

Efficient Condition-Based Consensus

ACHOUR MOSTÉFAOUI
IRISA, France

SERGIO RAJSBAUM
Compaq Research Lab, USA

MICHEL RAYNAL
IRISA, France

MATTHIEU ROY
IRISA, France

Abstract

The *condition-based* approach for consensus solvability (that we have introduced in a previous paper, ACM STOC'01) consists in identifying sets of input vectors for which it is possible to design a protocol solving the consensus problem for n processes despite the occurrence of up to f process crashes. For each value of f these conditions actually defines a hierarchy. This paper continues our investigation of this approach. It has two main contributions. It first show that it is possible to define conditions from very simple weight functions. Interestingly any weight function define an acceptable condition, i.e., a condition that allows to solve the consensus problem. The second contribution is an efficient protocol whose wait-free part has a number of shared memory accesses upper bounded by $O(n \log_2(f + 1))$ (interestingly, this upper bound is rarely attained).

Keywords

asynchronous distributed system, condition, consensus problem, early decision, fault-tolerance, input vector, shared memory

1 Introduction

Context of the study The *Consensus* problem lies at the heart of many distributed computing problems one has to solve when designing reliable applications on top of unreliable distributed asynchronous systems. There is a large literature dedicated to studying theoretical and practical aspects of this problem, that can be informally stated in terms of three requirements. Each process proposes a value, and has to decide on a value (termination) such that there is a single decided

value (agreement), and the decided value is one of the proposed values (validity). One of the most fundamental impossibility results in distributed computing says that this apparently simple problem has actually no deterministic solution in an asynchronous system even if only one process may crash [8]. To circumvent this impossibility, known as FLP, two main approaches have been investigated. One of them consists of relaxing the requirements of the problem, by either allowing for probabilistic solutions (e.g., [3]), or for approximate solutions (ϵ -agreement [6], or k -set agreement [5]). Another approach consists of enriching the system with synchrony assumptions until they allow the problem to be solved [7]. This approach has been abstracted in the notion of unreliable failure detectors [4].

We have recently introduced a new *condition-based approach* to tackle the consensus problem [9]. 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 a process p_i . A *condition* (which involves the parameters f and n) is a set of such vectors that can be proposed under normal operating conditions. This approach is interested in f -fault tolerant protocols that (1) solve consensus at least when such a condition holds, and (2) are always safe. Safe means that the protocol guarantees agreement, whether the proposed input vector is allowed by the condition or not. In addition, this approach is also concerned in protocols that make their “best effort” to terminate (for example, they should terminate in all failure-free executions). This is the best we can hope for, since the FLP impossibility result says we cannot require that a consensus protocol terminates always, for every input vector. But, by guaranteeing that safety is never violated, the hope is that such a protocol should be useful in applications. For example, let us consider the condition “more than a majority of the processes propose the same value.” It is not hard to see that consensus can be solved when the inputs satisfy this condition, when $f = 1$. 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).

Previous work on the condition-based approach In [9], we characterized the conditions that accept a consensus protocol with the above properties. That is, we described a set of conditions, denoted here \mathbb{C}_f , and proved that there is a consensus protocol for a condition C if and only if $C \in \mathbb{C}_f$. We presented two equivalent combinatorial descriptions of the class \mathbb{C}_f , and described two natural conditions $C1$ and $C2$ in \mathbb{C}_f that might be useful in practice, and proved them to be maximal (in the sense they cannot contain more input vectors). Basically, $C1$ includes the vectors that favor their greatest value, while $C2$ is made up of the vectors that favor their most common value. The class \mathbb{C}_f is quite rich, since it includes every condition for which there exists a condition-based consensus protocol. The protocol that we have presented in [9] can be instantiated for each particular condition $C \in \mathbb{C}_f$. It has the same step complexity, whatever the condition it is instanti-

ated with, namely $O(n \log_2(f + 1))$ read/write shared memory operations in the wait-free part of each process.

Although a priori it could be that all conditions of \mathbb{C}_f are equally difficult to solve, an interesting question is the following “*Are there conditions that are more difficult to solve than others?*”. If this is the case, there are more efficient protocols, specially tailored for particular classes of conditions. In practice, one would be interested in identifying the simplest classes of conditions whose input vectors occur frequently, because such classes would perhaps have very efficient consensus protocols. This question has been partially answered in [10] where we have shown that \mathbb{C}_f actually defines a hierarchy of classes of conditions, each one of some degree d ($0 \leq d$), namely, $\dots \subset \mathbb{C}_f^{[d+1]} \subset \mathbb{C}_f^{[d]} \subset \dots \subset \mathbb{C}_f^{[1]} \subset \mathbb{C}_f^{[0]}$, where $\mathbb{C}_f^{[0]}$ is the class \mathbb{C}_f of the weakest conditions.

We have also presented in [10] a condition-based consensus protocol that can be used for any condition of degree d , and shown that the value $(f - d)$ measures the “*difficulty*” of the class $\mathbb{C}_f^{[d]}$. More precisely, it is shown that the number of read/write operations that are executed by a consensus protocol is related to d . Roughly speaking, for any condition C in the class $\mathbb{C}_f^{[d]}$, the number of read/write invocations of the wait-free part of the protocol is proportional to $n \log_2(f - d + 1)$. Hence, when we progress in the hierarchy from the largest class $\mathbb{C}_f = \mathbb{C}_f^{[0]}$, there are more and more efficient consensus protocols, until the class $\mathbb{C}_f^{[f]}$ ¹.

Content of the paper This paper continues our study of the condition-based approach. It has two main contributions. For lack of space, the proofs are in Appendix II and a third contribution of the paper is given in Appendix III.

First contribution. A generic form to express conditions is first presented. This generic expression is based on weights associated with the values that can be proposed by the processes. Whatever the weight function used to instantiate the generic formula we get, for any pair (f, d) , a condition that belongs to $\mathbb{C}_f^{[d]}$. Interestingly, and in addition to its simplicity, this generic formulation not only includes the two previous particular conditions that we have investigated in previous works [9, 10] (namely, $C1$ and $C2$), but also allows to define new conditions to solve the consensus problem. While there are conditions that cannot be derived from the generic expression, it seems that the generic expression captures a meaningful set of practically relevant conditions.

Second contribution. The second contribution is related to the efficiency of condition-based consensus protocols. As noticed previously, the protocol presented in [9] solves the consensus problem for any condition in \mathbb{C}_f , but requires $O(n \log_2(f + 1))$ read/write shared memory operations per process in its wait-free part. Differently, the protocol described in [10] costs only $O(n \log_2(f - d + 1))$

¹In a sequel paper [12] we showed that for $f < n/2$, there is a more efficient protocol whose wait-free part has no operation.

read/write shared memory operations per process, but guarantees termination only when the actual input vector I belongs to $\mathbb{C}_f^{[d]}$. Hence, if $I \in \mathbb{C}_f - \mathbb{C}_f^{[d]}$, it is possible that the protocol tailored for $\mathbb{C}_f^{[d]}$ does not terminate.

Hence, the following question: “*Is it possible to design a protocol for the conditions in \mathbb{C}_f , that, when compared to the protocol presented in [9], can allow processes to decide earlier and at a lower cost when the input vector belongs to $\mathbb{C}_f - \mathbb{C}_f^{[1]}$?*”. It appears that the answer to this question is “yes”. To answer it, the paper presents such an adaptive protocol. Only in the worst case, a process has to execute $O(n \log_2(f + 1))$ read/write shared memory operations (in its wait-free part). The actual number of such operations actually depends on the number of actual failures and their perception by the processes.

Organization of the paper The paper is made up of 5 sections. Section 2 introduces the computation model, while Section 3 presents the condition-based approach. Then, Section 4 proposes a generic formula to derive conditions, and shows that all its instances belong to \mathbb{C}_f . Section 5 presents an adaptive condition-based protocol and proves its correctness. This section shows also an interesting relation between the weights used to define a condition and a critical parameter used by the protocol. Due to space limitations, proof are omitted, they can be found in [11].

2 Computation Model

The computation model is a standard asynchronous shared-memory system with n ($n > 1$) processes, where at most f ($0 \leq f < n$) processes can crash. The shared memory consists of single-writer, multi-reader atomic registers. The executions are assumed to be linearizable. (For details of this model see any standard distributed computing textbook.)

The shared memory is organized into arrays. The j -th entry of such a shared array $X[1..n]$ can be read by any processes p_i with an operation $\text{read}(X[j])$. Only p_i can write to the i -th component, $X[i]$, it uses the operation $\text{write}(v, X[i])$ for this. In addition to the shared memory, each process has a local memory. The subindex i is used to denote p_i 's local variables.

To simplify the notation we also consider the following non-primitive, non-atomic **collect** operation which can be invoked by any process p_i . It can only be applied to a whole array $X[1..n]$, and is an abbreviation for $\forall j : \text{do read}(X[j])$ **enddo**. Hence, it returns an array of values $[a_1, \dots, a_n]$ such that a_j is the value returned by $\text{read}(X[j])$.

When we count the number of operations executed by a process we only count the number of read/write operations that access the shared memory. Local operations are not considered.

3 The Condition-Based Approach

3.1 The Approach

In the consensus problem there is a set V of values that can be *proposed* by the processes, $\perp \notin V$, and $|V| \geq 2$. In an execution, every correct process p_i proposes a value $v_i \in V$ and all correct processes have to *decide* on the same value v , that has to be one of the proposed values. The proposed values in an execution are represented as an *input vector*, such that the i -th entry contains the value proposed by p_i , or \perp if p_i did not take any step in the execution. We usually denote with I an input vector with all entries in V , and with J an input vector that may have some entries equal to \perp . If at most f processes can crash, we consider only input vectors J with at most f entries equal to \perp , called *views*. Let V^n be the set of all possible input vectors with all entries in V , and V_f^n the set of all possible vectors with at most f entries equal to \perp . For $I \in V^n$, let I_f be the set of possible views, i.e., the set of all input vectors J with at most f entries equal to \perp , and such that I agrees with J in all the non- \perp entries of J . 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 $J \in V_f^n$ is a possible input vector.

The *condition-based* approach consists of considering subsets C of V^n , called *conditions*, that represent common input vectors in a particular distributed application. We are interested in 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. More precisely, we say that an *f -fault tolerant protocol solves the consensus problem for a condition C* if in every execution whose input vector J is in V_f^n , the protocol satisfies the following properties:

- **P-Validity:** A decided value is one of the proposed values (consensus validity).
- **P-Agreement:** Two processes cannot decide different values (consensus agreement).
- **P-Best_Effort_Termination:** 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.

We use the following notations in the rest of the paper. For vectors $J1, J2 \in V_f^n$, $J1 \leq J2$ if $\forall k: J1[k] \neq \perp \Rightarrow J1[k] = J2[k]$. The expression $\#_x(J)$ denotes number of entries of J whose value is x , with $x \in V \cup \{\perp\}$.

3.2 Acceptability of a Condition

Given a condition C , a value of f and a degree d , acceptability (is a combinatorial property that) defines the constraints C has to satisfy in order the consensus problem can be solved for C . Operationally, it is defined in terms of a predicate P and a function S that have to satisfy some properties. (A combinatorial characterization of acceptability is given in [9, 10].) Those properties are related to the termination, validity and agreement of the protocol, respectively.

The intuition for the first property is the following. The predicate P allows a process p_i to test if a decision value can be computed from its view. Thus, P returns true at least for all those input vectors J such that $J \in I_f$ for $I \in C$.

- Property $T_{C \rightarrow P}$: $I \in C \Rightarrow \forall J \in I_f : P(J)$.

The second property is related to validity.

- Property $V_{P \rightarrow S}$: $\forall I \in V^n : \forall J \in I_f : P(J) \Rightarrow S(J) = \text{a non-}\perp \text{ value of } J$.

The next property concerns agreement. Given an input vector I , if two processes p_i and p_j get the views $J1$ and $J2$, and both belong to I_f such that $P(J1)$ and $P(J2)$ are satisfied, these processes have to decide the same value of V , from $J1$ for p_i and $J2$ for p_j , whenever the following holds (where $d \geq 0$ is the *degree* of the condition).

- Property $A_{P \rightarrow S}^{[d]}$: $\forall I \in V^n : \forall J1, J2 \in I_f \text{ s.t. } P(J1) \wedge P(J2)$:
 $((J1 \leq J2) \vee (\#_{\perp}(J1) + \#_{\perp}(J2) \leq f + d)) \Rightarrow S(J1) = S(J2)$.

Definition 1 A condition C is (f, d) -acceptable if there exist a predicate P and a function S satisfying the properties $T_{C \rightarrow P}$, $A_{P \rightarrow S}^{[d]}$ and $V_{P \rightarrow S}$ for f . An (f, d) -acceptable condition is denoted $C^{[d]}$. The class $\mathbb{C}_f^{[d]}$ consists of all the (f, d) -acceptable conditions.

3.3 Summary of Previous Results

A main result of [9] is the following theorem, which states that (given f) $\mathbb{C}_f = \mathbb{C}_f^{[0]}$ is the largest set of conditions that allows to solve the consensus problem.

Theorem 1 The consensus problem is solvable for C iff C is $(f, 0)$ -acceptable.

As indicated in the Introduction, the condition-based consensus protocol presented in [9] is of the most general form as it works with any condition $C \in \mathbb{C}_f$. As noted, the cost of its wait free part is $O(n \log_2(f + 1))$ read/write shared memory operations per process.

The theorem that follows (stated and proved in [10]) shows that \mathbb{C}_f is actually made up of a strict hierarchy of conditions.

Theorem 2 $\forall d \geq 0 : \dots \mathbb{C}_f^{[d+1]} \subset \mathbb{C}_f^{[d]} \subset \dots \subset \mathbb{C}_f^{[0]} = \mathbb{C}_f$.

The consensus protocol presented in [9] assumes a fixed d and works with any condition in $\mathbb{C}_f^{[d]}$. As noted previously, its cost is only $O(n \log_2(f - d + 1))$ read/write shared memory operations per process. The quantity $(f - d)$ actually measures the difficulty of the condition: The stronger the condition (i.e., when d increases from 0 to f), the more efficient the protocol (there is no additional gain when $d > f$). Hence, there is a tradeoff relating the degree of a condition and the cost of the associated consensus protocol.

4 A Generic Formulation of Conditions

Given d and f , this section introduces a generic formulation for conditions. It also proves that all the conditions $C^{[d]}$ derived from this formulation are (f, d) -acceptable, i.e., belong to $\mathbb{C}_f^{[d]}$. A few conditions derived from the generic formulation are exhibited.

4.1 Weight-Based Definition of Conditions

Let w be a function from V to \mathbb{R}^+ . It associates a positive weight $w(a)$ with each value a that can be proposed². In order not to introduce cumbersome notations in the following, an input vector is sometimes considered as the set of the values it contains.

The idea of the generic weight-based formulation of a condition C is to provide a simple way to distinguish a value that can be safely decided from a vector. Given a degree d , we define the condition $C^{[d]}$ to be the set of vectors I that satisfy:

$$\boxed{\exists a \in I, \forall b \in I \setminus \{a\} : (\#_a(I) w(a) - \#_b(I) w(b)) > (f + d) \max(w(a), w(b)) .}$$

The intuition that underlies this definition is the following. For a value a of an input vector I to be decided (i.e., for I to belong to the condition $C^{[d]}$), this value has to “bypass” any other value b present in I , (1) despite up to f process crashes, and (2) whatever the occurrence number of that “adversary” value b . This means that:

- The value a has to be “present enough” despite the previous situations. This is expressed by the following constraint: $(\#_a(I) - (f + d)) w(a) > \#_b(I) w(b)$.
 - The value a has to be “distant enough” from any b to be distinguishable. This is expressed by the following constraint: $\#_a(I) w(a) > (\#_b(I) + (f + d)) w(b)$.
- The weights are used to represent the respective “power” of each value of V . The generic definition is simply the combination of the two previous constraints involving the weights and the parameter $f + d$.

The theorem that follows shows that any condition defined from the previous generic weight-based expression defines an (f, d) -acceptable condition, i.e., belongs to $\mathbb{C}_f^{[d]}$.

Theorem 3 $\forall d \geq 0$, any function w from V to \mathbb{R}^+ defines a condition $C^{[d]}$ in $\mathbb{C}_f^{[d]}$. For the proof, see [11].

4.2 Examples of Conditions

We consider here three weight functions. They differ in the way they a priori favor values. Other weight functions can easily be defined. Each of these weight

²A value needs to have a positive weight to be decided. So, in the following, we consider only positive weights in order not to a priori discard values from being decided. If one is interested in preventing some values x to be decided, the corresponding weight $w(x)$ has to be set to 0.

functions defines a condition pattern C that combined with the degree d provides the following hierarchy of conditions: $\dots \subset C^{[d+1]} \subset C^{[d]} \dots \subset C^{[0]}$.

Favoring no value (Condition pattern C2) Let us first consider the *uniform weight* function: $\forall a \in V: w(a) = 1$. This function a priori favors no value. The generic expression simplifies and becomes:

$$(I \in C2^{[d]}) \equiv (\exists a \in I: \forall b \in I \setminus \{a\}: \#_a(I) - \#_b(I) > (f + d)).$$

As we can see, this is the condition pattern called C2 in [9, 10]. It favors the value that appears the most often in an input vector. To be decided, despite up to f crashes and the presence of any other value b , the most common value a has to appear $(f + d)$ times more than b . Hence, this weighting function defines the hierarchy of conditions: $\dots \subset C2^{[d+1]} \subset C2^{[d]} \dots \subset C2^{[0]}$.

$S2(J) =$ the most common value of J , and $P2^{[d]}(J) \equiv (\#_{1st}(J) - \#_{2nd}(J) > f + d - \#_{\perp}(J))$ constitute (f, d) -acceptability parameters for $C2^{[d]}$ ($\#_{1st}(J)$ and $\#_{2nd}(J)$ denote the occurrence number of the most common value and second most common value of a vector, respectively).

Slightly favoring a single value (Condition pattern C3) Let us associate the same weight (namely, 1) to all values but one, e.g., a whose weight is 2. This is a new condition, not investigated before, that we call C3. It very slightly favors the distinguished value a . Increasing the weight of a , would favor it more and more when it appears in an input vector.

Largely spacing out the favors (Condition pattern C1) Let $V = \{a_1, \dots, a_p\}$, and let us assume that these values are ranked: $a_1 < a_2 < \dots < a_{p-1} < a_p$. Moreover, let $w(a_i) = n^i$. This condition favors a_p (the largest value in V). Then, if a_p is not proposed it favors a_{p-1} (the second largest value in V), etc.

As all weights are powers of n , we can simplify the generic formula. Let a_i be the largest value that appears in a vector I , and a_j another value in I (hence $i > j$). The constraint part in the formula becomes: $\#_{a_i}(I) n^i - \#_{a_j}(I) n^j > (f + d) \max(n^i, n^j)$, which simplifies into $\#_{a_i}(I) - \#_{a_j}(I) (1/n^{i-j}) > (f + d)$. As $i - j > 1$ and $\#_{a_j}(I) < n$, we have $0 < \#_{a_j}(I) (1/n^{i-j}) < 1$. This allows to simplify once more and we get:

$$(I \in C1^{[d]}) \equiv (a_i = \max(I) \Rightarrow \#_{a_i}(I) > (f + d))$$

that is the condition pattern C1 investigated in [9, 10]: a vector belongs to the condition $C1^{[d]}$ if its greatest value appears more than $(f + d)$ times. $S1(J) =$ the largest value of J , and $P1^{[d]}(J) \equiv (a_i = \max(J) \Rightarrow \#_{a_i} > f + d - \#_{\perp}(J))$ constitute (f, d) -acceptability parameters for $C1^{[d]}$ (assuming \perp is smaller than any value of V).

Remark Interestingly, when $n > 3$, $|V| = 2$ and $f = 1$, we have $C1^{[0]} \cup C2^{[0]} = V^n$. This shows that the set of $(f, 0)$ -acceptable conditions is not closed under union (due to the FLP impossibility result, V^n is not acceptable).

5 An Efficient Condition-Based Protocol

5.1 Structure of the Protocol

The condition-based consensus protocols presented in [9, 10] are made of three parts. In their first part, a process builds a view of the input vector. This view includes at least $(n - f)$ non- \perp values. The second part is a wait-free part where each process tries to get a decision value without compromising the consensus safety requirement. The last part is the best effort part of the protocol: a process that has not decided looks for the value decided by another process (if any), or else waits until it knows all processes have deposited the value they propose in the shared memory.

As noticed previously, the cost of these protocols is measured by the number of shared memory accesses issued in their wait-free part: the protocol presented in [9] works with any condition $C \in \mathbb{C}_f = \mathbb{C}_f^{[0]}$ and costs $O(n \log_2(f + 1))$ read/write shared memory operations per process, while the protocol presented in [10] depends on d : it is tailored for the conditions of $\mathbb{C}_f^{[d]}$ and costs only $O(n \log_2(f - d + 1))$. Let us consider an input vector that belongs to $\mathbb{C}_f^{[d']}$. If $d' \geq d$, both protocols work but the second is more efficient. If $d' < d$, only the first protocol guarantees the termination property.

The aim of this section is the design of a protocol that, when $C \in \mathbb{C}_f$, always terminates, but whose wait-free part does not always cost $O(n \log_2(f + 1))$. This protocol (described in Figure 1) has the three-part structure just described. A process p_i starts executing the protocol by invoking the function *Consensus*(v_i) where v_i is the value it proposes. It terminates when it executes the statement **return** (at line 5, 7 or 10) which provides it with the decided value r . The three-parts of the protocol are:

- **Part 1** (lines 1-2): A process p_i first writes its input value v_i to the shared array V (initialized to $[\perp, \dots, \perp]$). Then p_i repeatedly reads V until at least $(n - f)$ processes (including itself) have written their input values in V , from which it constructs its initial view J_i , where $J_i[j]$ is the input value of p_j , or \perp if p_j has not yet written its input value.
- **Part 2** (lines 3-5): Now, p_i enters its wait-free condition-dependent protocol part. It uses the underlying **Decision.Chasing** function which is the core of the protocol from safety and efficiency point of views. This abstraction returns the decided value if p_i can decide by itself, or \top if it cannot. Whatever the output w_i provided by **Decision.Chasing**, p_i writes it into the shared variable $W[i]$ (line 4). The array W is initialized to $[\perp, \dots, \perp]$. If $w_i \neq \top$, p_i can decide by itself: this writing will help processes that cannot decide by themselves. If $w_i = \top$, this writing informs the other processes that p_i cannot decide by itself but has deposited its proposed value. Then, if p_i can decide, it does it (line 5). Otherwise it proceeds

to the best effort termination part.

• **Part 3** (lines 6-10): In this section, p_i enters a loop to look for a decision value (i.e., a value different from \perp, \top) provided by another process p_k in the shared variable $W[k]$. If, while waiting for a decision, p_i discovers that every process has written a value to W , and no process can directly decide (all these values are \top), p_i concludes that every process has deposited its initial value in the shared array V in line 1. Then, p_i reads V (line 10) to get the full input vector, and proceed to decide according to a fixed deterministic rule F (such as max).

```

Function Consensus( $v_i$ ) return( $r$ ):
(1) write( $v_i, V[i]$ );
(2) repeat  $J_i \leftarrow \text{collect}(V)$  until ( $\#_{\perp}(J_i) \leq f$ ) endrepeat;
(3)  $w_i \leftarrow \text{Decision\_Chasing}(J_i)$ ;
(4) write( $w_i, W[i]$ );
(5) case  $w_i \neq \top$  then return( $r = w_i$ )
(6)      $w_i = \top$  then repeat  $X_i \leftarrow \text{collect}(W)$ ;
(7)         if ( $\exists k : X_i[k] \neq \perp, \top$ ) then return( $r = X_i[k]$ ) endif
(8)         until ( $\perp \notin X_i$ ) endrepeat;
(9)         [ $a_1, \dots, a_n$ ]  $\leftarrow \text{collect}(V)$ ;
(10)        return( $r = F([a_1, \dots, a_n])$ )
(11) endcase

```

Figure 1: Efficient Consensus-Based Protocol

5.2 Adapting Attiya-Rachman's Synchronization Tree

The `Decision_Chasing` function used in Figure 1 relies on the traversal of a classifier/improver tree close to Attiya-Rachman's tree defined in [2]. The next subsection will show how and under which conditions the traversal of this adapted tree can be appropriately shortened (according to the information currently at the process's disposal) in order to save shared memory accesses. The aim of this binary tree is to allow the processes that call `Decision_Chasing` to improve their current view J_i and to classify them according to the relation " \leq " defined on views.

Structure and traversal of the classifier tree The tree structure is statically defined. It is a full binary tree with $\lceil \log_2 m \rceil$ levels and m leaves, where $m = \lceil f/2 \rceil + 1$ [2] (m is assumed to be a power of 2. Otherwise, the tree is not fully balanced). It is shared by all the processes; they access it from the pointer *root*. A process traverses the tree from the root downwards (as we will see, it can stop the traversal before a leaf). Hence, each non-leaf vertex v contains pointers to its children, namely, $v.left$ and $v.right$.

To allow the processes to classify their views, each vertex v of the tree is labeled with an integer interval: $[L(v), H(v)]$, and the labels are statically set to satisfy the following properties:

- $[L(\text{root}), H(\text{root})] = [n - f, n]$,
- $L(v.\text{left}) = L(v)$, $H(v.\text{left}) = L(v.\text{right}) - 1$, $H(v.\text{right}) = H(v)$ for any non-leaf vertex v .

Hence, the intervals associated with the leaves of the tree form a partition of the interval associated with the root. Such a labeled tree, with $f = 2^p - 2$ (for some p), is depicted in Figure 2. An interval is *trivial* if its size is 1.

Using the tree to improve the views From the “dynamic” point of view, each vertex v of the tree contains an array $v.R$ of n vectors. The vector $v.R[i]$ is a shared variable initialized to $[\perp, \dots, \perp]$ that can atomically be written by p_i (to store its current view) and read by any process. Let the size of a view J be $|J| = n - \#_{\perp}(J)$ (number of positions of J with a non- \perp value).

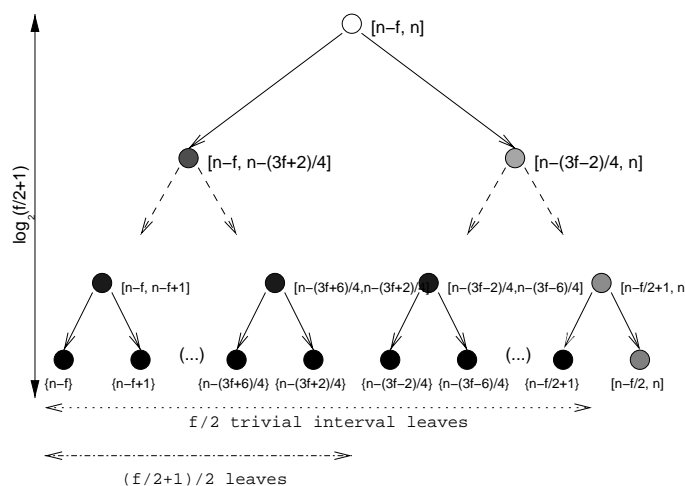


Figure 2: A Classifier/Improver Tree

The basic step of the traversal of the tree by a process p_i is described in Figure 3. As indicated, a process traverses it from the root downwards. When it visits the vertex v , p_i first deposits its current view in $v.R[i]$ (line 1). Then, p_i reads the other views $v.R[j]$ already deposited in $v.R$ by other processes p_j (line 2). According to what it has read, it progresses (classifies) to the right son or the left son of v . The notation $\cup \{J_1, \dots, J_n\}$ (used at lines 3 and 5), for views $J_i \in I_f$ denotes the view J that has a non- \perp value $v = J[j]$ in a position j if at least for one i , $J_i[j] = v$. If p_i knows more than $H(v.\text{left})$ entries different from \perp , it adopts the current content of $v.R$ as its current view and progresses to the right son of v (lines 4-5). In the other case, it keeps its previous view J_i and progresses to the left son of v (line 6).

```

Function  $v$ .Classifier_Improver( $J_i$ ) return ( $J'_i, dir_i$ ):
(1) write( $J_i, v.R[i]$ );
(2)  $R_i \leftarrow \text{collect}(v.R)$ ;
(3) if  $|\cup\{R_i[1], \dots, R_i[n]\}| > H(v.left)$ 
(4)   then  $R_i \leftarrow \text{collect}(v.R)$ ;
(5)     return ( $J'_i = \cup\{R_i[1], \dots, R_i[n]\}, dir_i = right$ )
(6)   else return ( $J'_i = J_i, dir_i = left$ )
(7) endif

```

Figure 3: An Implementation of Classifier_Improver

Properties associated with the tree Let us denote $p_i \in \text{visited}(v)$ the execution (if any) of v .Classifier_Improver by p_i . Moreover, let $J_{i,v}$ denote the actual value of the corresponding input parameter, while $dir_{i,v}$ denotes the value returned in the output parameter dir_i (i.e., at the end of the visit of v by p_i). The lemma that follows, proved in [10], states the main properties of the tree traversal.

Lemma 1 *Let us assume that the initial input view J_i of every p_i that calls Decision_Chasing belongs to I_f for some input vector I . For every vertex v and every process p_i we have the following.*

- (1) $L(v) \leq |J_{i,v}| \leq H(v)$ when $p_i \in \text{visited}(v)$,
- (2) $|\cup_{p_i \in \text{visited}(v)} \{J_{i,v}\}| \leq H(v)$,
- (3) $J_{i,v} < J_{j,u}$ whenever $H(v) < L(u) \wedge p_i \in \text{visited}(v) \wedge p_j \in \text{visited}(u)$,
- (4) $J_{i,v} = J_{j,v}$ whenever $L(v) = H(v) \wedge p_i, p_j \in \text{visited}(v)$.
- (5) *The cost of v .Classifier_Improver for a process is upper bounded by $(2n + 1)$ shared memory read/write operations³.*

5.3 Shortening the Tree Traversal

In the protocol presented in [9] a process traverses the previous tree once, from the root to a leaf. Hence, its cost (number of shared memory operations) lies between $(n + 1)D$ and $(2n + 1)D$, where D is the depth of the tree, namely, $D = \lceil \log_2(\lceil f/2 \rceil + 1) \rceil$. In the protocol tailored for the stronger conditions of $\mathbb{C}_f^{[d]}$ [10], a process also traverses the tree from the root to a leaf, but here the tree has a shorter depth, namely, $D' = \lceil \log_2(\lceil (f - d)/2 \rceil + 1) \rceil$.

The idea that underlies the design of the Decision_Chasing function described in Figure 4 is to allow a process to stop its tree traversal before having

³Those are the accesses to the array R associated with the vertex v : one write, one collect, plus possibly one more collect.

joined a leaf. This will obviously decrease the length of the path traversed by the process and consequently allow it to decide early thereby reducing the cost of its wait-free part. The difficulty in attaining this goal consists in ensuring that, whatever the vertices at which two processes p_i and p_j stop their tree traversal, they will decide the same non- \perp value: the consensus agreement property must be guaranteed whatever parts of the tree are traversed by processes. Moreover, given a condition $C \in \mathbb{C}_f$, the protocol must terminate at least when $I \in C$.

Underlying idea of the protocol Let $C \in \mathbb{C}_f$. As we have seen, C gives rise to a hierarchy of conditions $\dots \subset C^{[d+1]} \subset C^{[d]} \subset \dots \subset C^{[0]} = C$. Let $P^{[d]}$ be a predicate associated with $C^{[d]}$.

When a process p_i visits a vertex v and gets J as current view, the idea is to allow it to try to benefit from the fact that J can belong to some I'_f such that $I' \in C^{[d]}$, in order to stop its tree traversal. Only in the worst case a process has to traverse the tree from the root to a leaf. In order the agreement property be never violated by processes stopping at arbitrary vertices, the protocol requires that the predicates $P^{[d]}$ satisfy monotonicity properties.

Properties on the sequence of predicates When it visits a vertex v , a process has first to compute a value d to test if $P^{[d]}(J)$ holds (J being its current view). The determination of d is crucial for the protocol correctness: d has to be “as small as possible” to allow early tree exit, but large enough to guarantee agreement.

Let us consider two processes that exist the tree at v_1 and v_2 with J_1 and J_2 , respectively. This means that both $P^{[d_1]}(J_1)$ and $P^{[d_2]}(J_2)$ hold. To ensure that we also have $S(J_1) = S(J_2)$, the protocol requires the following properties be satisfied:

- Decreasing Degree Monotonicity:

$$M_{dd} \equiv (\forall d > 0, \forall J \in V_f^n : P^{[d]}(J) \Rightarrow P^{[d-1]}(J)).$$
- Increasing Vector Monotonicity:

$$M_{iv} \equiv \forall d \geq 0, \forall J \in V_f^n, \forall d' \leq d, \forall J' \geq J : \\ (P^{[d]}(J) \wedge \text{dist}(J, J') \leq \frac{d-d'}{\alpha(J)}) \Rightarrow P^{[d']}(J').$$

Let J be a view such that $P^{[d]}(J)$ holds. This means that $\exists I \in C^{[d]}$ with $J \in I_f$. As $C^{[d]} \subset C^{[d-1]}$, it follows that $I \in C^{[d-1]}$. Hence $P^{[d-1]}(J)$ holds, and the sequence of predicates $P^{[d]}$ of any condition pattern satisfies M_{dd} .

Differently from M_{dd} , the property M_{iv} involves a parameter α for each J . It follows that for the sequence of predicates $P^{[d]}$ to satisfy M_{iv} , the value $\alpha(J)$ associated with each view J actually depends on the condition pattern (this issue is addressed in Section 5.4). The intuition that underlies the M_{iv} property is the following. For each J , there is a “disc of augmented vectors” J' , centered at J and with radius $\frac{d-d'}{\alpha(J)}$. The property M_{iv} requires that all those augmented vectors satisfy $P^{[d']}$. As we start from d and J , and proceed to $d' \leq d$ and $J' \geq J$, M_{iv} states

a tradeoff relating how much an increase in the number of non- \perp values in a view (from J to J') affects a degree reduction (from d to d') when we want to have both $P^{[d]}(J)$ and $P^{[d']}(J')$.

Let p_i and p_j be two processes with current views J and J' , respectively. Let us consider the extreme case where $d' = 0$ and $d = \alpha(J) \text{ dist}(J, J')$. If $P^{[d]}(J)$ holds (allowing p_i to early decide), the property M_{iv} allows p_j to also decide. So, when p_i is visiting the vertex v and has J as current view, as it does not know the view J' , it has to use some value to consistently approximate $\text{dist}(J, J')$. To that aim, it considers the value $\#_{\perp}(J) + n - L(v) - f$. As d cannot be negative, it actually uses $\max(0, \alpha(J) \times (\#_{\perp}(J) + n - L(v) - f))$ to define the value of d it uses to compute $P^{[d]}(J)$. The proof (see [12]) will show that this heuristic value is consistent.

The core of the protocol The Decision_Chasing function (Figure 4) assumes that the sequence of predicates $P^{[d]}$ associated with the condition C satisfies M_{dd} and M_{iv} .

```

Function Decision_Chasing( $J_i$ ) return ( $w_i$ ):
(1)  $current_i \leftarrow J_i$ ;  $v \leftarrow root$ ;  $prev\_d\_val_i \leftarrow +\infty$ ;  $prev\_nb\_of\_bot_i \leftarrow (f + 1)$ ;
(2) while  $v$  is not a leaf do
(3)   ( $current_i, direction_i$ )  $\leftarrow v$ .Classifier_Improver( $current_i$ );
(4)   let  $d = \max(0, \alpha(current_i) \times (\#_{\perp}(current_i) + n - L(v) - f))$ ;
(5)   if ( $prev\_d\_val_i \neq d$ )  $\vee$  ( $prev\_nb\_of\_bot_i \neq \#_{\perp}(current_i)$ )
(6)     then if  $P^{[d]}(current_i)$  then return ( $w_i = S(current_i)$ ) endif;
(7)      $prev\_d\_val_i \leftarrow d$ ;  $prev\_nb\_of\_bot_i \leftarrow \#_{\perp}(current_i)$ 
(8)   endif;
(9)   if  $direction_i = right$  then  $v \leftarrow v.right$  else  $v \leftarrow v.left$  endif
(10) endwhile;
(11) if  $P^{[0]}(current_i)$  then return ( $w_i = S(current_i)$ ) else return ( $w_i = \top$ ) endif

```

Figure 4: The Decision_Chasing Function

A process p_i executing Decision_Chasing first initializes local variables (line 1): $current_i$ represents its current view, v the vertex it is about to visit (initially $root$), $prev_d_val_i$ the previous value of the parameter d , and $prev_nb_of_bot_i$ the occurrence number of the \perp value in its previous view.

Then p_i enters the tree traversal (lines 2-10). It first visits the vertex v (line 3), computes the current values of $\alpha(current_i)$ and d from its context (current values of $current_i$ and v). If something has changed (line 5) with respect to the previous predicate evaluation (either the value of d or $\#_{\perp}(current_i)$), then p_i tests $P^{[d]}(current_i)$, and if true, stops the tree traversal and considers the value $S(current_i)$ (line 6). Otherwise (line 9), p_i updates the relevant local loop variables, and progresses to the next level of the tree in the direction indicated by the previous call to Classifier_Improver.

Finally, if a process has not stopped during the tree traversal (it has joined a leaf), we are in the case where there is no possibility to have an early decision.

So, p_i checks $P^{[0]}(\text{current}_i)$ to see if it can decide. If it cannot, it sets w_i to \perp .

Theorem 4 *The Consensus protocol described in Figure 1 satisfies the P-Validity, P-Agreement and P-Best_Effort_Termination properties described in Section 3.1. Moreover, the number of shared memory accesses of its wait-free part is upper bounded by $O(n \log_2(f + 1))$ for each process. (Proof: see [11].)*

Example Let us consider the following example to illustrate the early decision possibility offered by the Decision_Chasing function. Let $V = \{a, b, \dots\}$, $n = 6$, $f = 1$. The condition considered is $C2$ defined by the uniform weight function (Section 4.2). This condition provides $\alpha(J) = 2, \forall J \in V_f^n$ (See below).

Let us consider the input vector $I = [a, a, a, a, b, b]$. As $\#_a(I) - \#_b(I) > f$, it follows that $I \in C2^{[0]}$. Moreover, it is easy to see that $I \notin C2^{[d]}$ for $d > 0$.

Let $\text{current}_i = [a, a, a, a, \perp, b]$ be the view of p_i after its first call to Classifier_Improver at line 3. As $v = \text{root}$ and $L(\text{root}) = n - f$, we get $d = 2$ at line 4. Hence p_i evaluates $P^{[d]}([a, a, a, a, \perp, b]) = (\#_a(\text{current}_i) - \#_b(\text{current}_i) > f + d - \#_{\perp}(\text{current}_i))$ which is true. Consequently, p_i stops its tree traversal and exits with the value a (line 6). The fact that $P^{[2]}([a, a, a, a, \perp, b])$ holds means that there is some $I' \in C2^{[2]}$ such that $J \in I'_f$ (e.g., $[a, a, a, a, a, b]$ is such an I').

5.4 From a Condition to an α Function

This section defines an α function that allows predicates P associated with a condition obtained from a weight function (Section 4.2) to satisfy M_{iv} , i.e., $\forall d \geq 0, \forall J \in V_f^n, \forall J' \geq J: (P^{[d]}(J) \wedge d' \leq d - \alpha(J) \text{ dist}(J, J')) \Rightarrow P^{[d']}(J')$.

Theorem 5 *Let C be a condition defined from a weight function w , with the following acceptability parameters (defined in Theorem 3):*

$$\begin{aligned} P^{[d]}(J) &\equiv \exists a \in J, \forall b \neq a \in J, \\ &\quad (\#_a(J) + \#_{\perp}(J))w(a) - \#_b(J)w(b) > (f + d) \max(w(a), w(b)); \\ S(J) &= a \text{ such that } \#_a(J)w(a) = \max\{\#_b(J)w(b) : b \in J\}. \end{aligned}$$

Let $J \in V_f^n$, $a = S(J)$, and M be the maximal weight associated with a value of V . Let α be a function from V_f^n to \mathbb{R}^+ defined as follows:

- $\alpha(J) = \max_{b \in J, b \neq a} \left(\frac{w(a) + w(b)}{\max(w(a), w(b))} \right)$, if $\exists x \in J: w(x) = M$.
- $\alpha(J) = (n - f)$, otherwise.

The sequence of predicates $P^{[d]}$ satisfies the monotonicity property M_{iv} with this function α . (Proof: see [11].)

Table 1 defines the α functions associated with the conditions $C1$, $C2$ and $C3$ defined in Section 4.2 (M denotes the maximal weight). As we can see, every condition provides an $\alpha(J)$ that is always ≥ 1 .

Condition	$\exists x \in J : w(x) = M$	$\forall x \in J : w(x) < M$
C1	$\alpha(J) = 1 + 1/n$	$\alpha(J) = (n - f)$
C2	$\alpha(J) = 2$	Cannot appear
C3	$\alpha(J) = 2$ or 1.5 (acc. to J)	$\alpha(J) = (n - f)$

Table 1: Values of $\alpha(J)$ for the three Conditions

References

- [1] Afek Y., Attiya H., Dolev D., Gafni E., Merritt M. and Shavit N., Atomic Snapshots of Shared Memory. *JACM*, 40(4):873-890, 1993.
- [2] Attiya H. and Rachman O., Atomic Snapshots in $O(n \log n)$ Operations. *SIAM Journal of Computing*, 27(2):319-340, 1998.
- [3] Ben-Or M., Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols. *Proc. 2nd ACM Symposium on Principles of Distributed Computing (PODC'83)*, ACM Press, pp. 27-30, Montréal (Canada), 1983.
- [4] Chandra T. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *JACM*, 43(2):225-267, 1996.
- [5] Chaudhuri S., More Choices Allow More Faults: Set Consensus Problems in Totally Asynchronous Systems. *Information and Computation*, 105:132-158, 1993.
- [6] Dolev D., Lynch N. A., Pinter S. S., Stark E. W., and Weihl W. E., Reaching Approximate Agreement in the Presence of Faults. *JACM*, 33(3):499-516, 1986.
- [7] Dwork C., Lynch N. and Stockmeyer L., Consensus in the Presence of Partial Synchrony. *JACM*, 35(2):288-323, 1988.
- [8] Fischer M.J., Lynch N.A. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *JACM*, 32(2):374-382, 1985.
- [9] Mostéfaoui A., Rajsbaum S. and Raynal M., Conditions on Input Vectors for Consensus Solvability in Asynchronous Distributed Systems. *Proc. 33rd ACM Symposium on Theory of Computing (STOC'01)*, ACM Press, Crete (Greece), July 2001.
- [10] Mostéfaoui A., Rajsbaum S., Raynal M. and Roy M., A Hierarchy of Conditions for Consensus Solvability. *Proc. 20th ACM Symposium on Principles of Distributed Computing (PODC'01)*, ACM Press, Newport (RI), August 2001.
- [11] Mostéfaoui A., Rajsbaum S., Raynal M. and Roy M., Efficient Condition-Based Consensus. *Research Report #1397*, IRISA, University of Rennes, France, April 2001, 22 pages. <http://www.irisa.fr/bibli/publi/pi/2001/1397/1397.html>.
- [12] Mostéfaoui A., Rajsbaum S., Raynal M. and Roy M., Condition-based Protocols for Set Agreement Problems. *Research Report #1393*, IRISA, University of Rennes, France, April 2001, 21 p. <http://www.irisa.fr/bibli/publi/pi/2001/1393/1393.html>.

Achour Mostéfaoui is Assistant Professor at the Computer Science Department of the University of Rennes, Campus de Beaulieu, 35042 Rennes, France. E-mail: achour@irisa.fr

Sergio Rajsbaum is with the Cambridge Research Lab of Compaq, One Kendall Square Build. 700, Cambridge, MA 02139, and Professor at the Instituto de Matemáticas, UNAM, Ciudad Universitaria, D.F. 04510, Mexico. E-mail: rajsbaum@compaq.com

Michel Raynal is Professor at the Computer Science Department of the University of Rennes, Campus de Beaulieu, 35042 Rennes, France. E-mail: raynal@irisa.fr

Matthieu Roy is a PhD student at the Computer Science Department of the University of Rennes, Campus de Beaulieu, 35042 Rennes, France. E-mail: mroy@irisa.fr

