An Automatic Real-Time Analysis of the Time to Reach Consensus

Tatsuhiro Tsuchiya *
Osaka University
1-5 Yamadaoka, Suita, 565-0871 Japan
tatsuhiro@ieee.org

André Schiper †
École Polytechnique Fédérale de Lausanne (EPFL)
1015 Lausanne, Switzerland
andre.schiper@epfl.ch

Abstract

Consensus is one of the most fundamental problems in fault-tolerant distributed computing. This paper proposes a mechanical method for analyzing the condition that allows one to solve consensus. Specifically, we model check a distributed algorithm that implements a communication predicate, which is an alternative system abstraction to failure detectors. This model checking problem is challenging because it involves both continuous time and unbounded integers. We solve the problem by reducing it to the satisfiability problem of linear arithmetic constraints over real and integer variables. The proposed method can be used to determine the length of a synchronous period required for implementing a communication predicate for solving consensus.

1. Introduction

Consensus is one of the most fundamental problems in fault-tolerant distributed computing. It is at the core of the state machine replication, the most general method for implementing fault tolerant services. This paper proposes a mechanical method for analyzing the timeliness requirement for solving consensus.

It is well-known that consensus cannot be solved by any deterministic algorithm in a pure asynchronous fault-prone distributed system [11]. Any practical model therefore must be augmented with synchrony assumptions to make consensus solvable.

In this paper a general asynchronous model is assumed where the system alternates between good periods and bad periods [13]. Our proposed method can determine whether a good period of a given length is sufficient for solving consensus.

More specifically, we focus on a distributed algorithm that implements a *communication predicate*, which encapsulates fault and synchrony conditions [4]. By model checking the algorithm, our proposed method determines if a good period of a given length allows the algorithm to implement a communication predicate that is sufficient for solving consensus. The advantages of communication predicates over failure detectors are elaborated in [3, 4, 13].

The contribution of the paper is twofold. The first is to provide an automatic analysis method for the timeliness properties required for solving consensus. Although considerable research has been conducted to investigate the performance of consensus algorithms, it is only recently that the issue of performance following bad periods has begun to get attention [9, 13, 14]. To our knowledge, our work is the first attempt to apply a formal method to this issue. In [13] mathematical proofs are provided for some timeliness properties of the predicate implementation. The model checking approach proposed in this paper allows finer analysis for specific situations with fixed parameter values.

The second contribution of the paper is the novel idea behind the proposed method. Compared to many problems that have been addressed by real-time model checking, the verification problem tackled in the paper is unique in that unbounded integer numbers must be treated. This feature prevents us from using well-established timed automaton-based model checking techniques [1]. We solve the problem by reducing it to the satisfiability problem of linear arithmetic constraints over real and integer variables.

The paper is structured as follows. Section 2 describes the system model and the concept and implementation of communication predicates. Section 3 describes the proposed verification method. Section 4 presents the results of experiments using the proposed method. Section 5 summarizes related work. Section 6 concludes the paper.

2. Preliminaries

This section summarizes some of the discussions made in [13]. We consider a two-layered distributed system con-

*This work was done when the first author was visiting EPFL with support from Scientist Exchange Program between JSPS and SNSF.

†Research funded by the Swiss National Science Foundation under grant number 200021-111701.

0-7695-3054-0/07 $25.00 © 2007 IEEE
DOI 10.1109/PRDC.2007.10

sisting of n processes. In the higher layer a consensus algorithm works in communication-closed rounds, while in the lower layer the communication predicate is implemented and provided to the higher layer. Let Π denote the set of these n processes. Processes and links can be faulty but do not behave maliciously. More detailed assumptions about faults are given in Section 2.2.

2.1. The HO Model and Consensus (The Upper Layer)

We adopt the *Heard-Of (HO) Model* [4] as the communication-closed round model in the upper layer. The HO model generalizes the asynchronous round model in [10]. In the HO model both synchrony degree and faults are represented in the form of *transmission faults*.

An algorithm for this model comprises, for each round r, a sending function S_p^r and a transition function T_p^r. In each round r, every process p sends messages according to $S_p^r(s_p)$, where s_p is the state of p. Then, p makes a state transition according to $T_p^r(R, s_p)$, where R is the collection of all messages that have been received in round r.

We denote by $HO(p,r)$ ($\subseteq \Pi$) the set of processes from which p receives a message in round r: $HO(p,r)$ is the "heard of" set of p in round r. A *transmission fault* refers to the situation where $q \notin HO(p,r)$ while q sent (or was supposed to send) a message to p in round r. Transmission faults can occur if messages missed a round due to the asynchrony of communication and processing, or if a process or a link is faulty.

Consensus is specified by the following three conditions:

Integrity Any decision value is the proposed value of some process.

Agreement No two processes decide differently.

Termination All processes eventually decide.

Note that the termination property requires all processes to decide. Discussion of the reason for this specification can be found in [3, 4].

As an example of a consensus algorithm, we consider the *OneThirdRule* algorithm (*OTR* for short) [4], see Algorithm 1. A sufficient condition for OTR to solve consensus can be represented in the form of a *communication predicate*, that is, a predicate over the collection of HO sets $(HO(p,r))_{p\in\Pi, r>0}$. Let:

$$\begin{aligned} \mathcal{P}_{su}(\Pi_0, r_0) &:= \forall p \in \Pi_0 : HO(p, r_0) = \Pi_0 \\ \mathcal{P}^2_{su}(\Pi_0) &:= \exists r_0, r_1 \text{ s.t. } 1 \le r_0 \le r_1 : \\ &\quad \mathcal{P}_{su}(\Pi_0, r_0) \wedge \mathcal{P}_{su}(\Pi_0, r_1) \end{aligned}$$

In words, $\mathcal{P}_{su}(\Pi_0, r_0)$ states the existence of a round in which all processes receive the messages from the same subset of the processes, while $\mathcal{P}^2_{su}(\Pi_0)$ states the existence of two such rounds.

Algorithm 1 The *OneThirdRule* algorithm [4]

```
1:  Initialization:
2:    x_p ∈ V, initially v_p            { v_p is the proposed value of p. }

3:  Round r:
4:    S_p^r :
5:      send ⟨ x_p ⟩ to all processes

6:    T_p^r :
7:      if |HO(p, r)| > 2n/3 then
8:        if the values received, except at most [(n−1)/3], are equal
          to x̄ then
9:          x_p ← x̄
10:       else
11:         x_p ← smallest x received
12:       if more than 2n/3 values received are equal to x̄ then
13:         DECIDE(x̄)
```

It is not difficult to see that communication predicate $P^2_{su}(\Pi_0)$ ensures the termination of processes in Π_0 if $|\Pi_0| > 2n/3$. Round r_0 allows every process $p \in \Pi_0$ to adopt the same value for x_p at the end of this round, while round r_1 ensures that p decides in that round. Integrity trivially holds. Agreement also holds because if some process decides some value v in round r, then only v can be assigned to x_p in any round $r' \ge r$.

2.2. The Asynchronous Model and Communication Predicate Implementation (The Lower Layer)

Communication-closed rounds and the communication predicate in the upper layer are implemented by an algorithm that works in the lower layer. In the lower layer, a process executes a sequence of atomic steps, which are either *send* or *receive* steps. Steps take no time but time elapses between steps. Each process p has two sets of messages: $network_p$ and $buffer_p$. If p executes $send_p(msg)$ to Π, then message msg is put into $network_q$ for all processes $q \in \Pi$. Messages in $network_p$ are transfered to $buffer_p$ by the network. If p executes a receive step, then p receives at most one message from $buffer_p$. If $buffer_p = \emptyset$ at the time when a receive step is executed, then an empty message is received. Thus receive steps are never blocked.

We consider a general fault model where good periods and bad periods alternate. In bad periods, processes can crash and recover, and suffer from send and receive omission. Also the network can loose messages.

A good period $I = [\tau_G, \tau_G + d\tau]$ is defined with respect to the set π_0 of "good" processes ($\pi_0 \subseteq \Pi$). During I the processes in π_0 are up and do not crash. Further, communication and processing in π_0 are synchronous, that is:

1. For two processes $p, q \in \pi_0$, if process p executes $send_p(msg)$ at time $t \in I$, then $msg \in buffer_q$ by

Algorithm 2 The algorithm for implementing $\mathcal{P}_{su}(\pi_0, *)$ [13]

```
1:  Initialization:
2:      msgsRcv_p ← ∅
3:      r_p ← 1
4:      next_r_p ← 1
5:      s_p ← init_p   { init_p is the initial state of p in the upper layer. }

6:  Task:
7:      while true do
8:          ct_p ← 0
9:          msg ← S_p^{r_p}(s_p)
10:         send ⟨msg, r_p⟩ to all
11:         while next_r_p = r_p do
12:             ct_p ← ct_p + 1
13:             if ct_p ≥ 2δ + (n + 2)φ then
14:                 next_r_p ← r_p + 1
15:             receive a message (highest round number first)
16:             if message is ⟨msg, r'⟩ from q and r' ≥ r_p then
17:                 msgsRcv_p ← msgsRcv_p ∪ {⟨msg, r', q⟩}
18:                 if r' > r_p then
19:                     next_r_p ← r'
20:         R ← {⟨msg', q'⟩ | ⟨msg', r_p, q'⟩ ∈ msgsRcv_p}
21:         s_p ← T_p^{r_p}(R, s_p)
22:         for all r' ∈ [r_p + 1, next_r_p − 1] do
23:             s_p ← T_p^{r'}(∅, s_p)
24:         r_p ← next_r_p
```

or at time $t + \delta$, provided that $t + \delta \in I$.

2. In any open contiguous sub-interval of I of length 1, every process in π_0 takes at most one step.
3. In any contiguous sub-interval of I of length ϕ (≥ 1), every process in π_0 takes at least one step.

Let $\overline{\pi_0}$ denote $\Pi \backslash \pi_0$. We assume that processes in $\overline{\pi_0}$ are down and do not recover during a good period I. Moreover, no messages from processes in $\overline{\pi_0}$ are in transit during I. As in [13], we refer to such a good period as a $\overline{\pi_0}$-*down good period*.

We let $\mathcal{P}_{su}(\pi_0, *)$ denote $\exists r > 0 : \mathcal{P}_{su}(\pi_0, r)$. In a sufficiently long $\overline{\pi}_0$-down good period, Algorithm 2, which was proposed in [13], can implement $\mathcal{P}_{su}(\pi_0, *)$. In the remainder of the paper, since $\mathcal{P}^2_{su}(\pi_0)$ holds iff $\mathcal{P}_{su}(\pi_0, r)$ holds for two different rounds $r = r_0, r_1$, we concentrate our discussion on the analysis of a $\overline{\pi}_0$-down good period that allows Algorithm 2 to implement $\mathcal{P}_{su}(\pi_0, *)$.

This algorithm works as follows: The function $S_p^{r_p}$ at line 9 is the send function of the consensus algorithm in the upper layer and returns the message to be sent. The message, together with the current round number r_p, is sent at line 10. Variable ct_p counts the number of receive steps during an iteration of the while loop between lines 11 to 19. Process p executes receive steps until (1) $\lceil 2\delta + (n+2)\phi \rceil$ receive steps have been executed or (2) a message of a round $r' > r_p$ is received. The round number will be updated to $r_p + 1$ in Case (1) or to r' in Case (2). Before r_p is updated to the new round number at line 24, the state transition function $T_p^{r_p}$ of the consensus algorithm is executed with the set R of the messages received in round r_p (line 21). In addition, in Case (2), the state transition function is executed for all rounds from $r_p + 1$ to $r' - 1$ with an empty set of messages (lines 22–23).

Figure 1 schematically shows a part of a run of this algorithm, where $\Pi = \{p_1, p_2, p_3, p_4\}$, $\pi_0 = \{p_1, p_2, p_3\}$, $\phi = 1.5$ and $\delta = 2$. A $\overline{\pi_0}$-down good period starts at time 20. The black, gray, and white dots respectively represent send steps, receive steps that actually received a message, and receive steps receiving an empty message. The number attached to a send step denotes the round number of the message sent by it, while the tuple associated with a receive step represents the round number of the received message and its sender process. $\mathcal{P}_{su}(\{p_1, p_2, p_3\}, 10)$ holds in this example.

3. Verification

The verification problem we address is defined as follows: Given π_0, n, δ, ϕ and $d\tau$, decide whether a $\overline{\pi_0}$-down good period $I = [\tau_G, \tau_G + d\tau]$ is large enough to implement the communication predicate $\mathcal{P}_{su}(\pi_0, *)$ for any τ_G.

We propose a conservative approximate solution to this problem. That is, if our proposed method outputs "Yes," then it is guaranteed that $\overline{\pi_0}$-down good period $I = [\tau_G, \tau_G + d\tau]$ implements $\mathcal{P}_{su}(\pi_0, *)$.

3.1. k-Step Sequence $V(k)$

The idea of our verification approach is to limit the search space to a range that can be represented by a collection of step sequences of bounded length. Parameter k (≥ 1) is used to denote the length of the step sequences. Specifically, we consider, for each $p \in \pi_0$, a sequence of the first k consecutive steps that occur at or after τ_G (the beginning of the good period). We let $V(k)$ denote the set of steps in these sequences. That is,

$$V(k) := \{step_{p,i} \mid \forall p \in \pi_0, \forall i \in [1, k]\}$$

where $step_{p,i}$ ($i \in [1, k]$) is the i-th step of p after time τ_G.

We associate each step $step_{p,i}$ in $V(k)$ with the following attributes:

- $t_{p,i}$: the time at which the step occurs.
- $ct_{p,i}$: the value of ct_p of Algorithm 2 when the step occurs. If $ct_{p,i} = 0$, $step_{p,i}$ is a send step; otherwise, $step_{p,i}$ is a receive step.

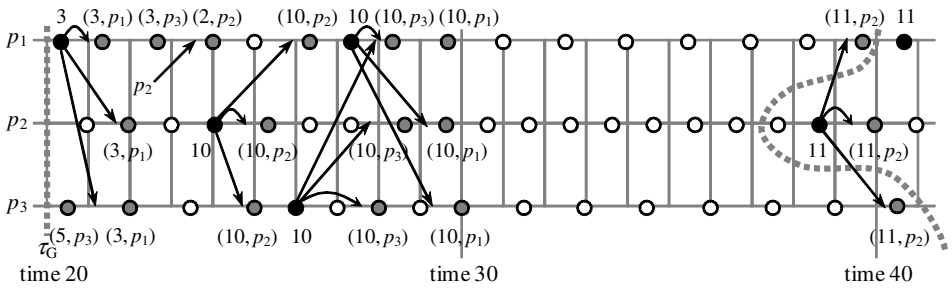

Figure 1. A part of a run of Algorithm 2

- $r_{p,i}$: the round number of process p, i.e., the value of r_p of Algorithm 2. Precisely, if $step_{p,i}$ receives a message of round $> r_p$, then $r_{p,i}$ represents the new value of r_p updated at line 24 of Algorithm 2. In this case $r_{p,i}$ equals the round number of the message.
- $ta^q_{p,i}$ ($\forall q \in \pi_0$): the arrival time of the message sent by $step_{p,i}$ to q. If $step_{p,i}$ is a receive step, then $ta^q_{p,i}$ is set to 0.
- $type_{p,i}$: the type of the message received by $step_{p,i}$ (explained below). If $step_{p,i}$ is a send step, then $type_{p,i} = \perp$.

The attributes $t_{p,i}$ and $ta^q_{p,i}$ are real values, while the other attributes are integers. A receive step is one of the following four types, where $q \in \pi_0$:

$\langle 1, q, j\rangle$: (Type 1) The received message has a round number $\geq r_p$ and is sent by $step_{q,j} \in V(k)$ (i.e., $j \in [1, k]$).

$\langle 2, q\rangle$: (Type 2) The received message has a round number $\geq r_p$ and is sent by q before $step_{q,1}$.

$\langle 3, q\rangle$: (Type 3) The received message has a round number $\geq r_p$ and is sent by q after $step_{q,k}$.

$\langle 4\rangle$: (Type 4) The received message has a round number $< r_p$ or is an empty message. Messages of this type are simply discarded by Algorithm 2 (see line 16).

Execution for $V(k)$: We define an *execution for* $V(k)$ as a value assignment to the attributes of all the steps in $V(k)$ that corresponds to a possible run of Algorithm 2 in $\overline{\pi_0}$-down good period $[\tau_G, \infty]$. We say that an execution implements a communication predicate by time t iff the message receptions that occur by time t in the execution guarantee that the communication predicate holds. If t is not important, we simply say that an execution implements a communication predicate.

In Figure 1, the dotted lines show which steps are contained in $V(16)$; that is, $V(16)$ contains, for each process, the first 16 steps that occurred at or after τ_G. The execution for $V(16)$ in Figure 1 implements $\mathcal{P}_{su}(\pi_0, 10)$ by time 30, where $\pi_0 = \{p_1, p_2, p_3\}$. Table 1 shows the attribute values of some steps of p_1 in this execution.

Table 1. Attributes of some steps of p_1 in Figure 1

i	$t_{p_1,i}$	$ct_{p_1,i}$	$r_{p_1,i}$	$ta^{p_1}_{p_1,i}$	$ta^{p_2}_{p_1,i}$	$ta^{p_3}_{p_1,i}$	$type_{p_1,i}$
1	20.2	0	3	21.0	21.6	21.1	$\perp$
2	21.3	1	3	0	0	0	$\langle 1, p_1, 1\rangle$
3	22.8	2	3	0	0	0	$\langle 2, p_3\rangle$
4	24.0	3	3	0	0	0	$\langle 4\rangle$
5	25.0	4	3	0	0	0	$\langle 4\rangle$
6	26.3	5	10	0	0	0	$\langle 1, p_2, 4\rangle$
7	27.3	0	10	28.0	29.1	29.2	$\perp$
8	28.3	1	10	0	0	0	$\langle 1, p_3, 5\rangle$
16	39.7	9	11	0	0	0	$\langle 3, p_2\rangle$

3.2. Arithmetic Constraints

The proposed method solves the verification problem by solving several instances of a satisfiability problem, which is the problem of deciding whether or not at least one value assignment exists that satisfies all linear arithmetic constraints in a given set. A constraint is a boolean combination of linear (in)equalities over real or integer variables and constants.

We use arithmetic constraints over the attributes of the steps in $V(k)$ to reason about the executions for $V(k)$. That is, the real and integer variables involved in the arithmetic constraints correspond to those attributes.

Specifically, we construct the following three constraint sets:

- $M(k)$: $M(k)$ models the behavior of Algorithm 2: $M(k)$ represents all possible executions for $V(k)$ in such a way that $M(k)$ is satisfied by any execution for $V(k)$.[1] The non real-time constraints that define $M(k)$ are shown in Figure 2. In this figure, lines 4–7, for example, specify how the value of ct_p changes. For instance, constraint $\forall p \in \pi_0, \forall i \in [1, k] : 0 \leq$

[1] Note that we do not guarantee that a satisfying assignment to $M(k)$ is an execution, since this would make $M(k)$ much more complex. In other words, $M(k)$ is an overapproximation of the executions for $V(k)$.

$ct_{p,i} \leq \lceil 2\delta + (n+2)\phi \rceil$ [2] (at line 5) specifies that ct_p varies between 0 to $\lceil 2\delta + (n+2)\phi \rceil$ (see lines 8 and 13 of Algorithm 2). Real-time constraints of $M(k)$ are presented in Section 3.4.

- $P(k)$: $P(k)$ specifies the condition for an execution for $V(k)$ *not* to implement communication predicate $\mathcal{P}_{su}(\pi_0, *)$. Precisely, $P(k)$ is satisfied by an execution for $V(k)$ iff the execution does not implement the communication predicate.
- $P_{d\tau}(k)$: $P_{d\tau}(k)$ is almost the same as $P(k)$ except that $P_{d\tau}(k)$ takes the length $d\tau$ of the good period into account: $P_{d\tau}(k)$ is satisfied by an execution for $V(k)$ iff the execution does not implement the communication predicate $\mathcal{P}_{su}(\pi_0, *)$ by time $\tau_G + d\tau$.

$P(k)$ and $P_{d\tau}(k)$ are described in Section 3.5.

The satisfiability problem of this class has NP-hard complexity; however, heuristics have been extensively studied and thus even large instances of the problem can be solved in a reasonable amount of time using a recent algorithm. Throughout the work, we use the YICES satisfiability solver [7].

3.3. Algorithm Overview

Algorithm 3 shows how the verification works. It consists of two phases.

Phase 1 determines the value of k such that $V(k)$ represents a sufficiently large search space for solving the verification problem. Specifically, we seek the least k such that all executions for $V(k)$ implement the communication predicate $\mathcal{P}_{su}(\pi_0, *)$, assuming that the good period lasts sufficiently long. We denote this value of k as k_{min}.

Phase 2 is then performed to determine if all executions for $V(k_{min})$ implement $\mathcal{P}_{su}(\pi_0, *)$ by time $\tau_G + d\tau$. If so, it is ensured that $\overline{\pi_0}$-down good period $I = [\tau_G, \tau_G + d\tau]$ implements $\mathcal{P}_{su}(\pi_0, *)$.

3.3.1 Phase 1

Phase 1 repeatedly checks the satisfiability of the constraint set:

$$M(k) \cup P(k), \tag{1}$$

starting from $k = 1$ up to k_{min}. Remember that $M(k)$ is satisfied by any execution for $V(k)$, while $P(k)$ is satisfied by any execution for $V(k)$ that *does not* implement $\mathcal{P}_{su}(\pi_0, *)$. Hence, Constraint set (1) is satisfied by any execution for $V(k)$ if the execution does not implement communication predicate $\mathcal{P}_{su}(\pi_0, *)$. If no satisfying assignment exists, one can conclude that *all* executions for $V(k)$ implement $\mathcal{P}_{su}(\pi_0, *)$.

[2] $\forall, \exists, \in$ are used only for simplicity of presentation.

Algorithm 3 The outline of the verification algorithm

```
1:  Phase 1:
2:    for k = 1, 2, ··· do
3:      Check the satisfiability of M(k) ∪ P(k)              {(1)}
4:      if Unsatisfiable then
5:        k_min ← k
6:        Break the loop
7:  Phase 2:
8:    Check the satisfiability of M(k_min) ∪ P_dτ(k_min)     {(2)}
9:    if Unsatisfiable then
10:     Output "Yes"   {Good period [τ_G, τ_G + dτ] ensures P_su.}
```

If Constraint set (1) turns out to be satisfiable, as a result of the satisfiability checking, then the satisfiability checking is repeated with $k \leftarrow k + 1$. If Constraint set (1) is unsatisfiable, on the other hand, the current k is k_{min} and Phase 1 terminates.

3.3.2 Phase 2

Once k_{min} has been obtained in Phase 1, the next phase is to determine whether *all* executions for $V(k_{min})$ implement the communication predicate by time $\tau_G + d\tau$. This can be done by checking the satisfiability of the constraint set:

$$M(k_{min}) \cup P_{d\tau}(k_{min}). \tag{2}$$

If an execution for $V(k_{min})$ exists that does not implement the communication predicate by time $\tau_G + d\tau$, then Constraint set (2) is satisfiable, because that execution satisfies both $M(k_{min})$ and $P_{d\tau}(k_{min})$. By contraposition, if Constraint set (2) is unsatisfiable, then all executions for $V(k_{min})$ implement the communication predicate by time $\tau_G + d\tau$.

If Constraint set (2) turns out to be satisfiable, however, it is not possible to immediately conclude that there is an execution that fails to implement $\mathcal{P}_{su}(\pi_0, *)$ in $I = [\tau_G, \tau_G + d\tau]$, because the satisfying assignment is not necessarily a possible execution (see Footnote 1). In this case further analysis will be needed if one wants to determine whether good period $I = [\tau_G, \tau_G + d\tau]$ is indeed insufficient for implementing the communication predicate. The procedure for this will be studied in future work.

Note that if one wants to check a different value for $d\tau$, then Phase 1 no longer needs to be executed; it suffices to repeat Phase 2 with the new value. In Section 4, we demonstrate that the upper bound on the minimum length of a sufficient good period can be obtained by iterating Phase 2 with different values for $d\tau$.

3.4. Real-Time Constraints of $M(k)$

The real-time constraints of $M(k)$ are the following.

Message types

$\forall p \in \pi_0, \forall i \in [1,k] : type^1_{p,i} \in [0,4] \wedge type^2_{p,i} \in [1, |\pi_0|] \wedge type^3_{p,i} \in [1,k]$

(Integer variable $type^j_{p,i}$ represents the j-th entry of $type_{p,i}$.)

The value of ct_p

$\forall p \in \pi_0, \forall i \in [1,k] : 0 \leq ct_{p,i} \leq m$ (m is a constant defined as $m := \lceil 2\delta + (n+2)\phi \rceil$.)

$\forall p \in \pi_0, \forall i \in [1,k-1] : \big((ct_{p,i} = m \vee r_{p,i} > r_{p,i-1}) \Rightarrow ct_{p,i+1} = 0\big) \wedge \big(\neg(ct_{p,i} = m \vee r_{p,i} > r_{p,i-1}) \Rightarrow ct_{p,i+1} = ct_{p,i} + 1\big)$

$\forall p \in \pi_0, \forall i \in [1,k] : ct_{p,i} = 0 \Leftrightarrow type_{p,i} = \perp$

Incrementing round number r_p

$\forall p \in \pi_0 : 1 \leq r_{p,0}$ (Integer variables $r_{p,0}$ represents r_p of the previous step of $step_{p,1}$.)

$\forall p \in \pi_0, \forall i \in [0,k-1] : r_{p,i} \leq r_{p,i+1}$

$\forall p \in \pi_0, \forall i \in [2,k] : type_{p,i} = \perp \Rightarrow$

$\quad \big((ct_{p,i-1} = m \wedge r_{p,i-1} = r_{p,i-2}) \Rightarrow r_{p,i} = r_{p,i-1} + 1\big) \wedge \big(\neg(ct_{p,i-1} = m \wedge r_{p,i-1} \neq r_{p,i-2}) \Rightarrow r_{p,i} = r_{p,i+1}\big)$

Highest round number first policy for message delivery

$\forall p \in \pi_0, i \in [1,k] : type_{p,i} \neq \{\perp, \langle 4 \rangle\} \Rightarrow$

$\quad \forall q \in \pi_0, \forall i' \in [1,k] : type_{q,i'} = \perp \Rightarrow (ta^p_{q,i'} > t_{p,i} \vee r_{q,i'} \leq r_{p,i} \vee \exists i'' < i : type_{p,i''} = \langle 1, q, i' \rangle)$

The properties of the receive steps of the four types

$\forall p, q \in \pi_0, \forall i, i' \in [1,k] : type_{p,i} = \langle 1, q, i' \rangle \Rightarrow (type_{q,i'} = \perp \wedge ta^p_{q,i'} \leq t_{p,i} \wedge\ r_{q,i'} = r_{p,i})$

$\forall p \in \pi_0, \forall i, i' \in [1,k] : (type_{p,i} = \langle 1, *, * \rangle \wedge type_{p,i'} = \langle 1, *, * \rangle) \Rightarrow type_{p,i} \neq type_{p,i'}$

$\forall p, q \in \pi_0, \forall i \in [1,k] : type_{p,i} = \langle 2, q \rangle \Rightarrow r_{p,i} \leq r_{q,0}$

$\forall p, q \in \pi_0, \forall i \in [1,k] : type_{p,i} = \langle 3, q \rangle \Rightarrow (t_{p,i} > t_{q,k} \wedge r_{p,i} \geq r_{q,k} \wedge p \neq q)$

$\forall p \in \pi_0, i \in [1,k] : type_{p,i} = \langle 4 \rangle \Rightarrow r_{p,i} = r_{p,i-1}$

$\forall p \in \pi_0, i \in [1,k] : type_{p,i} = \langle 4 \rangle \Rightarrow$

$\quad \forall q \in \pi_0, i' \in [1,k] : type_{q,i'} = \perp \Rightarrow (ta^p_{q,i'} > t_{p,i} \vee\ r_{q,i'} < r_{p,i} \vee\ \exists i'' < i : type_{p,i''} = \langle 1, q, i' \rangle)$

Figure 2. $M(k)$: Constraints Specifying Executions (Excluding Real-Time Properties)

Message Delay: Property 1 in Section 2.2 states that if a send step is executed at or after τ_G, then the message sent by the step is placed in $buffer_q$ at the receiver process q within time δ. This property is represented as follows:

$$\forall p \in \pi_0, \forall i \in [1,k] : \\ type_{p,i} = \perp \Rightarrow \forall q \in \pi_0 : 0 \leq ta^q_{p,i} - t_{p,i} \leq \delta$$

Step Execution Speed: Properties 2 and 3 in Section 2.2 impose the lower and upper bounds on the time between two consecutive steps executed by the same process. These bounds can be represented as follows:

$$\forall p \in \pi_0, \forall i \in [1,k-1] : 1 \leq t_{p,i+1} - t_{p,i} \leq \phi$$

The First Step Executed in the Good Period: For each process $p \in \pi_0$, $step_{p,1}$ is the first step that occurs at or after τ_G. Because of the upper bound ϕ on the step delay, $step_{p,1}$ must occur before or at time $\tau_G + \phi$:

$$\forall p \in \pi_0 : \tau_G \leq t_{p,1} \leq \tau_G + \phi$$

3.5. $P(k)$ and $P_{d\tau}(k)$: Constraints Specifying a Communication Predicate

$P(k)$ is satisfied by an execution for $V(k)$ iff the execution does not implement the communication predicate. We have:

$$P(k) := \\ \forall p \in \pi_0, \forall i \in [1,k], \exists q, q' \in \pi_0, \forall i' \in [1,k] : \\ \big((\forall i'' \in [1,k] : type_{q,i'} \neq \langle 1, q', i'' \rangle) \wedge type_{q,i'} \neq \langle 2, q' \rangle \\ \wedge type_{q,i'} \neq \langle 3, q' \rangle\big) \vee (r_{p,i} \neq r_{q,i'})$$

This formula can be explained as follows. Given an execution for $V(k)$, all round numbers occurring in that execution are represented as $\{r_{p,i} \mid p \in \pi_0, i \in [1,k]\}$. The execution fails to implement $\mathcal{P}_{su}(\pi_0, *)$ iff for any of these round numbers, say r, some process $q \in \pi_0$ exists that receives no message of round r from some process $q' \in \pi_0$. This happens iff at every step of q, that is, at $step_{q,i'}$ ($\forall i' \in [1,k]$), the received message is not from q' or the round number $r_{q,i'}$ is different from r.

$P_{d\tau}(k)$ is satisfied by an execution for $V(k)$ iff the execution does not implement the communication predicate by time $\tau_G + d\tau$. $P_{d\tau}(k)$ can be obtained by slightly modifying $P(k)$ to take time $\tau_G + d\tau$ into account as follows:

$$P_{d\tau}(k) := \\ \forall p \in \pi_0, \forall i \in [1,k], \exists q, q' \in \pi_0, \forall i' \in [1,k] : \\ \big((\forall i'' \in [1,k] : type_{q,i'} \neq \langle 1, q', i'' \rangle) \vee type_{q,i'} \neq \langle 2, q' \rangle \\ \vee type_{q,i'} \neq \langle 3, q' \rangle\big) \vee (r_{p,i} \neq r_{q,i'}) \vee (t_{q,i'} > \tau_G + d\tau)$$

4. Experimental Results

This section presents the results of experiments. All the measurements were performed using a Linux workstation

Table 2. Execution time (h:m:s) ($n = 4$, $|\pi_0| = 3$)

ϕ	δ	Phase 1	k_{min}	Phase 2 ($d\tau_1$)	Phase 2 ($d\tau_2$)	Total ($d\tau_1$)	Total ($d\tau_2$)
1.0	0.5	0:18:40	15	0:11:15	0:04:12	0:29:55	0:22:52
1.0	1.0	0:26:29	16	0:14:04	0:05:28	0:40:33	0:31:57
1.0	1.5	1:25:24	19	0:53:55	0:13:36	2:19:19	1:39:00
1.0	2.0	2:08:21	20	1:13:48	0:18:59	3:22:09	2:27:20
1.2	0.5	5:24:22	20	3:42:08	0:49:45	9:06:30	6:14:07
1.2	1.0	12:01:31	22	8:26:11	1:26:34	20:27:42	13:28:05

with an Intel Xeon processor 2.2GHz and 4Gbyte memory.

Table 2 summarizes the performance of the proposed verification method, where the execution time is shown for several combinations of the parameter values. The measurement was performed for two types of $d\tau$: $d\tau_1$ and $d\tau_2$. $d\tau_1$ is defined as follows:

$$d\tau_1 := 2(\lceil 2\delta + (n+2)\phi \rceil + 1)\phi + \delta + \phi \qquad (3)$$

This is the known upper bound on the minimum length of a $\overline{\pi_0}$-good period that allows Algorithm 2 to implement $\mathcal{P}_{su}(\pi_0, *)$ [13]. $d\tau_2$ was set to $d\tau_1/2$. In all the cases tested, our verification method confirmed that $\overline{\pi_0}$-good period $[\tau_G, \tau_G + d\tau_1]$ is sufficient to implement $\mathcal{P}_{su}(\pi_0, *)$. For $\overline{\pi_0}$-good period $[\tau_G, \tau_G + d\tau_2]$, on the other hand, no conclusive answer was obtained, since Constraint set (1) (Section 3.3.1) turned out to be satisfiable in all the cases.

As seen in Table 2, Phase 2 for $d\tau_2$ took less time to complete than in the case of $d\tau_1$. This can be explained by the fact that satisfiable instances of the satisfiability problem are usually easier to solve than those unsatisfiable, because finding a single satisfiable assignment is sufficient.

Next, we explored the upper bound on the minimum length of a sufficient good period by applying the proposed method with different values for $d\tau$, setting it first to some small value and increasing it by small steps. As stated in Section 3.3.2, only Phase 2 was needed to be iterated in this process.

Table 3 compares the obtained upper bounds and the known bounds. As shown in this table, we were successful in obtaining tighter bounds. The difference between these two bounds can be accounted for, to some extent, by the fact that the known bound does not take π_0 into consideration (see Formula (3)). In contrast, the proposed approach examines all possible executions in a given setting, thus yielding more precise results. We expect that these obtained bounds can be useful in finding a more precise formula for the upper bound.

Table 3. Upper bound on the minimum length of a good period ensuring $\mathcal{P}_{su}$ ($n = 4$, $|\pi_0| = 3$)

ϕ	δ	obtained bound	known bound [13]
1.0	0.5	13.5	17.5
1.0	1.0	15.0	20.0
1.0	1.5	17.5	22.5
1.0	2.0	19.0	25.0
1.2	0.5	19.0	25.7
1.2	1.0	21.2	28.6

5. Related Work

It is only very recently that the issue of performance of consensus following asynchronous periods has been studied. This issue was addressed in [9, 13, 14]. In [14] performance of consensus was analyzed with respect to the number of rounds, rather than time. In [9] an algorithm was proposed that reaches consensus within a constant number of message delays after the system becomes synchronous. In [13] timeliness properties of the predicate implementation were analyzed. None of the previous work discussed formal verification.

The work that seems most related to ours is that by Hendriks [12]. In [12] the UPPALL model checker [15] was used to verify the correctness of a consensus algorithm. The consensus algorithm and the underlying system model are different from ours in many respects. For example, the system model is a synchronous model where the message delay and the relative process speed are always bounded. Also, no concept similar to communication predicates appeared in [12].

The standard continuous real-time model checking is based on the well-established theory of timed automata [1]. UPPAAL is an example of a timed-automata-based real-time model checker. The verification problem we addressed in the paper involves unbounded integer variables, because a process can take an arbitrary round number at the beginning of a good period. Unbounded integer variables cannot be treated in timed automata.

In [8] and [17], different techniques were devised for continuous real-time model checking. Using the SAL model checker [6], these techniques were successfully applied to verification of the fault-tolerant start up protocol for the Time-Triggered Architecture (TTA). SAL uses

YICES [7] as a back-end solver; thus their approach is similar to ours in that the model checking problem is reduced to constraint satisfiability problems of a similar class. The TTA start up protocol is, however, completely different from the communication predicate implementation algorithm, and thus these techniques cannot be directly used in our context.

The proposed method borrows some ideas from *bounded model checking* [5]. The basic idea behind bounded model checking is to search a counterexample of length up to a given bound. Usually, this bounded version of the model checking problem is reduced to the boolean satisfiability problem (SAT). Since states are represented with boolean variables, only discrete time can be dealt with by conventional bounded model checking.

Extension of bounded model checking to continuous time was discussed in, for example, [2] and [16]. These studies adopted timed automata as the underlying computation model and thus cannot be used for our problem.

6. Conclusions

In this paper we model checked a distributed algorithm that implements a communication predicate that solves consensus. By doing this, we addressed the performance evaluation of consensus in a synchronous period following asynchronous periods. Our work is the first study to apply model checking to this issue. This model checking problem was challenging since it involved both continuous time and unbounded integers. We solved this problem by reducing it to several instances of the satisfiability problem of linear arithmetic constraints. An advantage of using model checking is that it allows fine analysis for specific parameter settings. We demonstrated this advantage through experiments, by obtaining more precise conditions for solving consensus than known before.

Future work needs to be carried out to improve the performance of verification. There are several techniques worth exploring. For example, we expect that the behavioral symmetry of processes can be exploited to reduce the solution space.

References

[1] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[2] G. Audemard, A. Cimatti, A. Kornilowicz, and R. Sebastiani. Bounded model checking for timed systems. In *Proceedings of the 22nd IFIP WG 6.1 International Conference Houston on Formal Techniques for Networked and Distributed Systems*, pages 243–259. Springer-Verlag, 2002.

[3] B. Charron-Bost and A. Schiper. Harmful dogmas in fault tolerant distributed computing. *SIGACT News*, 38(1):53–61, Mar. 2007.

[4] B. Charron-Bost and A. Schiper. The Heard-Of Model: Computing in Distributed Systems with Benign Failures. Technical report, 2007. Replaces TR-2006: The Heard-Of Model: Unifying all Benign Failures.

[5] E. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

[6] L. de Moura, S. Owre, H. Rueß, J. Rushby, N. Shankar, M. Sorea, and A. Tiwari. SAL 2. In R. Alur and D. Peled, editors, *Computer-Aided Verification, CAV 2004*, volume 3114 of *Lecture Notes in Computer Science*, pages 496–500, Boston, MA, July 2004. Springer-Verlag.

[7] B. Dutertre and L. M. de Moura. A fast linear-arithmetic solver for DPLL(T). In *Proc. of 18th Conf. on Computer Aided Verification (CAV 2006)*, volume 4144 of *LNCS*, pages 81–94, Seattle, USA, Aug. 2006. Springer.

[8] B. Dutertre and M. Sorea. Modeling and verification of a fault-tolerant real-time startup protocol using calendar automata. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 3253 of *Lecture Notes in Computer Science*, pages 199–214, Grenoble, France, Sept. 2004. Springer-Verlag.

[9] P. Dutta, R. Guerraoui, and L. Lamport. How fast can eventual synchrony lead to consensus? In *Proc. Int'l Conf. on Dependable Systems and Network (DSN 2005)*, pages 22–27, Yokohama, Japan, 2005. IEEE CS Press.

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

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

[12] M. Hendriks. Model checking the time to reach agreement. In P. Pettersson and W. Yi, editors, *3rd International Conference on the Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, volume 3829 of *LNCS*, pages 98–111. Springer–Verlag, 2005.

[13] M. Hutle and A. Schiper. Communication predicates: A high-level abstraction for coping with transient and dynamic faults. In *Proc. Int'l Conf. on Dependable Systems and Network (DSN 2007)*, pages 92–101, Edinburgh, UK, June 2007. IEEE CS Press. Full version in technical report LSR-REPORT-2006-006, EPFL, 2006.

[14] I. Keidar and A. Shraer. Timeliness, failure-detectors, and consensus performance. In *Proc. 25th ACM Symp. on Principles of Distributed Computing (PODC'06)*, pages 169–178, Denver, Colorado, USA, 2006. ACM Press.

[15] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

[16] M. Sorea. Bounded model checking for timed automata. *Electronic Notes in Theoretical Computer Science*, 68(5):116–134, 2003.

[17] W. Steiner, J. Rushby, M. Sorea, and H. Pfeifer. Model checking a fault-tolerant startup algorithm: From design exploration to exhaustive fault simulation. In *Proc. Int'l Conf. on Dependable Systems and Network (DSN 2004)*, pages 189–198, Florence, Italy, June 2004. IEEE Computer Society.