Primary Partition "Virtually-Synchronous Communication" harder than Consensus*

André Schiper and Alain Sandoz

Département d'Informatique
Ecole Polytechnique Fédérale de Lausanne
CH-1015 Lausanne (Switzerland)

Abstract. The paper considers the problem of implementing "Virtually Synchronous Communication" in the primary partition of an asynchronous system. Virtually Synchronous Communication was first introduced by the Isis system as a powerful mechanism for building fault-tolerant processes that mask failures by replication: it can be understood as a rule for ordering message deliveries (reliable multicasts) with respect to view changes, defined by a membership service. Primary partition Virtually Synchronous Communication, noted PP-VSC, is the problem of implementing Virtually Synchronous Communication in the case of totally ordered views. The paper formally defines the problem, and shows that surprisingly this problem is harder than consensus: (1) consensus is solvable whenever the PP-VSC problem is solvable, however (2) there are environments where consensus is solvable, but not PP-VSC. The paper also defines an environment in which PP-VSC can be solved. The practical consequences of the result are discussed.

1 Introduction

The paper considers the problem of implementing "Virtually Synchronous Communication" in the primary partition of an asynchronous system. It shows that this problem is harder than consensus. Virtually synchronous communication is a mechanism for building fault-tolerant processes that mask failures by replication [4, 3]. The idea (first introduced in the Isis system) is to use a membership service, responsible for establishing views of the operational processes in the system [11], and to order message deliveries (reliable multicasts) with respect to view changes. A view is a set of correct processes, as perceived by the membership service. The membership service typically reacts to process crashes and recoveries, or long communication delays. These situations lead it to define new views that are delivered to each process. The following definition is considered in [12]: Given two consecutive views V and V', communication is virtually-synchronous

* Research supported by the "Fonds national suisse" and OFES under contract number 21-32210.91, as part of the ESPRIT Basic Research Project BROADCAST (number 6360), and by SPP-IP under contract number 5003-34344.

if and only if all processes in V and in V' delivered the same set of multicasts in view V [2]. The same definition is considered in [1, 2].

To understand the definition, consider two consecutive views V and V', and two processes members of both views: $p_i, p_j \in V$ and $p_i, p_j \in V'$. By delivering V' and learning that $p_j \in V'$, process p_i knows that p_j delivered the same set of multicasts in view V as itself. This has two consequences. First, the multicasts in view V are *terminated*: no multicast m delivered by p_i in V has ever to be retransmitted, as p_i knows that p_j has already delivered in V the same set of multicasts as itself. Second, if p_i and p_j started view V in the same initial state, and if process state is determined by an initial state and the set of multicasts delivered to that process [3], then p_i knows that p_j starts view V' in the same state as itself. *Partial* virtually synchronous communication is the problem of implementing virtually synchronous communication when views are partially ordered (i.e. several concurrent views can be active at the same time, which models a system that has logically partitioned). Partial virtually synchronous communication is easier than consensus: it can be solved in an environment where consensus is not solvable [8]. Implementation of partial VSC is considered in [1, 10, 12].

It might however often be desirable to prevent logical partitions (i.e. concurrent views) from occurring. This corresponds to the so called *primary partition* model which defines a unique totally ordered sequence of views in which progress is possible on behalf of the whole system [11]. *Linear* virtually synchronous communication or *Primary partition* VSC, noted PP-VSC, is the problem of implementing virtually synchronous communication in the case of totally ordered views. Informally we define PP-VSC as the following problem: given a view V, and for every process p_i in V a set Msg_i of multicasts delivered by p_i in view V, define a unique view V' such that every process in view V and in view V' has delivered the same set of multicasts in view V. Thus an instance of the PP-VSC problem occurs for each view change in the system [4]. Surprisingly it turns out that this problem is harder than consensus: (1) consensus is solvable whenever the PP-VSC problem is solvable, however (2) there are environments where consensus can be solved [6], but not PP-VSC. This paper formally establishes (1) and (2).

The paper is structured as follows. Section 2 presents the system model, and formally defines the PP-VSC problem. Section 3 shows that consensus is solvable whenever the PP-VSC problem is solvable. Section 4 shows the impossibility result: the PP-VSC problem is not solvable in an environment where the consensus problem is solvable. Section 5 considers an environment where the PP-VSC

[2] A message is delivered in view V if it is delivered after the delivery of view V, and before the delivery of the next view V'.

[3] If messages don't commute, total order is additionally required.

[4] Joins can easily be handled within the framework defined by the PP-VSC problem. Consider process p_k wanting to join while view V is defined. The request $join(p_k)$ can be considered as an ordinary reliable multicast issued in view V by some process $p_i \in V$. Let V' be the view output by the PP-VSC problem. Define the view subsequent to V as $V' \cup \{p_k\}$.

problem is solvable, and gives a solution to the problem. Section 6 discusses the practical consequences of the result.

2 System model and problem definition

The distributed system is composed of a finite set $\mathsf{S} = \{p_1, \ldots, p_n\}$ of processes completely connected through a set of channels. Communication is by message passing, asynchronous (there is no bound on the transmission delays), and reliable[5]. Processes fail by crashing (the paper does not consider the problem of process recovery after a crash). A process $p_i \in \mathsf{S}$ may (1) send a message to another process, (2) deliver a message sent by another process p_j, (3) perform some local computation, or (4) crash, which is modeled by by the local event $crash_i$.

The process *history* of $p_i \in \mathsf{S}$ is a sequence of events $h_i = e_i^0 \cdot e_i^1 \cdots e_i^k$. Histories of correct processes are infinite. If not infinite, the process history of p_i is terminated with event $crash_i$. A *cut* is an n-tuple of process history prefixes, one for each $p_i \in \mathsf{S}$. We assume familiarity with the notions of inter-event causality [9] and of consistent cuts [7]. Global predicates are evaluated on consistent cuts.

The primary partition-virtually synchronous communication problem (PP-VSC) is defined on S by:

1. an input from each process of S;
2. an output on some subset of processes of S;
3. a set of conditions linking inputs and outputs.

We start by describing inputs and outputs (I- stands below for input, O- for output):

Input. From every process $p_i \in \mathsf{S}$, PP-VSC takes as input a set of messages $\mathsf{I\text{-}Msg}_i \neq \emptyset$. To simplify we assume that for $p_i \neq p_j$, $\mathsf{I\text{-}Msg}_i \cap \mathsf{I\text{-}Msg}_j = \emptyset$ [6].

Output. On every process p_i of some non-empty subset of S, PP-VSC outputs (1) a set of messages $\mathsf{O\text{-}Msg}_i$ and (2) a set of processes $\mathsf{O\text{-}S}_i \subseteq \mathsf{S}$ (to avoid more notations, we make no distinction between the *set of processes* $\mathsf{O\text{-}S}_i$ and the *set of process ids* of processes in $\mathsf{O\text{-}S}_i$). $\mathsf{O\text{-}Msg}_i$ can be output in several steps. We note $\mathsf{O\text{-}Msg}_i(c)$ the set of messages output on p_i on cut c, and $\mathsf{O\text{-}Msg}_i$ the complete set of messages finally output on p_i.

This relates to the previous section as follows. Consider that $V = \mathsf{S}$ is the current view of the system, and assume that a new view V' has to be defined

[5] A reliable channel can be implemented by retransmitting lost or corrupted messages. A reliable channel ensures that a message sent by p_i to p_j is eventually received by p_j if p_i and p_j are correct. This does not exclude link failures, if we require that any link failure is eventually repaired.

[6] The impossibility result of Sect. 4 only requires $\mathsf{I\text{-}Msg}_i \not\subseteq \bigcup_{p_j \neq p_i} \mathsf{I\text{-}Msg}_j$.

(e.g. because some process in S is suspected to have crashed). Switching from V to V' requires to solve an instance of PP-VSC, i.e. all processes in both V and V' must have delivered the same set of messages in view V before delivering V'. Therefore, one must know what messages each process $p_i \in \mathsf{S}$ has *already* delivered in V on the cut on which the PP-VSC problem is defined. Input set $\mathsf{I\text{-}Msg}_i$ is precisely the set of multicasts delivered by p_i in view V on this cut. Using these inputs, a solution of the PP-VSC problem outputs on each process p_i a set of messages $\mathsf{O\text{-}Msg}_i$ that p_i is supposed to deliver *before* switching to the new view $V' \equiv \mathsf{O\text{-}S}_i$, which is also an output of PP-VSC.

This informal description translates into the six conditions $C1 - C6$ below defining a solution to the PP-VSC problem. Condition *Order* below states that $\mathsf{O\text{-}Msg}_i$ is output on p_i before $\mathsf{O\text{-}S}_i$.

C1. Order. Consider a cut c and the predicate $\text{TERMINATED}_i(c)$ such that $\text{TERMINATED}_i(c)$ holds iff $\mathsf{O\text{-}S}_i$ has been output on p_i (predicate TERMINATED_i is stable). Then, once TERMINATED_i holds, no more messages are output to p_i.
Formally, $\text{TERMINATED}_i(c) \;\Rightarrow\; \mathsf{O\text{-}Msg}_i(c) = \mathsf{O\text{-}Msg}_i$. □

C2. Termination. There exists at least one correct processes p_i, such that $\text{TERMINATED}_i()$ eventually holds and $p_i \in \mathsf{O\text{-}S}_i$. □

Conditions *Validity 1* and *Validity 2* below characterize the set of messages $\mathsf{O\text{-}Msg}_i$ output at p_i, with respect to the set of all input messages $\bigcup_{p_j \in \mathsf{S}} \mathsf{I\text{-}Msg}_j$. Validity 1 is states that any output message in $\mathsf{O\text{-}Msg}_i$ must have been input to the problem through some $\mathsf{I\text{-}Msg}_j$. Validity 2 is a *no-undo* condition: if a process p_i has already delivered a multicast m when the PP-VSC problem is defined, p_i should not learn later that m is not part of the complete set of multicasts it must deliver.

C3. Validity 1. Consider a cut c. For every process p_i and for every message m in $\mathsf{O\text{-}Msg}_i(c)$, there exists a process p_j such that $m \in \mathsf{I\text{-}Msg}_j$:

$$\bigwedge_{cut\ c} \bigwedge_{p_i \in \mathsf{S}} \left(\mathsf{O\text{-}Msg}_i(c) \subseteq \bigcup_{p_j \in \mathsf{S}} \mathsf{I\text{-}Msg}_j \right)$$

□

C4. Validity 2. Consider a cut c. For every process p_i, the input messages $\mathsf{I\text{-}Msg}_i$ are included in the output messages $\mathsf{O\text{-}Msg}_i(c)$. This condition states that the messages input by p_i must be included in the set of messages output at p_i:

$$\bigwedge_{cut\ c} \bigwedge_{p_i \in \mathsf{S}} \left(\mathsf{I\text{-}Msg}_i \subseteq \mathsf{O\text{-}Msg}_i(c) \right)$$

□

Agreement 1 below is (1) a consensus condition on $\mathsf{O\text{-}Msg}_i$ for all p_i together with (2) a *termination* condition. When $\mathsf{O\text{-}S}_i$ is delivered on p_i, process p_i knows (1) that an agreement on the messages to output has been reached, and (2) that the output of messages is terminated: every process $p_j \in \mathsf{O\text{-}S}_i$ has already output the same set of messages as itself. *Agreement 2* is a consensus condition on $\mathsf{O\text{-}S}_i$.

C5. Agreement 1. Consider a cut c and a process p_i such that $\text{TERMINATED}_i(c)$ holds. If $p_j \in \mathsf{O\text{-}S}_i$, then p_i and p_j have output the same set of messages:

$$\bigwedge_{cut\ c} \bigwedge_{p_i \in \mathsf{S}} \left(\text{TERMINATED}_i(c) \;\Rightarrow\; \left(\bigwedge_{p_j \in \mathsf{O\text{-}S}_i} \mathsf{O\text{-}Msg}_i(c) = \mathsf{O\text{-}Msg}_j(c) \right) \right)$$

$\Box$

C6. Agreement 2. Consider a cut c and two processes p_i, p_j such that $\text{TERMINATED}_i(c)$ and $\text{TERMINATED}_j(c)$ hold. Then p_i and p_j agree on the output set of processes:

$$\bigwedge_{cut\ c} \bigwedge_{p_i, p_j \in \mathsf{S}} \left(\text{TERMINATED}_i(c) \wedge \text{TERMINATED}_j(c) \;\Rightarrow\; (\mathsf{O\text{-}S}_i = \mathsf{O\text{-}S}_j) \right)$$

$\Box$

It follows directly from Agreement 1, Agreement 2 and Termination that a solution to the PP-VSC problem leads a subset of processes in S to reach an agreement on a set of output messages $\mathsf{O\text{-}Msg}$:

Lemma 2.1 *Consider the* PP-VSC *problem defined on* S. *Let* $p_i, p_j \in \mathsf{S}$ *be such that* TERMINATED_i *and* TERMINATED_j *both hold. Then* $\mathsf{O\text{-}Msg}_i = \mathsf{O\text{-}Msg}_j$.

PROOF. Assume that TERMINATED_i and TERMINATED_j both hold on a cut c. By Agreement 2, $\mathsf{O\text{-}S}_i = \mathsf{O\text{-}S}_j$. By Termination (C2), $\mathsf{O\text{-}S}_i$ and $\mathsf{O\text{-}S}_j$ are not empty; let $p_k \in \mathsf{S}$ be such that $p_k \in \mathsf{O\text{-}S}_i$, $p_k \in \mathsf{O\text{-}S}_j$. By Agreement 1 we have $\mathsf{O\text{-}Msg}_i = \mathsf{O\text{-}Msg}_k(c)$ and $\mathsf{O\text{-}Msg}_j = \mathsf{O\text{-}Msg}_k(c)$, i.e. $\mathsf{O\text{-}Msg}_i = \mathsf{O\text{-}Msg}_j$. $\Box$

Lemma 2.2 gives an important property of every solution to the PP-VSC problem. Let $p_i \in \mathsf{S}$ such that TERMINATED_i holds: if $p_j \in \mathsf{S}$, $p_j \neq p_i$, does not have its input messages $\mathsf{I\text{-}Msg}_j$ in $\mathsf{O\text{-}Msg}_i$, then p_j cannot be in $\mathsf{O\text{-}S}_i$.

Lemma 2.2 *Consider the* PP-VSC *problem defined on* S. *Let* $p_j \in \mathsf{S}$ *such that* $\mathsf{I\text{-}Msg}_j \neq \emptyset$. *If there exists a cut* c *and* $p_i \in \mathsf{S}$ *such that* TERMINATED_i *holds on* c *and* $\mathsf{I\text{-}Msg}_j - \mathsf{O\text{-}Msg}_i \neq \emptyset$, *then* $p_j \notin \mathsf{O\text{-}S}_i$.

PROOF. Consider $p_i \in \mathsf{S}$ and a cut c such that $\text{TERMINATED}_i(c)$ holds. Let p_j such that $\mathsf{I\text{-}Msg}_j \neq \emptyset$ and assume $p_j \in \mathsf{O\text{-}S}_i$. By condition Agreement 1, $\mathsf{O\text{-}Msg}_i = \mathsf{O\text{-}Msg}_j(c)$. By condition Validity 2, $\mathsf{I\text{-}Msg}_j - \mathsf{O\text{-}Msg}_j(c) = \emptyset$. Thus $\mathsf{I\text{-}Msg}_j - \mathsf{O\text{-}Msg}_i = \emptyset$. $\Box$

3 Reduction of consensus to PP-VSC

This section shows how to reduce consensus to the PP-VSC problem, i.e. how any solution of PP-VSC can be used to solve the consensus problem. In this prospect, consider that each $p_i \in \mathsf{S}$ proposes a value v_i taken from a set of possible values. The consensus problem consists in deciding on some value v such that the following three properties hold [6]:

Termination. Each correct process eventually decides.

Validity. If a process decides v, then v was proposed by some process.

Agreement. No two correct processes decide differently.

The reduction goes as follows:

- for every $p_i \in \mathsf{S}$, define $\mathsf{I\text{-}Msg}_i = \{< i, v_i >\}$;
- given a solution to the PP-VSC problem, consider any process p_i such that TERMINATED$_i$ holds. Define for p_i the decision value v of the consensus problem as the value v_j such that $j = min\ \{k \mid\ < k, v_k > \in\ \mathsf{O\text{-}Msg}_i\}$;
- once p_i has determined v, it broadcasts the decision value to S (recall that the channels are reliable).

Proposition 3.1 *The above reduction leads to a solution of the consensus problem.*

PROOF. Agreement holds because of lemma 2.1. Validity holds because of condition C3 (Validity 1) of PP-VSC. Because of the condition C2 (Termination), TERMINATED$_i(c)$ holds on some cut c for some correct process p_i and p_i broadcasts the decision value v. Since p_i is correct, every correct process eventually receives v. Thus the termination property of the consensus problem also holds. □

Notice that neither $\mathsf{O\text{-}S}$ output by the solution of PP-VSC nor condition Validity 2 have been used in the proof.

4 PP-VSC harder than consensus

The previous section shows that whenever the PP-VSC problem can be solved, the consensus problem can also be solved. Thus the consensus problem is not harder than the PP-VSC problem.

We now show that PP-VSC is harder than consensus, i.e. that there exists an environment in which the consensus problem can be solved, but not the PP-VSC problem. It is well known that consensus is not solvable in an asynchronous system with a single process crash failure [8]. Chandra and Toueg have shown that by adding the failure suspector $\diamond\mathcal{W}$ (see below) to the asynchronous environment, the consensus problem becomes solvable if the number of process crashes is bounded by f with $f < n/2$ [6]. We show that the PP-VSC problem is not solvable in this environment. Thus PP-VSC is harder than the consensus problem.

4.1 The hierarchy of failure suspectors

The definitions are taken from [6]. A failure suspector FS_i is a local module attached to process $p_i \in \mathsf{S}$, which maintains a list of processes that it currently suspects to have crashed. *Process p_i suspects process p_j* at some instant t, means that at t process p_j is in the list of suspected processes maintained by FS_i. A failure suspector can make mistakes by incorrectly suspecting a process. Suspicions are not stable: if at a given instant FS_i suspects p_j, it can later learn that the suspicion was incorrect: p_j is then removed by FS_i from the list of suspected processes.

[6] defines a hierarchy of failure suspectors ordered by reducibility. Let FS and FS' be two failure suspectors. FS' is said to be *reducible* to FS if there exists an algorithm $A_{FS \to FS'}$ that transforms FS into FS'. FS' is also said to be *weaker* than FS, noted $FS' \preceq FS$. From the hierarchy in [6] we need to consider $\diamond\mathcal{W}$ and the class $\mathcal{SF}(k)$ of failure suspectors:

Eventual Weak $\diamond\mathcal{W}$. The $\diamond\mathcal{W}$ failure suspector satisfies the following properties: (1) *weak completeness*: eventually every crashed process is permanently suspected by some correct process, and (2) *eventual weak accuracy*: there is a time after which some correct process is not suspected by any correct process. $\diamond\mathcal{W}$ is the weakest failure suspector that makes it possible to solve consensus in an asynchronous system with $f < n/2$ [5].

Strongly k-mistaken $\mathcal{SF}(k)$. A failure suspector FS is *Strongly k-mistaken*, noted $\mathcal{SF}(k)$, iff (1) it satisfies the *weak completeness* property, and (2) it does not make more than k mistakes. Recall that the failure suspector FS_i at process p_i makes a mistake at an instant t, if it incorrectly includes some process p_j in the list of suspected processes. A continuous retention of p_j in the list of suspected process does not count as additional mistakes. Thus p_i can make multiple mistakes about p_j only by removing p_j from its list of suspected processes, and later adding p_j again to the list of suspected processes.

The following relation holds [6]:

$$\diamond\mathcal{W} \preceq \ldots \preceq \mathcal{SF}(k+1) \preceq \mathcal{SF}(k) \preceq \ldots \preceq \mathcal{SF}(0).$$

When $f < n/2$, consensus is solvable using $\diamond\mathcal{W}$ (or any stronger failure suspector). When $f \leq n/2$, consensus is solvable using a failure suspector not weaker than $\mathcal{SF}(n-f)$. Finally when $f < n$, consensus is solvable using a failure suspector not weaker than $\mathcal{SF}(n-f-1)$.

4.2 PP-VSC not reducible to consensus

We show now that the PP-VSC problem is not reducible to consensus. By lemma 2.1 and condition C6 (Agreement 2), the PP-VSC problem consists in reaching agreement both on a set of messages O-Msg and a set of processes O-S. We show first that it is not possible to solve PP-VSC by reaching agreement simultaneously on O-Msg and O-S (Proposition 4.1). We consider then an algorithm $\mathcal{A}$ that tries to solve PP-VSC by first reaching agreement on O-Msg and then (i.e. by condition C1 (Order), once agreement on O-Msg has been reached), agreement on a set of processes O-S that have output O-Msg. We exhibit an environment where the consensus problem has a solution, but where the algorithm $\mathcal{A}$ cannot solve PP-VSC. The environment is defined by the failure suspector $\mathcal{SF}(2\lceil n/3 \rceil)$ and we consider $f = n - 2\lceil n/3 \rceil$. Because $f < n/2$ and $\mathcal{SF}(2\lceil n/3 \rceil)$ is stronger than $\diamond\mathcal{W}$, consensus is solvable in this environment. However PP-VSC is not, as shown by Proposition 4.2.

Proposition 4.1 *Consider the* PP-VSC *problem defined on* S. *The problem cannot be solved in an environment with* $\diamond\mathcal{W}$ *and* $f < n/2$ *by simultaneous agreement on* O-Msg *and* O-S.

Proof. Consider a run R that solves PP-VSC by reaching agreement in one step. Agreement in one step means that there exists and a cut c_{agr} such that (1) agreement has not been reached before c_{agr} and (2) agreement has been logically reached on c_{agr} (O-Msg and O-S are implicitly defined on c_{agr}; O-S has to be such that for every $p_i \in$ O-S, $\mathsf{I\text{-}Msg}_i \subseteq$ O-Msg). Because the input messages $\mathsf{I\text{-}Msg}_i$ are disjoint (see Sect. 2), and because O-Msg is not defined before c_{agr}, there exists a run R' indistinguishable from run R in which there exists a process $p_i \in$ O-S such that $\mathsf{O\text{-}Msg}_i(c_{agr}) \neq$ O-Msg. Let the adversary delay in R' any message m such that (1) on c_{agr} message m is in a channel to p_i, or (2) m is sent to p_i after c_{agr}. Then for any process p_j and cut c such that $\text{TERMINATED}_j(c)$, one has $\mathsf{O\text{-}Msg}_j(c) = \mathsf{O\text{-}Msg} \neq \mathsf{O\text{-}Msg}_i(c)$ and $p_i \in$ O-S, in contradiction with C5 (Agreement 1). □

Proposition 4.2 *Consider the* PP-VSC *problem defined on* S. *If* $f = n - 2\lceil n/3 \rceil$, *there is no algorithm that solves* PP-VSC *by reaching agreement first on* O-Msg *and then on* O-S, *using the failure suspector* $\mathcal{SF}(2\lceil n/3 \rceil)$.

Proof. The proof is by contradiction.
Consider an algorithm $\mathcal{A}$ that solves PP-VSC by reaching agreement in two steps. We construct a run $R_{\mathcal{A}}$ of algorithm $\mathcal{A}$ that respects $f = n - 2\lceil n/3 \rceil$ and the number of incorrect suspicions imposed by $\mathcal{SF}(2\lceil n/3 \rceil)$, and such that $R_{\mathcal{A}}$ does not satisfy the specification of PP-VSC. Partition S into three sub-sets Π_1, Π_2 and Π_3, such that:

- Π_1 and Π_2 are of size $\lceil n/3 \rceil$: $|\Pi_1| = |\Pi_2| = \lceil n/3 \rceil$
- Π_3 is of size $|\mathsf{S}| - |\Pi_1| - |\Pi_2|$, i.e. equal to f: $|\Pi_3| = f = n - (2\lceil n/3 \rceil)$

and construct a run $R_{\mathcal{A}}$ of algorithm $\mathcal{A}$ as follows:

– *$R_{\mathcal{A}}$ is split into three phases:*
  - *Phase 1* starts at the beginning of the algorithm, and ends on the consistent cut c_{agr_1} such that before c_{agr_1} no agreement on $\mathsf{O\text{-}Msg}$ was reached, and on c_{agr_1} $\mathsf{O\text{-}Msg}$ is implicitly defined.
  - *Phase 2* begins on c_{agr_1} and ends on the cut c_{agr_2} such that before c_{agr_2} no agreement on $\mathsf{O\text{-}S}$ was reached, and on c_{agr_2} $\mathsf{O\text{-}S}$ is implicitly defined.
  - *Phase 3* begins on c_{agr_2}.

– *Communications and crashes in $R_{\mathcal{A}}$:*
  - *Phase 1.* No process crashes, no message from any process in Π_2 is received in phase 1 by any process in $\Pi_1 \cup \Pi_3$.
  - *Phase 2.* No process crashes, no message from any process in Π_1 is received in phase 2 by any process in $\Pi_2 \cup \Pi_3$.
  - *Phase 3.* The adversary crashes all the processes in Π_3.

– *Failure suspector outputs in $R_{\mathcal{A}}$:*
  - *Phase 1.* Processes in Π_2 don't suspect any process. Processes in $\Pi_1 \cup \Pi_3$ suspect all processes in Π_2.
  - *Phase 2.* Processes in Π_1 don't suspect any process. Processes in $\Pi_2 \cup \Pi_3$ suspect all processes in Π_1.
  - *Phase 3.* Irrelevant, but for example: processes in $\Pi_1 \cup \Pi_2$ suspect all processes in Π_3.

Run $R_{\mathcal{A}}$ satisfies the basic assumptions:

– only processes in Π_3 crash, i.e. the number of process crashes is bounded by $n - 2\lceil n/3 \rceil$;

– in phase 1 processes in $\Pi_1 \cup \Pi_3$ incorrectly suspect processes in Π_2. In phase 2 processes in $\Pi_2 \cup \Pi_3$ incorrectly suspect processes in Π_1. This sums up to a total number of incorrect suspicions which is $2\lceil n/3 \rceil$, i.e. the failure suspector is in the equivalence class $\mathcal{SF}(2\lceil n/3 \rceil)$.

Run $R_{\mathcal{A}}$ of algorithm $\mathcal{A}$ does not satisfy the specifications of the PP-VSC problem:

1. By definition of phase 1 and condition C3 (Validity 1), agreement on the set of messages $\mathsf{O\text{-}Msg}$ can only include the initial messages $\mathsf{I\text{-}Msg}_i$ of processes in $\Pi_1 \cup \Pi_3$:
$$\mathsf{O\text{-}Msg} \subseteq \left(\bigcup_{p_i \in \Pi_1 \cup \Pi_3} \mathsf{I\text{-}Msg}_i \right)$$
2. By lemma 2.2 and because of 1, only processes in $\Pi_1 \cup \Pi_3$ can be included in the set $\mathsf{O\text{-}S}$ of processes that agree to have output $\mathsf{O\text{-}Msg}$. By definition of phase 2 (no message from processes in Π_1 is received by any process in $\Pi_2 \cup \Pi_3$) and condition C6 (Agreement 2), the set of processes $\mathsf{O\text{-}S}$ can

only include processes in Π_3 (there is no way for processes in Π_3 to know if processes in Π_1 have received $\mathsf{O\text{-}Msg}$, i.e. processes in Π_1 cannot be in $\mathsf{O\text{-}S}$). Thus:

$$\mathsf{O\text{-}S} \subseteq \Pi_3$$

3. By definition of phase 3, all processes in Π_3 crash.

Thus run $R_{\mathcal{A}}$ does not satisfy condition C2 (Termination) and hence the specification of the PP-VSC problem. A contradiction. □

5 Solving the PP-VSC problem with the $\mathcal{SF}(k)$ failure suspector

5.1 Sketch of the algorithm

Section 4 shows that when the number of process crashes in the system is bounded by $f < n/2$, the PP-VSC problem cannot be solved with a failure suspector as weak as $\diamond\mathcal{W}$. In this section we show how PP-VSC can be solved with the failure suspector $\mathcal{SF}(n - f - 1)$ when the number of process crashes is bounded by $f < n$ [7] [8]. We present the algorithm in a modular way, based on two algorithms: a *collect* algorithm that solves the *collect problem*, and a *consensus* algorithm.

Definition (Collect problem). We define the collect problem on a set of processes S, based on a failure suspector FS, as follows. Every process $p_i \in \mathsf{S}$ proposes an initial value v_i. The collect problem consists in defining for every process p_i an output set of values $\mathsf{O\text{-}Coll}_i$ such that for every process $p_j \in \mathsf{S}$, either (1) the initial value v_j of p_j is in $\mathsf{O\text{-}Coll}_i$, or (2) p_j is suspected by FS_i, the failure suspector module of p_i.

Solving the PP-VSC problem can be done in four phases, preceded by an initialization phase. In phases 1 and 3 a collect problem is solved with $\mathcal{SF}(n - f - 1)$; in phases 2 and 4 a consensus problem is solved [9]:

- Initialization phase: p_i outputs externally $\mathsf{I\text{-}Msg}_i$;

---

[7] This is not in contradiction with the result of section 4, because if $f = n - 2\lceil n/3 \rceil$ then $(n - f - 1) = 2\lceil n/3 \rceil - 1$, so $\mathcal{SF}(n - f - 1)$ is stronger than $\mathcal{SF}(2\lceil n/3 \rceil)$.

[8] It might appear surprising that we consider $f < n$ rather than the more restrictive $f < n/2$. However when $f < n$ consensus is solvable using $\mathcal{SF}(n - f - 1)$. Because of this result, the PP-VSC problem is also solvable with $\mathcal{SF}(n - f - 1)$ (see the PP-VSC algorithm and the proofs in Sect. 5.3).

[9] In order to distinguish the outputs of the intermediary problems, from the PP-VSC output, the former are called hereafter *internal* outputs, whereas the second are called *external* outputs.

– Phase 1: the collect problem with, for every $p_i \in \mathsf{S}$, the initial value $v_i \leftarrow \mathsf{I\text{-}Msg}_i$ is solved. We note $\mathsf{Ph1\text{-}O\text{-}Msg}_i$ the internal output of the collect problem on process p_i.

– Phase 2: the consensus problem with, for every $p_i \in \mathsf{S}$, the initial value $\mathsf{Ph1\text{-}O\text{-}Msg}_i$ is solved. We note $\mathsf{Ph2\text{-}O\text{-}Msg}$ the internal output of the consensus problem. As soon as the output of the consensus is known by p_i, the set of messages $\left(\mathsf{Ph2\text{-}O\text{-}Msg} - \mathsf{I\text{-}Msg}_i\right)$ are externally output on p_i.

– Phase 3: the collect problem is solved with, for every $p_i \in \mathsf{S}$, the following initial value:

$$\text{if } \left(p_i \text{ has output externally } \mathsf{Ph2\text{-}O\text{-}Msg}\right) \text{ and } \left(\mathsf{I\text{-}Msg}_i \subseteq \mathsf{Ph2\text{-}O\text{-}Msg}\right)$$
$$\text{then } \quad v_i \leftarrow p_i \text{ else } \quad v_i \leftarrow nil$$

We note $\mathsf{Ph3\text{-}O\text{-}S}_i$ the internal output of the collect problem on process p_i.

– Finally in phase 4, the consensus problem is solved with, for every $p_i \in \mathsf{S}$, $\mathsf{Ph3\text{-}O\text{-}S}_i$ as initial value. We note $\mathsf{Ph4\text{-}O\text{-}S}$, and also $\mathsf{O\text{-}S}$, the output of the consensus problem. If $p_i \in \mathsf{O\text{-}S}$, then $\mathsf{O\text{-}S}$ is externally output on p_i.

We describe here only the collect algorithm. The Chandra/Toueg consensus algorithm can be used in the phases 2 and 4 [6].

5.2 The collect algorithm

Consider the collect problem defined on S and based on a failure suspector FS. The problem is solved as follows. For every process $p_i \in \mathsf{S}$:

1. send v_i to every $p_j \in \mathsf{S}$;
2. for every $p_j \in \mathsf{S}$, wait either (1) to receive v_j, or (2) a notification of FS that p_j is suspected. Define $\mathsf{O\text{-}Coll}_i$ as the set of v_j received by p_i.

This trivially solves the collection problem.

5.3 Proof of the algorithm

Proposition 5.1 *On every cut c and for every process $p_i \in \mathsf{S}$, conditions C3 (Validity 1) and C4 (Validity 2) hold.*

Proof. Condition C3 is ensured by the initialization phase. The collect algorithm of phase 1, together with the consensus algorithm of phase 2, ensure condition C4. □

Proposition 5.2 *If* $\text{TERMINATED}_i(c)$ *holds on some cut c for some process $p_i \in \mathsf{S}$, conditions C5 (Agreement 1) and C6 (Agreement 2) hold for p_i.*

PROOF. Assume that $\textsc{terminated}_i(c)$ holds for some process p_i on some cut c, and that p_i has externally output $\mathsf{O\text{-}Msg}$ and $\mathsf{O\text{-}S}$. By the consensus algorithm of phase 4 (consensus on $\mathsf{O\text{-}S}$), C6 is satisfied.

Consider now $p_j \in \mathsf{O\text{-}S}_i$. By definition of the consensus problem of phase 4, if $p_j \in \mathsf{O\text{-}S}_i$, then there exists a process p_k such that $p_j \in \mathsf{Ph3\text{-}O\text{-}S}_k$. By definition of the collect problem of phase 3, if there exists a process p_k such that $p_j \in \mathsf{Ph3\text{-}O\text{-}S}_k$, then p_j has output $\mathsf{Ph2\text{-}O\text{-}Msg}$. By definition of the consensus phase 2, $\mathsf{O\text{-}Msg}_i = \mathsf{Ph2\text{-}O\text{-}Msg}$. Thus C5 is also satisfied. □

Proposition 5.3 *Conditions C1 (Order) and C2 (Termination) of* PP-VSC *are satisfied.*

PROOF. Condition C1 is trivially satisfied. For C2, we must prove that $\textsc{terminated}_i$ eventually holds for some correct process p_i such that $p_i \in \mathsf{O\text{-}S}_i$. We proceed in two steps. We prove first that phase 4 of the PP-VSC algorithm eventually outputs $\mathsf{O\text{-}S}$ on every correct process. Then we show that there is at least one correct process p_i such that $p_i \in \mathsf{O\text{-}S}$.

i) $\mathsf{O\text{-}S}$ *is eventually output on every correct process.*
The $\mathcal{SF}(n-f-1)$ failure suspector satisfies the weak completeness property (Sect. 4.1). Thus the collect algorithm of phase 1 eventually terminates on every correct process. Consensus can be solved with $f < n$ using $\mathcal{SF}(n-f-1)$ [6]. Thus the consensus algorithm of phase 2 eventually terminates on every correct process. The same arguments apply to the collect algorithm of phase 3, and to the consensus algorithm of phase 4, which completes the first part of the proof.

ii) There is at least one correct process $p_i \in \mathsf{O\text{-}S}$.
We prove the result by contradiction. Consider there exists a process p_i such that p_i is correct, but $p_i \notin \mathsf{O\text{-}S}$. Thus there exists $p_k \in \mathsf{S}$ such that $p_i \notin \mathsf{Ph3\text{-}O\text{-}S}_k$ (if for all p_k we have $p_i \in \mathsf{Ph3\text{-}O\text{-}S}_k$, then by definition of consensus in phase 4, $p_i \in \mathsf{Ph4\text{-}O\text{-}S}_k$, i.e. $p_i \in \mathsf{O\text{-}S}$).

If $p_i \notin \mathsf{Ph3\text{-}O\text{-}S}_k$, by definition of the collect problem of phase 3, either (1) p_k did incorrectly suspect p_i in Phase 3, or (2) $\mathsf{I\text{-}Msg}_i \not\subseteq \mathsf{Ph2\text{-}O\text{-}Msg}$. Case (1) accounts for one mistake of the failure suspector. In case (2), there exists $p_l \in \mathsf{S}$ such that $p_i \notin \mathsf{Ph1\text{-}O\text{-}Msg}_l$ (if for all p_l, $p_i \in \mathsf{Ph1\text{-}O\text{-}Msg}_l$, then by definition of the consensus of phase 2, $p_i \in \mathsf{Ph1\text{-}O\text{-}Msg}_l$). If $p_i \notin \mathsf{Ph1\text{-}O\text{-}Msg}_l$, by definition of the collect problem of phase 1, p_l did incorrectly suspect p_i in Phase 1. Thus case (2) also accounts for one mistake of the failure suspector.

Altogether, every p_i correct not in $\mathsf{O\text{-}S}$ accounts at least for one mistake of the failure suspector $\mathcal{SF}(n-f-1)$. If there are no correct processes in $\mathsf{O\text{-}S}$, then this accounts at least for $n-f$ mistakes of $\mathcal{SF}(n-f-1)$. A contradiction. □

6 Discussion

The paper formally defines the Primary Partition "Virtually Synchronous Communication" problem, noted PP-VSC, and shows that the problem is harder than consensus: the consensus problem is solvable whenever the PP-VSC problem is solvable, whereas the PP-VSC problem cannot be solved in some environments where consensus can be solved. More specifically the paper shows that PP-VSC cannot be solved with the eventual weak failure suspector $\diamond\mathcal{W}$ and $f < n/2$ (f is the maximum number of processes that may crash). The paper also shows that the PP-VSC problem can indeed be solved with the failure suspector $\mathcal{SF}(n-f-1)$ and $f < n$. We don't claim that $\mathcal{SF}(n-f-1)$ is the weakest failure suspector for solving PP-VSC. Establishing the weakest failure suspector for solving PP-VSC has still to be done.

The result of the paper has a very practical consequence in a large-scale system (WAN), where physical partitions are not unlikely to occur. If two processes p_i and p_j are partitioned, the probability of p_i incorrectly suspecting p_j, and p_j incorrectly suspecting p_i, is almost inevitable (suspicions are based on timeouts). Thus incorrect suspicions might be frequent in a large-scale system, and the property of $\mathcal{SF}(n-f-1)$ is very unlikely to be ensured.

As it is pointed out at the end of Section 3, the difficulty of PP-VSC is related to condition Validity 2: if a process p_i has already delivered a multicast m in view V when the PP-VSC problem is defined, p_i should not learn later that m is not part of the complete set of multicasts it must deliver in view V. The difficulty of PP-VSC is thus related to the *early* delivery of multicasts in a view V. Early delivery can be avoided if messages are multicast using a *uniform* (reliable) multicast [13] instead of just a reliable multicast. This leads to a slightly modified PP-VSC problem, and our intuition is that this modified problem is equivalent to consensus. This result has still to be established. If true, it would strongly argue for exclusively using a uniform multicast whenever virtually synchronous communication must be ensured in the primary partition model of a large scale distributed system.

Acknowledgments: We would like to thank the anonymous referees for their useful comments.

References

1. Y. Amir, D. Dolev, S. Kramer, and D. Malki. Membership Algorithms for Multicast Communication Groups. In *6th Intl. Workshop on Distributed Algorithms proceedings (WDAG-6), (LCNS, 647)*, pages 292–312, November 1992.
2. Y. Amir, L.E. Moser, P.M. Melliar-Smith, D.A. Agarwal, and P.Ciarfella. Fast Message Ordering and Membership Using a Logical Token-Passing Ring. In *IEEE 13th Intl. Conf. Distributed Computing Systems*, pages 551–560, May 93.
3. K. Birman. The Process Group Approach to Reliable Distributed Computing. *Comm. ACM*, 36(12):37–53, December 93.
4. K. Birman, A. Schiper, and P. Stephenson. Lightweight Causal and Atomic Group Multicast. *ACM Trans. Comput. Syst.*, 9(3):272–314, August 1991.
5. T. D. Chandra, V. Hadzilacos, and S. Toueg. The Weakest Failure Detector for Solving Consensus. In *proc. 11th annual ACM Symposium on Principles of Distributed Computing*, pages 147–158, 1992.

6. Tushar D. Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. Technical Report 93-1374, Department of Computer Science, Cornell University, August 1993. A preliminary version appeared in the *Proceedings of the Tenth ACM Symposium on Principles of Distributed Computing*, pages 325–340. ACM Press, August 1991.
7. K. M. Chandy and L. Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Trans. Comp. Syst.*, 3(1):63–75, February 1985.
8. M. Fischer, N. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM*, 32:374–382, April 1985.
9. L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Comm. ACM*, 21(7):558–565, July 78.
10. P. M. Melliar-Smith, L. E. Moser, and V. Agrawala. Membership Algorithms for Asynchronous Distributed Systems. In *IEEE 11th Intl. Conf. Distributed Computing Systems*, pages 480–488, May 91.
11. A. M. Ricciardi and K. P. Birman. Using Process Groups to Implement Failure Detection in Asynchronous Environments. In *proc. annual ACM Symposium on Principles of Distributed Computing*, pages 341–352, August 1991.
12. A. Schiper and A. Ricciardi. Virtually-Synchronous Communication Based on a Weak Failure Suspector. In *IEEE 23rd Int Symp on Fault-Tolerant Computing (FTCS-23)*, pages 534–542, June 93.
13. A. Schiper and A. Sandoz. Uniform Reliable Multicast in a Virtually Synchronous Environment. In *IEEE 13th Intl. Conf. Distributed Computing Systems*, pages 561–568, May 93.