

## 4 Consensus and Reliable Broadcasts

### 4.1 Introduction

Consider a complete *asynchronous* network of  $n$  servers  $\mathcal{P} = \{P_1, \dots, P_n\}$ . Up to  $t$  servers may *fail* by silently *crashing* (and they do not recover). A server that never crashes is called *correct*. Every pair of servers is linked by a reliable point-to-point communication channel (i.e., if a correct server sends a message to another correct server, it will *eventually* receive the message).

Coordination of all (correct) servers in this model has received considerable attention. Many relevant practical problems, such as atomic broadcast or decentralized atomic commitment of transactions, can be reduced to the problem of reaching consensus.

### 4.2 Consensus

Consensus is defined in terms of two events, *propose* and *decide*; every server  $P_i$  executes *propose*( $v$ ), where  $v$  is the value that  $P_i$  “proposes,” and every server  $P_i$  executes *decide*( $v$ ), where  $v$  is the value for which  $P_i$  “decides.”

**Definition 4.1 (Consensus).** A *consensus* protocol satisfies:

*Validity:* If a server *decides*  $v$ , then  $v$  was *proposed* by some server.

*Agreement:* No two servers decide differently.

*Termination:* Every correct server eventually *decides*.

This actually defines *uniform* consensus, which means that the properties hold also for faulty servers until they fail; in *non-uniform* consensus, *agreement* is restricted to the correct servers, which is sometimes easier to achieve.

It is not possible to implement Definition 4.1 in asynchronous systems [FLP85], even if  $t = 1$ . Possible solutions are (1) to use randomization or (2) to make *timing assumptions*. We explore (2) here and discuss (1) in the context of Byzantine agreement.

**Example 4.2 (Non-Blocking Atomic Commitment using Consensus).** At the end of a distributed computation, a group of processes (servers) enters a protocol to commit the changes of their local state. Every process may propose to *commit* or to *abort* the computation; if one process *aborts*, then all others must also *abort*, otherwise they must *commit*. Some processes may fail (we assume here that they *never* recover) and if a process believes that another process has failed (e.g., by using a “failure detector,” see below), it is also possible to *abort*.

This is a variation of consensus with domain  $\{\text{commit}, \text{abort}\}$  and the following notion of *validity*:

- If a process decides *commit*, then all processes have proposed *commit*.
- If all processes propose *commit* and none of them is believed to have failed, they must decide *commit*.

The following algorithm implements distributed non-blocking atomic commitment using consensus:

1. Every process sends its proposed action, *commit* or *abort*, to all others.
2. When a process receives  $n$  messages indicating *commit*, it starts consensus and proposes to *commit*; otherwise, when at least one process sent a message that indicates *abort* or when a process believes that another process has failed, it starts consensus and proposes to *abort*.
3. Return the decision of the consensus protocol.

However, the database literature usually allows recoveries, which makes this a different problem. Practical database systems use a single transaction coordinator that implements the decision [BHG87]; when the coordinator fails, the 2PC protocol blocks until the coordinator recovers, and the 3PC protocol needs a complex and synchronous recovery procedure when the backup coordinators fail as well. See also [BT93, GL04].

### 4.3 Failure Detectors

**Definition 4.3 (Failure Detector [CT96]).** Every server  $P_i$  has a local *failure detector* module  $\mathcal{D}_i$  that (periodically) outputs a list of servers that it suspects to have crashed. We say  $P_i$  *suspects*  $P_j$  whenever  $j \in \mathcal{D}_i$

A failure detector (FD) represents an abstraction of a timing assumption; a FD is described by its abstract properties rather than through an implementation. We usually speak of “the” failure detector  $\mathcal{D}$  when every server has access to a local FD module  $\mathcal{D}_i$  with the properties of  $\mathcal{D}$ ; note that the outputs of the modules at different servers may differ from each other.

**Definition 4.4 (Completeness).**

- A failure detector satisfies *strong completeness* if eventually *every* server that crashes is permanently suspected by *every* correct server.
- A failure detector satisfies *weak completeness* if eventually *every* server that crashes is permanently suspected by *some* correct server.

Completeness alone is trivial to satisfy and hence not useful.

**Definition 4.5 (Accuracy).**

- A failure detector satisfies *strong accuracy* if *no* server is suspected before it crashes.
- A failure detector satisfies *weak accuracy* if *some* correct server is never suspected.

Such failure detectors must never output false suspicions about any server (strong accuracy) or about one particular server (weak accuracy). Therefore they are rather difficult to implement, and one considers also the following relaxation.

**Definition 4.6 (Eventual Accuracy).**

- A failure detector satisfies *eventual strong accuracy* if there is a time after which *no* correct server is suspected by any correct server.
- A failure detector satisfies *eventual weak accuracy* if there is a time after which *some* correct server is never suspected.

A failure detector is characterized by a completeness and by an accuracy condition. Two notions of completeness and four forms of accuracy define eight classes of FD:

<i>completeness</i>	<i>accuracy</i>			
			<i>eventually</i>	
	<i>strong</i>	<i>weak</i>	<i>strong</i>	<i>weak</i>
<i>strong</i>	$\mathcal{P}$	$\mathcal{S}$	$\diamond\mathcal{P}$	$\diamond\mathcal{S}$
<i>weak</i>	$\mathcal{Q}$	$\mathcal{W}$	$\diamond\mathcal{Q}$	$\diamond\mathcal{W}$

$\mathcal{P}$  is also called the [class of] “perfect,”  $\mathcal{S}$  the [class of] “strong,” and  $\mathcal{W}$  the [class of] “weak” failure detectors; read  $\diamond$  as “eventually.”

**Definition 4.7 (Reducibility).** If there exists an algorithm that emulates all properties of a FD  $\mathcal{D}'$  using only the output from a FD  $\mathcal{D}$ , we say that  $\mathcal{D}'$  is *reducible* to  $\mathcal{D}$  and that  $\mathcal{D}'$  is *weaker* than  $\mathcal{D}$ , written  $\mathcal{D}' \leq \mathcal{D}$ .

Similarly for classes of FD: if every FD in a class  $\mathcal{C}'$  is reducible to a FD in a class  $\mathcal{C}$ , we say that  $\mathcal{C}'$  is *reducible* to  $\mathcal{C}$  and write  $\mathcal{C}' \leq \mathcal{C}$ .

If  $\mathcal{D} \leq \mathcal{E}$  and  $\mathcal{E} \leq \mathcal{D}$ , then  $\mathcal{D}$  and  $\mathcal{E}$  are *equivalent*, written  $\mathcal{D} \equiv \mathcal{E}$ .

Trivially, we have  $\mathcal{Q} \leq \mathcal{P}$ ,  $\mathcal{W} \leq \mathcal{S}$ , etc.

**Theorem 4.8.** *Weak and strong completeness are equivalent, i.e.,  $\mathcal{P} \equiv \mathcal{Q}$ ,  $\mathcal{S} \equiv \mathcal{W}$ , etc.*

*Proof.* Reduce FD  $\mathcal{S}$  with strong completeness to FD  $\mathcal{D}$  with weak completeness as follows: every  $P_i$  periodically sends the output of  $\mathcal{D}_i$  to all servers; when  $P_i$  receives such a message with the output of  $\mathcal{D}_j$ , it updates  $\mathcal{S}_i$  to  $\mathcal{S}_i \cup \mathcal{D}_j \setminus \{P_j\}$ . □

## 4.4 Consensus using Failure Detectors

**Algorithm 4.9 (Consensus based on  $\mathcal{S}$  or  $\diamond\mathcal{S}$  [MR99]).** Every  $P_i$  has access to a failure detector  $\mathcal{D}_i$ ;  $\mathcal{D}_i$  is either in  $\mathcal{S}$  or in  $\diamond\mathcal{S}$  for all servers.  $P_i$  executes the following algorithm.

```

upon propose( $v$ ):
     $r \leftarrow 0$  // current round
    while not decided do
         $c \leftarrow (r \bmod n) + 1$  // current coordinator
         $u \leftarrow \perp$  // value received from coordinator  $P_c$  or  $\perp$  if none
        if  $i = c$  then
            send message (propose,  $r, v$ ) to all
            wait for message (propose,  $r, v'$ ) from  $P_c$  or  $c \in \mathcal{D}_i$ 
            if a message (propose,  $r, v'$ ) was received then
                 $u \leftarrow v'$ 
            send message (vote,  $r, u$ ) to all
            wait for messages (vote,  $r, u'$ ) from all  $P_j \in Q$  for some  $Q$  s.t.
                 $Q = \mathcal{P} \setminus \mathcal{D}_i$  with FD  $\mathcal{S}$ 
                 $|Q| = \lceil \frac{n+1}{2} \rceil$  with FD  $\diamond\mathcal{S}$ 
             $U \leftarrow$  set of values  $u'$  received in vote messages
            if  $U = \{u'\}$  for some  $u' \neq \perp$  then
                send message (decide,  $u'$ ) to all
            else if  $U = \{u', \perp\}$  then
                 $v \leftarrow u'$ 
             $r \leftarrow r + 1$ 

upon receiving a message (decide,  $v'$ ):
    if not decided then
        send the message (decide,  $v'$ ) to all
        decide( $v'$ )

```

The algorithm uses the “rotating coordinator” paradigm and provides “early termination.” The way in which the *decide* message is disseminated is a “reliable broadcast” that tolerates crash failures (see next section).

**Theorem 4.10.** *Algorithm 4.9 implements consensus with a strong failure detector ( $\mathcal{S}$ ) for  $n > t$ .*

*Proof idea.* Let  $P_c$  be the correct server that is never suspected and let  $v_c$  be its vote at begin of round  $c$ . All correct servers will decide in round  $c$ . Note that  $c \leq n$ .  $\square$

**Theorem 4.11.** *Algorithm 4.9 implements consensus with an eventually strong failure detector ( $\diamond\mathcal{S}$ ) for  $n > 2t$ .*

*Proof idea.* Agreement and termination are based on these facts:

- If two servers decide in the same round, then they decide the same value.

- Suppose some server decides  $v'$  in round  $r$ . Then the value  $v'$  is contained in the `propose` message of round  $r$  and has been “locked” in the sense that it is not possible for any server in round  $r' > r$  to decide  $u' \neq v'$  or to assign  $u' \neq v'$  to its  $v$  because every two sets of  $\lceil \frac{n+1}{2} \rceil$  servers intersect. (Such a set forms a “quorum.”)
- If some server decides, then every other server eventually decides (because it receives a `decide` message).
- There is some round in which the coordinator  $P_c$  is not suspected by any server; all correct servers decide in this round.

□

Combining Theorems 4.10 and 4.11 with Theorem 4.8 shows that consensus can also be implemented using the weak failure detectors  $\mathcal{W}$  and  $\diamond\mathcal{W}$ . Moreover, it has been shown that  $\diamond\mathcal{W}$  is the weakest failure detector that solves consensus in the sense of Definition 4.7 [CHT96].

**Corollary 4.12.** *Consensus can be implemented in asynchronous systems with a weak failure detector for  $n > t$  and with an eventually weak failure detector for  $n > 2t$ .*

## References

- [BHG87] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency control and recovery in database systems*, Addison-Wesley, 1987.
- [BT93] Ö. Babaoglu and S. Toueg, *Non-blocking atomic commitment*, Distributed Systems (S. J. Mullender, ed.), ACM Press & Addison-Wesley, New York, 1993.
- [CHT96] T. D. Chandra, V. Hadzilacos, and S. Toueg, *The weakest failure detector for solving consensus*, Journal of the ACM **43** (1996), no. 4, 685–722.
- [CT96] T. D. Chandra and S. Toueg, *Unreliable failure detectors for reliable distributed systems*, Journal of the ACM **43** (1996), no. 2, 225–267.
- [FLP85] M. J. Fischer, N. A. Lynch, and M. S. Paterson, *Impossibility of distributed consensus with one faulty process*, Journal of the ACM **32** (1985), no. 2, 374–382.
- [GL04] J. Gray and L. Lamport, *Consensus on transaction commit*, Technical Report MSR-TR-2003-96, Microsoft Research, 2004, Revised, 21 Feb 2005.
- [MR99] A. Mostfaoui and M. Raynal, *Solving consensus using Chandra-Toueg’s unreliable failure detectors: A general quorum-based approach*, Proc. 13th International Symposium on Distributed Computing (DISC) (P. Jayanti, ed.), Lecture Notes in Computer Science, vol. 1693, Springer, 1999.