Eventually Consistent Failure Detectors [*]

Mikel Larrea[†] Antonio Fernández[‡] Sergio Arévalo[§]

1 Introduction

The concept of *unreliable failure detector* was introduced by Chandra and Toueg [1] as a mechanism that provides information about process failures. This mechanism has been used to solve different distributed problems in asynchronous systems, in particular the Consensus problem.

This short paper presents a new class of unreliable failure detectors, which we call *Eventually Consistent* and denote by $\diamond\mathcal{C}$. This class adds to the failure detection capability of other classes an *eventual leader election* capability. To show the power of this new class of failure detectors, we propose an efficient Consensus algorithm based on an eventually consistent failure detector. This algorithm successfully exploits the leader election capability of the failure detector and performs better in the number of rounds than all the previously proposed Consensus algorithms for failure detectors with eventual accuracy [1, 2, 6]. This is due to the fact that, to our knowledge, it is the first Consensus algorithm for failure detectors with eventual accuracy that does not rely on the rotating coordinator paradigm.

Due to space limitation, the reader is referred to [3] for an in-depth presentation of the new class of failure detectors (relationship with other classes of failure detectors, equivalence between $\diamond\mathcal{C}$ and $\diamond\mathcal{S}$[1], implementations of $\diamond\mathcal{C}$), as well as for the correctness proof of the Consensus algorithm.

2 System Model

We consider a distributed system consisting of a finite totally ordered set Π of n processes, $\Pi = \{p_1, p_2, \ldots, p_n\}$. Processes communicate only by sending and receiving messages. Every pair of processes is assumed to be connected by a reliable communication channel. The system is *asynchronous*, i.e., there are no timing assumptions about neither the relative speeds of the processes nor the delay of messages. Processes can fail by *crashing*, that is, by prematurely halting. Crashes are permanent, i.e., crashed processes do not recover.

A *distributed failure detector* can be viewed as a set of n failure detection modules, each one attached to a different process in the system. These modules cooperate to satisfy the required properties of the failure detector. Upon request, each module provides its attached process with a set of processes it suspects to have crashed. These sets can differ from one module to another at a given time. Let us denote by $\mathcal{D}_p$ the set of suspected processes returned by a failure detector $\mathcal{D}$ to

[*]Research partially supported by the Spanish Research Council, contracts TIC99-0280-C02-02, TEL99-0582, and TIC98-1032-C03-01, and the Madrid Regional Research Council, contract CAM-07T/00112/1998.

[†]Universidad Pública de Navarra, 31006 Pamplona, Spain, mikel.larrea@unavarra.es

[‡]Universidad Rey Juan Carlos, 28933 Móstoles, Spain, afernandez@acm.org

[§]Universidad Rey Juan Carlos, 28933 Móstoles, Spain, s.arevalo@escet.urjc.es

[1]The *Eventually Strong* class of failure detectors, denoted $\diamond\mathcal{S}$, is the weakest class for solving Consensus.

a given process p. We also denote by $\mathcal{T}_p$ the set of trusted (non-suspected) processes of the failure detection module attached to process p, i.e., $\mathcal{T}_p = \Pi - \mathcal{D}_p$.

3 Eventually Consistent Failure Detectors

In this section, we introduce the eventually consistent class of failure detectors. The main characteristic of these failure detectors is the accuracy property they satisfy, which we call *Eventual Consistent Accuracy*. Informally, the eventual consistent accuracy guarantees that there is a correct process p that is eventually and permanently not suspected by any correct process, and that there is a function that each correct process can apply to the output of its local failure detection module that eventually and permanently returns p.

More formally, the eventual consistent accuracy property can be defined as follows. Let $\mathcal{P}(\Pi)$ be the power set of the set Π.

Definition 1 *A failure detector $\mathcal{D}$ satisfies* Eventual Consistent Accuracy *if there is a deterministic function leader* : $\mathcal{P}(\Pi) \rightarrow \Pi$, *a time t and a correct process p such that, after t, for every correct process q, $p \notin \mathcal{D}_q$ and $leader(\mathcal{T}_q) = p$.*

Definition 2 *We define the* Eventually Consistent *class of failure detectors, denoted $\diamond\mathcal{C}$, as those that satisfy both the strong completeness*[2] *and the eventual consistent accuracy properties.*

A failure detector of class $\diamond\mathcal{C}$ enhances the classical failure detection properties of previously defined classes with an eventual leader election mechanism. These failure detectors guarantee that after some point in time all correct processes can behave as a consistent leader election algorithm. This property can be used by algorithms in which the safety properties are not affected by the simultaneous existence of several leaders, and that guarantee termination if a unique leader exists. Furthermore, these failure detectors can be very useful to algorithms that have early termination when there is a unique leader. As it is well known, usually it is not necessary for the failure detector to reach permanent stability to be useful. Instead, many algorithms can successfully complete if the failure detector is stable (provides a unique leader) for long enough periods of time.

We have found several implementations of failure detectors in the literature that in fact implement an eventually consistent failure detector. Examples are the algorithms implementing $\diamond\mathcal{P}$ proposed in [1, 4], and the algorithms implementing $\diamond\mathcal{S}$ proposed in [4, 5].

4 Solving Consensus using $\diamond\mathcal{C}$

In this section, we present an algorithm that solves Consensus using an eventually consistent failure detector. In addition to the model defined in Section 2, we assume that the system is augmented with a failure detector $\mathcal{D}$ of class $\diamond\mathcal{C}$, to which processes have access. We also assume that all processes know the function *leader* associated with the failure detector, as specified in Definition 1. Finally, we assume that a majority of processes are correct, i.e., do not crash.

Figures 1 and 2 present the algorithm in detail. Each process runs an instance of this algorithm, which proceeds in asynchronous rounds. As the $\diamond\mathcal{S}$-Consensus algorithm of Chandra and Toueg [1], it goes through three asynchronous epochs, each of which may span several rounds. In the first epoch, several decision values are possible. In the second epoch, a value gets *locked*: no other decision value is possible. In the third epoch, processes decide the locked value.

[2] *Eventually every process that crashes is permanently suspected by every correct process.*

```
procedure $propose(v_p)$
  $estimate_p \leftarrow v_p$                    $\{estimate_p$ is $p$'s estimate of the decision value$\}$
  $state_p \leftarrow undecided$
  $r_p \leftarrow 0$                             $\{r_p$ is $p$'s current round number$\}$
  $ts_p \leftarrow 0$                            $\{ts_p$ is the last round in which $p$ updated $estimate_p$, initially 0$\}$

  while $state_p = undecided$                    $\{$Rotate until decision is reached$\}$
    $chosen_p \leftarrow false$
    $replied_p \leftarrow false$
    $r_p \leftarrow r_p + 1$

    Phase 0: $\{$Each process determines its coordinator for the round$\}$
      wait until [$p = leader(\mathcal{T}_p)$ or for a process $q$: received $(q, r_p, coordinator)$]    $\{$Query the failure detector$\}$
      if [for a process $q$: received $(q, r_p, coordinator)$] then
        $c_p \leftarrow q$
      else
        $c_p \leftarrow p$
        send $(p, r_p, coordinator)$ to all processes except $p$
      $chosen_p \leftarrow true$

    Phase 1: $\{$Each process $p$ sends $estimate_p$ to its current coordinator$\}$
      send $(p, r_p, estimate_p, ts_p)$ to $c_p$

    Phase 2: $\{$Each coordinator tries to gather $\lceil\frac{(n+1)}{2}\rceil$ estimates to propose a new estimate$\}$
      if $p = c_p$ then
        wait until [for $\lceil\frac{(n+1)}{2}\rceil$ processes $q$: received $(q, r_p, estimate_q, ts_q)$ or $(q, r_p, null\_estimate, 0)$]
        $msgs_p[r_p] \leftarrow \{(q, r_p, estimate_q, ts_q) \mid p$ received $(q, r_p, estimate_q, ts_q)$ from $q\}$
        if [for $\lceil\frac{(n+1)}{2}\rceil$ processes $q$: received $(q, r_p, estimate_q, ts_q)$] then
          $decidible_p \leftarrow true$
          $t \leftarrow$ largest $ts_q$ such that $(q, r_p, estimate_q, ts_q) \in msgs_p[r_p]$
          $estimate_p \leftarrow$ select one $estimate_q$ such that $(q, r_p, estimate_q, t) \in msgs_p[r_p]$
          send $(p, r_p, estimate_p)$ to all
        else                                     $\{p$ received null_estimate from some process$\}$
          $decidible_p \leftarrow false$
          send $(p, r_p, null\_estimate)$ to all

    Phase 3: $\left\{\begin{array}{l}\text{Each process waits for a new estimate proposed by a coordinator}\\ \text{or to receive null\_estimate from its coordinator or to suspect it}\end{array}\right\}$
      wait until [for a process $q$: received $(q, r_p, estimate_q)$ or received $(c_p, r_p, null\_estimate)$ from $c_p$ or $c_p \in \mathcal{D}_p$]
      if [for a process $q$: received $(q, r_p, estimate_q)$] then      $\{p$ received $estimate_q$ from a process $q\}$
        $estimate_p \leftarrow estimate_q$
        $ts_p \leftarrow r_p$
        send $(p, r_p, ack)$ to $q$
      else if [received $(c_p, r_p, null\_estimate)$ from $c_p$] then   $\{p$ received null_estimate from $c_p\}$
        discard message
      else                                       $\{p$ suspects that $c_p$ crashed$\}$
        send $(p, r_p, nack)$ to $c_p$
      $replied_p \leftarrow true$

    Phase 4: $\left\{\begin{array}{l}\text{The coordinator that can still decide (if any) waits for } \lceil\frac{(n+1)}{2}\rceil \text{ replies. If they indicate that}\\ \lceil\frac{(n+1)}{2}\rceil \text{ processes adopted its estimate, the coordinator R-broadcasts a decide message}\end{array}\right\}$
      if $(p = c_p)$ and $(decidible_p)$ then
        wait until [for $\lceil\frac{(n+1)}{2}\rceil$ processes $q$: received $(q, r_p, ack)$ or $(q, r_p, nack)$]
        if [for $\lceil\frac{(n+1)}{2}\rceil$ processes $q$: received $(q, r_p, ack)$] then
          R-broadcast$(p, r_p, estimate_p, decide)$
```

Figure 1: Solving Consensus using any $\mathcal{D} \in \diamond\mathcal{C}$.

```
when received $(q, r_q, coordinator)$ from $q$ such that $(r_q < r_p)$ or $((r_q = r_p)$ and $(chosen_p))$
  send $(p, r_q, null\_estimate, 0)$ to $q$

when received $(q, r_q, estimate_q)$ from $q$ such that $(r_q < r_p)$ or $((r_q = r_p)$ and $(replied_p))$
  send $(p, r_q, nack)$ to $q$

when R-deliver$(q, r_q, estimate_q, decide)$        {If p R-delivers a decide message, p decides accordingly}
  if $state_p = undecided$ then
    $decide(estimate_q)$
    $state_p \leftarrow decided$
```

Figure 2: Separate tasks for replying to late coordinators and taking the decision.

Each round of the algorithm is divided into five asynchronous phases. In Phase 0, every process determines its coordinator for the round. In Phase 1, every process sends its current estimate of the decision value timestamped with the round number in which it adopted this estimate, to its coordinator. In Phase 2, each coordinator tries to gather a majority of estimates. If it succeeds, then it selects an estimate with the largest timestamp and sends it to all the processes as a proposition. In Phase 3, each process waits for a proposition from a coordinator. If the process receives a non-null proposition from some coordinator (including its own), then it adopts it and sends an *ack* message to this coordinator. Finally, in Phase 4 the coordinator that succeeded in Phase 2 and sent a non-null proposition (if any, and at most one) waits for a majority of *ack*/*nack* messages. If it gathers a majority of *ack* messages, then it knows that a majority of processes adopted its proposition as their new estimate. Consequently, this coordinator broadcasts a request to decide its proposition. At any time, if a process delivers such a request, it decides accordingly.

With this algorithm, if the failure detector is stable Consensus is solved in only one round. On the other hand, with any $\diamond\mathcal{S}$-Consensus algorithm based on the rotating coordinator paradigm, the number of rounds can be $\Omega(n)$, until the correct and non-suspected process becomes the coordinator.

Acknowledgements. We are grateful to André Schiper for his valuable comments.

References

[1] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

[2] M. Hurfin and M. Raynal. A simple and fast asynchronous consensus protocol based on a weak failure detector. *Distributed Computing*, 12(4):209–223, 1999.

[3] M. Larrea. *Efficient Algorithms to Implement Failure Detectors and Solve Consensus in Distributed Systems*. PhD thesis, University of the Basque Country, San Sebastián, October 2000.

[4] M. Larrea, S. Arévalo, and A. Fernández. Efficient algorithms to implement unreliable failure detectors in partially synchronous systems. In *Proceedings of the 13th International Symposium on DIstributed Computing (DISC'99)*, pages 34–48. LNCS, Springer-Verlag, September 1999.

[5] M. Larrea, A. Fernández, and S. Arévalo. Optimal implementation of the weakest failure detector for solving consensus. In *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS'2000)*, Nurenberg, Germany, October 2000. To appear.

[6] A. Schiper. Early consensus in an asynchronous system with a weak failure detector. *Distributed Computing*, 10(3):149–157, April 1997.