Evaluating the Condition-Based Approach to Solve Consensus

Achour MOSTEFAOUI, Eric MOURGAY, Philippe RAÏPIN PARVÉDY, Michel RAYNAL
IRISA, Université de Rennes 1, Campus de Beaulieu, 35042 Rennes Cedex, France
{achour|emourgay|praipinp|raynal}@irisa.fr

Abstract

Several approaches have been proposed to circumvent the impossibility to solve consensus in asynchronous distributed systems prone to process crash failures. Among them, randomization, unreliable failure detectors, and leader oracles have been particularly investigated. Recently a new approach (called “condition-based”) has been proposed. Let an input vector be a vector whose i-th entry contains the value proposed by process \( p_i \). The condition-based approach consists in stating conditions on input vectors that make consensus solvable despite up to \( f \) process crashes. Several conditions have been proposed. (As an example, one of them requires that the greatest value in an input vector appears more than \( f \) times.)

This paper presents an evaluation of the condition-based approach to solve consensus. It shows that this approach is particularly attractive and very efficient when the probability of process crashes is low (a common fact in practice). In these cases, the probability for the condition-based protocol to terminate is practically equal to 1.

1 Introduction

Context of the paper

The consensus problem is a central paradigm of reliable distributed computing. Each process proposes a value, and all non-faulty processes have to decide a value (termination), such that there is a single decided value (uniform agreement) and that value has been proposed by a process (validity). This problem abstracts a family of problems (usually called agreement problems) where the processes have to agree on the same value, that value representing a state from which they can correctly coordinate their behavior, a view they share on the computation progress, or a common decision ensuring that from now on they are mutually consistent. Solving any of these problems in an asynchronous distributed system basically amounts to reduce it to the consensus problem. As an example, let us consider the atomic broadcast problem. Processes can broadcast messages, and message deliveries have to be such that (1) the non faulty processes deliver the same set of messages in the same order, and (2) at least the messages broadcast by the non faulty processes are delivered. Actually, atomic broadcast = reliable broadcast + consensus [2]. Reliable broadcast solves the message dissemination problem, while consensus allows the processes to agree on their delivery order. More generally, as consensus allows distributed processes to be mutually consistent despite faulty behaviors, it is pervasive in all distributed applications.

One of the most fundamental (and surprising) result in distributed computing is that this apparently simple problem has no deterministic solution in an asynchronous system even if only one process may crash. This is the well-known FLP impossibility result [7]. This means that any protocol that always guarantees the consensus safety properties (namely, agreement and validity) has at least one run that does not terminate.

Several approaches have been proposed to circumvent the FLP impossibility result. One of them consists in relaxing the problem requirements by allowing for either probabilistic solutions [1] or \( k \)-set agreement [3] (up to \( k \) different values can be decided). Another approach consists of enriching the underlying system with synchrony assumptions until they allow the problem to be solved [6]. This approach has been abstracted in the notion of unreliable failure detectors [2]. An alternative approach consisting in equipping the underlying system with a leader oracle has also been investigated [5, 9, 12, 16].

Recently, a new approach to address the consensus problem in asynchronous systems, and consequently to circumvent the FLP impossibility result, has been proposed [14]. This approach focuses on sets of input vectors that allow \( n \) processes to solve the consensus problem despite up to \( f \) process crashes, in a standard asynchronous model. Let an input vector be a size \( n \) vector, whose i-th entry contains the value proposed by process \( p_i \) (1 \( \leq \) i \( \leq \) n, where n
is the number of processes). A condition (which involves the parameters \( f \) and \( n \)) is a set of such vectors that can be proposed under normal operating conditions. In addition to the statement of conditions suited to consensus, the focus of the condition-based approach is the design of \( f \)-fault tolerant protocols that (1) are always safe (safe means that the protocol guarantees agreement and validity, whether the proposed input vector belongs to the condition or not), and (2) guarantee termination at least when the input vector belongs to the considered condition and there are at most \( f \) process crashes (or when there is no crash at all). As a very simple example of a condition, let us consider the following one “more than a majority of the processes propose the same value”. It is not hard to see that, when \( f = 1 \), consensus can be solved when the input vectors satisfy this condition\(^1\).

Major results concerning the condition-based approach are:

- The design of a generic condition-based consensus protocol, two “natural” conditions, and a characterization\(^2\) of the set of conditions that allow to solve consensus [14],
- A general method to define “linear” conditions [15],
- The formulation of conditions that allow to solve consensus as a family of error correcting codes [8]\(^3\).

**Motivation and content of the paper** As the consensus safety properties have never to be violated (this is a “minimum” requirement if we want to solve consensus!), the FLP impossibility result basically says that an asynchronous consensus protocol cannot always guarantee the termination property. So, given a consensus protocol, an interesting question is “Does this protocol terminates very often?”(Such a question has not been explicitly answered in previous works on the consensus problem. The quality of service of failure detectors is studied in [4] where the metrics investigated are “how fast” failures are detected and (2) “how well” false detections are avoided. Such a study could be used to evaluate failure detector-based consensus protocols.) Here, our aim is to answer the previous question in the context of the condition-based approach.

In some sense, a condition-based consensus protocol does “its best” with respect to termination: it guarantees termination when the input vector belongs to the condition and there is no more than \( f \) crashes, or when there is no crash at all (whatever the input vector). In the other cases it can sometimes terminate according to the failure pattern, the perception each process has of the failure pattern, and the fact that the input vector is not “too far” from the condition. (A condition could be seen as a “consistent heuristics” with respect to termination.) So, the aim is to evaluate the “quality” of the condition-based approach with respect to the consensus termination property.

Similarly to failure detector-based, oracle-based, or randomization-based consensus protocols, a condition-based protocol explicitly uses the parameter \( f \) in its code. This allows a process to wait for values from \( n - f \) processes, and so construct a partial view of the input vector. But, \( f \) is only a parameter assumed to be an upper bound on the number of faulty processes. So, when more than \( f \) processes crash, processes can block forever, despite the fact that they could have terminated if they had considered a greater value for \( f \). \(^4\) To summarize, there are two causes for a condition-based protocol not to terminate: (1) there are more than \( f \) process crashes, or (2) the input vector does not belong to the condition.

To answer the question “Does a condition-based protocol terminate very often?”, we consider the following parameters: \( \lambda \) (probability that a process crashes), \( n \) (total number of processes), and \( f \) (an a priori fixed value). With those parameters, the previous question refines into the following questions (where “good” means “provides a high probability of termination”, and “better than” means “allows the protocol to terminate more often than”):

- Given a pair \((n, f)\), and assuming \( f \) is an effective upper bound on the number of process crashes, (1) are there good conditions? (2) are there conditions better than others? The aim here is to know if the condition-based approach is good, and among several conditions that allow to solve consensus, which one is the best (if any).
- Given \( n, \lambda \), and a condition \( C \), which is a good value for the parameter \( f \)? Here, (1) according to the underlying system features (defined by \( n \)-the size of the system- and \( \lambda \)-a measure of the “quality” of each process with respect to failures-), and (2) for a given condition \( C \), the aim is to find a well “calibrated” value of the parameter \( f \) that, when plugged into the protocol code, provides a good protocol instance.

The main lessons learned by answering these questions are the following:

\(^1\)It is plausible to imagine an application that in some real system satisfies this condition most of the time; only when something goes wrong, the processes proposals get evenly divided, and only then should the protocol take longer to terminate (or even not terminate).

\(^2\)It is also shown that deciding if a condition allows to solve consensus can be done in polynomial time.

\(^3\)This formulation is pretty general as it includes the case of Byzantine process failures.

\(^4\)This is why many protocols consider \( f = \lfloor (n - 1)/2 \rfloor \), which is the greatest value the parameter \( f \) can take without compromising the termination of the non faulty processes.
Both the conditions $C1$ (which tries to decide the maximal proposed value) and $C2$ (which tries to decide the proposed value that appears the most often) are good with respect to the protocol termination property. Moreover, $C1$ is better than $C2$.

Given a condition $C$ and a pair $(n, \lambda)$ characterizing a system, it is possible to compute a value of $f$ that maximizes the probability that the protocol terminates. (That value of $f$ is usually much smaller than the upper bound $\lceil(n - 1)/2\rceil$.)

Roadmap The paper is made up of five sections. Section 2 introduces the computation model and the consensus problem. Section 3 presents the condition-based approach. Then, Section 4 presents an evaluation of the condition-based approach to solve consensus. It first sets questions that we want to answer and then shows how they can be answered. Finally, Section 5 concludes the paper. Interestingly, this paper can also be seen as a discussion on failure mode assumptions and assumption coverage as introduced and investigated by Powell [17].

2 Computation Model and the Consensus Problem

2.1 Computation Model

Processes The computation model we consider is basically the asynchronous system model of [2, 7, 13]. The system 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. Until it possibly crashes, the process behaves according to its specification. By definition, a correct process is a process that does never crash. A faulty process is one that is not correct. In the following $f$ denotes an integer deemed to be an upper bound on the number of processes that may crash.

Channels Processes communicate and synchronize by sending and receiving messages through channels. Each pair of processes is connected by a communication channel. Channels are assumed to be reliable: messages are not lost, altered or duplicated. There is no assumption about the relative speed of processes nor on message transfer delays.

The message-passing condition-based algorithm presented in Section 3.3 makes use of a communication abstraction (that can easily be built on top of the previously described channels [10]), named Uniform Reliable Broadcast. This communication abstraction is defined by two primitives: $UR_{Broadcast}(m)$ (resp. $Delivery(m)$) we say that it broadcasts $m$ (resp. delivers $m$). We assume that all the messages are different.

- Validity: If a process delivers $m$, then some process has broadcast $m$. (No spurious messages.)
- Integrity: A process delivers a message $m$ at most once. (No duplication.)
- Termination: If (1) a correct process broadcasts $m$, or if (2) a process delivers $m$, then all correct processes deliver $m$. (No message broadcast by a correct process or delivered by a process is missed by a correct process.)

2.2 The Consensus Problem

The Consensus problem has been informally stated in the Introduction. Every process $p_i$ is supposed to propose a value $v_i$ and the processes have to decide on some value $v$, that has to be one of the proposed values. In the following, $\mathcal{V}$ represents the set of values that can be proposed ($|\mathcal{V}| \geq 2$). In terms of the properties that any protocol solving the problem has to satisfy, consensus can be specified by two safety properties (validity and agreement) and a liveness property (termination) [2, 7] (this definition is sometimes called uniform consensus):

- Validity: If a process decides $v$, then $v$ was proposed by some process.
- Agreement: No two processes decide differently.
- Termination: Every correct process eventually decides on some value.

3 The Condition-Based Approach

3.1 Principles

The condition-based approach consists in considering conditions that make the consensus problem solvable despite up to $f$ process crashes. More precisely, it consists in identifying sets of input vectors for which it is possible to design a consensus protocol that does not require additional assumptions (such as a failure detector [2, 9, 11] or a leader oracle [12, 9, 16]).

An input vector is a size $n$ vector, whose $i$-th entry contains the value proposed by $p_i$, or $\perp$ if $p_i$ did not take any step in the execution ($\perp$ denotes a default value such that $\perp \notin \mathcal{V}$). We usually denote with $I$ an input vector with all entries in $\mathcal{V}$, and with $J$ an input vector that may have some entries equal to $\perp$. As (by assumption) at most $f$ processes are assumed to crash, we consider only input vectors $J$ with
at most \( f \) entries equal to \( \bot \), called views. Let \( V^n \) be the set of all possible input vectors with all entries in \( V \). For \( I \in V^n \), let \( I_{f} \) be the set of possible views, i.e., the set of all input vectors \( J \) such that \( J \) is equal to \( I \) except for at most \( f \) entries that are equal to \( \bot \). For a set \( C, C \subseteq V^n \), let \( C_{f} \) be the union of the \( I_{f} \)'s over all \( I \in C \). Thus, in the consensus problem, every vector \( J \in V^n_{f} \) is a possible input vector.

As already stated, the condition-based approach focuses on conditions \( C \) that, when satisfied (i.e., when the proposed input vector does belong to \( C_{f} \)), make the consensus problem solvable, despite up to \( f \) process crashes. According to the terminology of [14], we say “a protocol solves the consensus problem for a condition \( C \)” if in every execution whose input vector \( J \) belongs to \( V^n_{f} \), the protocol satisfies the consensus validity and agreement properties plus the following termination property: If (1) \( J \in C_{f} \) and no more than \( f \) processes crash, or (2) all processes are correct, or (3) a process decides, then every correct process decides.

Not all conditions allow to solve consensus (otherwise, it would contradict the FLP impossibility result). So, to solve consensus, a condition \( C \) has to meet constraints. It is shown in [14] that those constraints can be defined in terms of a predicate \( P \) and a deterministic function \( S \). Intuitively, \( P \) permits to test whether a decision value can be computed from a process’ view, while \( S \) applied to a view allows a process to get the corresponding decision value. More precisely, \( S(J) \) deterministically selects a value of \( J \) (with \( a \neq \bot \)). [14] provides a characterization of the largest set of conditions that allows to solve consensus. The fundamental property that allows a condition \( C \) to solve consensus despite up to \( f \) crashes is the following (where \( S \) is the function associated with \( C \)):

\[
\forall I \in C : \forall J_{1}, J_{2} \in I_{f} : S(J_{1}) = S(J_{2}).
\]

This means that, when it is not possible to associate such a deterministic function \( S \) with a set \( C \) of input vectors, then \( C \) cannot guarantee the consensus termination property in presence of process crashes.

Acceptability is the key notion that makes a condition work or not work. Intuitively a condition is acceptable if it imposes a sufficient distance between vectors from which different values are decided despite up to \( f \) process crashes. Two natural conditions are described in the next section to make more explicit the underlying principles of the condition-based approach.

### 3.2 Two Natural Conditions

Intuitively, a condition selects a proposed value to become the decided value. As two “natural” examples of a condition, we consider here the conditions \( C1 \) and \( C2 \) introduced in [14]. A general method to define linear conditions is described in [15]. In the following, \( \#_a(J) \) denotes the number of occurrences of \( a \) in the vector \( J \), with \( a \in V \cup \{ \bot \} \).

**Condition C1: The maximal value** This condition assumes that the set \( V \) (the values that can be proposed) is totally ordered. Considering an input vector \( I \in V^n \), \( C1 \) favors an extremal value of \( I \) (the greatest or the smallest value). This condition can be useful when the values proposed to consensus are not symmetric for example for the non-blocking atomic commit problem the proposed values can be commit or abort and we can favor one over the other. Considering here that “extremal”=“greatest”, let \( \text{max}(I) \) denote the greatest value of \( I \). \( C1 \) is formally defined as follows:

\[
(I \in C1) \overset{def}{=} \left[ a = \text{max}(I) \Rightarrow \#_a(I) > f \right].
\]

So, \( C1 \) states that, when at least \( f + 1 \) entries of the input vector \( I \) are equal to its greatest entry, then consensus can be solved despite up to \( f \) process crashes. Interestingly, this condition does not consider the other values of the input vector.

When we consider an input vector \( J \in V^n_{f} \) (that is to say an input vector \( J \) where some processes have initially crashed, or a vector \( J \) representing a view of the actual input vector that a process can obtain), \( C1 \) has to take into account the fact that some entries of \( J \) can be equal to \( \bot \). As shown in [14], the previous formulation of \( C1 \) can be converted into the following more general form. (to simplify notation we assume that \( \bot \) is smaller than any value of \( V \)):

\[
(J \in C1_{f}) \Leftrightarrow (a = \text{max}(J)) \Rightarrow \#_a(J) > f - \#_\bot(J).
\]

So, when \( C1 \) is considered, a process tries to decide the value defined by \( S1(J) = \text{max}(J) \), where \( J \) is its current view of the input vector.

**Condition C2: The most frequent value** Differently from \( C1 \), the aim of \( C2 \) is to favor the value that appears the most frequently in a vector. Let \( \#_{1st}(J) \) be the number of occurrences of the most frequent non-\( \bot \) value of \( J \), and \( \#_{2nd}(J) \) be the number of occurrences of its second most common non-\( \bot \) value. If \( \#_{1st}(J) = \#_{2nd}(J) \), the function \( S2 \) can deterministically output any of them. Considering a vector \( I \in V^n \), \( C2 \) is defined as follows:

\[
(I \in C2) \overset{def}{=} \left[ \#_{1st}(I) - \#_{2nd}(I) > f \right].
\]

This means that, when the input vector \( I \) has a value that appears at least \( f + 1 \) more times than any other value, consensus can be solved despite up to \( f \) process crashes. Interestingly, this condition does not assume that the set \( V \) is ordered.
As previously, when we consider an input vector $J \in \forall^n_j$, $C2$ takes the following more general form [14]:

$$(J \in C2) \iff \#1st(J) - \#2nd(J) > f - \#\bot(J).$$

So, when $C2$ is considered, a process tries to decide $S2(J) = a$, where $a$ is the most common value of its view $J$ of the input vector.

**Remarks** The previous condition $C2$ and a refined version of $C1$ are maximal in the sense that adding to any of them an input vector that they do not include, provides conditions that do no longer allow to solve consensus [14].

When we consider the case $n > 3$, $|\forall| = 2$ and $f = 1$, we have $C1 \cup C2 = \forall^n$. This shows that the set of conditions that allows to solve consensus cannot be composed (as a condition-based consensus protocol using such a composition would contradict the FLP impossibility result [15].

### 3.3 A Message-Passing Protocol

A variant of a message-passing condition-based consensus protocol introduced in [14] is described in Figure 1. This protocol assumes $f < n/2$ which has been shown to be a necessary requirement for the condition-based approach to solve consensus in message-passing systems. The protocol is relatively simple. It is made up of two phases.

- The first phase allows each process $p_j$ to build its local view of the input vector. This view is kept in the vector $V_j$ (lines 2-4). Then, each process computes its “estimate” $w_j$ of the decision value. To this end, $p_j$ uses the function $S$ associated with the condition $C$ that is considered (line 5), e.g., $S$ is “take the maximal value of $V_j^n$”, when $C1$ is considered.

- Then, the processes start the second phase, during which they first exchange their estimates $w_j$ (line 6). If $p_i$ receives the same estimate $w$ from a majority of processes, it decides that value (lines 9-10). If the input vector does not belong to the condition and no process crashes, then $V_j$ eventually contains the “full” input vector (without $\bot$ entries), and the processes can thereby decide the same value from this vector.

**Theorem 1** The protocol satisfies the consensus validity and agreement properties plus the following termination property: If (1) $J \in C_f$ and no more than $f$ processes crash, or (2) all processes are correct, or (3) a process decides, then every correct process decides.

**Proof** (Sketch) It is easy to see that the protocol satisfies consensus validity, as due to $S$, an estimate is always a proposed value. The proof of uniform agreement comes from the following observations: (1) If a process decides at line 10, then no process decides at line 12: (2) When processes decide at line 10, they necessarily decide the same value $w$, as $w$ is then a majority estimate.

As far as the termination is concerned we have the following. Let $J$ be the actual input vector.

- (1) If $J \in C_f$ and no more than $f$ processes crash, then $S(V_j)$ provides all processes with the same estimate $w$, and as there is a majority of correct processes, they terminate at line 10.

- (2) If all processes are correct, and no process has decided at line 10, then each $p_i$ gets a vector $V_j$ with no $\bot$ entry, and they all decide at line 12.

- (3) If process decides at line 10, it got PHASE2 messages carrying the same estimate $w$ from a majority of processes. But then, due to the termination property of the $UR_BROADCAST$ primitive, it follows that all correct processes have been delivered the same PHASE2 messages, and consequently decide. Similarly, if $p_i$ decides at line 12, all correct processes have been delivered a PHASE2 message from each process, and consequently each correct process decides. So, if a process decides, all correct processes decide.

\[\Box\text{Theorem 1}\]
4 Evaluation

4.1 Aim: the Questions we Want to Answer

As announced in the Introduction, we are interested in evaluating the condition-based protocol with respect to the consensus termination property, i.e., in answering the question “Does the basic condition-based protocol described in Figure 1 terminate very often?”

In the following, λ denotes the probability that a process crashes. We assume that the random events “crash of a process” are independent. Many papers on fault-tolerance (e.g., checkpointing) use such a probability. They commonly use a Poisson distribution whose average is around 10−9/second [18]. This gives for each process a probability λ to crash ranging from 0 to 0.1 for a period of time of 105 seconds.

Pterm denotes the probability that the condition-based protocol terminates. The function d(Vi, Vj) denotes the Hamming distance between the vectors Vi and Vj. Let us notice that, if Vi and Vj are the views (of an input vector) obtained at line 4 by pi and pj, respectively, we have 0 ≤ d(Vi, Vj) ≤ 2f.

Answering “how good is the condition-based approach with respect to termination?” (i.e., is Pterm close to 1) depends on several parameters, including the considered condition C, and the values of n, f, and λ. So, the previous question can be refined according to these parameters.

Question Q1 The first questions we want to answer is “How good is C1 (resp., C2) with respect to termination?” and “Which condition is the best (if any)?” To answer these questions, assuming that at most f (< n/2) processes crash, we compute Pterm(C1) and Pterm(C2) for (n, f, λ) triples.

Question Q2 The parameter f defines the upper bound on the number of crashes tolerated by the protocol, but more than f processes can crash during an execution. So, it is possible that, for a given execution whose input vector belongs to the condition, an instance of the protocol with f1 does not terminate (the correct processes blocking forever) because the actual number of crashes x is such that x > f1, while an instance with a greater value f2 > f1 would terminate if x is such that x ≤ f2. So, a second question is “Which value of the parameter f has to be incorporated in the protocol in order to get a good protocol instance?” To our knowledge there has been no (analytical or simulation) consensus protocol evaluation that uses the probability of a process to crash (λ) and tries to determine the best value of f.

To be more explicit let us look at the protocol. The termination of the first phase (lines 2-5) depends on (1) the number x of processes that have crashed, and (2) the value of the parameter f (line 3). (Let us observe that the probability that the first phase terminates is probability that x ≤ f, it does not depend on the condition C that is considered.) Differently, the termination of the second phase (when we consider the part specifically related to the condition-based approach, i.e., the lines 6-10) depends on the values w = S(Vi), i.e., depends on the fact that the input vector views V1, V2, . . . obtained by a majority of processes are such that S(Vi) = S(Vj) (which is always satisfied when the input vector belongs to the considered condition C). This means that there is a tradeoff between the termination of each phase. More precisely,

- Let us first consider a small value of f. In that case, if there is no more than f crashes, the views Vi of the input vector contain few ⊥, and consequently, given two processes pi and pj, the probability p to have S(Vi) = S(Vj) (that is equal to 1 when the input vector belongs to the condition) remains high when the input vector does not belong to the condition. This is because d(Vi, Vj) is small when f is small, and S is a deterministic selection function. Here, the important point is that the smallest the value of f, the highest the probability p, and so the highest the probability for the processes to compute the same w value, and consequently to decide during the second phase.

But, when f is small, according to the value of λ, it is possible that more than f processes crash, and consequently (as a process waits for (n − f) processes to terminate if (n − f) processes crash with respect to the value of the parameter f) and the probability that the second phase terminates (which depends on the condition and the number of crashes): when f increases from 1 to [(n − 1)/2], the probability for the first phase to terminate increases, while the probability for the second phase to terminate decreases (when we consider the case where the input vector does not belong to the condition). So, the question is “Given a pair (n, λ) characterizing the underly-
ing system, and a condition $C$, which is a good value of the parameter $f$, i.e., a value that maximizes $P_{\text{term}}(C, n, \lambda)$?\footnote{Alternatively, by starting with the random variable $y_1 = n - x - y_0$, the reasoning could be based on the value 1 instead of 0.}

The next sections answer the previous questions considering binary consensus, i.e., $V = \{0, 1\}$.

## 4.2 Analytical Model

The following analytical evaluation of the condition-based protocol assumes that (1) if a process crashes, it crashes before the protocol starts its execution; (2) the probability that $p_i$ proposes 0 (resp., 1) is $1/2$. The parameters $n$, $f$, and $\lambda$ have been previously defined (let us remind that $f \leq [(n - 1)/2]$ is an upper bound on the number of crashes tolerated by the protocol, and is not an upper bound on the actual number of crashes). Finally, $x$ denotes the actual number of crashes during an execution; so, there are $n - x$ correct processes during that execution. $C$ denotes a condition ($C1$ or $C2$).

Let $\text{DEC}(x)$ be the probability for the correct processes to terminate assuming exactly $x$ processes have initially crashed (the $n - x$ remaining processes are correct). Let us notice that $x$ depends on $\lambda$. Let us also notice that $\text{DEC}(0) = 1$, and $\text{DEC}(x) = 0$ for $x > f$ (as, in this latter case, all non crashed processes block forever in the first phase as there are less than $n - f$ correct processes). Hence, the probability $P_{\text{term}}(C, n, \lambda)$ that the protocol (instantiated with the condition $C$ and the parameter $f$) terminates in a distributed system (characterized by the parameters $n$ and $\lambda$), is

$$P_{\text{term}}(C, n, \lambda) = \sum_{x=0}^{[n/2]} \binom{n}{x} \lambda^x (1 - \lambda)^{n-x} \text{DEC}(x).$$

To compute $\text{DEC}(x)$ for $0 < x \leq f$, let us observe that $\text{DEC}(x)$ is the probability that, given $x$ initial crashes, a majority of processes compute the same $w$ value at the end of the first phase of the protocol (line 5). Let us notice that, during the second phase, a process waits for at most $n - x$ PHASE2 messages. It waits until it gets a majority value (if any), or blocks forever after it has been delivered $n - x$ PHASE2 messages and there is majority value. Among those $n - x$ messages, some carry $w_j = 0$ while others carry $w_j = 1$. In order to determine $\text{DEC}(x)$, we consider the two following possibilities:

- $P_{\text{OCC0}}(x, y_0)$: probability that exactly $y_0$ processes propose the value 0 at line 2, given that $x$ processes have initially crashed and none among the $n - x$ remaining processes crashes later (hence, $n - x - y_0$ propose the value 1)$^3$. As the process proposals are independent (a process proposes

$0$ or $1$ with probability $1/2$), it follows that (binomial distribution)

$$P_{\text{OCC0}}(x, y_0) = \binom{n-x}{y_0} (1/2)^{y_0} (1/2)^{n-x-y_0}.$$  

- $P_{\text{DEC}}(x, y_0)$: probability that at least one process decides given that exactly $y_0$ processes initially propose 0, $x$ processes have initially crashed and none among the $n - x$ remaining processes crashes later. $P_{\text{DEC}}(x, y_0)$ is actually the probability that a majority of processes compute the same $w$ value (0 or 1) at line 5.

To determine $P_{\text{DEC}}(x, y_0)$, let us define $s_0(x, y_0)$ as the probability that a process $p_i$ computes $w_i = 0$ at line 5 given that $x$ processes have crashed and $y_0$ processes have initially proposed 0. Each pair $(x, y_0)$ characterizes a class of input vectors from which the same value is decided. Let $I$ be one of those vectors. $s_0(x, y_0)$ is equal to the probability that a process gets a local view $J \leq f$ such that $S(J) = 0$ ($S$ is the function associated to the condition as defined Section 3). Let us notice that $s_0(x, y_0)$ is the only part involving the condition $C$ with which the protocol is instantiated$^6$. More explicitly,

- For $C1$ (maximal value): $s_0(x, y_0)$ is the probability for $p_i$ to get only 0 values in the $n - f$ messages it has been delivered during the first phase (as to get $w_i = 0$, $p_i$ must have received only 0 values at line 2, otherwise we would have $w_i = 1$).

- For $C2$ (most frequent value): $s_0(x, y_0)$ is the probability for $p_i$ to get more 0 values than 1 values among the $n - f$ messages it has been delivered during the first phase.

In the following, to simplify notation, $s_0(x, y_0)$ is abbreviated as $s_0$. Moreover, let $s = 1 - s_0$ (probability that a process $p_i$ computes $w_i = 1$, given that $x$ processes have crashed and $y_0$ processes have proposed 0).

From the previous discussion we get

$$P_{\text{DEC}}(x, y_0) = \sum_{k=\lceil \frac{n-x}{2} \rceil}^{n-x} \binom{n-x}{k} (s_0^k s^1 s_1^{n-x-k} + s_1^k s_0^{n-x-k}).$$

Finally, as $0 \leq y_0 \leq n - x$, we get

$$\text{DEC}(x) = \sum_{y_0=0}^{n-x} P_{\text{OCC0}}(x, y_0) \times P_{\text{DEC}}(x, y_0).$$

## 4.3 Answering Q1

To answer Q1 (namely, how good are $C1$ and $C2$ with respect to termination, and which one is the best) we have

$^3$The determination of $s_0(x, y_0)$ can be easily done for all weight-based conditions [15].
considered the protocol instantiated with the maximal value
that the protocol parameter $f$ can take, i.e., $f = \lfloor (n-1)/2 \rfloor$. The results are depicted on the Figures 2 and 3. On each of
these figures, the left side concerns $C_1$ while the right side concerns $C_2$.

Figure 2.a (resp. 2.b) displays the probability $P_{\text{term}}$ that
the protocol instantiated with $C_1$ (resp. $C_2$) terminates ac-
cording to $\lambda$ (process crash probability) for $0.01 \leq \lambda \leq 0.1$. Three curves are shown on each figure, corresponding to
$n = 10, 14$ and $18$, respectively. These figures show that, for both $C_1$ and $C_2$, $P_{\text{term}}$ is very close to 1. Moreover, it
is extremely close to 1 for $C_1$.

Let us now consider Figure 3 that displays $P_{\text{term}}$ according to $n$, the number of processes ($6 \leq n \leq 18$), for
given values of $\lambda$ (namely, $\lambda = 0.005$, 0.01 and 0.05). This figure complements Figure 2, and shows how the protocol s-
cales when $n$ increases. We can see that (1) $P_{\text{term}}$ remains
close to 1 in all cases, and (2) $P_{\text{term}}$ increases with $n$ when we consider $C_1$, while it lightly decreases when $n$ increases when we consider $C_2$. This is not counter-intuitive as $C_1$ requires that a majority of processes see the same maximal value, while $C_2$ is “more constraining” as a majority of process have to see the same most frequent value. This suggests that the use of a particular condition may depend on the application.

So, as far as the protocol termination is concerned, two main lessons can be learned from Figures 2-3 (let us remind

that $\lambda$ is very small in practice):
- First lesson: Both $C_1$ and $C_2$ are good.
- Second lesson: $C_1$ is better than $C_2$.

The intuition behind the first lesson is that it is enough
that a majority of processes see the same distinguished val-
ue (maximum or most frequent one) to decide. As the only possible values are 0 and 1 this is very likely to happen. However, this probability can never be one as it will con-
tradict the FLP impossibility result. Concerning the second lesson, $C_1$ is better than $C_2$ but $C_1$ is asymmetric with re-
gard to the probability to decide 0 or 1. The probability to
decide the greatest value is much higher than the probability
to decide the issmallest value. When using $C_2$, the probabil-
ity to decide 1 is equal to the probability to decide 0 (no
distinguished value). The question whether there exists a
condition better than $C_1$ is an open problem.

4.4 Answering Q2

In order to determine a well calibrated value for the pa-
rameter $f$ that, when plugged into the protocol will pro-
vide a good protocol instance with respect to termination,
we have computed $P_{\text{term}}$ for different values of $f$ ($1 \leq f \leq \lfloor (n-1)/2 \rfloor$). The result of these computations are
described in Figure 4.

The simulations show that there is no best of value of $f$
if we consider only the size $n$ of the system whether we use
$C_1$ or $C_2$. The value of $f$ that provides an optimal proto-
Figure 3. How $P_{term}$ varies according to $(n, \lambda)$

The condition-based approach to solve consensus consists in stating conditions on input vectors (vectors of proposed values) that make consensus solvable despite up to $f$ process crashes. Considering two natural conditions (decide the greatest value, or decide the most common value), this paper has presented an evaluation of the condition-based approach. It appears that this approach is practically very relevant as the probability that a condition-based protocol terminates is very close to 1. The paper has also shown how the optimal value of critical protocol parameters (such as $f$, the maximal number of processes that the protocol allows to crash) can be determined.

Acknowledgments

We would like to acknowledge the referees and Neeraj Suri whose comments helped us improve the presentation of the paper.

References


Figure 4. Determining $f$ for $C1$ according to a pair $(n, \lambda)$


