One for All and All for One: Scalable Consensus in a Hybrid Communication Model

Michel Raynal[⋆,†], Jiannong Cao[†]

[⋆]Univ Rennes IRISA, 35042 Rennes, France
[†]Department of Computing, Polytechnic University, Hong Kong

Abstract—**This paper addresses consensus in an asynchronous model where the processes are partitioned into clusters. Inside each cluster, processes can communicate through a shared memory, which favors efficiency. Moreover, any pair of processes can also communicate through a message-passing communication system, which favors scalability. In such a "hybrid communication" context, the paper presents two simple binary consensus algorithms (one based on local coins, the other one based on a common coin). These algorithms are straightforward extensions of existing message-passing randomized round-based consensus algorithms. At each round, the processes of each cluster first agree on the same value (using an underlying shared memory consensus algorithm), and then use a message-passing algorithm to converge on the same decided value. The algorithms are such that, if all except one processes of a cluster crash, the surviving process acts as if all the processes of its cluster were alive (hence the motto "one for all and all for one"). As a consequence, the hybrid communication model allows us to obtain simple, efficient, and scalable fault-tolerant consensus algorithms. As an important side effect, according to the size of each cluster, consensus can be obtained even if a majority of processes crash.**

Keywords: Asynchronous system, Atomic register, Cluster, Binary consensus, Common coin, Compare and swap, Local coin, Hybrid communication, Message-passing, Modularity, Process crash failure, Scalability.

I. Introduction

Consensus in asynchronous systems: Consensus is one of the most fundamental problems of fault-tolerant asynchronous distributed computing. Assuming each process proposes a value, it consists in designing an algorithm such that all the processes that do not crash decide a value (termination), no two processes decide different values (agreement), and the decided value is a proposed value (validity).

Albeit its statement is very simple, it is impossible to solve consensus in the presence of asynchrony and even a single process crash, be the underlying communication medium message-passing [9], or read/write shared memory [18]. This means that the underlying systems must be enriched (from a computability point of view) for consensus to be solved. In crash-prone shared memory systems, it has been shown that enriching the read/write system with synchronization operations such as compare&swap(), fetch&add(), or LL/SC allows consensus to be solved for any number of processes, despite asynchrony and the crash of all except one process. More generally, the synchronization operations that allow consensus to be solved can be ranked according to their "consensus number", that is the maximal number of processes for which they can solve consensus despite any number of crashes (the previous operations have an infinite consensus number). This constitutes the famous consensus hierarchy introduced by M. Herlihy [14]. Consensus algorithms for enriched shared memory systems are described in several textbooks (e.g., [3], [21], [23]).

In crash-prone asynchronous message-passing systems the situation is different. No new operation can be provided by the network, which would allow consensus to be solved. Hence, the computability enrichment of an asynchronous crash-prone message-passing system must be provided another way. One consists in adding synchrony assumptions [7], [8]. Another one consists in providing processes with information on failures, this is the *failure detector*-based approach [6]. Another one consists in restricting the set of input vectors that can be proposed by the processes, this is the *condition*-based approach [19]. One of the very first approaches that was proposed (the one considered in this paper) consists in enriching the system with the power of *randomization* [4], [20]. In this case, the processes are allowed to draw random number in order to circumvent the non-determinism generated by the net effect of asynchrony and process crashes. These consensus algorithms are no longer deterministic. They are Las Vegas algorithms which ensure that the processes that do not crash decide with probability 1. Consensus algorithms with such computability enrichments are described in several textbooks (e.g., [3], [5], [22]). They all assume that a majority of processes do not crash, which is a necessary requirement for solving consensus in crash-prone asynchronous message-passing systems.

Content of the paper: This article considers an asynchronous crash-prone system in which the processes communicate through both shared memory and messages. More precisely, the processes are partitioned into clusters and each cluster provides its processes with a shared memory. Moreover, a communication system allows any process to send message to any process. Hence, processes in different clusters can communicate only by message-passing. This hybrid communication model was introduced in [16].

As advocated in [1], communication based on both shared memory and message-passing can be leveraged to design distributed algorithms that are both efficient and scalable. More precisely, on the one side, shared memory consensus algorithms can be efficient and tolerate any number of failures, but they do not scale due to memory hardware constraints. Differently, on the other side, message-passing consensus algorithms scale, but are less efficient due to message asynchrony, and work only if a majority of processes do not crash. This suggests a possible tradeoff between scalability and fault-tolerance.

Considering the previous two-dimension communication model, this article presents two simple randomized consensus algorithms (one based on local coins, the other one based on a common coin). These algorithms are simple compositions of existing round-based consensus algorithms (one is Ben-Or's randomized consensus algorithm [4], the other one is a simple adaptation of a Byzantine randomized consensus algorithm introduced in [10]). At every round, inside each cluster the processes first agree on the same value (using an efficient underlying shared memory consensus algorithm), and then use a cluster-independent message-passing algorithm involving all processes to try to converge on the same decided value. The algorithms are such that, if processes of a cluster crash, the surviving processes of this cluster act as if all the processes of this cluster were alive. The fact this is true even if a single process of a cluster does not crash, explains the motto "One for All and All for One"[1]). It follows that, when considering the number and the size of the clusters of the hybrid communication model, consensus can be obtained even if a majority of processes crash. As a simple example, let us consider the case where there is a cluster including a majority of processes. In the failure patterns where any number of processes crash, except one process belonging to the majority cluster, consensus can be solved. Let us additionally notice that, on a distributed software engineering point of view, the hybrid communication model favors a modular decomposition of distributed algorithms.

Related work: A two-dimension communication model, different from the previous cluster-based hybrid communication model, has recently been investigated in [1]. According to its authors, their model (called m&m model) is motivated by emerging technologies such Remote Direct Memory Access (RDMA) or disagregated memory [17]. The shared memories are defined from a communication graph, such that each process share a memory with all its neighbors. Hence, this amounts to have a shared memory per process, which can be accessed directly by this process and remotely by all its neighbors (only). It is easy to see that this two-dimension communication model is different from our hybrid model.

[1]This motto is the translation of the Latin expression *Unus pro omnibus, omnes pro uno*, which has been made famous in A. Dumas's novel "The Three Musketeers" (1844).

Roadmap: The article is composed of five sections. Section II introduces the hybrid communication-based computing model and a few coin-related definitions. Then two randomized consensus algorithms suited for this model are described. In the first one (Section III) each process uses a local coin, while the second one (Section IV) considers the processes share a common coin. As already said, these algorithms are straightforward extensions of existing randomized round-based consensus algorithms (the first one extends the consensus algorithm introduced in [4], while the second one extends a simplified version of a Byzantine consensus introduced in [10]). Section V concludes the paper.

II. Distributed Computing Model and Definitions

A. Process and Communication Model

Process model: The system is made up of a set Π of n processes denoted p_1, ..., p_n. In the notation "p_i", the integer i is called the index of p_i. Each process is sequential (which means it executes one step at a time), and asynchronous (which means it progresses at its own speed, which can vary with time and remains always unknown to the other processes).

A process can crash. A crash is a premature halt (after it crashed, if ever it does, a process executes no more steps).

Clusters: The n processes are partitioned into m, $1 \leq m \leq n$, non-empty subsets $P[1], \ldots, P[m]$ called clusters (i.e., $\cup_{1 \leq x \leq m} P[x] = \Pi$ and $\forall x, y : (x \neq y) \Rightarrow (P[x] \cap P[y] = \emptyset)$).

A process knows the number m of clusters, and the set of processes composing each cluster. When invoked by a process, the function $\mathsf{cluster}(i)$ returns the set of processes composing the cluster to which p_i belongs.

A shared memory MEM_x, made up of atomic registers, is associated with each cluster $P[x]$. Hence, the processes of $P[x]$, and only them, can communicate through MEM_x. Two examples of cluster-based decomposition are described in Fig. 1. These figures show two different cluster-based decompositions of $n = 7$ processes into in $m = 3$ clusters.

Memory operations: In addition to the basic read and write operations, the shared memory MEM_x of each cluster $P[x]$ is enriched with a synchronization operation whose consensus number is $+\infty$, e.g., $\mathsf{compare\&swap}()$. It follows that consensus can be solved by a deterministic

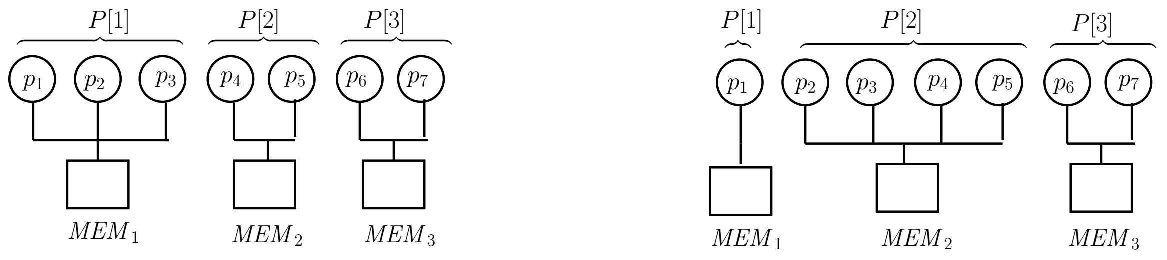

Figure 1. Two examples of cluster-based decomposition

algorithm within each cluster [3], [14], [15], [21], [23]. It is consequently assumed that each cluster provides its processes with cluster-limited consensus objects.

Message-passing communication: Processes can send and receive messages through channels. It is assumed that any pair of processes is connected by a bidirectional channel. Channels are reliable but asynchronous. Reliable means that messages are neither corrupted, nor duplicated, nor lost. Asynchronous means that, albeit finite, the transit duration of a message is arbitrary.

The sending and the reception of a message are atomic steps. The processes can also use a broadcast macro-operation, denoted broadcast (msg) where msg is a message, which is a shortcut for "**for each** $j \in \{1, ...n\}$ **do** send (msg) to p_j **end for**". Let us observe that this macro-operation is not reliable, namely, if the sender crashes while executing it, an arbitrary subset of processes (possibly empty) receive the message.

Extreme configurations: If $m = 1$ there is a single cluster, and the model boils down to the classical shared memory model. The message-passing facility becomes then useless.

If $m = n$ there is a cluster per process, and consequently the cluster shared memory dimension disappears. The model is then the classical message-passing model.

B. Local and Common Coins

Local coin: A local coin (LC) provides a process p_i with a function, denoted local_coin(), which returns the value 0 or 1, with probability 0.5.

The important points are here that no value is returned with probability 0 and the fact that the local coins of any two distinct processes are independent.

Common coin: A common coin (CC) is global function, denoted common_coin(), that delivers the same sequence of random bits b_1, b_2, ..., b_r, etc., to each process p_i, each bit b_r having the value 0 or 1 with probability 0.5. This means that the rth invocation of common_coin() by a process p_i and the rth invocation of common_coin() by a process p_j return them the very same bit. The construction of a distributed common coin is addressed in several textbooks (e.g., [3], [5], [22]).

III. An Algorithm Based on Local Coins

This section presents a round-based algorithm which implements binary consensus in the previous hybrid communication model. This algorithm can be seen as the composition of a message communication pattern (inspired from [1]), and a randomized message-passing consensus algorithm introduced by Ben-Or [4].

A. Communication Pattern

This pattern, called msg_exchange (), is an all-to-all communication pattern. It is defined by Algorithm 1, where r a round number, ph a phase number (a round is composed of two phases), and est a value in $\{0, 1, \perp\}$. 0 (resp. 1) means the invoking process supports 0 (resp. 1), while $\perp$ means it supports no value. The aim of this pattern is to provide the processes with a "weak" agreement on the value each of them selects just after it exits the pattern.

```
operation msg_exchange $(r, ph, est)$ is
(1) if $(ph = 1)$ then $(a, b) = (0, 1)$ else $(a, b) = (0$ or $1, \perp)$ end if;
(2) $supporters_i[a] \leftarrow \emptyset$; $supporters_i[b] \leftarrow \emptyset$;
(3) broadcast $(r, ph, est)$;
(4) repeat wait(messages carrying $(r, ph, -)$);
(5)         let $p_j$ be the sender of the msg and $v$ the value it carries;
(6)         $supporters_i[v] \leftarrow supporters_i[v] \cup$ cluster$(j)$
(7) until $|supporters_i[a] \cup |supporters_i[b]| > n/2$ end repeat;
(8) return().
```

Algorithm 1: Communication pattern

When a process p_i invokes msg_exchange (r, ph, est), it first initializes the two process sets $supporters_i[a]$ and $supporters_i[b]$ to $\emptyset$, where a and b depends on the phase ph of the current round r executed by p_i (lines 1-2). If $ph = 1$ (first phase of round r), a is 0 while b is 1. If $ph = 2$ (second phase of round r), a is either 0 or 1, while b is $\perp$. In this case, the value of a is dynamically defined, according to the values carried by the messages received by p_i. Then, p_i broadcasts a message carrying the triple (r, ph, est) (line3), and waits until it has received messages carrying triples $(r, ph, -)$ from "enough" processes (lines 4-7).

The aim of $supporters_i[v]$ is to contain the indexes of all the processes that support the value v. Those are the following processes (lines 5-6): if p_i receives a message (r, ph, v) from a process $p_j \in P[x]$, it is like if it received the very same message (r, ph, v) from all the processes in the cluster $P[x]$, despite the fact that some of them possibly crashed before sending such a message. This is due to the fact that, as we will see in the next section, the non-crashed processes of a cluster $P[x]$ cannot broadcast different messages such as $(r, ph, v1)$ and $(r, ph, v2)$, where $v1 \neq v2$. Hence, "One for All and All for One" inside each cluster.

Finally, when p_i has received messages carrying $(r, ph, -)$ from a set of processes $p_{i_1}, \cdots, p_{i_k}$ such that $\mathsf{cluster}(i1) \cup \cdots \cup \mathsf{cluster}(i_k)$ contains a majority of processes (line 7), it returns from the communication pattern (line 8).

B. Local Coin-Based Scalable Consensus Algorithm

This section presents Algorithm 2, which is a scalable local coin-based consensus algorithm suited to the hybrid communication model.

Variables shared inside a cluster: The shared memory MEM_x of a cluster $P[x]$ is composed of two arrays of consensus objects, denoted $CONS_x[r, 1]$ and $CONS_x[r, 2]$, where $r \geq 1$ is a round number.

As previously said, a round is made up of two phases. $CONS_x[r, 1]$ is used by the processes of the cluster $P[x]$ to agree (at the cluster level) on a common estimate of the decision. $CONS_x[r, 2]$ is used by the processes of $P[x]$ to agree on the same proposal (a proposed value or the default value $\perp$) used in the second phase of round r.

Local variables: Each process p_i manages four local variables.

- r_i: current round number executed by p_i.
- $est1_i$: current estimate of the decision value at the beginning of the round.
- $est2_i$: value championed by p_i to decide or default value $\perp$.
- rec_i set of (at most two) values received by p_i during the second phase of a round (moreover, if rec_i contains two values, one of them is 0 or 1, while the other is $\perp$).

Process behavior: initialization: When a process p_i invokes the consensus operation $\mathsf{propose}(v_i)$, where $v_i \in \{0, 1\}$ is the value it proposes, it first initializes $est1_i$ to v_i, and enters the sequence of asynchronous rounds.

Process behavior: phase 1 of round r: A process p_i invokes first the consensus object associated with the first phase of the current round r, namely $CONS_x[r, 1]$ (line 4). It follows that no two processes of a same cluster $P[x]$ can decide different values from $CONS_x[r, 1]$. Then, as the non-crashed processes of its cluster, p_i invokes the consensus pattern $\mathsf{msg\_exchange}$ $(r, 1, est1)$, where $est1$ is the value decided by $CONS_x[r, 1]$. When it exits the communication pattern, p_i appears as if it received messages from a majority of processes (let us recall that, due to the consensus object $CONS_y[r, 1]$, if it received a message carrying $(r, 1, est)$ from a process $p_j \in P[y]$, p_i considers it received the same message from all the processes in $P[y]$). If p_i sees a value $v \in \{0, 1\}$ supported by a majority of processes (i.e., $|supporters_i[v]| > n/2$), it adopts it in $est2_i$ (line 7). If there is no such v, it adopts $\perp$, whose meaning is "p_i has not enough information to champion a value". At the end of phase 1 of round r we have the following weak agreement:

$$WA1 \overset{\text{def}}{=}$$

$$\big((est2_i \neq \perp) \wedge\ (est2_j \neq \perp)\big) \ \Rightarrow\ (est2_i = est2_j = v).$$

This agreement follows from a simple observation. If $est_i = v$ and $est_j = v'$, each of v and v' is supported by a majority of processes. As all the processes of a cluster support the same value, and any two majority intersect, we necessarily have $v = v'$.

Process behavior: phase 2 of round r: The first part of this phase is the same as in phase 1. Namely, inside each cluster, the processes agree on the same $est2$ value (line 8), and then invoke the communication pattern $\mathsf{msg\_exchange}$ $(r, 2, est2_i)$ (line 9). Due to $WA1$, we can have at a process p_i only $rec_i = \{v\}$, $rec_i = \{v, \perp\}$, or $rec_i = \{\perp\}$. Let us notice that, if $rec_i = \{v\}$, p_i is such that $|supported_i[v]| > n/2$. And similarly for $rec_i = \{\perp\}$. Due to the intersection property of any two majorities, we have the following weak agreement at any round r:

$$WA2 \overset{\text{def}}{=}$$

$$\big((rec_i = \{v\}) \wedge\ (rec_j = \{\perp\})\big) \text{ are mutually exclusive.}$$

Then, there are three cases. If p_i sees a single value v, it decides it. To prevents possible deadlocks (due to the fact that no messages are sent from a cluster whose processes have decided or crashed), before deciding a value, a process is directed to broadcast the value it is about to decide (lines 12 and 17). If p_i sees a value v and $\perp$, it adopts v as its new estimate $est1_i$ (this is to ensure it will not decide another value in a future round, line 13). Finally, if it sees only $\perp$, no value was decided, and both values 0 and 1 were proposed. In this case, in order to break the non-determinism, p_i invokes $\mathsf{local\_coin}()$ and assigns the returned value to $est1_i$.

Algorithm 2 as an extension of Ben-Or's algorithm: If each cluster contains a single process, the system is a pure message-passing system. Consequently, the consensus objects used in each cluster (lines 4 and 8) are useless and

```
operation propose $(v_i)$ is % $p_i \in P[x]$ and $v_i \in \{0,1\}$ %
(1)  $est1_i \leftarrow v_i$; $r_i \leftarrow 0$;
(2)  loop forever
(3)    $r_i \leftarrow r_i + 1$;
       % $\boxed{\text{Phase 1: Try to champion a value}}$ %
(4)      $est1_i \leftarrow CONS_x[r_i,1]$.propose$(est1_i)$; % First, locally agree on $est1$ inside each cluster %
(5)      msg_exchange $(r_i, 1, est1_i)$;               % and then exchange across all clusters %
(6)      if $(\exists\ v : |supporters_i[v]| > n/2)$         % $v$ is supported by $> n/2$ processes %
(7)              then $est2_i \leftarrow v$ else $est2_i \leftarrow \perp$ end if;     % $p_i$ champions $v$ or no value %
       % Here, at any round $r_i$: $\big((est2_i \neq \perp) \wedge\ (est2_j \neq \perp)\big) \ \Rightarrow\ (est2_i = est2_j = v)$ %
       % $\boxed{\text{Phase 2: try to decide a value from the } est2 \text{ values}}$ %
(8)      $est2_i \leftarrow CONS_x[r_i,2]$.propose$(est2_i)$;  % Locally agree on $est2$ inside each cluster %
(9)      msg_exchange $(r_i, 2, est2_i)$;                 % and then exchange across all clusters %
(10)     let $rec_i$ = $\{est2 \mid$ PHASE2 $(r_i, est2)$ has been received$\}$;
(11)   % Here, at any round $r_i$, $\big((rec_i = \{v\}) \wedge\ (rec_j = \{\perp\})\big)$ are mutually exclusive %
(12)     case $(rec_i = \{v\})$        then broadcast DECIDE$(v)$; return$(v)$
(13)          $(rec_i = \{v, \perp\})$   then $est1_i \leftarrow v$
(14)          $(rec_i = \{\perp\})$      then $est1_i \leftarrow$ local_coin()
(15)     end case
(16) end loop.

(17) when DECIDE$(v)$ is received do broadcast DECIDE$(v)$; return$(v)$.
```

Algorithm 2: Local coin-based binary consensus for process $p_i \in P[x]$

can be suppressed. Moreover, the communication pattern can then be simplified by replacing the sets $supporters_i[a]$ and $supporters_i[b]$ by a simple counting of each value received during a phase. The algorithm then boils down to Ben-Or's algorithm [4]. In this sense Algorithm 2 is an extension of Ben-Or's algorithm to the hybrid communication model.

Main scalability and fault-tolerance property: The main property of Algorithm 2 with respect to failures is a consequence of the "One for All and All for One" principle, which translates as follows:

In all the executions in which there is a set of k distinct clusters $P[x_1]$, ..., $P[x_k]$, such that

- $|P[x_1]| + |P[x_2]| + \cdots + |[P[x_k]| > n/2$, and
- in each of these clusters at least one process does not crash,

Algorithm 2 solves consensus.

If there is no such set on clusters, the algorithm may not terminate. However, the algorithm is indulgent (whatever the failure pattern, it never terminates with an incorrect result [11], [12], [13]).

As already indicated in the Introduction, it follows that, if there is a cluster $P[x]$ such that $|P[x]| > n/2$ and one of its processes does not crash, Algorithm 2 solves consensus despite any failure patterns that occur in the other clusters.

On the proof: The proof of Algorithm 2 is left to the reader. It is obtained from the following observations.

- Due to the consensus objects used inside each cluster, at the same phase ph of the same round r, the processes of a cluster $P[x]$ send the same messages to the other processes, and, for any cluster $P[y]$, receive the same messages from the processes of $P[y]$.
- Given a cluster $P[x]$, As long as a process $p_i \in P[x]$ does not crash, all other processes receive messages, directly or indirectly thanks to the communication pattern (line 6 of Algorithm 1), from all the processes of $P[x]$ as if none of them crashed ("One for All and All for One" inside each cluster).
- The proof of Ben-Or's message-passing randomized binary consensus can be found in [2], [3], [4], [22]. Adding the two previous observations to this proof, proves that algorithm Algorithm 2 satisfies the validity, agreement, and termination properties defining consensus in the presence of crash failures.

C. Remark on Consensus in the m&m Comm. Model

The m&m communication model: In the uniform version of the m&m (messages and shared memories) model presented in [1], the memories shared by processes are defined by a graph G, whose vertices are the processes and two processes are neighbors if they share registers. Roughly speaking, there are n shared memories, each one associated with a process and its neighbors. As an example, if a process p has two neighbors q and r, there is a "p-centered" memory shared by p, q, and r, and if q has three neighbors p, s, and t, there is another'q-centered" memory shared by q, p, s and t. (From a hardware point of view, the memory shared by p, q, and r is accessed directly by p and remotely by its two neighbors q, and r. Similarly, the memory shared by q, p, s, and t is accessed directly by q and remotely by its three neighbors p, s, and t.) More information on the m&m communication model is given in appendix.

```
operation propose ($v_i$) is % $p_i \in P[x]$ and $v_i \in \{0,1\}$ %
(1)   $est_i \leftarrow v_i$; $r_i \leftarrow 0$;
(2)   loop forever
(3)     $r_i \leftarrow r_i + 1$;
(4)     $est_i \leftarrow CONS_x[r_i]$.propose($est1_i$); % First, locally agree on $est$ inside each cluster %
(5)     msg_exchange ($r_i, est_i$);                    % then exchange among all clusters %
(6)     $s_i \leftarrow$ common_coin();                 % and invoke the common coin %
(7)     if (during $r_i$ the same estimate $v$ is supported by $> n/2$ processes)
(8)                       then $est_i \leftarrow v$;
(9)                            if ($s_i = v$) then broadcast DECIDE ($v$); return ($v$) end if
(10)                      else $est_i \leftarrow s_i$
(11)    end if
(12)  end loop.

(13)  when DECIDE($v$) is received do broadcast DECIDE($v$); return($v$).
```

Algorithm 3: Common coin-based binary consensus for process $p_i \in P[x]$

Impact of the communication models on consensus algorithms: While both Algorithm 2 and the m&m consensus algorithm presented in [1] share message communication patterns, they are different and provide us with different properties. More precisely, due to its underlying shared memory model, the m&m consensus algorithm cannot ensure the “One for All and All for One” property, and consequently cannot benefit from its agreement power. Moreover, the number of underlying shared memory consensus objects accessed in each phase of a round is n in the m&m model, while (thanks to the cluster-based partitioning, namely a process accesses exactly one shared memory), this number is m (the number of clusters) in Algorithm 2. Finally, in the m&m model, a process p_i invokes $\alpha_i + 1$ consensus objects (where α_i is the number of its neighbors in the graph G) at each phase of a round. Differently, a process invokes a single consensus object in each phase of a round in the hybrid communication model.

IV. An Algorithm Based on a Common Coin

This section presents a consensus algorithm for the hybrid communication model, which is based on a common coin. This algorithm is a simple extension of a version of a pure message-passing consensus algorithm described in [22] (which is itself a simplified adaptation to crash failures of of a Byzantine consensus algorithm [10]).

Presentation of the algorithm: As Algorithm 2, Algorithm 3 consists in a sequence of asynchronous rounds, each made up of a single phase. Hence, there is no notion of a phase, and consequently the communication pattern (captured by Algorithm 1) simplifies, namely its first line becomes $(a, b) = (0, 1)$. Moreover, the shared memory of a cluster $P[x]$ is now made up of a single array of consensus objects $CONS_x[r]$, where $r \geq 1$ is a round number.

Lines 1-5 are the same in both algorithms. Hence, after line 4, the processes in the same cluster have the same estimate value est. Moreover, after line 5, a process p_i is such that, for any cluster $P[y]$, $supporters_i[0]$ contains all the processes $p_j \in P[y]$ such that p_i receives the value 0 from a process $p_k \in [y]$, ans similarly for $supporters_i[1]$.

Then, at line 6, a process p_i invokes the common coin and stores its value in s_i. Let us remind that, during the current round all the processes obtain the same bit b_r. If no value is supported by a majority of processes (i.e., $|supporters_i[0]| \leq n/2$ and $|supporters_i[1]| \leq n/2$), p_i considers $s_i = b_r$ as its new estimate (line 10), where b_r is the value obtained from the common coin by all processes at round r. Let us observe that, in this case, both values were present in the est variables at the beginning of the round. Differently, if there is value v supported by a majority of processes (i.e., $|supporters_i[v]| > n/2$), p_i adopts it as its new estimate (line 8). In this case, it also decides v if $v = s_i$. As in Algorithm 2, the messages DECIDE() are used to prevent possible deadlocks.

The reader can check that, due to lines 4-5, the termination-related property “Main scalability and modularity property”, stated in Section III-B, is satisfied by this algorithm.

On the proof of the algorithm: Let us observe that, as in Algorithm 2, the combination of the consensus objects used at each round inside each cluster with the message exchange pattern ensures that, given any cluster $P[y]$ in which at least one process has not crashed, every process receives the same messages directly or indirectly from the processes of $P[y]$ (“One for All and All for One” inside each cluster). With this observation, the proof of Algorithm 3 is the same as the one of a message-passing algorithm described in [22], from which it inherited its structure.

The consensus termination property is obtained in two stages. In the first stage, the non-crashed processes adopt the same value v an estimate. During the second stage the random bit must have the same value as v. The expected number of rounds for this to happen during the second stage is 2.

V. Conclusion

This paper was on the scalability of consensus algorithms in asynchronous systems where processes are partitioned into clusters and can communicate via a hybrid communication system. Namely, processes communicate via a shared memory inside each cluster, and via a message-passing system which allows any process to send message to any process.

The scalability is obtained with consensus objects used inside each cluster, and a simple communication pattern, which satisfies the following property: If a process p_i receives a message m from a process belonging to a partition $P[x]$, it is as if p_i received the very same message m from all the processes of the partition $P[x]$ (be them alive or crashed). Hence the motto "All for One and One for All". Scalability results from the fact that agreement inside a cluster can be done efficiently, but does not scale (due to hardware constraints). Differently, agreement through a message-passing system scales, but -due to message asynchrony– cannot be done efficiently.

According to the composition of the clusters, this hybrid communication model allows to circumvent the "majority of correct processes" assumption, which is a necessary requirement for consensus to be solved in non-hybrid crash-prone asynchronous message-passing systems. As a simple example (cited in the Introduction), let us consider the case where there is a cluster including a majority of processes (for example the cluster $P[2] = \{p_2, p_3, p_4, p_5\}$ in the system depicted at the right of Figure 1). In the failure patterns where any number of processes crash, except one process belonging to the majority cluster, consensus can be solved.

To illustrate these ideas, two scalable consensus algorithms have been presented. Both use randomization to circumvent consensus impossibility in the presence of asynchrony and process crashes. These algorithms are simple extensions of existing algorithms. One is based on local coins, while the other is based on a common coin. More generally, it would be interesting to investigate the scalability benefits of the hybrid communication model for other distributed computing problems.

Last but not least, from a software engineering point of view, the hybrid communication model favors a modular decomposition/composition of distributed algorithms.

Acknowledgments

The authors want to thank the referees for their constructive comments. This work has been partially supported by the French ANR project DESCARTES devoted to layered and modular structures in distributed computing.

References

[1] Aguilera M.K., Ben-David N., Calciu I., Guerraoui R., Petrank E., and Toueg S., Passing messages while sharing memory. *Proc. 37th ACM Int'l Symposium on Principles of Distributed Computing (PODC'18)*, ACM Press, pp. 51-60 (2018)

[2] Aguilera M.K. and Toueg S., The correctness proof of Ben-Or's randomized consensus algorithm. *Distributed Computing*, 25(5):371-381 (2012)

[3] Attiya H. and Welch J., *Distributed computing: fundamentals, simulations and advanced topics*, (2d Edition), Wiley-Interscience, 414 pages (2004)

[4] 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 (1983)

[5] Cachin Ch., Guerraoui R., and Rodrigues L., *Reliable and secure distributed programming*, Springer, 367 pages, ISBN 978-3-642-15259-7 (2011)

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

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

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

[9] Fischer M.J., Lynch N.A., and Paterson M.S., Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374-382 (1985)

[10] Friedman R., A. Mostéfaoui A., and Raynal M., Simple and efficient oracle-based consensus protocols for asynchronous Byzantine systems. *IEEE Transactions on Dependable and Secure Computing*, 2(1):46-56 (2005)

[11] Guerraoui R., Indulgent algorithms. *Proc. 19th Annual ACM Symposium on Principles of Distributed Computing (PODC'00)*, ACM Press, pp. 289-297 (2000)

[12] Guerraoui R. and Lynch N., A general characterization of indulgence. *ACM Transactions on Autonomous and Adaptive Systems*, 3(4), article 20, 19 pages (2008)

[13] Guerraoui R. and Raynal M., The information structure of indulgent consensus. *IEEE Transactions on Computers*, 53(4):453-466 (2004)

[14] Herlihy M.P., Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124-149 (1991)

[15] Herlihy M. and Shavit N., *The art of multiprocessor programming*. Morgan Kaufmann, 508 pages, ISBN 978-0-12-370591-4 (2008)

[16] Imbs D. and Raynal M., The weakest failure detector to implement a register in asynchronous systems with hybrid communication. *Theoretical Computer Science*, 512:130-142 (2013)

[17] Lim K., Chang J., Mudge T., Ranganathan R., Reinhardt S.K., and Wenisch T.F., Disaggregated memory for expansion and sharing in blade servers. *Proc. 36th International Symposium on Computer Architecture (ISCA'09)*, ACM Digital Library, pp. 267-278 (2009)

[18] Loui M. and Abu-Amara H., Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research*, 4:163-183, JAI Press (1987)

[19] Mostéfaoui A., Rajsbaum S. and Raynal M., Conditions on input vectors for consensus solvability in asynchronous distributed systems. *Journal of the ACM*, 50(6):922-954 (2003)

[20] Rabin M., Randomized Byzantine generals. *Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS'83)*, IEEE Computer Society Press, pp. 116-124 (1983)

[21] Raynal M., *Concurrent programming: algorithms, principles and foundations*. Springer, 515 pages, ISBN 978-3-642-32026-2 (2013)

[22] Raynal M., *Fault-tolerant message-passing distributed systems: an algorithmic approach*. Springer, 492 pages, ISBN 978-3-319-94140-0 (2018)

[23] Taubenfeld G., *Synchronization algorithms and concurrent programming*. Pearson Prentice-Hall, 423 pages, ISBN 0-131-97259-6 (2006)

APPENDIX

The *shared memory domain* $\mathcal{S}$ considered in [1] is defined as a set of process subsets, each $S \in \mathcal{S}$ defining a subset of processes sharing a specific common memory. While in the general case $\mathcal{S}$ can be be arbitrary, the algorithms in [1] consider the cases where the shared memory domain $\mathcal{S}$ is *uniform*.

"Uniform" means that the shared memory domain can be represented by an undirected graph $G = (V, E)$ where V is the set of proceses, and a process $p_i \in V$ can share registers with its neighbors as defined by E. More explicitly, S_i being $\{S_i\} = \{p_j : (p_i, p_j) \in E\}$, we have $\mathcal{S} = \{S_i : p_i \in V\}$.

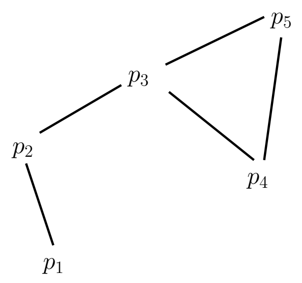

Figure 2. Example of a uniform shared memory domain

An example of a graph $G = (V, E)$, where $V = \{p_1, \cdots, p_5\}$ (from [1]) is given in Fig. 2. We have:

- $S_1 = \{p_1, p_2\}$,
- $S_2 = \{p_1, p_2, p_3\}$,
- $S_3 = \{p_2, p_3, p_4, p_5\}$,
- $S_4 = \{p_3, p_4, p_5\}$,
- $S_5 = \{p_3, p_4, p_5\}$.

Consequently, the shared memory domain $\mathcal{S}$ defined in Fig. 2 is $\{S_1, S_2, S_3, S_4, S_5\}$. More explicitly, $\mathcal{S} = \{\{p_1, p_2\}, \{p_1, p_2, p_3\}, \{p_2, p_3, p_4, p_5\}, \{p_3, p_4, p_5\}\}$.