Eventually Consistent Failure Detectors [*]

Mikel Larrea
Univ. Pública de Navarra
31006 Pamplona, Spain
mikel.larrea@unavarra.es

Antonio Fernández
Univ. Rey Juan Carlos
28933 Móstoles, Spain
afernandez@acm.org

Sergio Arévalo
Univ. Rey Juan Carlos
28933 Móstoles, Spain
s.arevalo@escet.urjc.es

ABSTRACT

The concept of *unreliable failure detector* was introduced by Chandra and Toueg [2] as a mechanism that provides (possibly incorrect) information about process failures. This mechanism has been used to solve different problems in asynchronous systems, in particular the Consensus problem.

In this paper, we present 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 capabilities of other classes an eventual leader election capability. We study the relationship between $\diamond\mathcal{C}$ and other classes of failure detectors. We also propose an efficient algorithm to transform $\diamond\mathcal{C}$ into $\diamond\mathcal{P}$. Finally, to show the power of this new class of failure detectors, we present an efficient Consensus algorithm based on $\diamond\mathcal{C}$. Due to space limitation, the reader is referred to [4] for an in-depth presentation of the algorithms. Here, we only present their general idea.

1. 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. We denote by $\mathcal{D}_p$ the set of suspected processes returned by a failure detector $\mathcal{D}$ to 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., $\mathcal{T}_p = \Pi - \mathcal{D}_p$.

[*]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.

2. EVENTUALLY CONSISTENT FAILURE DETECTORS

We introduce now the class of eventually consistent 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*[1] *and the eventual consistent accuracy properties.*

3. RELATION BETWEEN $\diamond\mathcal{C}$ AND OTHER FAILURE DETECTOR CLASSES

With an eventually consistent failure detector, from the eventual consistent accuracy, eventually there is a correct process that will be never suspected by any correct process, and from the strong completeness, eventually all the crashed processes are permanently suspected by every correct process. From this, it is simple to see that every failure detector of class $\diamond\mathcal{C}$ belongs also to the class $\diamond\mathcal{S}$ (and hence to $\diamond\mathcal{W}$).

Note also that any failure detector of class $\diamond\mathcal{P}$ belongs to the class $\diamond\mathcal{C}$ as well. With $\diamond\mathcal{P}$, eventually the set of suspected processes by every correct process becomes the same, containing only all the processes that actually crash. A possible *leader* function could always choose the first (with respect to the order $p_1, \ldots, p_n$ assumed in the system model) non-suspected process.

Observation 1. $\diamond\mathcal{P} \subseteq \diamond\mathcal{C} \subseteq \diamond\mathcal{S} \subseteq \diamond\mathcal{W}$.

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

3.1 Equivalence of $\diamond\mathcal{C}$ and Ω

In [1], Chandra et al. defined a new class of failure detectors, denoted Ω, and used it to prove that $\diamond\mathcal{W}$ is the weakest failure detector class for solving Consensus. The output of the failure detector module of Ω at a process p is a *single* process q, that p currently considers to be *correct* (we say that *p trusts q*). The failure detector Ω satisfies the following property:

PROPERTY 1. *There is a time after which all the correct processes always trust the same correct process.*

It is straightforward to show that failure detector classes $\diamond\mathcal{C}$ and Ω are equivalent, i.e., one can be transformed into the other and vice versa. In some sense, the class $\diamond\mathcal{C}$ can be viewed as a redefinition of Ω in terms of lists of suspected processes instead of a single trusted process.

LEMMA 1. $\Omega \cong \diamond\mathcal{C}$.

3.2 Equivalence of $\diamond\mathcal{C}$ and $\diamond\mathcal{S}$

We now show that any failure detector of class $\diamond\mathcal{S}$ can be transformed into a failure detector of class $\diamond\mathcal{C}$. Since, by definition, $\diamond\mathcal{C}$ is a subclass of $\diamond\mathcal{S}$, i.e., every failure detector in $\diamond\mathcal{C}$ is in $\diamond\mathcal{S}$, this shows that failure detector classes $\diamond\mathcal{C}$ and $\diamond\mathcal{S}$ are actually equivalent.

LEMMA 2 ([2]). $\diamond\mathcal{S} \cong \diamond\mathcal{W}$.

LEMMA 3 ([1]). $\diamond\mathcal{W} \cong \Omega$.

THEOREM 1. $\diamond\mathcal{C} \cong \diamond\mathcal{S}$.

Thus, having any failure detector of class $\diamond\mathcal{S}$, it is always possible to build a failure detector of class $\diamond\mathcal{C}$. However, instead of running a transformation protocol on top of any failure detector of class $\diamond\mathcal{S}$, we show in the next section that there are very efficient implementations of $\diamond\mathcal{C}$.

3.3 Implementations of $\diamond\mathcal{C}$

There are several algorithms implementing failure detectors in the literature that also implement $\diamond\mathcal{C}$. For instance, since $\diamond\mathcal{P}$ is a subclass of $\diamond\mathcal{C}$, the $\diamond\mathcal{P}$ algorithms of Chandra and Toueg [2] and Larrea et al. [3], implement also a failure detector of class $\diamond\mathcal{C}$.

Concerning the ring-based algorithm implementing $\diamond\mathcal{S}$ proposed in [3], the set of non-suspected processes can be different in different processes, but the algorithm guarantees that eventually the first (starting from the *initial candidate to leader* and following the order defined by the ring) non-suspected process is the same for every correct process, and that it is correct. From this, it is easy to derive a deterministic function *leader* that returns the first non-suspected process. Hence, the ring-based algorithm implementing $\diamond\mathcal{S}$ of [3] implements also a failure detector of class $\diamond\mathcal{C}$.

In [5], an even more efficient algorithm implementing a failure detector of class $\diamond\mathcal{S}$ is proposed. In this case, the set of non-suspected processes contains only one process, that will eventually and permanently be the same correct process for all correct processes, which trivially gives us a possible *leader* function. Again, this algorithm implements also a failure detector of class $\diamond\mathcal{C}$.

3.4 Transforming $\diamond\mathcal{C}$ into $\diamond\mathcal{P}$

In this section, we present an efficient algorithm that transforms any failure detector of class $\diamond\mathcal{C}$ into a failure detector of class $\diamond\mathcal{P}$. The algorithm works as follows. Each *leader* process (i.e., each process that considers itself as leader by consulting its failure detection module) builds a local list of suspected processes by using time-outs, and sends its list periodically to the rest of processes. Concurrently, each non-leader process periodically sends an I-AM-ALIVE message to its leader process. Finally, when a process receives a list of suspected processes from its leader process, it adopts this list as its own list of suspected processes.

4. SOLVING CONSENSUS USING $\diamond\mathcal{C}$

In this section, we present an efficient Consensus algorithm based on $\diamond\mathcal{C}$. The algorithm proceeds in asynchronous rounds. Each round 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 to its coordinator. In Phase 2, each coordinator tries to gather a majority of estimates. If it succeeds, then it selects an estimate and sends it to all the processes as a proposition. In Phase 3, each process waits for a proposition from a coordinator or to suspect its own coordinator. If the process receives a proposition from some coordinator, then it adopts it and sends an *ack* message to this coordinator. Otherwise, it sends a *nack* message to its coordinator. Finally, in Phase 4 the coordinator that succeeded in Phase 2 and sent a proposition (if any) waits for a majority of *ack/nack* messages. If it gathers a majority of *ack*, then it broadcasts a request to decide its proposition.

Note that this $\diamond\mathcal{C}$-Consensus algorithm does not rely on the rotating coordinator paradigm, but on the eventual leader election functionality provided by the failure detector. As a result, in the case of stability of the failure detector, Consensus is solved in only one round, providing early consensus. In any $\diamond\mathcal{S}$-Consensus algorithm based on the rotating coordinator paradigm, the number of rounds can be $\Omega(n)$ once the failure detector is stable.

5. REFERENCES

[1] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, July 1996.

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

[3] 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*, pages 34–48. Bratislava, September 1999.

[4] M. Larrea, A. Fernández, and S. Arévalo. Eventually consistent failure detectors. Technical Report, Universidad Pública de Navarra, April 2000. Brief Announcement, 14th International Symposium on Distributed Computing, Toledo, Spain, October 2000.

[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*, pages 52–59, Nurenberg, Germany, October 2000.