$ and $T2$. $T1$ implements the core of the protocol. Line 4 and $T2$ implement the reliable broadcast.

<table>
<thead>
<tr>
<th>Function Consensus($v_i$)</th>
</tr>
</thead>
</table>
| **Task T1:** | \[(1) \text{broadcast } \text{PROPOSED} (v_i);\]  
| | \[(2) \text{wait until a } \text{PROPOSED} \text{ message has been received from } (n-f) \text{ processes};\]  
| | \[(3) \text{if (the same estimate value } v \text{ has been received from } (n-f) \text{ processes} \]  
| | \[(4) \text{then } \text{broadcast } \text{DECISION} (v); \text{return} (v)\]  
| | \[(5) \text{else if (the same value } v \text{ has been received from at least } (n-2f) \text{ processes} \]  
| | \[(6) \text{then } v_i \leftarrow v \text{ endif};\]  
| | \[(7) \text{return(Underlying-Consensus} (v_i))\]  
| | \[(8) \text{endif}\]  
| **Task T2:** | \[(9) \text{upon reception of } \text{DECISION} (v): \text{broadcast } \text{DECISION} (v); \text{return} (v)\]  

| Figure 1: The Consensus Protocol |

### 3.3 One Communication Step Decision

Let us consider the case where all the processes that propose a value (those are the processes that have not initially crashed) propose the same value. The protocol makes the processes that do not crash decide in exactly one communication step.

More generally, let us consider the following scenario where there is a set $S$ of $(n-f)$ processes that propose the same value $v$, and during the first phase of the first round, all processes receive the PROPOSED messages from the processes of $S$. It follows from lines 2-4 that all processes also decide after one communication step.

### 4 Proof

The proof assumes $f < n/3$. The proof of the Validity property (a decided value is a proposed value) and Integrity property (no process decides more than once) are left to the reader.

#### 4.1 Termination

**Theorem 1** If a process $p_i$ is correct, then it eventually decides.

**Proof** As there are at least $(n-f)$ correct processes, let us first note that no correct process can block forever at line 2. Hence, they all execute line 3. According to the results of the test there are two cases:

- A process decides at line 4.
  In that case, this process has previously sent a DECISION message to all other processes. Due to the reliable channel assumption, it follows that if a correct process has not yet decided when it receives this message, it executes line 9 and consequently decides.

- No process decides at line 4.
  In that case, all the processes that have not crashed during the first communication step invoke the underlying consensus protocol. Due to its Termination property, all the correct processes eventually decide.
4.2 Uniform Agreement

**Theorem 2** Let \( f < n/3 \). No two processes decide differently.

**Proof** Let us first notice that a process that decides at line 9, decides a value \( v \) that has been sent by a process at line 4. So, we only consider the decision at line 4 and line 7. The proof considers three cases.

- Let us first consider the case where two processes \( p_i \) and \( p_j \) decide at line 4. This means \( p_i \) received \((n - f)\) PROPOSED messages carrying the same value \( v \). Similarly, \( p_j \) received \((n - f)\) PROPOSED messages carrying the same value \( w \). Moreover, each process sends a single PROPOSED message to the other processes. As \( f < n/3 \), we have \((n - f) > n/2\). It follows that at least one PROPOSED(\( v \)) message and one PROPOSED(\( w \)) message have been sent by the same process. Consequently \( v = w \).

- If no process executes line 4, then the processes that decide execute line 7. In that case, due to the Uniform Agreement property of the underlying consensus protocol, they decide the same value.

- Let us now consider the case where some processes decide a value (say \( v \)) at line 4, while other processes decide at line 7. We claim\(^1\) that the variable \( v_j \) of any process \( p_j \) that executes line 7 has been previously set to \( v \) at line 6. Then, all the processes that execute the underlying protocol propose the same value \( v \) to this consensus. Due to the Validity property of the underlying consensus, they can only decide \( v \).

*Proof of the claim.* Let \( p_i \) be a process that executes line 4 and \( p_j \) be a process that executes line 5. We have the following:

1. \( p_i \) received \((n - f)\) PROPOSED(\( v \)) messages. Hence, no more than \( f \) PROPOSED messages carry a value different from \( v \).
2. \( p_j \) received \((n - f)\) PROPOSED messages. Due to (1), at most \( f \) of them carry a value different from \( v \). (In the worst case, those \( f \) values are equal.)
3. From (1) and (2) we conclude that at least \((n - 2f)\) PROPOSED messages received by \( p_j \) carry the value \( v \).
4. As \( n > 3f \), we have \((n - 2f) > f \). This means that the value \( v \) is a majority value among the values received by \( p_j \).
5. From the test done at line 5, we conclude that \( p_j \) updates \( v_i \) to \( v \), which concludes the proof of the claim.

\(^1\)Using traditional terminology [3], this claim states how a value decided during the first communication step is "locked".
4.3 Remark

When we consider the protocol described in Figure 1, the requirement $f \leq \lceil (n-1)/3 \rceil$ allows to replace the value $(n-f)$ (resp. $(n-2f)$) used at lines 2 and 3 (resp. line 5) with the value $\lceil (2n+1)/3 \rceil$ (resp. $\lceil (n+1)/3 \rceil$). These replacements can allow processes to decide earlier when the value of $f$ is smaller than its upper bound $\lceil (n-1)/3 \rceil$.

5 A Necessary Condition

Theorem 3 Let us consider an asynchronous distributed system made up of $n$ processes from which at most $f$ may crash. Let $P$ be any Consensus protocol that allows the processes to decide in one communication step when at least $(n-f)$ of them propose the same value. $P$ requires $f < n/3$.

Proof Let us first introduce the following parameters related to $P$:
- $\ell$: number of processes from which a process has to receive a value before deciding after one communication step (note that $\ell \leq (n-f)$, otherwise the protocol could block forever).
- $x$: number of messages containing the same value $v$, that allows a process $p_i$ to decide that value after the first communication step (note that $x \leq \ell$).

Let us observe that, as two processes that decide at the end of the first communication step have to decide the same value, it is necessary that $x > n/2$. (If this was not the case, $p_i$ could decide $v_1$ because it received $x$ copies of it, while $p_j$ could independently decide $v_2 \neq v_1$ because it received $x$ copies of it).

The proof is by contradiction. Let us assume that $P$ works in a system made up of $n = 3k$ processes with $k \leq f$. The processes are partitioned into three subsets $G_1$, $G_2$ and $G_3$ of size $k$. Combining $3f \geq 3k = n$ with $x \leq \ell \leq (n-f)$ and $x > n/2$, we get $k = n/3 < n/2 < x \leq \ell \leq (n-f) \leq (n-k) = 2k \leq 2f$. From $\ell \leq 2k$, we deduce $\ell - k \leq k < x$. Hence, $\max(k, \ell - k) < x$.

Let us consider the following scenario:

- No process has initially crashed. The processes of $G_1$ propose $v$; the processes of $G_2$ propose $v$; and the processes of $G_3$ propose $w$ (\(\neq v\)).

- Each process $p_i \in G_1$ receives values from $\ell \leq 2k \leq 2f$ processes of $G_1$ and $G_2$. As $x \leq \ell$, each process $p_i \in G_1$ receives enough copies of $v$ to decide (definition of $x$). So each process of $G_1$ decides $v$. Then, after having decided, the processes of $G_1$ crash.

- Each process $p_i \in G_2 \cup G_3$ receives values from $\ell \leq 2k \leq 2f$ processes of $G_2$ and $G_3$. More precisely, let us consider the scenario where:
  - Each process of $G_2$ receives $k$ copies of $v$ and $(\ell-k)$ copies of $w$.
  - Each process of $G_3$ receives $(\ell-k)$ copies of $v$ and $k$ copies of $w$.

From $\max(k, \ell - k) < x$, we conclude that no process $p_i \in G_2 \cup G_3$ can decide. It follows that any $p_i \in G_2 \cup G_3$ neither decides nor is blocked during the first communication step. Consequently, the processes of $G_2 \cup G_3$ continue executing $P$. Moreover, there is no way for them to know whether processes of $G_1$ have decided. The subsets of processes $G_2$ and $G_3$ are symmetric with respect to the number of copies of $v$ and $w$ they have. Hence, whatever $P$, the processes of $G_2 \cup G_3$ can indistinctly decide $v$ or $w$. The Uniform Agreement property is violated in all the runs of $P$ that decide $w$. 

\(\Pi n^- 1321\)
This shows there is no protocol \( P \) when \( n = 3k \) with \( k \leq f \). A contradiction. \( \square \) Theorem 3

**Corollary 1** Let us consider the family of the Consensus protocols that, in asynchronous systems, allow the processes to decide after one communication step when at least \( (n-f) \) of them propose the same value. Within this family, the protocol presented in Section 3 is optimal with respect to the number of process crashes that can be tolerated.

**Proof** Follows directly from Theorem 3. \( \square \) Corollary 1

## 6 Circumventing The Necessary Condition

### Function `Consensus(v_i)`

<table>
<thead>
<tr>
<th>Task T1:</th>
<th>broadcast ( \text{PROPOSED}(v_i) );</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>wait until (a ( \text{PROPOSED} ) message has been received from a majority of processes);</td>
</tr>
<tr>
<td></td>
<td>if (all the received values are equal to ( z ))</td>
</tr>
<tr>
<td></td>
<td>then broadcast ( \text{DECISION}(z) ); return(( z ))</td>
</tr>
<tr>
<td></td>
<td>else if (( z ) received from a process) then ( v_i \leftarrow z ) endif;</td>
</tr>
<tr>
<td></td>
<td>return(( \text{Underlying_Consensus}(v_i) ))</td>
</tr>
<tr>
<td>Task T2:</td>
<td>upon reception of ( \text{DECISION}(v) ); broadcast ( \text{DECISION}(v) ); return(( v ))</td>
</tr>
</tbody>
</table>

Figure 2: Use of a Privileged Value \( (f < n/2) \)

This section shows that the previous necessary condition can be circumvented when the system satisfies additional assumptions. Those assumptions basically define an “a priori agreement” among the processes. We give here two protocols that, given such an appropriate additional assumption, allow one step decision when \( f < n/2 \). (Their proofs are left to the reader.)

### 6.1 Existence of a Privileged Value

Let us assume that there is a value (say \( z \)) that is privileged among the values that can be proposed. Moreover, all the processes know it. The a priori knowledge of this distinguished value can help expedite the decision when \( f < n/2 \) as shown by the protocol described in Figure 2. The idea of the protocol is very simple: it strives to decide this value, when it has been proposed by a majority of processes.

### 6.2 Predefined Set of Processes

Let us assume that there is a predefined set of processes (say \( S \)) that is initially known by each process. The protocol described in Figure 3 uses this a priori knowledge to decide in one communication step when all the processes of \( S \) propose the same value. It requires \( f < n/2 < |S| \).

## 7 Concluding Remark

This paper has presented a consensus protocol that makes the processes decide in one communication step when the processes that propose a value propose the same value. It has been shown that
Function Consensus($v_i$)

Task T1: broadcast PROPOSED($v_i$);

wait until (a PROPOSED message has been received from a $(n - f)$ processes);

if (a message has been received from each process $\in S$ and they those carry the same value $v$)

then broadcast DECISION($v$); return($v$)

else $v_i \leftarrow$ a value from a process $\in S$; return(Underlying Consensus($v_i$))

endif

Task T2: upon reception of DECISION($v$): broadcast DECISION($v$); return($v$)

Figure 3: Predefined Set of Processes ($f < n/2$)

this protocol requires $f < n/3$ and that this requirement is necessary. As noted in the Introduction, in practice failures occur but are rare. Moreover, in some practical agreement problems (e.g., Atomic Commit [5, 8]), processes usually propose the same value (e.g., COMMIT). The consensus protocol that has been presented is particularly attractive to solve these agreement problems in real systems. It has also been shown how additional assumptions allow to circumvent the $f < n/3$ requirement.

Acknowledgments

We would like to thank our colleague Michel Hurfin (from Mayenne) whose constructive comments helped us improve the content of the paper.

References


