CHAPTER

15 Failure detectors

15.1 Introduction

This chapter deals with the design of fault-tolerant distributed systems. It is widely known that the design and verification of fault-tolerent distributed systems is a difficult problem. Consensus and atomic broadcast are two important paradigms in the design of fault-tolerent distributed systems and they find wide applications. Consensus allows a set of processes to reach a common decision or value that depends upon the initial values at the processes, regardless of failures. In atomic broadcast, processes reliably broadcast messages such that they agree on the set of messages delivered and the order of message deliveries.

This chapter focuses on solutions to consensus and atomic broadcast problems in asynchronous distributed systems. In asynchronous distributed systems, there is no bound on the time it takes for a process to execute a computation step or for a message to go from its sender to its receiver. In an asynchronous distributed system, there is no upper bound on the relative processor speeds, execution times, clock drifts, and delay during the transmission of messages although they are finite. This is mainly casued by unpredictable loads on the system that causes asynchrony in the system and one cannot make any timing assumptions of any types. On the other hand, synchronous systems are characterized by strict bounds on the execution times and message transmission delays.

The asynchronous model of distributed system has simpler semantics when compared to synchronous model. Applications based on the asynchronous model are easily portable because there are no strict timing assumptions to take care of. The asynchronous model of distributed systems is very popular and has attracted lot of attention due to these reasons. Inspite of the attractiveness of asynchronous distributed systems, it is well known that consensus, atomic broadcast, and several other reliable broadcast problems cannot be solved deterministically even for a single process failure due to the unbounded timing characteristics. The main cause of this impossibility result is that it is very difficult to determine in asynchronous systems whether a process has failed or is simply taking a long time for execution; so it is difficult to deal with failures in these systems. On the other hand, in synchronous systems due to strict timing constraints, failures can easily be detected.

The asynchronous model of distributed systems is widely used, and such systems are prone to failures. Thus, detection and/or prevention of failures in these systems is of vital importance. The detection of process failures is a crucial task in the design of fault tolerant distributed systems. Detection of crashed processes is especially difficult in asynchronous systems as it is impossible to determine whether a process has really crashed or is very slow (as there are no timing constraints present).

In this chapter, we discuss the concept of unreliable failure detectors to deal with the impossibility results in asynchronous distributed systems with crash failures. Basically, the asynchronous model of computation is extended with a failure detection mechanism that is prone to errors in the sense that a process can brand another process as crashed even though the process is running. We study failure detectors in asynchronous distributed systems. We investigate two major problems faced in asynchronous distributed environments, namely, consensus and atomic broadcast. We study several solutions for these problems.

15.2 Unreliable failure detectors

Chandra and Toueg [3] introduced the concept of unreliable failure detectors and showed how unreliable failure detectors can be used to solve two fundamental paradigms of asynchronous distributed systems with crash failures, namely, consensus and atomic broadcast.

15.2.1 The system model

We consider asynchronous distributed systems in which there is no bound on message delay, clock drift, or the time taken to execute a step. The system consists of a finite set of n processes, $Q = \{p_1, p_2, ..., p_n\}$. Each pair of processes is connected by a reliable communication channel. A process can fail by crashing only, i.e., by prematurely halting. A process behaves correctly (i.e., according to its specification) until it crashes.

A discrete global clock is assumed, and the range of the clock's ticks, Φ, is the set of natural numbers. The global clock is used for the sake of simplicity of presentation and reasoning and is not accessible to the processes.

A process p_i is said to crash at time t if p_i does not perform any action after time t. Process failures are permanent; once a process crashes, it does not recover. A *correct* process is a process that has not crashed.

Informally, a run is an infinite execution of the system. Given any run σ, *Crashed*(t, σ) is the set of processes that have crashed by time t and $Up(t, \sigma)$ is the set of processes that are correct (i.e., have not crashed) by time t, that is, $Up(t, \sigma) = Q - Crashed(t, \sigma)$. $Crashed(\sigma)$ is the set of processes that have crashed in a run σ and is equal to $\bigcup_t Crashed(t, \sigma)$. $Up(\sigma)$ is the set of processes that are correct in a run σ and is equal to $Q - Crashed(\sigma)$. If a process $p \in Crashed(\sigma)$, we say that p is a faulty process in σ. If a process $p \in Up(\sigma)$, we say that p is a correct process in σ. We consider only execution runs where at least one process is correct.

Failure patterns and environments

A failure pattern is a function F from Φ to 2^Q, where $F(t)$ denotes the set of processes that have crashed through time t. An environment E is a set of failure patterns. Environments describe the crashes that can occur in a system. In general, we consider the environments that contain all possible failure patterns, i.e., there is no bound on the number of processes that crash.

Each process p_i has a local failure detector module of D, denoted by D_i. Associated with each failure detector D is a range R_D of values output by the failure detector. A failure detector history H with range R is a fuction H from $\Omega X \Phi$ to R. $D(F)$ denotes the set of possible failure detector histories permitted for the failure pattern F, i.e., each history represents a possible behavior of D for the failure pattern F. For any failure detector D, any failure pattern F, and any history H in $D(F)$, $H(p_i, t)$ is the set of processes suspected by process p_i at time t.

15.2.2 Failure detectors

A failure detector D is a distributed oracle that gives hints about failure patterns. Each process p_i in the distributed environment has its own local failure detector D_i, which monitors all other processes and maintains a list of processes, currently p_i suspects to have crashed. The suspicion is based on relative timeouts of other processes at p_i.

Thus, a failure detector D as the vector $D = \langle D_{p1}, D_{p2}, ...D_{pn} \rangle$, where D_i is the failure detector module at process p_i, that outputs the set of processes that it currently suspects to have crashed. Formally, a failure detector is a function "from time and the set of all runs" to 2^Q. $D_p(t, \sigma)$ is the set of processes that are suspected to have crashed by p's failure detector module at time t in run σ. If $q \in D_p(t, \sigma)$, we say that p suspects q at time t in run σ. After a process crashes, it is immaterial what its failure detector module indicates. We formalize this by assuming that if $p \in Crashed(t, \sigma)$, then $D_p(t, \sigma) = \phi$.

The failure detectors can make mistakes, i.e., a correct process may be added to the list of suspects and can later be removed if the failure detector realizes that it was a mistake. Thus, a failure detector may continually add and remove processes from its list of suspects. Processes can be added and removed from the list of suspects by each failure detector module any number of times. At any time, failure detector modules at two processes may have different lists of suspects.

It should be noted that the addition of a correct process to the list of suspects by any other process or by all other processes should not prevent this process from behaving correctly, according to its specifications.

15.2.3 Completeness and accuracy properties

Chandra and Toueg [3] classified failure detectors in terms of their completeness and accuracy properties. Informally, completeness requires that a failure detector eventually suspects all processes that have crashed and accuracy resticts the mistakes a failure detector can make (i.e., a correct process suspect another correct process). They define two types of completeness and four types of accuracy properties, giving rise to eight classes of failure detectors.

Chandra and Toueg [3] introduced the concept of reducibility among failure detectors. Informally, a failure detector D is *reducible* into another failure detector D' if there exists a distributed algorithm that can transform D into D'. In this case, any problem that can be solved using D' can also be solved using D. If two failure detectors are reducible to each other, they are said to be *equivalent*.

Chandra and Toueg [3] put failure detectors into eight classes and ordered them into a hierarchy according to the reducibility relationship. In this hierarchy, some failure detectors can solve the consensus problem with any number of process failures, while others require a certain number of correct processes to solve the consensus problem. This requirement and the boundary where this requirement becomes necessary have been clearly specified.

We now define completeness and accuracy properties of a failure detector.

Completeness

Definition 15.1 (Completeness) There is a time after which every process that has crashed is permanently suspected by a correct process.

Completeness can be of two types:

- **Strong completeness** Eventually every process that crashes is permanently suspected by every correct process. Notationally,

$$\forall \sigma, \forall p \in Crashed(\sigma), \forall q \in Up(\sigma), \exists t \textit{ such that } \forall t' \geq t : p \in D_q(t', \sigma).$$

- **Weak completeness** Eventually every process that crashes is permanently suspected by some correct process. Notationally,

$$\forall \sigma, \forall p \in Crashed(\sigma), \exists q \in Up(\sigma), \exists t \textit{ such that } \forall t' \geq t : p \in D_q(t', \sigma).$$

Note that completeness by itself may not be of much use. For example, a failure detector may satisfy the strong completeness property by having every process permanently suspect all other processes. Such a failure detector is useless because it provides no information about actual failures. Thus, a failure detector must satisfy some accuracy property to be useful. We define this property next.

Accuracy

Definition 15.2 (Accuracy) *There is a time after which a correct process is never suspected by any correct process.*

There are two types of accuracy property:

- **Strong accuracy** Correct processes are never suspected by any correct process. Formally,

$$\forall\sigma, \forall t, \forall p, q \in Up(t, \sigma) : p \notin D_q(t, \sigma).$$

Since in any practical system it is extremely difficult to achieve accuracy, we weaken it as follows:

- **Weak accuracy** Some correct process is never suspected by any correct process. Formally,

$$\forall\sigma, \exists p \in Up(\sigma), \forall t, \forall q \in Up(t, \sigma) : p \notin D_q(t, \sigma).$$

We collectively refer to strong accuracy and weak accuracy as the *perpetual accuracy* properties because these properties hold all the time. Note that even weak accuracy is difficult to achieve, because a failure detector (even at a correct process) may suspect a correct process and then later correct its mistake. The weak accuracy property does not permit this. Thus, we further weaken the accuracy requirement and allow failure detectors that may suspect a correct process at some points in the run, but they *eventually* satisfy the strong and weak accuracy properties.

Eventual accuracy

Definition 15.3 (Eventual accuracy) *We need not require accuracy property to be satisfied by each process at all the time. Instead, we require the accuracy property to be eventually satisfied.*

There are two types of eventual accuracy:

- **Eventual strong accuracy** There is a time after which correct processes are not suspected by any correct process. Formally,

$$\forall\sigma, \exists t, \forall t' \geq t, \forall p, q \in Up(t', \sigma) : p \notin D_q(t', \sigma).$$

- **Eventual weak accuracy** There is a time after which some correct process is not suspected by any correct process. Formally,

$$\forall\sigma, \exists t, \forall t' \geq t, \exists p \in Up(\sigma), \forall q \in Up(\sigma) : p \notin D_q(t', \sigma).$$

We collectively refer to eventual strong accuracy and eventual weak accuracy as the *eventual accuracy* properties because these properties hold eventually.

15.2.4 Types of failure detectors

Based on types of accuracies and completeness defined above, failure detectors can be classified into the following categories:

- **Perfect failure detectors** (P) Failure detectors that satisfy the strong completeness and the strong accuracy properties are called perfect failure detectors.
- **Eventually perfect failure detectors** ($\Diamond P$) Failure detectors that satisfy the strong completeness and the eventual strong accuracy properties are called eventually perfect failure detectors.
- **Strong failure detectors** (S) Failure detectors that satisfy the strong completeness and the weak accuracy properties are called strong failure detectors.
- **Eventually strong failure detectors** ($\Diamond S$) Failure detectors that satisfy the strong completeness and the eventual weak accuracy properties are called eventually strong failure detectors.
- **Weak failure detectors** (W) Failure detectors that satisfy the weak completeness and the weak accuracy properties are called weak failure detectors.
- **Eventually weak failure detectors** ($\Diamond W$) Failure detectors that satisfy the weak completeness and the eventual weak accuracy properties are called eventually weak failure detectors.
- Another class of failure detector is the one that satisfies weak completeness and strong accuracy properties. This class is denoted by ϑ.
- The last class is the set of failure detectors that satisfy weak completeness and eventually strong accuracy properties. This class is denoted by $\Diamond\vartheta$.

15.2.5 Reducibility of failure detectors

A failure detector D is reducible to another failure detector D' if there is an algorithm that transforms a failure detector D into another failure detector D'. A natural question is: what does it mean that an algorithm transforms D into D'? An algorithm $T_{D \rightarrow D'}$ transforms a failure detector D into another failure detector D' if and only if for every run R of $T_{D \rightarrow D'}$ under a failure pattern F using D, $output^R \in D'(F)$, where $output^R$ is the output of run R using failure detector D and $D'(F)$ denotes the set of histories of failure detector D' for failure pattern F. That is, variable $output_p$ at process p emulates the output of D'. Thus, $T_{D \to D'}$ can emulate D' using D. $T_{D \to D'}$ need not emulate all failure detector histories of D'; however, all failure detector histories it emulates must be histories of D'. Algorithm $T_{D \to D'}$ is called the *reduction algorithm*.

Given a reduction algorithm $T_{D \to E}$, any problem that can be solved using E, can also be solved using D. We illustrate this with an example: suppose a given algorithm A requires failure detector E, but only failure detector D is available. We can execute A using failure detector D as follows. Concurrently with A, processes run $T_{D \to E}$ to transform D to E. Algorithm A is modified at process p as follows: whenever A requires that p queries its failure detector module, p reads the current value of $output_p$, which is concurrently maintained by $T_{D \to E}$.

Since $T_{D \to E}$ is able to use D to emulate E, D must provide at least as much information about process failures as E does. Thus, if there is an algorithm $T_{D \to E}$ that transforms D into E, we say that E is weaker than D and denote it by $D \sqsubseteq E$. Note that $\sqsubseteq$ is a transitive relation. If $D \sqsubseteq E$ and $E \sqsubseteq D$, then we say that D and E are *equivalent* and denote it by $D \equiv E$.

If D and ε are two classes of failure detectors and there exists an algorithm $T_{D \to E}$ that can transform every failure detector $D \in D$ into a failure detector $E \in \varepsilon$, then we say that the class of failure detectors D is reducible to the class of failure detectors ε and this is denoted by $D \sqsubseteq \varepsilon$. In this case, ε is weaker than D. If $D \sqsubseteq \varepsilon$ and $\varepsilon \sqsubseteq D$, then D and ε are equivalent and this is denoted by $D \equiv \varepsilon$.

From a trivial reduction algorithm, where each process p periodically writes the current output of its failure detector module into $output_p$, the following relations between the classes of failure detectors are obvious:

Observation 15.1 $P \sqsubseteq \vartheta$, $S \sqsubseteq W$, $\diamond P \sqsubseteq \diamond\vartheta$, $\diamond S \sqsubseteq \diamond W$.

15.2.6 Reducing weak failure detector *W* to a strong failure detector *S*

In Algorithm 15.1, we give a reduction algorithm $T_{D \to D'}$ (due to Chandra and Toueg [3]) that transforms any given failure detector D that satisfies weak completeness, into a failure detector D' that satisfies strong completeness. D' satisfies the same accuracy property that D satisfies. Thus, this algorithm strenghtens the completeness while preserving the accuracy.

Informally, the conversion of any weak failure detector W to a strong failure detector S is as follows: initially, for every process p, $output_p$ is set to null. (Recall that $output_p$ is the variable emulating the output of the failure detector module D'_p.) Every process p periodically sends (p, $suspects_p$) to every process, where $suspects_p$ denotes the set of processes that p suspects according to its failure detector module D_p. When a process p recieves a message (q, $suspects_q$) from a process q, process p adds the suspect list of process q, $suspects_q$, to its output, $output_p$, and removes the process q from its output as it is a correct process.

Every process p executes the following:
$output_p \leftarrow \phi$
cobegin
||Task 1: repeat forever
$suspects_p \leftarrow D_p$ {p queries its local failure detector module D_p}
$send(p, suspects_p)$ to all other processes.
||Task 2: when receive $(q, suspects_q)$ for a process q
$output_p \leftarrow (output_p \cup suspects_q) - \{q\}$ {$output_p$ emulates E_p}
coend

Algorithm 15.1 Transforming weak completeness to strong completeness [3].

A correctness argument

The correctness proof of the algorithm involves showing the following three properties:

1. It transforms weak completeness into strong completeness.
2. It preserves the perpetual accuracy.
3. It preserves the eventual accuracy.

We show these properties in the following three lemmas.

Lemma 15.1 *Let p be any process that crashes. If eventually some correct process permanently suspects p in H_D, then eventually all correct processes permanently suspect p in $output^R$, where H_D is the history of failure detector D and $output^R$ is the output of an arbitrary run R using failure detector D.*

Since process p crashes, there is a time t' after which no process recieves a message from p. Suppose there is a correct process q that permanently suspects p in H_D after time t. Consider the execution of *task* 1 by process q after time $t_p = max(t, t')$. Process q sends a message $(q, suspects_q)$ such that $p \in suspects_q$ to all processes. Eventually, every correct process recieves $(q, suspects_q)$ and adds p to output (in *task* 2). Since no correct process recieves any messages from p after time t' and $t_p \geq t'$, no correct process removes p from its *output* after t_p. Thus, there is a time after which every correct process permanently suspects p in $output^R$.

Lemma 15.2 *Let p be any process. If no process suspects p in H_D before time t, then no process suspects p in $output^R$ before time t.*

Suppose there is a time t before which no process suspects process p in H_D. Thus, no process sends a message of type $(-, suspects)$ such that $p \in suspects$ before time t. Thus, no process q adds p to $output_q$ before time t.

Lemma 15.3 *Let p be a correct process. If there is a time after which no correct process suspects p in H_D, then there is a time after which no correct process suspects p in $output_R$.*

Suppose there is a time t after which no correct process suspects p in H_D. Thus, all processes that suspect p after time t eventually crash. Thus, there is time after which no process will send messages of type $(-, suspects)$ such that $p \in suspects$. Thus, there is a time t' after which no correct process recieves a message of type $(-, suspects)$ such that $p \in suspects$. Let q be a correct process. We need to show that there is a time after which q does not suspect p *in* $output^R$. Consider the execution of *task* 1 by process p after time t'. Process p sends the message $(p, suspects_p)$ to q. When q receives this message, it removes p from $output_q$ if p is present in $output_q$ (*task* 2). Note that q does not receive any messages of type $(-, suspects)$ such that $p \in suspects$ after time t'; therefore, q does not add p to $output_q$ after time t'. Thus, there is a time after which q does not suspect p in $output^R$.

Theorem 15.1 $\vartheta \sqsubseteq P, W \sqsubseteq S, \diamond\vartheta \sqsubseteq \diamond P$ *and* $\diamond W \sqsubseteq \diamond S$.

Proof Let D be any failure detector in ϑ, W, $\diamond\vartheta$, or $\diamond W$. We show that $T_{D \to E}$ transforms D into a failure detector E in P, S, $\diamond P$, or $\diamond S$. Since D satisfies weak completeness, E satisfies strong completeness (from Lemma 15.1). We now argue that D and E have the same accuracy properties. If D is in ϑ or W, then D and E have the same accuracy property (from Lemma 15.2). If D is in $\diamond\vartheta$ or $\diamond W$, then D and E have the same accuracy property (from Lemma 15.3).

Thus, we have:

$$\vartheta \sqsubseteq P, W \sqsubseteq S, \diamond\vartheta \sqsubseteq \diamond P \text{ and } \diamond W \sqsubseteq \diamond S. \qquad \square$$

From Theorem 15.1 and Observation 15.1, we have the following result:

$$P \equiv \vartheta, S \equiv W, \diamond P \equiv \diamond\vartheta, \text{ and } \diamond S \equiv \diamond W.$$

A significance of this result is that if we solve a problem for the four failure detectors with strong completeness, the problem is automatically solved for the remaining four failure detectors.

15.2.7 Reducing an eventually weak failure detector $\diamond W$ to an eventually strong failure detector $\diamond S$

Algorithm 15.2 gives an algorithm that converts any eventually weak failure detector $D \in \diamond W$ into an eventually strong failure detector $E \in \diamond S$. Q is the set of all processes.

At process p, variable $suspected_p(r, q)$ denotes how many times process q has suspected process r and variable $refuted_p(r, q)$ denotes how many times process r has refuted process q. Both variables are initialized to zero. S_p denotes the suspect list of process p.

Process p runs the following:

for all q, $r \in Q$
{Number of times q suspected r according to p}
$suspected_p(r, q) \leftarrow 0$
{Number of times r refuted q according to p}
$refuted_p(r, q) \leftarrow 0$

cobegin
||Task 1: repeat forever
if ($r \in D_p$ and $refuted_p(r, p) \leq suspected_p(r, p)$) then
    p rbcasts (p, *suspects*, r, $refuted_p(r, p) + 1$)

||Task 2: when p rbdelivers (q, *suspects*, r, k)
    $suspected_p(r, q) \leftarrow$ k
    if $p = r$ then p rbcasts (p, *refutes*, q, k)

||Task 3: when p redelivers (r, *refutes*, q, k)
    $refuted_p(r, q) \leftarrow k$

||Task 4: repeat forever
    for all processes r
        if $\exists\ q : suspected_p(r, q) > refuted_p(r, q)$
            then $S_p \leftarrow S_p \bigcup \{r\}$
        else $S_p \leftarrow S_p - \{r\}$
coend

Algorithm 15.2 An algorithm to reduce an eventually weak failure detector into an eventually strong failure detector [3].

An explanation of the algorithm

The algorithm consists of four tasks.

In task 1, a process p continuously performs the following for every process r that it suspects according to its failure detector module D_p: if the number of times process r is suspected by p is greater than the number of times r has refuted p, then p broadcasts a suspect message that contains the incremented refuted value.

In task 2, when process p receives a suspect message (q, *suspects*, r, k) from a process q, it updates $suspected_p(r, q)$ to k. If process p discovers that it was erroneously suspected by process q, p broadcasts an appropriate refutation, refuting the suspicion of process q.

In task 3, when process p receives a refutation message (r, *refutes*, q, k) from process r, it updates $refuted_p(r, q)$ to k.

In task 4, the following is repeatedly done for every process r: if there exists a process q such that the number of times q suspects process r is greater than the number of times the process r refutes q according to p, then process r is added to the suspect list of process p. Otherwise, r is removed from the suspect list of process p.

Correctness argument

A correctness argument of the algorithm is as follows. When a process q receives a suspect message accusing process p, process q may add p to its list of suspects S_q. However, upon receiving p's refutation, process q will remove p from its list of suspects S_q. However, p can be suspected again and added to S_q a second time. However, a further refutation from p will cause p to be again removed from S_q. Thus, a possibly infinite sequence of suspicions followed by corresponding refutations may occur, resulting in p being repeatedly added to and removed from S_q. However, from the eventual weak accuracy property of D, there is a time after which some correct process is not suspected. That is, there is a process p such that there is a time after which no correct process receives a message of type (*, *suspects*, p, k), suspecting p. Thus, after a time no correct process adds process p to its suspect list. Together with the refutation mechanism, this ensures the eventual weak accuracy property of the constructed E.

Now let us see why E satisfies the strong completeness property. Since D satisfies the weak completeness property, eventually every process that crashes is permanently suspected by some correct process, say p. Thus, eventually process p will repeatedly broadcast (p, *suspects*, *, k) messages for these crashed processes and since these processed have crashed, no one will send refute messages for them. Thus all crashed processes will eventually belong to the suspect list of all correct processes. Thus, due to the broadcast of suspect messages and weak completeness property of D, E satisfies the strong completeness property. Thus E satisfies strong completeness and weak accuracy.

15.3 The consensus problem

In the consensus problem, each correct process proposes a value and all processes must reach a unanimous and irrevocable decision on a value that is related to the proposed values [9]. The consensus problem is defined in terms of the following properties:

- **Termination** Every correct process eventually decides some value.
- **Uniform integrity** Every process decides at most once.
- **Agreement** No two correct processes decide differently.
- **Uniform validity** If a process decides a value v, then some process proposed v.