Abstract

This paper presents an evaluation of several communication-optimal algorithms implementing the \(3P\) class of failure detectors. The first algorithm is based on a Reliable Broadcast primitive, involving a quadratic number of messages to manage a suspicion. The second algorithm uses exclusively one-to-one communication, involving a linear number of messages to manage a suspicion, but with a higher latency to propagate the suspicion to the rest of processes. A third algorithm reduces this latency using an additional one-to-all communication mechanism. We evaluate the quality of service provided by these algorithms, in terms of the capability of the failure detector to provide right answers and the reaction time after a failure.

1. Introduction

Unreliable failure detectors, proposed by Chandra and Toueg [4], have been used to address the consensus problem [11] and several related problems in asynchronous crash-prone distributed systems. In this paper, we mainly focus on the Eventually Perfect failure detector class, denoted \(3P\), which satisfies (1) strong completeness: eventually every process that crashes is permanently suspected by every correct process, and (2) eventual strong accuracy: there is a time after which correct processes are not suspected by any correct process.

Consensus can also be solved with a weaker failure detector class called Eventually Strong, denoted \(3S\), which satisfies strong completeness and eventual weak accuracy: there is a time after which some correct process is not suspected by any correct process. Specifically, a particular failure detector called \(\Omega\), equivalent to \(3S\), has been proved to be the weakest failure detector to solve consensus [3]. The \(\Omega\) failure detector provides eventual agreement on a common leader among all non-faulty processes in a system. Specific algorithms for implementing \(\Omega\) and/or \(3S\) have been proposed in the literature [1, 2, 8]. However, note that since \(3P\) is strictly stronger than \(3S\), any implementation of \(3P\) trivially implements \(3S\). Observe also that \(3P\) can also be easily transformed into \(\Omega\), e.g., by choosing as leader the non-suspected process with lowest identifier. Also, for certain problems [7] and consensus protocols [12] failure detector \(3P\) is required. These facts lead us to look for \(3P\) based solutions.

In [9] it has been proposed a family of heartbeat-based algorithms which implement \(3P\) using a logical ring arrangement of processes. In these algorithms, every process \(p\) tries to determine its correct successor in the ring, i.e., the process to which \(p\) should send heartbeats forever, and also its correct predecessor in the ring, i.e., the process from which \(p\) should receive heartbeats forever. The algorithms are communication-efficient [1], i.e., eventually only \(n\) unidirectional links carry messages forever. Recently, it has been proposed in [10] a communication-optimal implementation of \(3P\), in which eventually only \(C\) unidirectional links carry messages forever, being \(C\) the number of correct processes in the system.

The first algorithm presented in this paper is the communication-optimal implementation of \(3P\) proposed in [10], where every process communicates suspicions (and refutations) to the rest of processes using a Reliable Broadcast primitive [4], i.e., a reliable form of one-to-all communication. A characteristic of the algorithm is that, due to the use of Reliable Broadcast, the number of messages exchanged when a suspicion occurs is quadratic. This can be a serious drawback in some scenarios, e.g., very large networks, in which traffic-load is a critical issue.

As a second algorithm, we propose a communication-optimal implementation of \(3P\) which uses exclusively
one-to-one communication, even for communicating suspicions and refutations. In this algorithm, information about suspicions will be included into heartbeat messages and propagated around the ring. Since Reliable Broadcast is not used, the overhead to manage suspicions is reduced considerably. A drawback of this algorithm is its linear crash detection time. We reduce it by defining a new algorithm that introduces sporadic one-to-all communication.

We evaluate the performance of these three communication-optimal implementations of $\diamond P$ in terms of QoS measures, comparing them to Chandra-Toueg’s all-to-all based $\diamond P$ algorithm.

The rest of the paper is organized as follows. In Section 2, we describe the system model considered in this work. In Section 3, we present the three communication-optimal algorithms implementing $\diamond P$. In Section 4, we analyze the complexity and evaluate the performance of the algorithms. Finally, Section 5 concludes the paper.

2. System model

We consider a distributed system composed of a finite set $\Pi$ of $n$ processes, $\Pi = \{p_1, p_2, \ldots, p_n\}$, that communicate only by sending and receiving messages. Every pair of processes $(p_i, p_j)$ is connected by two unidirectional and reliable communication links $p_i \rightarrow p_j$ and $p_j \rightarrow p_i$.

Processes can only fail by crashing, that is, by prematurely halting. Moreover, crashes are permanent, i.e., crashed processes do not recover. In every run of the system we identify two complementary subsets of $\Pi$: the subset of processes that do not fail, denoted correct, and the subset of processes that do fail, denoted crashed. We use $C$ to denote the number of correct processes in the system in the run of interest, which we assume is at least one, i.e., $C \geq 1$.

We consider that processes are arranged in a logical ring. Without loss of generality, process $p_1$ is preceded by process $p_{n-1}$, and followed by process $p_{n+1}$. As usual, $p_1$ follows $p_n$ in the ring. In general, we will use the functions $\text{pred}(p)$ and $\text{succ}(p)$ respectively to denote the predecessor and the successor of a process $p$ in the ring.

Concerning timing assumptions, we consider a partially synchronous model [4, 6] which stipulates that, in every run of the system, there are bounds on relative process speeds and on message transmission times, but these bounds are not known and they hold only after some unknown but finite time (called GST for Global Stabilization Time). Actually, the bounds must exist and hold only for the $C$ links that eventually form the ring of correct processes, i.e., the links from every correct process to its correct successor in the ring.

Finally, in the algorithms presented in this paper we assume that a local clock that can measure real-time intervals is available to each process. Clocks are not synchronized.

3. Communication-optimal implementations of $\diamond P$

3.1. Reliable Broadcast based optimal $\diamond P$

We describe here a first communication-optimal algorithm that implements $\diamond P$ using Reliable Broadcast [10].

\begin{algorithm}
\caption{Communication-optimal $\diamond P$ using Reliable Broadcast.}
\text{\{Every process $p$ executes the following\}}
\begin{algorithmic}
\Procedure{update pred and succ}{}
\If{$\forall r: \text{Balance}_r(r) > 0$}
\State $\text{pred}_p \rightarrow p$
\State $\text{succ}_p \leftarrow p$
\Else
\State $\text{pred}_p \rightarrow p$'s nearest predecessor $r$ in the ring such that $\text{Balance}_r(r) \leq 0$
\State $\text{succ}_p \rightarrow p$'s nearest successor $r$ in the ring such that $\text{Balance}_r(r) \leq 0$
\EndIf
\EndProcedure
\State $\text{pred}_p \leftarrow \text{pred}(p)$
\State $\text{succ}_p \leftarrow \text{succ}(p)$
\ForAll{$q \in \Pi$}
\State $\Delta_q(q) \leftarrow$ default time-out interval
\State $\text{Balance}_q(q) \leftarrow 0$
\EndForAll
\State \textbf{Task 1:} repeat periodically
\If{$\text{succ}_p \neq p$}
\State send \{ALIVE, $p$\} to $\text{succ}_p$
\EndIf
\State \textbf{Task 2:} repeat periodically
\If{$(\text{pred}_p \neq p$ and $p$ didn’t receive \{ALIVE, $\text{pred}_p$\} during the last $\Delta_p(\text{pred}_p)$ ticks of $p$’s clock}
\State $\text{r-broadcast (SUSPICION, $p$, pred$_p$)}$
\EndIf
\State \textbf{Task 3:} when $\text{r-deliver}$\{SUSPICION, $q$, $r$\}
\State $\text{Balance}_q(r) \leftarrow \text{Balance}_q(r) + 1$
\State update pred and succ()
\If{$r = p$}
\State $\text{r-broadcast (REFUTATION, $p$)}$
\EndIf
\State \textbf{Task 4:} when $\text{r-deliver}$\{REFUTATION, $q$\}
\State $\text{Balance}_q(q) \leftarrow \text{Balance}_q(q) - 1$
\State $\Delta_q(q) \leftarrow \Delta_q(q) + 1$
\State update pred and succ()
\EndProcedure
\end{algorithmic}
\end{algorithm}

As shown in Algorithm 1, each process sends heartbeats to its successor in the ring, and monitors its predecessor by hearing heartbeats from it. Every process $p$ uses a $\text{Balance}_p$ variable for every process $q$, accounting suspicions and refutations for $q$. If $\text{Balance}_p(q) > 0$, with $q \neq p$, then $p$ suspects $q$; else, $q$ is trusted...
3.2. One-to-one communication based optimal $\Diamond P$

In this section, we propose a second communication-optimal algorithm implementing $\Diamond P$, that uses one-to-one communication exclusively. As shown in Algorithm 2 every process $p$ uses a Suspected$^p$ list to provide the properties of $\Diamond P$. Every process $p$ starts sending periodically an (ALIVE, $p$, Suspected$^p$) message to its successor in the ring, denoted by the variable succ$^p$ (Task 1). Also, every process $p$ waits for periodical (ALIVE, pred$^p$) messages from its predecessor in the ring, denoted by the variable pred$^p$. If $p$ does not receive such a message on a specific time-out interval of $\Delta_p$ (pred$^p$), then $p$ suspects that pred$^p$ has crashed, and broadcasts a (SUSPICION, $p$, pred$^p$) message (Task 2). In Task 3, when $p$ receives a (SUSPICION, $q$, $r$) message, $p$ increments $\text{Balance}_p(r)$ and calls the $\text{update}_{\text{pred}_{\text{and succ}}}$ procedure. Besides this, if $r = p$, i.e., $p$ has been erroneously suspected by $q$, $p$ re-broadcasts a (REFUTATION, $p$) message. In Task 4, when $p$ delivers a (REFUTATION, $q$) message, $p$ decrements $\text{Balance}_p(q)$, increments $\Delta_p(q)$, and calls the $\text{update}_{\text{pred}_{\text{and succ}}}$ procedure. Variables pred$^p$ and succ$^p$ are updated from $\text{Balance}_p$ to the nearest predecessor and the nearest successor in the ring having a non-positive balance respectively. If all the components of the $\text{Balance}_p$ vector are positive, then $p$ sets both pred$^p$ and succ$^p$ to $p$.

![Algorithm 2](https://example.com/algorithm2.png)

Algorithm 2: Communication-optimal $\Diamond P$ using exclusively one-to-one communication.

(1) Procedure $\text{update}_{\text{pred}_{\text{and succ}}}$()
(2) pred$^p$ ← $p$'s nearest predecessor $r$ in the ring such that $r \notin \text{Suspected}_p$
(3) succ$^p$ ← $p$'s nearest successor $r$ in the ring such that $r \notin \text{Suspected}_p$
(4) pred$^p$ ← pred$^p$
(5) succ$^p$ ← succ$^p$
(6) for all $q \in \text{II}$ do
(7) $\Delta_p(q)$ ← default time-out interval
(8) Suspected$^p$ ← $\emptyset$
(9) cobegin
(10) || Task 1: repeat periodically
(11) if succ$^p$ ≠ $p$ then
(12) send (ALIVE, $p$, Suspected$^p$) to succ$^p$
(13) || Task 2: repeat periodically
(14) if (pred$^p$ ≠ $p$) and $p$ didn’t receive (ALIVE, pred$^p$), − $\Delta_p$ (pred$^p$) ticks of $p$’s clock then
(15) Suspected$^p$ ← Suspected$^p$ union (pred$^p$)
(16) send (SUSPICION, $p$, pred$^p$) to pred$^p$
(17) || Task 3: when receive (SUSPICION, $q$, $p$) for some $q$
(18) send (REFUTATION, $p$) to $q$
(19) succ$^p$ ← $q$
(20) || Task 4: when receive (REFUTATION, $q$) for some $q$
(21) Suspected$^p$ ← Suspected$^p$ - $\{q\}$
(22) $\Delta_p(q)$ ← $\Delta_p(q) + 1$
(23) update$\_p$ pred$^p$ and succ$^p$
(24) || Task 5: when receive (ALIVE, pred$^p$, Suspected$^p$) from pred$^p$
(25) for all $q \in \text{II}$ except pred$^p$ and $p$ do
(26) if $q \notin $ Suspected$^p$ and $q \notin $ Suspected$^p$
(27) Suspected$^p$ ← Suspected$^p$ union $q$
(28) send (SUSPICION, $p$, $q$) to $q$
(29) update$\_p$ pred$^p$ and succ$^p$
(30) coend
list piggybacked into heartbeat messages is received by Task 5, resulting in a higher latency for the detection of a crashed process by the rest of processes. We address this issue in the following subsection.

3.3. One-to-all communication to reduce the detection latency

We present here a modification to Algorithm 2 that reduces the detection latency of real failures by sending additional messages upon suspicions. We will evaluate the improvement of the modified algorithm in Section 4.2. The modification, presented in Algorithm 3, affects Task 2 and introduces a new task (Task 6).

Algorithm 3: Reducing the detection latency using one-to-all communication of suspicions.

```alg
{ Every process p executes the following }
cobegin

... Task 2: repeat periodically
| if (predp ≠ p) and p didn’t receive (ALIVE, predp, −1) during the last \( \Delta p \) ticks of p’s clock then
| Suspectedp ← Suspectedp ∪ \{ predp \}
| send (SUSPICION, p, predp) to predp,
| send (SUSP_TO_ALL, p, predp) to all except predp and p
| update_pred_and_succ()

... Task 6: when receive (SUSP_TO_ALL, q, r) for some
| r ≠ p for some q
|Suspectedq ← Suspectedq ∪ \{ r \}
|send (SUSPICION, p, r) to r
|update_pred_and_succ()

cobend
```

In Task 2, when \( p \) suspects that \( \text{pred}_p \) has crashed, besides sending the (SUSPICION, \( p, \text{pred}_p \)) message to \( \text{pred}_p \) (Line 16), \( p \) also sends a message (SUSP_TO_ALL, \( p, \text{pred}_p \)) to all processes except \( \text{pred}_p \) and \( p \). This new type of message is handled in Task 6. There, when a process \( p \) receives a (SUSP_TO_ALL, \( q, r \)) message for some \( q \), \( p \) includes \( r \) in \( \text{Suspected}_p \), sends a (SUSPICION, \( p, r \)) to verify if \( r \) has crashed, and calls the procedure update_pred_and_succ. Observe that if \( r \) has really crashed then the suspicion will be permanent, because \( r \) will not send any REFUTATION message to \( p \).

It is simple to see that the proposed modification speeds-up the detection of real failures, since the rest of processes will send almost simultaneously the SUSPICION message to the suspected process, while in Algorithm 2 the SUSPICION messages are sent following the ring by means of Task 5. On the other hand, in the case of an erroneous suspicion the proposed modification introduces an additional overhead of approximately \( 3n \) messages (SUSPICION and REFUTATION).

The modification does not affect neither the correctness of the algorithm nor its communication optimality, since after the stabilization of the ring no more SUSPICION or SUSP_TO_ALL messages will be sent.

4. Analysis and performance evaluation

4.1. Complexity analysis

Table 1 summarizes the communication costs of the communication-optimal algorithms presented in this paper, in terms of the number of unidirectional links used forever (which corresponds eventually to the number of regular heartbeat messages exchanged periodically) and the number of messages needed to manage an erroneous suspicion. Chandra-Toueg’s all-to-all algorithm is also included for comparative purposes.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Periodic cost (#links used forever)</th>
<th>Sporadic cost (#msgs to manage a suspicion)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm 1</td>
<td>( c )</td>
<td>( 2n^2 )</td>
</tr>
<tr>
<td>Algorithm 2</td>
<td>( c )</td>
<td>( 2n )</td>
</tr>
<tr>
<td>Algorithm 3</td>
<td>( c )</td>
<td>( 3n )</td>
</tr>
<tr>
<td>Chandra-Toueg [4]</td>
<td>( \mathcal{O}(n - 1) )</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1. Communication costs of different algorithms implementing \( \Diamond \mathcal{P} \).

As it can be observed, Algorithms 2 and 3, while communication-optimal, have a linear overhead for managing an erroneous suspicion. The benefits obtained with respect to the quadratic communication-optimal Algorithm 1 are evident, and can be explained by the fact that now suspicions and refutations are managed following one-to-one (Algorithm 2) one-to-all (Algorithm 3) communication patterns, respectively. As explained in the introduction, Algorithm 1 uses a reliable one-to-all communication pattern, which considering the implementation of Reliable Broadcast, results in practice in an all-to-all pattern.

4.2. Performance evaluation

Besides communication optimality, there are QoS measures that are of interest when evaluating the performance of failure detector algorithms. We have considered here two different performance measures to compare the algorithms presented. The first measure is related to the accuracy of the information provided to querying processes. In particular, we focus on the query accuracy probability, defined as the probability that a
failure detection module which is queried by its associated process gives the right answer. This measure is based on [5], but has been extended in this work to scenarios with more than two processes. The second measure tries to quantify how fast the failure detector reacts. This has been measured by the time interval between the crash of a process and the time in which the rest of the processes suspect it in a permanent way.

To test the comparative performance of the algorithms, we have used the ns-2 simulator (http://www.isi.edu/nsnam/ns/). In Table 2 we show the simulation settings for a typical local area network scenario. The simulation generates message delays at random with a uniform distribution. However, we have set minimum and maximum message bounds. Apparently, this contradicts our partially synchronous system model. Nevertheless, the algorithms do not exploit the knowledge of the maximal message delay when initializing the timeouts. This allows us to generate erroneous suspicions under the same conditions for both algorithms. Moreover, from a practical point of view the setting of a maximum message delay allows to determine the duration of the simulations.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum message delay</td>
<td>0.001</td>
</tr>
<tr>
<td>Maximum message delay</td>
<td>0.005</td>
</tr>
<tr>
<td>Periodicity of ALIVE</td>
<td>0.5</td>
</tr>
<tr>
<td>Initial timeouts</td>
<td>0.5</td>
</tr>
<tr>
<td>Timeout increment</td>
<td>0.001</td>
</tr>
</tbody>
</table>

Table 2. Simulation settings (in seconds).

The tests have been carried out for a number of nodes going from 3 to 24, using the settings of Table 2. The bad answer probability has been measured executing the algorithm during 2000 seconds, that has been empirically proved to be sufficient for comparative purposes. In fact, after this time the simulations have either stabilized or are near stabilization. We assume that no process crashes during the 2000 seconds. This assumption does not really lose any generality. On the one hand, in our algorithms erroneous suspicions are actually more complex to handle than real crashes. On the other hand, although a crash during the execution of the Reliable Broadcast may delay the delivery of the message, the probability of such a failure in practice is very low. Also, this delay is really small in a LAN, thus our assumption has not any impact in the accuracy of the failure detector. The crash detection time has been measured in a longer execution, introducing a crash in a time instant (2500 seconds) in which the system is stabilized. In both cases, every simulation has been executed a sufficiently large number of times.

Figures 1 and 2 show the average results obtained.

In Figure 1, for clarity, values express the complement of the right answer probability, i.e., the probability that a failure detection module gives a wrong answer. The bad answer probability is low for all the communication optimal algorithms, and does not increase with the number of processes. Although Algorithm 2 uses less messages to manage suspicions, the bad answer probability is lower for this algorithm than for Algorithm 1. This result is due to the fact that in Algorithm 2 when a process suspects its predecessor in the ring the rest of processes do not receive any information about the new suspicion most of the times. In fact, in practice the refutation message from a suspected process usually arrives to the suspecting process before the next heartbeat message is sent by it. This reduces the number of false suspicions in the system and makes the bad answer probability near
negligible. For Chandra-Toueg’s algorithm the bad answer probability is negligible too, at the cost of using an all-to-all communication pattern periodically and forever.

In Figure 2, it can be observed that Algorithm 2 has a higher crash detection latency than the other algorithms. Even worse, the detection latency increases linearly with the number of processes, hence the algorithm does not scale well for a large number of processes. The same mechanism that makes Algorithm 2 more accurate when false suspicions occur makes it slower for real crashes. If we consider the improvement presented in Algorithm 3, we observe that the crash detection latency is constant in this case and similar to the crash detection latency of Algorithm 1. Note that Reliable Broadcast involves a quadratic number of messages to manage suspicions, while the improvement used to reduce the crash detection latency of Algorithm 3 keeps the extra messages linear. Going back to Figure 1, it can be observed that the bad answer probability of Algorithm 3 is slightly higher because it takes a bit longer to correct false suspicions. In order to get an optimal performance, it could be interesting to switch from Algorithm 2 to Algorithm 3 once the system is considered stabilized.

5. Conclusion

In this paper, we have analysed three communication-optimal algorithms implementing the ◦P failure detector class. The first algorithm uses Reliable Broadcast to communicate suspicions and refutations, involving a quadratic number of messages. The second algorithm uses one-to-one communication exclusively, involving a lower overhead to manage suspicions. The third algorithm consists in adding sporadic one-to-all communication to the second one, in order to improve the crash detection time.

We have evaluated the performance of the algorithms in terms of two QoS measures: one of them is related to the accuracy of the information provided by the failure detector and the other concerns the crash detection time. Although the algorithm using one-to-one communication is better in terms of accuracy, it does not scale well when the crash detection latency is considered. That’s the reason why we have proposed the third algorithm, that uses some extra messages when a suspicion occurs, reducing the crash detection latency from linear to constant. This new algorithm also involves a linear number of messages to manage a suspicion.

An interesting aspect regarding the second and third algorithms presented in this paper is that they could be used together, switching from the second to the third once the system is considered stabilized.

References