Time Cost Analysis of a Consensus Algorithm*

Julie Vachon and André Schiper

Laboratoire de Systèmes d'Exploitation
Ecole Polytechnique Fédérale de Lausanne
Lausanne, CH-1015 Switzerland
{Julie.Vachon, Andre.Schiper}@di.epfl.ch

Abstract. This paper describes the time analysis of *Chandra & Toueg*'s asynchronous consensus, an algorithm of major importance in the field of fault-tolerant distributed computing. Contrary to other consensus protocols assuming semi-synchronous or synchronous conditions, *Chandra & Toueg*'s algorithm considers an asynchronous system, thus requiring no bounds on communication delays nor on processes' speeds. Introducing an adversary model to describe the system's load, our analysis leads to the evaluation of an upper bound on the execution time of *Chandra & Toueg*'s protocol. We compare our results concerning the asynchronous consensus with the results of former works dealing with semi-synchronous agreement algorithms.
In a broader perspective, we consider this time analysis of *Chandra & Toueg*'s consensus as a first step towards the development of specialized tools for automatic time analysis and simulation of complex agreement protocols and services in distributed computing.

Keywords: Analytical technique, execution time evaluation, time upper bound, asynchronous system, agreement algorithm, fault-tolerance, distributed system.

1 Introduction

1.1 Context

Driven by the fast evolution of large scale communication networks and the need for rapid information retrieval, the interest for fault-tolerant distributed systems doesn't stop increasing. Hence, these systems are used today for databases management, business applications, distributed control software, etc. These autonomous decentralized computing entities not only need high technological means but also require the conception of high-performance reliable communication protocols for which real-time guarantees can be automatically computed.

The autonomy of today's systems as well as their decentralization both justify the choice of an asynchronous model for protocol development. By fixing no bounds on communication delays nor on the speeds of processes, the asynchronous model remains very general but unfortunately complicates the resolution of some distributed problems. To this day, most problems have been solved in a synchronous or semi-synchronous context ([AL89], [ADLS94], [Att94], [Lyn96], [Pon91]). In the face of this new kind of decentralized and autonomous system, synchronous and semi-synchronous models prove to be inadequate : it is difficult, indeed even unconceivable, to bound the interaction delays between elements which are disparate and arbitrarily distant from one another. This motivates our specific interest for asynchronous systems and for the (time) analysis of protocols implemented in this model.

1.2 Problematics

Be it for transaction or distributed control applications, fault-tolerant agreement protocols are among the major problems of interest in the community of distributed systems. Hence, we have decided to focus on the fundamental consensus problem : Given a system composed of n processes each starting with an

* Research supported by the Fonds National Suisse, under contract number 21-43196.95.

initial value, a solution for the consensus has to verify some properties concerning agreement, validity and termination. (cf. Sect. 2).

But as said previously, the asynchronous model is very general and hence, offers almost no guarantee : the system's delays are indeed finite, but remain unbounded. This model is so little constraining that some problems have no solution in this context. In fact, Fischer, Lynch and Paterson have proven the impossibility of asynchronous consensus, even when a single process can crash. This result is inherent in the impossibility to distinguish a crashed process from one that is simply slow or dependent upon a slow communication link.

However, *Chandra & Toueg* have continued to work on the problem and have identified the minimal conditions required to make the consensus solvable. Hence, for a system with a majority of correct processes and with reliable communication links, *Chandra & Toueg* have proposed in [CT96] a consensus solution using unreliable failure detectors $\diamond S$ (cf. Section 3). In short, these detectors are oracles (detecting processes faults) which exhibit the minimal completeness and accuracy properties required to ensure the consensus termination. Thus, the genius of *Chandra & Toueg*'s solution is to encapsulate the problem's indecidability within the failure detector concept.

In theory, we know that under *Chandra & Toueg*'s conditions, the general asynchronous consensus problem has a solution which does not depend on the system's load. In practice however, we can't predict to which extent these hypotheses will be verified by the execution context. On the other hand, for consensus execution which terminate, *we would like to evaluate in advance the maximal time required by the consensus protocol for a given system's load.*

1.3 Objectives of This Work

Considering the asynchronous model augmented with *Chandra & Toueg*'s conditions, how can we offer real-time guarantees on the consensus termination time? This paper specifically attempts to answer this question and exhibits the corresponding results. The consensus impossibility result in asynchronous systems ([FLP85]) will always remain; and it reminds us of the necessity to strengthen the context with new hypotheses, if we want to bound its execution time.

Inspired by techniques commonly used in game theory, our analytical approach consists in modeling the system's load by an adversary. The adversary represents the system's opposing force which tries to prevent the consensus from terminating. A consensus execution submitted to an adversary with unlimited power obviously would not terminate and no temporal bounds could therefore be computed. Hence, we propose an adversary which has some constrained power : it models a system whose load is bounded. We define the parameters bounding this adverse power and explain the optimal strategy used by the adversary to fight against the protocol's termination. *Given this adversary model, we are able to derive an upper bound on the execution time of Chandra & Toueg's consensus.*

This work lies within the scope of a larger objective aiming at automating the time analysis of various distributed protocols. Indeed, let's mention that the consensus may be considered as a basic primitive for implementing more complex protocols such as group membership, non blocking total order broadcast/multicast primitives, etc. Among others, Schiper and Guerraoui, in [GS96], explain how the consensus may be used as a building block for other fault-tolerant primitives and introduce the new concept of consensus service. Thus, the time analysis of the asynchronous consensus constitutes a first step toward the development of tools for automatic time analysis and simulation of complex agreement protocols and services in distributed computing.

The rest of this paper is structured as follows: Section 2 gives a short description of *Chandra & Toueg*'s consensus algorithm. Section 3 concentrates on the definition of the adversary model proposed to analyze time in asynchronous systems. The full time analysis of *Chandra & Toueg*'s algorithm and the upper bound derived are explained in Section 4. Section 5 treats of related works, whereas Section 6 concludes the paper and comments the results obtained.

2 Consensus and Chandra & Toueg's Algorithm

As said previously, the consensus is a well-known problem in fault-tolerant distributed computing. Given an asynchronous system comprising n processes, every process proposes a value, and all the correct ones must finally agree on one of the proposed values. A solution for the consensus has to verify the three following properties :

Agreement All processes that decide, decide the same value.
Validity If a process decides v, then v was proposed by some process.
Termination Every correct process eventually decides some value.

Chandra & Toueg have proposed an algorithm to solve the consensus problem in an asynchronous system augmented with $\diamond S$ failure detectors and in which a majority of processes is correct (cf. Section 3). Of course, their protocol verifies the later properties. Hereafter, we briefly explain *Chandra & Toueg*'s algorithm, which is given in Figure 1. All the details may be found in [CT96].

```
procedure propose(vp)
    estimatep ← vp
    statep ← undecided
    rp ← 0
    tsp → 0

Task 1
    while statep = undecided
    (* Phase 0 *)
        rp ← rp + 1
        cp ← rp mod n

    (* Phase 1 *)
    if p ≠ cp then
        send(p, rp, estimatep, tsp) to cp

    (* Phase 2 *)
    if p = cp then
        wait until
          [for a majority-1 of proc. q :
           received (q, rp, estimateq, tsq) from q]
        estimatep ← select one estimateq
          such that (q, rp, estimateq, tsq) has
          the largest tsq in the rcved msgs
        send (p, rp, estimatep) to all

    (* Phase 3 *)
    if p ≠ cp then
        wait until [received (cp, rp, estimatec_p)
                 or cp ∈ 3S]
        if [received (cp, rp, estimatec_p) from cp]
        then estimatep ← estimatec_p
            tsp ← rp
            send (p, rp, ack) to cp
        else send (p, rp, nack) to cp

    (* Phase 4 *)
    if p = cp then
        wait until
            [for majority-1 of proc. q :
            received (q, rp, ack) or (q, rp, nack)]
        if [for majority-1 of proc. q :
            received (q, rp, ack)]
        then
         broadcast(p, rp, estimatep, decide)

Task 2
    when deliver(q, rq, estimateq, decide)
        decide(estimateq)
        statep ← decide
```

Fig. 1. *Chandra & Toueg*'s consensus algorithm

Every process participating in the consensus proposes an initial value v_p and thus executes procedure *propose*(v_p). This procedure is composed of two concurrent tasks. The main task consists in a loop of which each iteration is called a round. A round r is composed of five phases and is coordinated by one of the participating processes c_r, such that $c_r := r \, mod \, n$.

Phase 0 is a short initialization phase during which each process simply updates its local variables before starting the round. The round protocol really begins with **phase 1** where each process sends, to the coordinator c_r, its proposed value ($estimate_p$) with the number of the current round (r) and a time-stamp[2]. **Phase 2** strictly concerns the coordinator c_r, which is then waiting for the propositions of

[2] Indicating the number of the round during which the process has last updated its estimate

a majority of processes. It then selects the most recent proposition (largest time-stamp) among the ones received, and sends it to all the participants. Thus, during **phase 3**, each process p either receives c_r's estimate or finally comes to suspect it (notation : $c_r \in \diamond S_p$). In the first case, a process p updates its estimate and sends an "*ack*" answer to c_r. On the contrary, if a process p suspects c_r, p sends a *nack* answer to it and starts a new round. During this time, the coordinator c_r waits, in **phase 4**, for the answers of a majority minus one (i.e. excluding itself) of processes. If at least one of these answers is a *nack*, then c_r starts a new round.[3] Otherwise, it means that c_r has received *ack* answers from a majority of processes and can therefore decide and execute a spontaneous reliable broadcast of its decision value : thus, every correct process will eventually receive this decision message and will decide on the same value. The delivery of this decision message (task 2) is concurrent to the main task (main loop, task 1).

Before going on with the next section, we introduce some terminology used in the following. We call a round r a *winning* round if its coordinator initiates a reliable broadcast of its decision value at the end of its phase 4. Otherwise, we say that r is a *losing* round. We also refer to *losing* rounds as *failed* rounds.

3 The Adversary Model

3.1 Definition

An asynchronous distributed system is one in which message transmission times and relative processes' speeds are unbounded. Also assuming potential crash failures, this kind of system indeed constitutes a very flexible and general model. In fact, it is so little constraining that no algorithm may guarantee the termination of the consensus problem in such a environment. As mentioned in the introduction, *Chandra & Toueg* have however identified the minimal conditions which must be added to the asynchronous model to solve the consensus problem. These conditions are :

1. A majority of processes is correct.
2. Communication links are reliable.
3. Each process has a failure detection module $\diamond S$, which it can consult. Failure detector $\diamond S$ ensures *strong completeness* and *eventual weak accuracy*:
   - *strong completeness*: Eventually every process that crashes is permanently suspected by every correct process.
   - *Eventual weak accuracy*: There is a time after which some correct process is not suspected by any correct process.

These assumptions ensure the termination of *Chandra & Toueg*'s consensus algorithm but still do not allow one to give an upper bound on the execution time required by the protocol. The algorithm proposed by *Chandra & Toueg* terminates of course... but when? In practice, we are not completely satisfied with simply knowing that there exists a finite time t such that all the conditions required for a decision to be broadcast (phase 4) will be met by time t [4]. We would like to define an upper bound on the value of t. This is precisely the aim of our analysis.

In fact, the system may be seen as an adversary playing against the termination of the consensus : it can cause crashes, can delay processes and communications as it pleases, therefore it can also cause the failure detectors to make errors, etc. To obtain real-time bounds on the consensus execution time, one needs to restrict this adverse power. The idea here is to estimate the load and the possible behavior of the system. More precisely, we will make some assumptions on the bounds of the system delays. Our hypothesis about the system will be gathered in a model which we will call the *adversary.* Given this model, it will be an easy matter to evaluate the maximal execution time of *Chandra & Toueg*'s protocol. The upper bound derived this way will hold granted the system behaves as specified by the adversary model.

[3] This part of the algorithm is not optimized. In fact, to prevent c_r from starting a new round simply because one process suspected it, for each process p_i, c_r should wait either to receive an *ack/nack* answer from p_i or to suspect p_i. However, we do not consider this last alternative in our paper.

[4] Even-though we consider this as an important result!

Bounding the system's parameters...

In an asynchronous system augmented by *Chandra & Toueg*'s conditions, four temporal parameters represent finite values but remain unbounded. They are the following :

- the **communication delays**;
- the **processes' speeds**;
- the **time-out value**, or *the failure detection latency*, of the suspector ($\diamond S$'s *strong completeness* ensures this is a finite value);
- **the accuracy latency** i.e. the smallest time from which some correct process p is not erroneously suspected anymore ($\diamond S$'s *eventual weak accuracy* ensures this is a finite value).

The finite but unbounded nature of the these parameters is implied 1) by the asynchronous system definition, in the case of the two first parameters; and 2) by $\diamond S$'s properties, in the case of the two last ones.

To this list, we can add a fifth parameter, the **number of crash failures**, which already has an upper limit ($\lfloor n/2 \rfloor$) but could favorably be bounded more closely.

Hence, in our model, we first propose to define bounds, $\boxed{\delta}$ and $\boxed{c}$ respectively, for the communication delays and the processes' speeds. We will also assume a certain failure pattern F in which there is $\boxed{N_f}$ ($\leq \lfloor n/2 \rfloor$) **crash failures**. N_f is thus a tighter bound than the pessimistic $\lfloor n - 1/2 \rfloor$.

Definitions, notations	Comments
$\Pi = \{p_1, ..., pn\}$	The set of processes in the system.
$\mathcal{T} = \mathcal{N}$	The range of clock ticks which is taken to be the set of natural numbers.
$F : \mathcal{T} \rightarrow 2^{\Pi}$	Failure pattern. $F(t)$ denotes the set of processes that have crashed though time t.
$crashed(F) = \bigcup_{t \in \mathcal{T}} F(t)$	The set of crashed processes in F.
$correct(F) = \Pi - crashed(F)$	The set of correct processes in F.
$H : \Pi \times \mathcal{T} \rightarrow 2^{\Pi}$	Failure detector history. $H(p,t)$ is the failure detector module of process p at time t; it is the set of processes which p suspects at time t.

Table 1. *Chandra & Toueg*'s model : definitions and notations

The same way, we propose to make some assumptions on the failure detector's history. In Table 1, we recall what is a failure detector history as well as other definitions and notations introduced by *Chandra & Toueg* in the description of their model. For the purposes of our analysis, we define $\mathcal{T}'_p$ as the set of all the initial times t_1 of every interval $[t_1, t_2]$ during which a process p interrogates its failure suspector module (c.f. phase 3 of the algorithm.).

$$\mathcal{T}'_p = \{t_1 \mid p \text{ interrogates } \diamond S_p \text{ during } [t_1, t_2],\ \ t_1, t_2 \in \mathcal{T}\}$$

Let R be the range of rounds, taken to be the set of natural numbers. Since a process consults its failure detector module at most once during a round (i.e. only in phase 3), we can define a injective function $Ro_p : \mathcal{T}'_p \rightarrow R$, which maps the beginning of each time intervals during which a process p consults its detector, to the corresponding round number.

Thus, for a given failure pattern F, we can quantify the number of false suspicions a failure detector makes in history H by evaluating the cardinality of set $\mathcal{FS}$:

$$\mathcal{FS}(F,H) = \{Ro_p(t) \mid p \in \Pi,\ t \in \mathcal{T}'_p,\ c = Ro_p(t)\, mod\, n,\ \ p \neq c\ \ .\ \ (c \notin F(t)) \wedge (c \in H(p,t))\}$$

In our model, we assume that the cardinality of $\mathcal{FS}$ is bounded by constant $\boxed{N_s}$ (even though in practice, it is not possible to bound $\mathcal{FS}$'s cardinality a priori). To ensure $\Diamond S$'s *strong completeness*, the failure detection latency must obviously be finite. Assuming that failure detectors' implementation are based on a time-out mechanism, the failure detection latency, as well as the time any process has to wait before being informed of a suspicion (following a crash or a pure timeliness fault) is thus bounded by constant $\boxed{\tau_{t\text{-}out}}$ in our model.

Remark. The number of suspicions and the *time-out* value directly depend on the the failure detectors implementation. By fixing arbitrary bounds on the number of false suspicions and on the maximal failure detection latency of the suspector, we free ourselves from implementation details and from a non negligible number of correlations between the system's temporal parameters and those of the suspector. This choice is made at the cost of a less precise analysis. But, at this stage we do not envisage considering all these correlated factors. Simulation techniques will prove to be more efficient and appropriate for a finer evaluation of these bounds.

All the assumptions just stated are summarized in the definition of our adversary model (Def. 1).

Definition 1 (Adversary) *We define $adv(N_f, N_s, c, \delta, \tau_{t\text{-}out})$as an adversary to Chandra & Toueg 's consensus algorithm, which can*

- ◇ *provoke N_f ($< n/2$) process crashes,*
- ◇ *generate N_s false suspicions (during phase 3),*
- ◇ *delay processes' step time*[5] *up to c,*
- ◇ *delay communication latency up to δ,*
- ◇ *delay failure detection latency up to $\tau_{t\text{-}out}$.*

The adversary uses its power following an optimal strategy so as to maximize the execution time of the algorithm.

This adversary specifies the expected load of the system in which *Chandra & Toueg*'s protocol is to be run. It is important to recall that our adversary is simply a model of the possible load that may be imposed by the executing system. This model resumes the hypotheses made on the behavior of the system, and by no means implies bounds on communication delays and processes' speeds in the real system running the consensus. *Chandra & Toueg*'s algorithm has been designed for an asynchronous system and does not rely on our adversary's bounds for its correctness. We know that the algorithm terminates when the theoretical requirements are met (see Section 1.2). But in practice, when does it really end? To answer this question, one need to define the system's load (which is unpredictable is asynchronous environments) as we do it with our adversary model.

...to bound the consensus execution time.

The power limitations imposed to the adversary are sufficient to bound the execution time of *Chandra & Toueg*'s algorithm. The two following lemmas are introduced to prove this.

Lemma 1 *Assuming no process has decided before round r, if the coordinator of round r is correct and no process suspects it, then a decision value is necessarily broadcast by this coordinator during round r.*

Proof : If no process suspects the coordinator of round r then all correct processes will send an *ack* answer to it during phase 3. Since, by hypothesis, there exists at least a majority minus one (the coordinator) of other correct processes which have not decided yet and since communication links are reliable, the coordinator c_r (which is correct during round r) will eventually receive an *ack* answer from at least a majority minus one of processes (but no *nack*!) and will therefore broadcast a decision value by the end (phase 4) of its round r. □

[5] Time for an elementary operation.

Lemma 2 *If message transmission delays, processes' speeds and failure detection latency are all bounded values then, for any process p, the execution time of any round r is also bounded.*

Proof :

- *Phases 0 and 1 :* Since processes' speeds are bounded, the processes execute all their elementary operations, as well as their `send` instruction, in bounded time.
- *Phase 2 :* Since transmission delays are bounded, obviously p can't block in phase 2. Indeed, since a majority of processes is assumed to be correct, the required number of estimates will reach the coordinator in bounded time. Henceforth, since the time of each step is bounded, any coordinating process takes a bounded laps of time to reach the end of phase 2.
- *Phase 3 :* The `wait` statement of phase 3 is the only instruction on which p could possibly block. But since the failure detection latency and the communication delays are bounded, p will weather receive the coordinator's proposal or suspect it after a bounded amount of time.
- *Phase 4 :* Since we assume a majority of processes is correct and since communication delays are bounded, no coordinator can block on the `wait` statement. The required *ack/nack* answers are thus received after a bounded laps of time and consequently, the **end of the round** (which is followed by the beginning of a new round or by the the reliable broadcast of a decision) **is reached in bounded time**. □

Lemma 1 implies that if a decision is not broadcast during round r (i.e. if the protocol does not terminate in round r), then one of the two following cases necessarily holds :

- Case 1. Round r 's coordinator (c_r) has crashed before or during round r;
- Case 2. There is a process suspecting coordinator c_r during round r, and c_r is correct.

In other words, an adversary that wants to prevent the protocol from terminating during round r, simply has to provoke the occurrence of case 1 or 2 during this round. We can imagine the adversary as being a card player having two kinds of cards representing cases 1 and 2. Our adversary has N_f cards of type 1 and N_s of type 2. At each round, it can play a card. Since we restrict the number of cards the adversary has in its hands (this is what we do by imposing bounds N_f and N_s), the algorithm will necessarily reach a round during which the adversary won't be able to play because it has exhausted all its cards. By Lemma 1, a decision value will then be broadcast and the algorithm will thus terminate.

But what about the duration of these rounds? Indeed, it is not worth bounding the number of rounds executed if even a single one can last an eternity! Lemma 2 answers this question : it is possible to limit the duration of a round by bounding the appropriate time parameters of the system i.e. transmission delays, processes' speeds and failure detection latency. According to Lemma 2, the bounds δ, c and $\tau_{t\text{-}out}$ that we impose to the adversary, are **sufficient** conditions to ensure the termination of each round.

3.2 Strategy

We have defined the power of our adversary. But how and when does it use it precisely? The following paragraph explains the strategy used by the adversary to maximize the execution time of *Chandra & Toueg*'s algorithm.

Definition 1 (Sect. 3.1) says that the adversary refers to an optimal strategy. A strategy is optimal if it maximizes the execution time of *Chandra & Toueg*'s algorithm. The adversary's goal is therefore to maximize, (as far as its power allows it) the number and the duration of the rounds executed before the first broadcast of a decision (cf. phase 4). Let G, described by Figure 2, be the greedy algorithm which schedules the opposing actions that undertakes an adversary $adv(N_f, N_s, c, \delta, \tau_{t\text{-}out})$. The resulting schedule $Strat$ is a sequence of events which constitutes the adversary's strategy[6].
More precisely, during a round i, an adversary takes one of the following actions :

[6] G tells the adversary which card to play at the right moment.

- ◇ $crash_i(c_i)$: The adversary provokes the crash of round i 's coordinator.
- ◇ $suspect_i(c_i)$: The adversary generates a false suspicion about the coordinator c_i : c_i is erroneously suspected by a process whose *nack* answer is received by c_i in phase 4.
- ◇ $do\_nothing_i$: The adversary undertakes no action during round i.
- ◇ end_i : The adversary stops acting on the system. Consequently, a decision value can be broadcast during round i.

Hence, an event e is a triple (act, r, t) which indicates that the adversary will undertake action *act*, associated to round r, at time t.[7] In fact, by mean of procedure *plan_action*, the algorithm G always schedules the *crash* or the *false_suspicion* actions at the most costly time of a round, so as to **maximize the duration of that round**. Moreover, and as proven by Lemma 3, an adversary adopting strategy *Strat*, defined by G, entails the failure of a **maximal number of successive rounds**.

```
procedure G (n, Nf, Ns)

   powerNf ← Nf ;
   powerNs ← Ns;
   finished ← false;
   r ← 0;
   cr ← r mod n;

   Strat ← ∅

   while not finished do
   {
      r ← r − 1;
      If the cr is UP then
         If powerNf > 0 then
         { (act, r, t) ← plan_action(crash(cr),r);
            add (act, r, t) to Strat;
            powerNf ← powerNf − 1 }
         Else
            If powerNs > 0 then
            { (act, r, t) ← plan_action(false_suspicion(cr),r);
               add (act, r, t) to Strat
               powerNs ← powerNs − 1 }
            Else
            { (act, r, t) ← plan_action(end,r);
               add (act, r, t) to Strat;
               finished ← true }
      Else
      { (act, r, t) ← plan_action(do_nothing,r);
         add (act, r, t) to Strat }
   } (* end while *)

   return(Strat)

   end (* G *)
```

Fig. 2. The adversary's strategy

[7] $(do\_nothing_{r}, r, t)$ and (end_{r}, r, t) are events which could be removed form the schedule. They are only useful to trace the behavior of the adversary at each round.

Lemma 3 *Applied by an adversary* $adv(N_f, N_s, c, \delta, \tau_{t\text{-}out})$*on an asynchronous system composed of n processes, strategy G is optimal, i.e. it maximizes the number and the duration of the rounds executed before the broadcast of a decision.*

Proof : In appendix.

3.3 Maximal Number of Rounds

Based on the adversary model and its optimal strategy G, the next lemma defines the maximal number of rounds required by *Chandra & Toueg*'s consensus.

Lemma 4 *Given an adversary* $adv(N_f, N_s, c, \delta, \tau_{t\text{-}out})$*, referring to an optimal strategy, the maximal number of rounds executed before the first broadcast of a decision value (phase 4) in Chandra & Toueg's consensus involving* n *processes is given by* R *:*

$$R = \lfloor \frac{N_s}{n - N_f} \rfloor * n \; + \; N_s \, mod \, (n - N_f) + N_f \; + \; 1$$

Proof : To delay the consensus, the adversary seeks to generate the longest sequence of *losing*[8] rounds. Following the optimal strategy G, the adversary starts by provoking the failure of the first N_f rounds, by making the corresponding coordinators crash.[9]

As illustrated by Figure 3, let's consider the rounds of the algorithm by groups of n successive rounds. Therefore, since crashed processes do not recover and since each process is periodically[10] designated coordinator, the N_f first rounds of each group (of n rounds) are all failed ones.

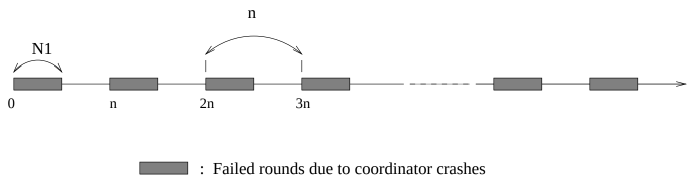

Fig. 3. Rounds failures due to coordinators crashes

To generate the longest sequence of *losing* rounds, the adversary must also try to fill the remaining subsequences with failed rounds. To do this, strategy G allows the adversary to use its power N_s and thus generate the appropriate false suspicions (one per round) that will make the first N_s remaining rounds fail (see Fig.4). The total length of the longest sequence of successive *losing* rounds therefore depends on how many subsequences of $n - N_f$ rounds the adversary can fill, using its power N_s.

In fact, it can completely fill the subsequences of $\lfloor \frac{N_s}{n - N_f} \rfloor$ groups of n processes. The last incomplete group consists of $N_f + N_s \, mod \, (n - N_f)$ failed rounds. Adding to this the final *winning* round, we conclude that the first broadcast of a decision occurs (when the adversary has no more power left) no later than round R, with R defined as follows :

$$R = \lfloor \frac{N_s}{n - N_f} \rfloor * n \; + \; N_s \, mod \, (n - N_f) + N_f \; + \; 1$$

□

[8] As mentioned at the end of Section 2, a round is *losing* if no decision is broadcast at the end of this round.

[9] i.e. just before the latest processes start their phase 3 and initiate a time-out on the coordinator.

[10] once each n rounds

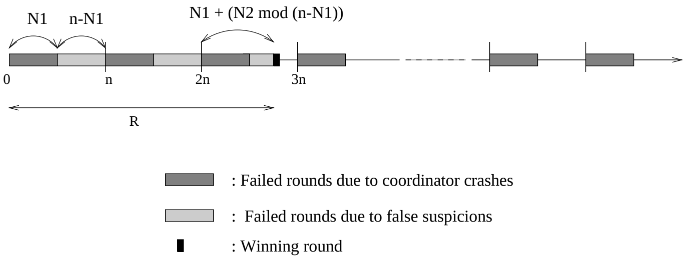

Fig. 4. Rounds failures due to false suspicions

4 Time analysis

4.1 Preliminaries

The following time analysis takes into account *Chandra & Toueg*'s initial assumptions concerning process failures, reliability of links and failure detection. In addition to these hypotheses, we also assume the following conditions :

- ◇ In the system, processes communicate by exchanging messages on a diffusion network. This choice is arbitrary but, for example, allows to consider a unique time cost for any `send` operation, whatever is the number of processors a message is addressed to.
- ◇ The reliable broadcast used in phase 4 is taken to be *Chandra & Toueg*'s spontaneous reliable broadcast primitive ([CT96]).
- ◇ All the participating processes begin the consensus at the same time t_{beg} by executing procedure $propose(v_p)$. The consensus terminates at time t_{end}, when all correct processes have decided. Our objective is to find an upper bound on $T_{consensus} = t_{end} - t_{beg}$.

To evaluate the maximal time required by *Chandra & Toueg*'s algorithm, we focus our attention on the execution of the first *winning* round's coordinator, which we denote by c_R. Hence c_R is the first[11] coordinator to broadcast a decision value. During the execution of *Chandra & Toueg*'s algorithm submitted to adversary $adv(N_f, N_s, c, \delta, \tau_{t\text{-}out})$, c_R starts ($\rightarrow t_{beg}$) procedure $propose(v_{c_r})$, goes through some initialization instructions, execute the maximal number of *losing* rounds and terminates by a final *winning* round. From then on, a reliable broadcast is initiated and the consensus is said to be terminated once all correct processes have decided ($\rightarrow t_{end}$).

Hence, according to Lemma 4 and our former comments, we can express the maximal time required by *Chandra & Toueg*'s algorithm, given an adversary $adv(N_f, N_s, c, \delta, \tau_{t\text{-}out})$, by this first formula :

$$\boxed{T_{consensus} = \tau_{init} + \Delta(R) + \tau_{bcast} + \tau_{decide}}$$

$$\begin{array}{rl} \tau_{init}: & \text{Initialization time of local variables.} \\ \tau_{bcast}: & \text{M.t.r.}^{12}\text{for all correct processes to receive the} \\ & \text{decision value broadcast.} \\ \tau_{decide}: & \text{M.t.r. by the local decision procedure of any process.} \\ \Delta(R): & \text{M.t.r. to execute R rounds, where R is the maximal number of} \\ & \text{rounds of any consensus execution submitted to } adv(N_f, N_s, c, \delta, \tau_{t\text{-}out}). \end{array} \tag{1}$$

Assuming an adversary $adv(N_f, N_s, c, \delta, \tau_{t\text{-}out})$, we can evaluate τ_{init}, τ_{bcast} and τ_{decide} by these equations :

$$\tau_{init} = 4 * c \qquad \tau_{bcast} = \delta + 2 * c \qquad \tau_{decide} = g(c) \tag{2}$$

[11] not in time, but relatively to the round number
[12] Maximal time required

We assume c_R remains correct until the end of the algorithm (see the remark below). Since c_R is correct and is the initiator of the spontaneous reliable broadcast and since links are reliable, the maximal time required for the broadcast simply consists of the time entailed to put the message on the network (c), transmitting it (δ) and waiting for its reception (c). Hence $\tau_{bcast} = \delta + 2 * c$. As for the time cost of the decision procedure, we simply express it as a function (g) of the maximal step time of a process. Finally, $\Delta(R)$ is the only parameter which still needs to be defined. This is precisely the aim of the next subsection.

Remark : Legitimately, one might ask why the adversary doesn't use its power Nf to provoke the failure of processes during the broadcast in phase 4 instead of using it during a round. The reason is simple: we are interested by a worst case analysis. For each crashed process in a broadcast, the additional time cost is $c + \delta$ (where $c \ll \delta$). Whereas, in a round, each crash necessitates the execution of a new round which costs at least $3 * \delta$.

Afterwards, we will often make reference to the maximal time required by phases 0 to 4 of a round in *Chandra & Toueg*'s protocol. Here below, we give the formulas evaluating the time cost of each individual phase. Parameter t_{wait} in t_{phase_2}, t_{phase_3} and t_{phase_4} allows to abstract, for the moment, the time spent "waiting" during a phase. As we will see later, the value of t_{wait} depends itself on the time spent in the preceding phase and on communication delays.

$$\begin{array}{ll} t_{phase_0} = 3 * c & t_{phase_3}(t_{wait}) = t_{wait} + 3 * c \\ t_{phase_1} = c & t_{phase_4}(t_{wait}) = t_{wait} + c \\ t_{phase_2}(t_{wait}) = t_{wait} + (n - f) * c + 3 * c & \end{array} \tag{3}$$

4.2 Maximal time required to execute R rounds : $\Delta(R)$

In the precedent section (see lemma 4), we have defined an upper bound R on the number of rounds required by *Chandra & Toueg*'s algorithm when opposed to an adversary $adv(N_f, N_s, c, \delta, \tau_{t\text{-}out})$. Let's consider c_R's execution. We are precisely interested in evaluating the maximal amount of time spent, by c_R, in the execution of these R rounds. Among these R rounds the first ones are *losing* rounds whereas the last is a *winning* one. Thus we will first concentrate on the time required by the *losing* rounds and then consider the final *winning* round.

Maximal time required by the coordinator of some *losing* round r to execute its first r rounds

Let $D(r)$ be the maximal time required by the coordinator of some *losing* round r to execute its r first *losing* rounds.

The evaluation of the waiting time, spent by processes in phase 2, 3 and 4, makes our analysis a little more intricate. We recall that a coordinator executes phases 0, 2 and 4, whereas the other processes execute phases 0, 1 and 3. Since we are interested by a worst case analysis, we assume that all processes are working at the minimal speed (c) and that all the communications take the maximal delay δ. Considering this, let's evaluate $D(r)$.

• **Case** $r = 1$. In the worst case scenarios and as stated by our initial hypothesis, all processes start at the same time (none of them gets ahead) and all of them execute phase 0 simultaneously. In phase 2, the coordinator of round 1, c_1, has to wait t_{wait}^{phase2} units of time to receive the other processes' estimates : that is to say the coordinator has to wait for processes to end their phase 1 and to transmit their estimate.

$$t_{wait}^{phase2} = t_{phase_1} + \delta + c$$

In phase 3, since the round considered here is a *losing* one, there exists at least one process that suspects c_1, due to a false suspicion or to the crash of c_1. Therefore, in the worst case, each non-coordinating process is informed of the suspicion by its failure detector only $\tau_{t\text{-}out}$ units of time later (we suppose $\tau_{t\text{-}out} > \delta + c$) :

$$t_{wait}^{phase3} = \tau_{t\text{-}out}$$

In phase 4, the coordinator c_1 has to wait for the other processes to execute their phase 3 and send their *nack* answer to it.

$$t_{wait}^{phase4} = t_{phase_3}(t_{wait}^{phase3}) + \delta + c = t_{phase_3}(\tau_{t\text{-}out}) + \delta + c$$

Thus, $D(1)$ can be defined by this formula :

$$\begin{aligned}
D(1) &= t_{phase_0} + t_{phase_2}(t_{wait}^{phase2}) + t_{phase_4}(t_{wait}^{phase4}) \\
&= t_{phase_0} + t_{phase_2}(t_{phase_1} + \delta + c) + t_{phase_4}(t_{phase_3}(\tau_{t\text{-}out}) + \delta + c) \\
&= t_{phase_0} + t_{phase_1} + t_{phase_2}(0) + t_{phase_3}(0) + t_{phase_4}(0) \\
&\quad + \tau_{t\text{-}out} + 2 * (\delta + c) \qquad (4)
\end{aligned}$$

• **Case** $r > 1$. When $r = 2$, the coordinator, c_2, has to wait in phase 2 for at most the time required by the latest process to transmit its estimate. The coordinator of round 1, c_1, starts sending its round 2's estimate at time $X_1 = D(1) + t_{phase_0}$ in the worst case. The other processes do it no later than time $X_2 = 2 * t_{phase_0} + t_{phase_1} + t_{phase_3}(\tau_{t\text{-}out})$. Since $X_1 \geq X_2$, c_1 appears to be the last process to send its estimate in phase 2. In fact, it is easy to see that, for each round r, the last process to send its estimate is the coordinator of the previous round, which, at its turn, lost time waiting for the slowest processes of that round, etc.

Hence, in the worst case[13] :

$$\begin{aligned}
D(r) &= t_{phase_2}[D(r-1) + t_{phase_0} + t_{phase_1} + \delta + c] + t_{phase_4}[t_{phase_3}(\tau_{t\text{-}out}) + \delta + c] \\
&= D(r-1) + t_{phase_0} + t_{phase_1} + t_{phase_2}(0) + t_{phase_3}(0) + t_{phase_4}(0) \\
&\quad + \tau_{t\text{-}out} + 2 * (\delta + c) \\
&= D(r-1) + D(1) \qquad (5)
\end{aligned}$$

This recursive equation (c.f. (4), (5)) is easily solved. We thus have :

$$\begin{aligned}
D(r) &= r * D(1) \\
&= r * [t_{phase_0} + t_{phase_1} + t_{phase_2}(0) + t_{phase_3}(0) + t_{phase_4}(0) \\
&\qquad + \tau_{t\text{-}out} + 2 * (\delta + c)] \qquad \text{for all } r \geq 1 \qquad (6)
\end{aligned}$$

We can now use D(r) to evaluate c_R's execution time.

[13] For the sake of simplicity, we make here a very pessimistic analysis when assuming the coordinator of a round must wait for the slowest process' estimate. Indeed, this is not always the case since only a majority of estimates is required (thus not necessarily including the slowest process' estimate)

Maximal time required by c_{R} to execute its R rounds : $\Delta(R)$

We first focus our attention on round R : R is a *winning* round and is coordinated by c_R, which has executed $R-1$ *losing* rounds.

Therefore, c_R is blocked in round R's phase 2 until it has receive all the required estimates. As mentioned before, in the worst case, the slowest sending process is the coordinator of the previous round. Hence, c_R waits until time $D(R-1) + t_{phase_0} + t_{phase_1} + \delta + c$.

Regarding phase 3, since round R is a winning round, processes receives c_R's estimate no later then $\delta + c$ units of time after c_R has sent it. Consequently,

$$t_{wait}^{phase3} = \delta + c$$

$$t_{wait}^{phase4} = t_{phase_4}(t_{phase_3}(\delta + c) + \delta + c)$$

The maximal time spent by c_R, in executing the R required rounds, can be evaluated by the following formula :

$$\begin{aligned}
\Delta(R) &= t_{phase_2}(D(R-1) + t_{phase_0} + t_{phase_1} + \delta + c) \\
&\quad + t_{phase_4}(t_{phase_3}(\delta + c) + \delta + c) \\
&= D(R-1) + 3 * (\delta + c) \\
&\quad + t_{phase_0} + t_{phase_1} + t_{phase_2}(0) + t_{phase_3}(0) + t_{phase_4}(0) \\
&= D(R-1) + 3 * (\delta + c) + (n + 11) * c
\end{aligned}$$

$$\boxed{\Delta(R) = R * [(n + 11) * c + 2 * (\delta + c)] + (R - 1) * \tau_{t\text{-}out} + \delta + c} \tag{7}$$

We can now give the complete formula evaluating the maximal time required to execute *Chandra & Toueg*'s consensus when opposed to an adversary $adv(N_f, N_s, c, \delta, \tau_{t\text{-}out})$. Theorem 1 is obtained by substituting equations 2 and 7 in equation 1.

Theorem 1 (Upper bound on *Chandra & Toueg*'s consensus execution time) *Let E be an execution of Chandra & Toueg's consensus algorithm in an asynchronous system composed of n processes augmented with failure detectors whose behavior in E respect $\diamond S$ properties. Given an adversary $adv(N_f, N_s, c, \delta, \tau_{t\text{-}out})$ using an optimal strategy, the maximal execution time of E is bounded by $T_{consensus}$ which is defined as follows :*

$$T_{consensus} = \tau_{init} + \Delta(R) + \tau_{bcast} + \tau_{decide}$$

$$\begin{aligned}
\tau_{init} = &\quad 4 * c \\
\tau_{bcast} = &\quad \delta + 2 * c \\
\tau_{decide} = &\quad g(c) \ \text{ where } g \text{ is the maximal time cost function} \\
&\quad \text{of the local decision procedures} \\
R = &\quad \lfloor \tfrac{N_s}{n - N_f} \rfloor * n \ + \ N_s \, mod\, (n - N_f) + N_f \ + \ 1 \\
\\
\Delta(R) = &\quad R * [(n + 11) * c + 2 * (\delta + c)] + (R - 1) * \tau_{t\text{-}out} + \delta + c
\end{aligned}$$

5 Related work : the timing-based model

5.1 Timing-based model vs asynchronous model

Real-time analysis of agreement protocols in asynchronous systems has not been much explored until now. The work of *Attiya et al.* entitled "*Bounds on the Time to Reach Agreement in the Presence of Timing Uncertainty* (cf.[ADLS94, Att94]) is probably the one whose aim is the closest to ours.

However, the comparison of our respective results remains difficult since the model and the initial assumptions considered by *Attiya et al.* are different from ours. Indeed, the algorithm introduced by these authors is written for the *timing-based model.* This model implies the following assumptions:
Note: The symbol § is used in front of variables to distinguish parameters of *Attiya et al.*'s model from our model's parameters.

1. It is assumed that the amount of time between any two consecutive process steps of any non faulty process is at least $\S c_1$ and at most $\S c_2$. It is also assumed that the time for message delivery is at most $\S d$.
2. Hence, a process is faulty if it doesn't answer in a fixed delay determined accordingly to $\S c_1, \S c_2$ and $\S d$. A faulty process may be one that crashes or one that is simply late (be it because of a slow link, etc.). All messages which might be issued by a faulty process are ignored by processes that have detected its fault. A faulty process is therefore considered as halted.
3. Fault detection is realized by a time-out mechanism. In a system with such strong constraints, perfect failure detection is obviously possible. Given bounds $\S c_1, \S c_2$ and $\S d$, a time-out may be implemented which will detect **all** (strong completeness) and **only** (strong accuracy) faulty processes (as defined by 2). In [ADLS94], the authors introduce a simple time-out strategy which is proven to respect the following properties:
   - **T1.** If any process p_i adds p_j to its list `faulty_proc` at time t, then p_j halts, and every message sent from p_j to p_i is delivered strictly before time t.
   - **T2.** There is a constant $\S\tau_{t\text{-}out}$ such that, if p_j halts at time t, then every p_i either halts or adds j to `faulty_proc` by time $t + \S\tau_{t\text{-}out}$.

The algorithm proposed in [ADLS94] ensures consensus *only between non faulty processes* i.e. processes whose answers are always received within the delay first fixed by the time-out mechanism. If a process fails to send some message or take a computation step in time, it might be considered as halted and no agreement guarantee is given concerning the possible decision this process might take.

This is indeed the weakness of the semi-synchronous model which the asynchronous model palliates to. In asynchronous systems, "pure" timing failures (i.e. not due to crashes) are authorized and **not considered as faults**. From this originates the impossibility of perfect failure detection : indeed, it is now necessary to distinguish slow processes from crashed ones and this is impossible. Hence, *Chandra & Toueg*'s asynchronous consensus allows for imperfect failure detection. To ensure the consensus resolution, as mentioned in Section 2, *Chandra & Toueg*'s algorithm requires failure detectors $\diamond S$, for which no implementation exists that can guarantee its properties. A time-out mechanism, as the one proposed in [ADLS94] for example, is a good compromise and may be used to implement these failure detectors. With an imperfect failure detection strategy based on time-outs, *Chandra & Toueg*'s algorithm ensures consensus between *all non crashed processes*, for all the executions during which the number of "pure" timing failures (false suspicions) made by the failure detection mechanism is finite (without needing to be bounded) and during which a majority of processes remains correct. Indeed, contrarily to *Attiya et al.*'s protocol, *Chandra & Toueg*'s algorithm requires a majority of correct processes, otherwise it may not terminate. This condition is essential to ensure agreement among all correct processes.

Let $\mathcal{F}$ be the sum of crashes (N_f) and suspicions (N_s) authorized in a protocol. Table 2 compares the constraints imposed on these values by *Attiya et al.*'s and *Chandra & Toueg*'s respective algorithms.

As explained in the next Subsection, *Attiya et al.*'s and *Chandra & Toueg*'s algorithms do not only differ in the model they are lying on, but also in the type of consensus each one solves.

Attiya et al.	*Chandra & Toueg*
$\mathcal{F} = N_S + N_f$	
$\mathcal{F} \leq n-1$	$\mathcal{F} < \infty$
$N_f \leq n-1$	$N_f < n/2$
$N_S \leq n-1$	$N_S < \infty$

Table 2. Faults constraints

5.2 Binary vs general consensus: extension of *Attiya et al.*'s binary consensus to multiple values

Chandra & Toueg's algorithm solves consensus among a set of processes proposing arbitrary initial values. On the contrary, *Attiya et al.*'s algorithm is binary: every participating process starts with 0 or 1 and the decision is therefore 0 or 1. *Attiya et al.*'s algorithm solves a specialized version of the consensus and thus, was allowed some optimizations. Thus, *Attiya et al.*'s algorithm (contrarily to other trivial binary consensus) has a time upper bound in which the time-out value is not multiplied by the number of timing failures $\mathcal{F}$. More precisely, $T_{\text{Att's\_BCons}} \in O(\mathcal{F}\S d + \S\tau_{t\text{-}out})$.

Near the end of [ADLS94], the authors explain how their algorithm may be extended to handle multiple initial values, i.e. not only 0 and 1. In the general consensus, each participating process p_i starts with some arbitrary value v_i. *Attiya et al.*'s extension consists in executing n binary consensus $A_1, A_2, ..., A_n$, where p_j is called the *source* of A_j. For each A_j, the source p_j proposes its initial value v_j (value associated to 0 in the binary consensus) while the other processes start with $\perp$ (value associated to 1 in the binary consensus). It is important to recall that *Attiya et al.*'s binary consensus is such that if at least one non faulty process proposes 0 (resp. $v_j \neq \perp$) then the decision is necessarily 0 (resp. v_j). Let w_j (v_j or $\perp$) be the decision of the binary consensus A_j, the decision of *Attiya et al.*'s consensus extension for multiple values is w_k such that $w_k \neq \perp$ and $\forall i < k,\ w_i = \perp$.

5.3 Time upper bounds comparison

One of the main contributions of paper [ADLS94] was to introduce an optimized binary consensus which could be solved within time $T_{\text{Att's\_BCons}} = (2\mathcal{F}-1)(\S d + \S c_2) + max\{\S\tau_{t\text{-}out}, 3(\S d + \S c_2)\}$. In other words : $T_{\text{Att's\_BCons}} \in O(\ \mathcal{F}(\S d + \S c) + \S\tau_{t\text{-}out}\ )$. Contrarily to some other trivial binary consensus algorithms, the time-out is not multiplied by the number of faults $\mathcal{F}$ in this time upper bound. But what is the time upper bound of *Attiya et al.*'s consensus when handling multiple values and how does it compare to the upper bound of *Chandra & Toueg*'s consensus execution time?

Remark:

To be compared, *Attiya et al.*'s and *Chandra & Toueg*'s general (i.e. not binary)consensus must be analyzed under common hypotheses. Thus, merging the failure constraints of Table 2 and assuming equivalent maximal system delays and time-outs, we obtain the set of constraints resumed in Table 3. We compare *Attiya et al.*'s and *Chandra & Toueg*'s time upper bounds (resp. $T_{\text{Att's\_BCons}}$ and $T_{\text{C\&T's\_cons}}$) in the context of executions respecting constraints of Table 3.

$\mathcal{F} = N_f + N_S$	$\delta = \S d$ (maximal communication delay)
$\mathcal{F} \leq n-1$	$c = \S c_2$ (maximal step time)
$N_f < n/2$	$\tau_{t\text{-}out} = \S\tau_{t\text{-}out}$ (time-out value)
$N_S < n-1$	

Table 3. Comparison context assumptions

Attiya et al.'s **general consensus** Let's assume all the A_i in *Attiya et al.*'s extended algorithm (Sect. 5.2) are run sequentially. In the worst case, we can imagine that the sources of the $\mathcal{F}$ first binary consensus A_i are faulty[14]. Hence a sequence of $\mathcal{F}$ binary consensus, with one faulty process each, needs to be executed. $A_{\mathcal{F}+1}$ will be the first binary consensus with an non-faulty source i.e. such that the decision is different from $\perp$: $w_{\mathcal{F}+1} = v_{\mathcal{F}+1} \neq \perp$. Therefore, *Attiya et al.*'s consensus extension for arbitrary values is upperly bounded by $T_{\text{Att's\_cons}} \approx \mathcal{F}(\tau_{t\text{-}out} + \delta + c) + 2(\delta + c)$.
Hence : $T_{\text{Att's\_cons}} \in O(\ \mathcal{F}(\tau_{t\text{-}out} + \delta + c)\ )$

Chandra & Toueg's **general consensus** On the other hand, let's consider *Chandra & Toueg*'s algorithm under the assumptions of Table 3. Since $\mathcal{F} \leq n-1$, $N_s \leq n-1$ and considering equation of Lemma 4, it follows that the maximal number of rounds that can be executed is $R = N_s + N_f + 1 = \mathcal{F} + 1$. From Theorem 1, we have $\Delta(R) = (\mathcal{F} - 1) * [(n + 11) * c + 2 * (\delta + c)] + \mathcal{F} * \tau_{t\text{-}out} + \delta + c$ and therefore : $T_{\text{C\&T's\_cons}} \in O(\ \mathcal{F}(\tau_{t\text{-}out} + \delta + c)\ )$

As we can see, under the restrictions of Table 3 (mostly inherited from *Attiya et al.*'s timing-based model), both algorithms have an execution time upper bound in $O(\ \mathcal{F}(\tau_{t\text{-}out} + \delta + c)\ )$. Nevertheless, *Chandra & Toueg*'s algorithm still has the advantage to ensure agreement between all uncrashed processes **including** untimely ones (contrarily to *Attiya et al.*'s).

One may remark that the n binary consensus A_i of *Attiya et al.*'s extension for multiples values can be run in parallel, thus giving an upper bound in which the time-out value is in $O(\ \mathcal{F}(\delta + c) + \tau_{t\text{-}out}\ )$. Indeed, this is what [ADLS94] suggests. Under the constraints of *Attiya et al.*'s timing-based model (which, among others, does not require a consensus protocol to ensure agreement with untimely (i.e. suspected) processes), it is possible to extract from *Chandra & Toueg*'s algorithm a new simplified protocol in which rounds could be run in parallel. A description of the parallelization of *Chandra & Toueg*'s protocol is beyond the scope of this article and shall be discussed in an other paper of the first author.

6 Conclusion

We have presented an analytical approach to evaluate an upper bound on the execution time of *Chandra & Toueg*'s asynchronous consensus. As explained, the asynchronous model advantageously offers a very general and flexible context for distributed programming. The results obtained for the asynchronous model also hold in more constraining contexts such as the one offered by semi-synchronous and synchronous systems. On the other hand, time analysis in total asynchrony is more complex and necessitates the specification of the potential system's load. Presumptions on the delays of the executing system must be set down as expected bounds. Hence, in our approach, we have proposed to describe the system's load by an adversary model. We define the power assigned to our adversary and the optimal strategy it uses to delay the consensus termination. Given this adversary model, we have shown how to evaluate an upper bound on the time requirements of *Chandra & Toueg*'s consensus.

Among other things, the following conclusions may be drawn from our analysis :

Let $\mathcal{F}$ represent the sum of processes timeliness faults (be they false suspicions or real crash detections) allowed to arise during the execution of an algorithm. Under the conditions imposed by our adversary $adv(N_f, N_s, c, \delta, \tau_{t\text{-}out})$, we have $\mathcal{F} = N_f + N_s$, where $N_f \leq \lceil n/2 \rceil$ to ensure agreement, and $N_s < \infty$ to ensure termination of the algorithm. Hence, recalling that $T_{consensus}$ is an upper bound on the time required to solve *Chandra & Toueg*'s consensus given $adv(N_f, N_s, c, \delta, \tau_{t\text{-}out})$and that R is the maximal number of rounds executed for this, based on Theorem 1 we observe that

- ⋄ $R \in O(\mathcal{F})$ (indeed, $\forall N_f, N_s,\ \exists k\ .\ R \leq k * (N_f + N_s) = k * \mathcal{F}$)
- ⋄ $T_{consensus} \in O(R * (\tau_{t\text{-}out} + 2 * (\delta + c))$.
  Consequently : $T_{consensus} \in O(\mathcal{F} * (\tau_{t\text{-}out} + 2 * (\delta + c))$.

[14] For example, $\mathcal{F}$ timeliness faults.

These results conform to what is naturally expected from an asynchronous consensus protocol (in particular, $T_{consensus} \in O((N_f + N_s) * \tau_{t\text{-}out})$). Among others, *Chandra & Toueg*'s consensus doesn't impose constant values for $\tau_{t\text{-}out}$ and N_s : these parameters depend on the implementation of the failure detectors, whose specification has advantageously been kept at an abstract level (i.e. resumed by completeness and accuracy properties). Thus, the algorithm's performances not only depend on the system (i.e. N_f, δ and c) but also on the implementation of the failure detection mechanism. Obviously, *Chandra & Toueg*'s algorithm doesn't pretend to be optimized. In fact, randomized algorithms ([Bra87]) or optimized protocols written considering the presence of timing uncertainty ([AL89]) may show as good or better performance results under some specific conditions. But as we recall, *Chandra & Toueg*'s algorithm is more general since it is based on the pure asynchronous model (thus making no assumption on the executing system's delays) and since termination and agreement do not rely on failure detectors' implementation matters. Their work was a main contribution to the theory and development of asynchronous algorithms and to the formalization of indecidability in the asynchronous model. Optimized versions for specific execution contexts can also be derived from *Chandra & Toueg*'s basic consensus algorithm. Hence, the analysis we have presented here can be seen as a first step towards the time analysis of other protocols based on *Chandra & Toueg*'s general consensus algorithm and the use of failure detectors.

Moreover, this analytical exercise has proven very formative. It has helped us to identify and understand the various parameters influencing the behavior of *Chandra & Toueg*'s algorithm. It has also brought to light the limitations of the analytical approach to model complex systems with non deterministic comportment, probabilistic and interdependent parameters, etc. Indeed, to come to a tractable analytical model, many simplifications have been made, the first of them being to concentrate our efforts on a *worst case* analysis. Consequently, we have been assuming processes to work at the same minimal speed and all communications to take the same maximal time. Also for the sake of simplicity, we have restricted ourselves to consider a trivial and non optimized version of *Chandra & Toueg*'s algorithm. We have put aside detectors' implementation matters (by abstracting N_s and $\tau_{t\text{-}out}$), limiting our considerations to theoretical assumptions about failure detectors' history.

Therefore, after exploring the analytical approach, we now feel the need to tackle the problem with more precision. Hence, to come to more detailed observations, we plan to adopt a simulation approach. Indeed, simulation allows to push away some of the problem's complexity limits since it doesn't require general mathematical formulas to express complex behaviors. Simulation offers appropriate tools to take into account probabilities and interdependence properties.

To conclude, we are convinced that the analytical model presented here and the simulation model envisaged will both provide helpful tools to obtain real-time guarantees not only for *Chandra & Toueg*'s consensus, but also for the numerous asynchronous algorithms (cf.[GS96]) which can be built on this fundamental agreement protocol.

References

[ADLS94] Hagit Attiya, Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. *Journal of the ACM*, 41(1):122–152, January 1994. A preliminary version appeared in the Proceedings of the 23rth ACM Symposium On Theory Of Computing, New York, 1991, pp. 359-369.

[AL89] Hagit Attiya and Nancy Lynch. Time bounds for real-time process control in the presence of timing uncertainty. In *Proceedings of the 10th IEEE Real-Time Systems Symposium*, pages 268–284, New York, 1989.

[Att94] Hagit Attiya. Time bounds for decision problems in the presence of timing uncertainty and failures. Technical report, CS Department, The Technion, Haifa 32000, Israel, November 1994. A preliminary version appeared in the Proceedings of the 7th International Workshop on Distributed Algorithms, September 1993, Lausanne, Switzerland, LNCS no.725, pp. 204-218.

[Bra87] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and computation*, (75):130–143, 1987.

[CT96] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for asynchronous systems. *Journal of the ACM*, March 1996.

[FLP85] M.J. Fischer, Nancy Lynch, and M.S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[GS96] Rachid Guerraoui and André Schiper. Consensus service: a modular approach for building agreement protocols in distributed systems. In *IEEE 26th Int Symp on Fault-Tolerant Computing (FTCS-26)*, pages 168–177, June 1996.

[Lyn96] Nancy Lynch. *Distributed Algorithms.* Data Management Systems Serie. Morgan Kaufmann Publixhers, Inc., 340 Pine Street, San Francisco, CA, 1996.

[Pon91] Stephen Ponzio. Consensus in the presence of timing uncertainty: omission and byzantine failures. In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*, pages 125–138, August 1991.

Appendix

A Proofs

Lemma 3 *Applied by an adversary* $adv(N_f, N_s, c, \delta, \tau_{t\text{-}out})$*on an asynchronous system composed of n processes, strategy G (Fig. 2) is optimal, i.e. it maximizes the number and the duration of the rounds executed before the broadcast of a decision.*

Proof : Each round i is associated an event e_i. The action associated to each e_i is attributed a minimal cost given by function C :

$$C(crash_i(c_i)) = C(suspect_i(c_i)) = 1 \text{ and } C(do\_nothing_i) = 0$$

G's optimality is proven by contradiction. Let's G be a non optimal strategy. Procedure *plan_action* (Fig. 2) guarantees the time cost maximization of each round. Therefore, if G is not optimal it is because it doesn't **maximize the number of rounds executed**.

Let $\sigma = e_1, e_2, ...e_N$ be the events scheduled by G and $\sigma' = e'_1, e'_2, ...e'_{N'}$ be the ones scheduled by an optimal strategy G'. Given adversary $adv(N_f, N_s, c, \delta, \tau_{t\text{-}out})$, both strategies must respect the following conditions :

$$\forall i < N,\ i' < N'\ \ e_i \neq end_i,\ e'_{i'} \neq end_{i'}.\text{ And } e_N = end_N,\ e'_{N'} = end_{N'} \tag{8}$$

$$\forall i < N',\ \ e'_i \neq end_i \text{ and } e_N = end_N \tag{9}$$

$$\sum_{\substack{e_i = crash_i(c_i) \\ 1 \le i \le N}} C(e_i) = \sum_{\substack{e'_i = crash_i(c_i) \\ 1 \le i \le N'}} C(e'_i) = N_1 \tag{10}$$

$$\sum_{\substack{e_i = suspect_i(c_i) \\ 1 \le i \le N}} C(e_i) = \sum_{\substack{e'_i = suspect_i(c_i) \\ 1 \le i \le N'}} C(e'_i) = N_2 \tag{11}$$

If G is not optimal then $N < N'$ and there exists at least a round $i \le N$ such that $e_i \neq e'_i$ (at least for $i = N$)

Let (e_k, e'_k) be the first pair of events such that $e_k \neq e'_k$.

- Case 1 : $e_k = do\_nothing_k$. G compels the adversary to remain inactive only if the coordinator of the current round is already down. Consequently : $\exists j < k,\ e_j = e'_j = crash_j(c_k)$. If $e'_k = suspect_k(c_k)$ or $e'_k = crash_k(c_k)$ then strategy G' makes the adversary use one unit of its power for nothing, since the coordinator is already down and the round is therefore automatically failed. Hence, this contradicts G''s optimality.
- Case 2 : $e_k = crash_k(c_k)$. According to G, an adversary attempts to provoke the crash of a process only if it is up still[15] : $\forall j < k,\ e_j = e'_j \neq crash_j(c_k)$. If $e'_k = do\_nothing_k$ then the round is not failed and a decision value can be broadcast since the coordinator is up and no false suspicion is generated. Hence a contradiction : $e'_k = end_k$ with $k < N < N' \Rightarrow k \neq N'$ (contradiction of condition 9). Otherwise, suppose $e'_k = suspect_k(c_k)$ and let $j = k + n$ be the next round coordinated by c_k. G' will provoke the failure of round j by using the adversary's power N_1 or N_2, whereas G will have nothing to do since c_i has already been crashed during round $k < j$. Strategy G' 's optimality is again contradicted.

[15] Obviously!

- $\diamond$ Case 3 : $e_k = suspect_k(c_k)$. According to G, an adversary generates a false suspicion only once it has completely exhausted its power N_1. Since G' has behaved the same way as G up to event k, it has also exhausted the adversary's power N_1. Therefore, $e'_k \neq crash_k(c_k)$. On the other hand, if $e'_k = do\_nothing_k$, we are lead to the same contradiction as the one in case 2 : $e'_k = end_k$ with $k < N < N' \Rightarrow k \neq N'$!
- $\diamond$ Case 4 : $e_k = end_k$ where $k = N$. Strategy G ends when it has exhausted all the power of the adversary. Since $\forall k < N,\ e_k = e'_k$, strategy G' has no more power either to use. Therefore, $e'_k = end_k$.

By contradiction, it follows that $\forall k \leq N,\ e_k = e'_k$ and $N = N'$. Consequently, strategy G maximizes the number of rounds executed before the first broadcast of a decision value. G **is optimal.** $\square$