

Group membership failure detection: a simple protocol and its probabilistic analysis

This article has been downloaded from IOPscience. Please scroll down to see the full text article.

1999 Distrib. Syst. Engng. 6 95

(<http://iopscience.iop.org/0967-1846/6/3/301>)

View [the table of contents for this issue](#), or go to the [journal homepage](#) for more

Download details:

IP Address: 38.107.179.211

The article was downloaded on 20/02/2012 at 07:55

Please note that [terms and conditions apply](#).

Group membership failure detection: a simple protocol and its probabilistic analysis

M Raynal and F Tronel

IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France

E-mail: raynal@irisa.fr and ftronel@irisa.fr

Received 6 September 1999

Abstract. A *group membership failure* (in short, a *group failure*) occurs when one of the group members crashes. A *group failure detection protocol* has to inform all the non-crashed members of the group that this group entity has crashed. Ideally, such a protocol should be *live* (if a process crashes, then the group failure has to be detected) and *safe* (if a group failure is claimed, then at least one process has crashed).

Unreliable asynchronous distributed systems are characterized by the impossibility for a process to get an accurate view of the system state. Consequently, the design of a group failure detection protocol that is both safe and live is a problem that cannot be solved in all runs of an asynchronous distributed system.

This paper analyses a group failure detection protocol whose design naturally ensures its liveness. We show that by appropriately tuning some of its duration-related parameters, the safety property can be guaranteed with a probability as close to one as desired. This analysis shows that, in real distributed systems, it is possible to achieve failure detection with a negligible probability of wrong suspicions.

1. Introduction

A *group* is a set of processes[†] that cooperate to implement a given task. This concept is used in several domains such as distributed systems, parallelism and cooperative support for cooperative work (CSCW). Let us consider, as an example, the distributed system context: in that context the group concept is usually used to make a service fault-tolerant (this is done by coordinating a group of processes duplicated on different nodes [12]). In the CSCW context, the group concept is usually used to define a set of workstations (users) that, during some time, cooperate to realize a common job.

Here we are interested in groups, and more specifically in *static* groups. A group is static if its membership does not change during its lifetime: there is neither leave nor join operation with respect to a group. However, a process can crash. In that case we consider that the group to which it belongs also crashes, this is a *group membership failure* (in short, a *group failure*). A new group has to be defined[‡].

The aim of this paper is to present and to analyse a protocol that detects the failure of a group. This protocol (which has been designed and implemented in our research group in the context of a CSCW system [10]) is very simple.

[†] The term *process* refers to any abstract entity such as a node, a site, a physical process, a software component, an object, etc.

[‡] The problem of defining a new group is not addressed in this paper. Usually, the new group includes all the non-crashed processes of the previous group. Moreover, the composition of the new group is usually called the new view of the group [1].

It is based on ‘silence propagation’ and uses two durations (namely, T_e and T_r). The protocol has three rules. The first rule is: every T_e time units, each process broadcasts an ‘*I am alive*’ message to all the other group members. The second rule is: if a process p has not received an ‘*I am alive*’ message from some process for T_r time units, then process p becomes silent, i.e. it ceases to send ‘*I am alive*’ messages. Note that if a process crashes it becomes silent; due to the second rule, this ‘silence’ propagates through the whole group, and eventually, each process becomes silent. Finally, the third rule defining the protocol is the following: when a non-crashed process becomes silent, it suspects something went wrong and claims a *group failure*.

Ideally, any group failure detection protocol should ensure the following two properties. (1) A liveness property: namely, if a process crashes, then all the non-crashed processes have to claim a group failure. (2) A safety property: namely, if a group failure is claimed, then at least one process of the group has crashed.

It is easy to see that the previous protocol exhibits different behaviours according to the values of T_e and T_r . When these are set to (arbitrary but) finite values, the protocol always satisfies the liveness property. When $T_r = +\infty$, the protocol satisfies the safety property but cannot satisfy the liveness one. In the following we assume that T_r has a finite value, so liveness is ensured. In that case, the fact that the safety property is satisfied depends on the underlying system model. Here we consider a variant of the *timed*

asynchronous system model [3]. This variant is characterized by the following points: (1) processes have access to local clocks with bounded drifts with respect to real time, and (2) process speeds and message delays are arbitrary. In such a context, due to slow messages, to slow processes or to a bad choice of the values of T_e and T_r , it is possible that processes claim a group failure while actually there is none (this is called a *false* failure detection). When this occurs, the safety property associated with the group failure detection problem is violated by the protocol. More generally, it appears that, for a given execution, the satisfaction of the safety property depends not only on the values of T_e and T_r but also depends strongly on the actual pattern of process speeds and of message transfer delays.

So, given a timed asynchronous distributed system, the problem is to define a relation on T_e and T_r such that the probability [4, 11, 13] of detecting a false group failure is as close to zero as desired. We address this problem in the paper. Section 2 introduces the underlying system model and section 3 presents the protocol. Then, section 4 provides a probabilistic evaluation of the group failure detection protocol. This evaluation relies on a probabilistic model for message transfer delays (the distribution function of transfer delays is assumed to be known[†]). Section 5 presents numerical results. Finally, section 6 concludes the paper.

The main lesson learned from this study is that, in real systems, a very simple protocol can achieve group failure detection with a negligible probability of wrong suspicions. This shows that liveness and safety are not necessarily antagonistic properties when one has to implement them in a practical context.

2. System model

2.1. Timed asynchronous systems

The underlying system is composed of a finite set of n processes $\{p_1, \dots, p_n\}$. A process may fail by crashing. When it crashes it definitively stops its activity. Processes synchronize and cooperate by exchanging messages through a communication network. Each node has a local memory and a hardware clock. Such a clock has a bounded drift with respect to real time. Process clocks are not synchronized. Process speeds and communication delays are arbitrary (this includes the case of message loss[‡]). So, the system is *asynchronous*.

Although process speeds and message transfer delays are arbitrary, *most of the time* one can rely on an upper bound for transfer delays. When these bound values are actually satisfied, then the system behaves as a synchronous system. This system model can be seen as a simple variant of the *timed asynchronous system model* [3] (which additionally considers process recoveries and performance failures).

2.2. An impossibility result

Asynchronous systems are characterized by an impossibility result, usually called the ‘FLP result’ (due to Fischer

[†] In fact, only the standard deviation has to be known.

[‡] A lost message is considered as having an infinite transfer delay.

et al) [5]. This result states that it is not possible to design a deterministic *consensus* protocol in such a system if even a single process can crash [5]. The consensus problem is a fundamental agreement problem to which a lot of practical problems can be reduced (e.g., Atomic Broadcast [2], Atomic Multicast [6], View Synchrony [8], Non-Blocking Atomic Commitment [8, 9]).

Intuitively, this impossibility result comes from the fact that it is impossible in an asynchronous distributed system to distinguish a crashed process from a process that is very slow or from a process with which communications are very slow. (It is for this reason that in some systems a process suspected to have crashed can be expelled from a group [1].)

FLP is usually interpreted as ‘*given a problem P , there is no protocol to solve it*’, or as ‘*it is impossible to design a protocol that ensures the liveness property associated with P* ’. Actually, FLP can be interpreted in a slightly different way: ‘*it is impossible to design a protocol that ensures both the liveness property and the safety property associated with P* ’. For a given problem P , this gives rise to the design of protocols that trade safety against liveness or to protocols that trade liveness against safety. The former approach has been explored in [2, 6, 8]. The latter approach is the one adopted in this paper.

In our context, the FLP impossibility result translates in the following way: there is no group failure detection protocol that is both live and safe. As we assume T_r is finite (i.e., the protocol is live), this means that the protocol cannot guarantee safety in all circumstances (i.e., in any pattern of process crashes and message transfer delays). The next section provides an in-depth mathematical analysis that allows one to quantify the probability of false failure detection occurrences. Let us call (H) the assumption that the patterns of process crashes and message transfer delays always allow the protocol to satisfy the safety property. This probabilistic analysis can be seen as an analysis of the coverage of (H) as defined by [11].

3. The algorithm

The algorithm is detailed by figure 1. We assume a reliable FIFO channel[§]. Moreover, we assume that we have access to the reception date of each message. This is the date on which the underlying layer effectively received the message, and not the date on which it has been delivered to our algorithm. Indeed, one can imagine that the lowest communication layer does not provide any ordering (or reliability) capabilities (this is typically the case if one uses UDP or Multicast IP).

A brief description of the different variables used by the algorithm is given by table 1.

One can see that each process executes a *while* loop until a group failure is detected by at least one process. Indeed, this can happen when a process has crashed or when a process (or a message) was too slow. Because we assume an asynchronous system these situations are not to be excluded, and moreover, cannot be distinguished. In fact this is not a real problem, because we do not want slow processes to remain within the group. They typically miss an important

[§] This assumption can be easily implemented by an underlying layer.

```

Group Failure Detection
begin
(1)  $groupFailure_i \leftarrow false$ ;  $currentDate \leftarrow getCurrentDate()$ ;
    $nextBroadcast_i \leftarrow currentDate$ ;  $timeout_i \leftarrow [+∞, +∞, \dots, +∞]$ ;
    $nextTimeout_i \leftarrow \min(timeout_i)$ ;  $r_i \leftarrow [0, \dots, 0]$ ;
    $b_i \leftarrow 0$ ;
(2) while ( $\neg groupFailure_i$ ) do
(3)   wait until ( $nextTimeout_i$  or receive a message  $m = (Alive_j, r_i[j])$  from  $p_j$  or  $nextBroadcast_i$ )
(4)   case receive a message  $m = (Alive_j, r_i[j])$ 
(5)      $receptionDate \leftarrow m.getReceptionDate()$ ;
(6)      $timeout_i[j] \leftarrow receptionDate + T_r$ ;
(7)      $nextTimeout_i \leftarrow \min(timeout_i)$ ;
(8)      $r_i[j] \leftarrow r_i[j] + 1$ ;
(9)   case  $nextBroadcast_i$ 
(10)    broadcast( $Alive_i, b_i$ );
(11)     $nextBroadcast_i \leftarrow nextBroadcast_i + T_e$ ;
(12)     $b_i \leftarrow b_i + 1$ ;
(13)   case  $nextTimeout_i$ 
(14)     $groupFailure_i \leftarrow true$ ;
(15) done
end

```

Figure 1. Group failure detector algorithm.

Table 1.

$groupFailure_i$	A Boolean set to <i>false</i> as long as no group failure is detected
$nextBroadcast_i$	Date of the next 'Alive' message to be broadcast
$timeout_i$	A real vector clock that stores for each process the date before which next awaited 'Alive' message must be received
$nextTimeout_i$	Next timeout to expire
r_i	A logical vector clock that stores for each process the timestamp of the next 'Alive' message to be received
b_i	A counter that gives the timestamp of the next 'Alive' message to be broadcast

number of deadlines, lose too many messages, and therefore increase the number of retransmissions. This leads to a poor overall performance of fault-tolerant algorithms.

Let us remark that because the vector clock $timeout_i$ is initialized to $[+\infty, \dots, +\infty]$, processes that are slow to start or processes that do not start simultaneously are not an issue. No group failure will be claimed in this case. On the other hand, initially dead processes are not detected. Within the loop, processes are waiting for three kinds of events:

- (i) Expiration of the next deadline $nextBroadcast_i$.
- (ii) Expiration of the next deadline $nextTimeout_i$.
- (iii) Reception of an awaited 'Alive' message from a process p_j .

In the first case, p_i simply broadcasts its b_i th 'Alive' message, adds T_e to $nextBroadcast_i$ and finally increments b_i and goes to sleep. In the second case, a deadline has expired. A group failure has been detected, $groupFailure_i$ is set to *true*. By contamination every process will do the same. In the third case, p_i has received the next awaited 'Alive' message from p_j before expiration of the deadline (actually, $timeout_i[j]$ for this message). p_i updates $timeout_i[j]$ by adding t_r to the date of reception of m , $r_i[j]$ is incremented, and $nextTimeout_i$ is computed by taking the minimum over $timeout_i$.

This algorithm is actually implemented in a platform, called Argo, and developed in our team. It is an object oriented fault-tolerant platform, written in Java, that provides total order broadcast within a set of processes. The group of processes is allowed to grow by the joining of new processes,

and to shrink by the deliberate departure of correct processes, or exclusion of suspected crashed processes (or supposed to be crashed). Changes of membership are also ordered like messages (view synchrony). Ordering is provided by a timed version of Chandra–Toueg consensus algorithm. Failure detection is ensured by an improved version of this algorithm, in the sense that we do not rely on special 'Alive' messages, but rather on the control messages that are already periodically broadcast to ensure consistency of decisions. Moreover, we relax the assumption on FIFO ordering of 'Alive' messages. Indeed, this assumption simply makes a probabilistic analysis of the algorithm behaviour easier, but also makes its implementation more complicated. Thus, whereas in this paper 'successive' is related to FIFO ordering, in the implementation we only require that two successive (reception order) 'Alive' messages are not separated from more than T_r times units. Because this assumption is weaker than FIFO ordering, bounds on false failure detection given in this paper are still valid for our implementation.

4. Probabilistic analysis

4.1. Key parameters

The aim of this analysis is to show that it is possible to tune the algorithm parameters so as to reduce the probability of false group failure detections down to an *a priori* given rate. This probability depends on the following fundamental parameters:

T_e = the amount of time between two successive broadcasts.

T_r = the maximal amount of time between the receptions of two successive broadcasts.

$$\Delta = T_r - T_e.$$

Z = the random variable ‘transfer delay of a message’.

f = the density function associated with Z .

$\mathbf{V}Z$ = the standard deviation of Z .

$(\lambda = \frac{\mathbf{V}Z}{\Delta^2})$ = the tuning parameter (an upper bound on the probability of one false failure detection within a group of two processes).

From the point of view of a process, the time interval between two successive broadcasts is called a *round*; its length is equal to T_e time units. We perform a probabilistic analysis of the algorithm behaviour with respect to a fixed number of rounds. This analysis computes the probability of false group failure detection.

- (i) We first show that Δ has to be positive, for the protocol to exhibit a meaningful behaviour.
- (ii) Moreover, we show that the protocol behaviour strongly depends on the difference of transfer delays between two successive messages[†]. In particular, we show that a false group failure is detected if the transfer delays of two such messages are separated by more than Δ time units. This condition is referred to as condition (4) in the following.
- (iii) Then, we compute the probability P_{couple} for condition (4) to be violated for a given pair of processes during a round.
- (iv) From P_{couple} we deduce the probability P_{group} for condition (4) to be violated within the entire group.
- (v) Then, we derive a lower bound on the probability of the following event: ‘there is no false group failure detection during at least r rounds’.
- (vi) Finally, we give a lower bound on the mean number of rounds without false group failure detection.

4.2. Analysis

Notations. To analyse the algorithm we introduce the following auxiliary notations. Dates refer to real time.

E_i^α = the date of the α th broadcast by the process p_i .

$R_{i,j}^\alpha$ = the date of reception by p_j of the α th broadcast issued by p_i .

$\delta_{i,j}^\alpha$ = the transmission delay of the α th broadcast from p_i to p_j .

Condition associated with a false group failure detection.

Let us assume that there exists a group \mathcal{G} at real time t_0 . By definition of E_i^α and T_e , one has

$$\forall \alpha \geq 0 \quad \forall i \in \mathcal{G} \quad E_i^{\alpha+1} = E_i^\alpha + T_e. \quad (1)$$

Of course, because we are dealing here with asynchronous systems, it is not possible to enforce a strict respect of scheduling. That means that it is not always possible to broadcast a message exactly T_e time units after the previous one. However, because our approach is probabilistic, it is

[†] Let us remind ourselves that transfer delays cannot be precisely determined in asynchronous distributed systems.

possible to take the jitter associated with the OS scheduler within transmission delays.

Moreover, because T_r is the maximal amount of time between the reception of two successive broadcasts, one has

$$\forall \alpha \geq 0 \quad \forall (i, j) \in \mathcal{G}^2 \quad R_{i,j}^{\alpha+1} \leq R_{i,j}^\alpha + T_r. \quad (2)$$

The algorithm claims a *group failure* when the previous condition is not satisfied. Furthermore, by definition of $\delta_{i,j}^\alpha$, one has

$$\forall \alpha \geq 0 \quad \forall (i, j) \in \mathcal{G}^2 \quad R_{i,j}^\alpha = E_i^\alpha + \delta_{i,j}^\alpha. \quad (3)$$

Thus it is possible to redefine condition (2) according to (1) and (3). So we get

$$\boxed{\forall \alpha \geq 0 \quad \forall (i, j) \in \mathcal{G}^2 \quad \delta_{i,j}^{\alpha+1} - \delta_{i,j}^\alpha \leq \Delta} \quad (4)$$

One can see that if T_r and T_e are chosen such that $\Delta \leq 0$, the algorithm exhibits a meaningless behaviour. (Indeed, in this case condition (4) is equivalent to stating that the sequence $(\delta_{i,j}^\alpha)_{\alpha \in \mathbb{N}}$ is monotonically decreasing.) In the following we will consider that $\Delta > 0$.

Taking into account message transfer delays. The previous discussion has not taken into account the probabilistic model that can be associated with the behaviour of the underlying system. We now consider the existence of a distribution function f associated with the transfer delay Z of messages. It means that a message m sent by a process p_i to a process p_j is subject to a transfer delay of less than x times units with probability

$$F(x) = \int_0^x f(y) dy.$$

We assume that channels are not lossy. This assumption can be formally stated as

$$\lim_{x \rightarrow +\infty} F(x) = 1.$$

Functions f and F are meaningless for negative values. In order to simplify further mathematical developments, we extend them over \mathbb{R} by considering: $\forall x < 0$, $f(x) = F(x) = 0$.

Let X be the random variable measuring the ‘difference of transfer delays between two message transmissions’. Let us associate with X the following density function g :

$$P[X \leq x] = G(x) = \int_{-\infty}^x g(y) dy.$$

4.2.1. The probability of a false group failure detection for a two-process group.

We want to evaluate the probability that condition (4) be violated for a pair of processes (i.e. for a group of two processes). Let it be P_{couple} . It means that we are interested in associating an upper bound with $P_{couple} = P[X \geq \Delta]$. This probability is given by formula (5). To compute it, we use some preliminary lemmas whose proofs are given in the appendix.

Lemma 4.1 states that g can be derived from f as follows.

Lemma 4.1.

$$\forall y \in \mathbb{R} \quad g(y) = \int_{-\infty}^{+\infty} f(z)f(z-y) dz.$$

Because g is an even function, the following lemma holds.

Lemma 4.2.

$$\forall y \in \mathbb{R} \quad g(y) = g(-y).$$

The following lemma is a direct consequence of the previous one.

Lemma 4.3.

$$\forall y \in \mathbb{R}^+ \quad P[0 \leq X \leq y] = P[-y \leq X \leq 0].$$

Corollary 4.1.

$$P[X \leq 0] = G(0) = \frac{1}{2}.$$

The following lemma is a consequence of lemma 4.1.

Lemma 4.4.

$$EX = 0 \quad \text{and} \quad VX = 2VZ.$$

One can apply the Chebyshev inequality to the random variable X :

$$\forall c > 0 \quad P\left[|X - EX| \geq c\sqrt{VX}\right] \leq \frac{1}{c^2}.$$

Let us choose $c = \frac{\Delta}{\sqrt{2VZ}}$. When $c > 1$, due to lemma 4.4, it is possible to simplify the previous expression which becomes

$$P[|X| \geq \Delta] \leq \frac{1}{c^2}.$$

Moreover,

$$\begin{aligned} P[X \geq \Delta] &= 1 - P[X \leq \Delta] \\ &= 1 - P[X \leq -\Delta] - P[|X| \leq \Delta] \\ &= 1 - \underbrace{P[X \leq 0]}_{\frac{1}{2}} - P[0 \leq X \leq \Delta] \\ &= \frac{1}{2} - \frac{1}{2}P[|X| \leq \Delta] \\ &= \frac{1}{2} - \frac{1}{2}(1 - P[|X| \geq \Delta]) \\ &= \frac{1}{2}P[|X| \geq \Delta]. \end{aligned}$$

Thus, an upper bound for P_{couple} is $\frac{1}{2c^2} = \lambda$:

$$\boxed{P_{couple} \leq \lambda}. \quad (5)$$

The probability of a false group failure detection for an n -process group. We have an upper bound for P_{couple} , the probability of breaking condition (4) during a round for a given couple of processes. We consider that two violations of condition (4) by two different pairs of processes are independent events[†]. Consequently, it is possible to find an expression (that is function of P_{couple}) giving the probability P_{group} that condition (4) be violated within the entire group during a single round. Indeed, P_{group} is a disjunction of independent cases.

[†] When false failure detections are considered.

- Either a single (directed) pair among[‡] the $n^2 - n$ possible pairs of processes violates condition (4) while this condition is not violated by the $n^2 - n - 1$ others.
- Either two among the $n^2 - n$ possible pairs of processes violate condition (4) while this condition is not violated by the $n^2 - n - 2$ others.
- And so on

Consequently,

$$\begin{aligned} P_{group} &= \sum_{1 \leq i \leq n^2 - n} C_i^{n^2 - n} P_{couple}^i (1 - P_{couple})^{n^2 - n - i} \\ &= \sum_{0 \leq i \leq n^2 - n} C_i^{n^2 - n} P_{couple}^i (1 - P_{couple})^{n^2 - n - i} \\ &\quad - (1 - P_{couple})^{n^2 - n} \\ &= 1 - (1 - P_{couple})^{n^2 - n}. \end{aligned}$$

We want to show that

$$\boxed{P_{group} \leq (n^2 - n)\lambda}.$$

Let n be a positive integer and $l(x) = (1 - x)^n - (1 - nx)$ the continuously differentiable function defined over $[0, 1]$. One has $l'(x) = -n(1 - x)^{n-1} + n$. Thus

$$\begin{aligned} l'(x) &\geq 0 \\ -(1 - x)^{n-1} + 1 &\geq 0 \\ (1 - x)^{n-1} &\leq 1. \end{aligned}$$

So, l is monotonically increasing over $[0, 1]$ and, because $l(0) = 0$, l is positive for $x \leq 1$. That is why $P_{group} = 1 - (1 - P_{couple})^{n^2 - n} \leq (n^2 - n)P_{couple} \leq (n^2 - n)\lambda$.

Let us denote by P_i the probability that condition (4) is violated within the group after exactly i rounds:

$$P_i = (1 - P_{group})^i P_{group}. \quad (6)$$

We can now compute the probability P_r^* that that condition (4) is not violated during at least r rounds within an n -process group:

$$\begin{aligned} P_r^* &= \sum_{i \geq r} P_i \\ &= P_{group} \sum_{i \geq r} (1 - P_{couple})^{i(n^2 - n)} \\ &= P_{group} \left(\sum_{i \geq 0} (1 - P_{couple})^{i(n^2 - n)} \right. \\ &\quad \left. - \sum_{i=0}^{i=r-1} (1 - P_{couple})^{i(n^2 - n)} \right) \\ &= P_{group} \left(\frac{1}{1 - (1 - P_{couple})^{n^2 - n}} \right. \\ &\quad \left. - \frac{1 - (1 - P_{couple})^{r(n^2 - n)}}{1 - (1 - P_{couple})^{n^2 - n}} \right) \\ &= (1 - P_{couple})^{r(n^2 - n)}. \end{aligned}$$

Finally, by (5), one concludes that

$$\boxed{P_r^* \geq (1 - \lambda)^{r(n^2 - n)}}. \quad (7)$$

[‡] The pair (i, i) does not have to be considered.

Mean time without false failure detection. It is now quite easy to compute the mean time \bar{T} without false failure detection exhibited by the algorithm. Indeed

$$\begin{aligned}\bar{T} &= T_e \sum_{i \geq 0} (i+1) P_i \\ &= T_e P_{group} \sum_{i \geq 0} (i+1) (1 - P_{group})^i.\end{aligned}$$

Let us introduce the generating series [7], $H(z)$ defined by

$$\begin{aligned}H(z) &= \sum_{i \geq 0} (1 - P_{group})^i z^i \\ &= \frac{1}{1 - (1 - P_{group})z}.\end{aligned}\quad (8)$$

Let us consider $H'(z)$:

$$\begin{aligned}H'(z) &= \sum_{i \geq 1} i (1 - P_{group})^i z^{i-1} \\ &= \sum_{i \geq 0} (i+1) (1 - P_{group})^{i+1} z^i.\end{aligned}$$

Thus, $\bar{T} = T_e \frac{P_{group}}{1 - P_{group}} H'(1)$. Furthermore, as a consequence of (8) one knows that

$$H'(z) = \frac{1 - P_{group}}{(1 - (1 - P_{group})z)^2}.$$

Finally,

$$\begin{aligned}\bar{T} &= T_e \frac{P_{group}}{1 - P_{group}} \frac{1 - P_{group}}{(1 - (1 - P_{group}))^2} \\ &= \frac{T_e}{P_{group}}.\end{aligned}$$

It is possible to find a lower bound for \bar{T} :

$$\bar{T} \geq \frac{T_e}{(n^2 - n)\lambda}.$$

5. Numerical results

Let us assume that VZ is in the order of $100 \mu s^2$, and $\Delta = 10$ s with $T_e = 60$ s. One obtains $\lambda = 10^{-6}$, and we get a mean number of rounds without false failure detection (i.e. \bar{T}/T_e) greater than $\frac{10^6}{n^2 - n}$. This is illustrated by figure 2 (with logarithmic scales on both axes), for values of λ ranging from 10^{-7} to 10^{-4} . Downwards, the number of processes within the system is: 2, 7, 12, 17, 22, 27. One can see that even with many processes in the system, claimed performances are always quite good. To increase the mean time before a false failure group detection, one can increase the value of T_e . This way, one can trade safety against the quality of service associated with the liveness property

To illustrate figure 2, let us consider the case where there are 27 processes in the system (the curve at the bottom of the figure) and the particular value $\lambda = 10^{-6}$ (the vertical line denoted '-6'). We see that \bar{T} is greater than $1600T_e$. Since $T_e = 60$ s, this means that one can expect, on average, that

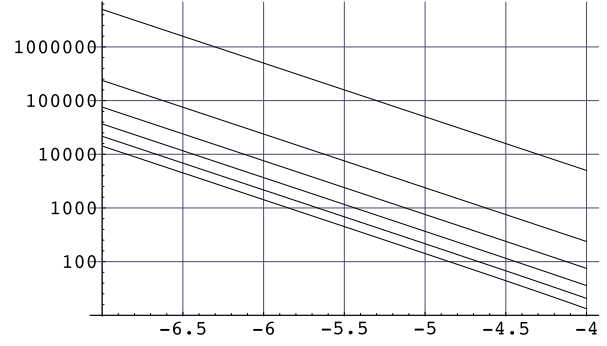


Figure 2. Mean number of rounds for values of λ between 10^{-7} and 10^{-4} .

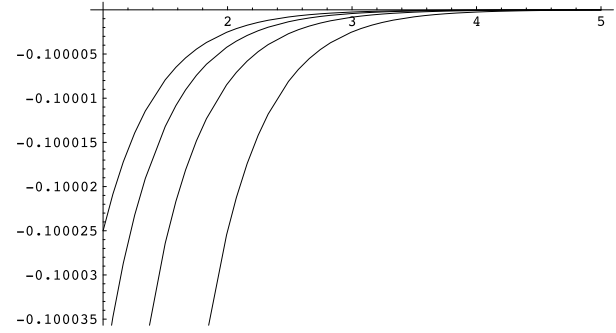


Figure 3. Reaching an *a priori* bound on P_r^* by choosing an appropriate λ .

$\bar{T} > 96\,000$ s (i.e. a little bit less than one day) without a false failure detection

Condition (7) shows that by choosing an appropriate value for λ , it is possible to obtain an *a priori* given bound for P_r^* . Taking $\lambda = \varepsilon/(r(n^2 - n))$, we get $\lim_{r \rightarrow \infty} P_r^* \geq 1 - \varepsilon$. This is depicted by figure 3 (with logarithmic scales on both axes) which gives for different numbers (downwards, from 2 to 5) of processes in the system, the lower bound on P_r^* according to the value of r (the number of rounds). The figure corresponds to the case where ε is 0.1. One can see that all the curves tend towards $e^{-0.1} \approx 1 - \varepsilon = 0.9$.

6. Conclusion

This paper has considered the *group failure detection* problem (a group fails as soon as one of its members crashes). Because ensuring both the liveness and the safety of such a detection is impossible in asynchronous distributed systems, we have considered a protocol that trades liveness against safety. A probabilistic analysis of this protocol has been done. This analysis uses the distribution function of message transfer delays. It has been shown which parameters are critical and to which values they have to be set in order that the probability of false group failure detection be as close to zero as desired (a *false* group failure detection occurs when the protocol claims a group failure while actually no process has crashed).

The proposed protocol can be used as a basic element to build a *membership service* in an asynchronous system. The probabilistic analysis shows that, although impossibility results place fundamental limits on what can be achieved

in asynchronous distributed systems, those results can be practically circumvented when one knows the value of critical parameters (such as the distribution function associated with message transfer delays). More precisely, the probabilistic analysis has shown that, in real distributed systems, it is possible to achieve failure detection with a negligible probability of wrong suspicions. This is an encouraging result favouring the use of simple failure detection protocols.

Acknowledgments

The authors want to thank Christof Fetzer (University of California at San Diego, CA) and Gerardo Rubino (IRISA, Rennes, France) and the anonymous referees whose comments helped improve the paper.

Appendix. Proofs

Proof of lemma 4.2. By definition

$$g(-y) = \int_{-\infty}^{+\infty} f(z) f(z+y) dz$$

and by change of variable $z = Z - y$:

$$\begin{aligned} g(-y) &= \int_{-\infty}^{+\infty} f(Z-y) f(Z) dZ \\ &= g(y). \end{aligned}$$

□

Proof of lemma 4.3. For $z > 0$:

$$P[0 \leq X \leq z] = \int_{-\infty}^z g(y) dy - \int_{-\infty}^0 g(y) dy,$$

and by change of variable $y = -u$ in both integrals and by using lemma 4.1, one gets

$$\begin{aligned} P[0 \leq X \leq z] &= \int_{-z}^{+\infty} g(u) du - \int_0^{+\infty} g(u) du \\ &= \int_{-z}^0 g(y) dy \\ &= \int_{-\infty}^{-z} g(y) dy - \int_{-\infty}^0 g(y) dy \\ &= P[-z \leq X \leq 0]. \end{aligned}$$

□

Proof of corollary 4.1. If one applies lemma 4.3 with $y \rightarrow +\infty$, one obtains

$$\begin{aligned} P[-\infty \leq X \leq +\infty] &= 1 \\ &= P[X \leq 0] + P[0 \leq X] \\ &= 2P[X \leq 0]. \end{aligned}$$

□

Proof of lemma 4.4. By definition

$$\begin{aligned} EX &= \int_{-\infty}^{+\infty} yg(y) dy \\ &= \underbrace{\int_{-\infty}^0 yg(y) dy}_{(1)} + \int_0^{+\infty} yg(y) dy, \end{aligned}$$

and by change of variable $u = -y$ in integral (1), one gets

$$\begin{aligned} EX &= -\int_0^{+\infty} yg(y) dy + \int_0^{+\infty} yg(y) dy \\ &= 0. \end{aligned}$$

Furthermore, by definition

$$\begin{aligned} VX &= EX^2 - (EX)^2 \\ &= \int_{-\infty}^{+\infty} y^2 g(y) dy \\ &= \int_{-\infty}^{+\infty} y^2 \int_{-\infty}^{+\infty} f(z) f(z-y) dz dy \\ &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} y^2 f(z) f(z-y) dz dy \\ &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} y^2 f(z) f(z-y) dy dz \\ &= \int_{-\infty}^{+\infty} f(z) \int_{-\infty}^{+\infty} y^2 f(z-y) dy dz, \end{aligned}$$

and by change of variable $u = z - y$

$$\begin{aligned} VX &= \int_{-\infty}^{+\infty} f(z) \int_{-\infty}^{+\infty} (z-u)^2 f(u) du dz \\ &= \int_{-\infty}^{+\infty} f(z) \int_{-\infty}^{+\infty} (z^2 + u^2 - 2zu) f(u) du dz \\ &= \int_{-\infty}^{+\infty} f(z) \left(\underbrace{z^2 \int_{-\infty}^{+\infty} f(u) du}_{=1} + \underbrace{\int_{-\infty}^{+\infty} u^2 f(u) du}_{=EZ^2} \right. \\ &\quad \left. - 2z \underbrace{\int_{-\infty}^{+\infty} u f(u) du}_{=EZ} \right) dz \\ &= \underbrace{\int_{-\infty}^{+\infty} z^2 f(z) dz}_{EZ^2} + EZ^2 \underbrace{\int_{-\infty}^{+\infty} f(z) dz}_{=1} \\ &\quad - 2EZ \underbrace{\int_{-\infty}^{+\infty} z f(z) dz}_{EZ} \\ &= 2EZ^2 - 2(EZ)^2 \\ &= 2VZ. \end{aligned}$$

□

References

- [1] Birman K P 1996 *Building Secure and Reliable Network Applications* (Greenwich, CT: Manning)
- [2] Chandra T and Toueg S 1996 *J. ACM* **34** 225–67

- [3] Cristian F and Fetzer C 1999 The timed asynchronous distributed system model *IEEE Trans. Parallel Distrib. Syst.* **10** 642–57
- [4] Duggal H, Cukier M and Sanders W H 1998 Probabilistic verification of a synchronous round-based consensus protocol *Proc. 16th IEEE Symp. on Reliable Distributed Systems* pp 1165–74
- [5] Fischer M J, Lynch N and Paterson M S 1985 *J. ACM* **32** 374–82
- [6] Fritzke U, Ingels P, Mostefaoui A and Raynal M 1998 Fault-tolerant total order multicast to asynchronous groups *Proc. 17th IEEE Symp. on Reliable Distributed Systems (West Lafayette, IN)* (Los Alamitos, CA: IEEE Computer Society Press) pp 228–35
- [7] Graham R L, Knuth D E and Patashnik O 1994 *Concrete Mathematics* 2nd edn (Reading, MA: Addison-Wesley)
- [8] Guerraoui R and Schiper A 1996 Consensus service: a modular approach for building fault-tolerant agreement protocols in distributed systems *Proc. 26th IEEE Symp. on Fault-Tolerant Computing (Sendai)* (Los Alamitos, CA: IEEE Computer Society Press) pp 168–77
- [9] Hurfin M and Tronel F 1997 A solution to atomic commitment based on an extended consensus protocol *Proc. 7th IEEE Int. Workshop on Future Trends of Distributed Computing Systems (Tunis)* (Los Alamitos, CA: IEEE Computer Society Press) pp 90–103
- [10] Lorcy S and Plouzeau N 1998 A distributed algorithm for managing group membership with multiple groups *Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications (Las Vegas, NV)* (Los Alamitos, CA: IEEE Computer Society Press) pp 1643–9
- [11] Powell D 1992 Failure mode assumptions and assumption coverage *Proc. 22th IEEE Symp. on Fault-Tolerant Computing (Boston, MA)* (Los Alamitos, CA: IEEE Computer Society Press) pp 386–92
- [12] Powell D (guest editor) 1996 Special issue on group communication *Commun. ACM* **39** 50–3
- [13] van Renesse R, Minsky Y and Hayden M 1998 A gossip-style failure detection service *Technical Report 98-1687* Cornell University, Department of Computer Science