Chapter 9

Synchronous agreement

Here we'll consider synchronous agreement algorithm with stopping failures, where a process stops dead at some point, sending and receiving no further messages. We'll also consider Byzantine failures, where a process deviates from its programming by sending arbitrary messages, but mostly just to see how crash-failure algorithms hold up; for algorithms designed specifically for a Byzantine model, see Chapter 10.

If the model has communication failures instead, we have the coordinated attack problem from Chapter 8.

9.1 Problem definition

We use the usual synchronous model with n processes with binary inputs and binary outputs. Up to f processes may fail at some point; when a process fails, one or one or more of its outgoing messages are lost in the round of failure and all outgoing messages are lost thereafter.

There are two variants on the problem, depending on whether we want a useful algorithm (and so want strong conditions to make our algorithm more useful) or a lower bound (and so want weak conditions to make our lower bound more general). For algorithms, we will ask for these conditions to hold:

Agreement All non-faulty processes decide the same value.

Validity If all processes start with the same input, all non-faulty processes decide it.

Termination All non-faulty processes eventually decide.

For lower bounds, we'll replace validity with **non-triviality** (often called validity in the literature):

Non-triviality There exist failure-free executions A and B that produce different outputs.

Non-triviality follows from validity but doesn't imply validity; for example, a non-trivial algorithm might have the property that if all non-faulty processes start with the same input, they all decide something else.

In §9.2, we'll show that a simple algorithm gives agreement, termination, and validity with f failures using $f+1$ rounds. We'll then show in §9.3 that non-triviality, agreement, and termination imply that $f+1$ rounds is the best possible. In Chapter 10, we'll show that the agreement is still possible in $f+1$ rounds even if faulty processes can send arbitrary messages instead of just crashing, but only if the number of faulty processes is strictly less than $n/3$.

9.2 Solution using flooding

The flooding algorithm, due to Dolev and Strong [DS83] gives a straightforward solution to synchronous agreement for the crash failure case. It runs in $f+1$ rounds assuming f crash failures. (This algorithm is also described in more detail in [AW04, §5.1.3] or [Lyn96, §6.2.1].)

Each process keeps a set of (process, input) pairs, initially just $\{(\mathsf{myId}, \mathsf{myInput})\}$. At round r, I broadcast my set to everybody and take the union of my set and all sets I receive. At round $f+1$, I decide on $f(S)$, where f is some fixed function from sets of process-input pairs to outputs that picks some input in S: for example, f might take the input with the smallest process-id attached to it, take the max of all known input values, or take the majority of all known input values.

Lemma 9.2.1. *After $f+1$ rounds, all non-faulty processes have the same set.*

Proof. Let S_i^r be the set stored by process i after r rounds. What we'll really show is that if there are no failures in round k, then $S_i^r = S_j^r = S_i^{k+1}$ for all i, j, and $r > k$. To show this, observe that no faults in round k means that all processes that are still alive at the start of round k send their message to all other processes. Let L be the set of live processes in round k. At the end of round k, for i in L we have $S_i^{k+1} = \bigcup_{j \in L} S_j^k = S$. Now we'll consider some round $r = k+1+m$ and show by induction on m that $S_i^{k+m} = S$; we

already did $m = 0$, so for larger m notice that all messages are equal to S and so S_i^{k+1+m} is the union of a whole bunch of S's. So in particular we have $S_i^{f+1} = S$ (since some failure-free round occurred in the preceding $f+1$ rounds) and everybody decides the same value $f(S)$. □

Flooding depends on being able to trust second-hand descriptions of values; it may be that process 1 fails in round 0 so that only process 2 learns its input. If process 2 can suddenly tell 3 (but nobody else) about the input in round $f + 1$—or worse, tell a different value to 3 and 4—then we may get disagreement. This remains true even if Byzantine processes can't fake inputs (e.g., because an input value is really a triple $(i, v, \mathsf{signature}(v))$ using an unforgeable digital signature)—the reason is that a Byzantine process could horde some input $(i, v, \mathsf{signature}(v))$ until the very last round and then deliver it to only some of the non-faulty processes.

9.3 Lower bound on rounds

Here we show that synchronous agreement requires at least $f + 1$ rounds if f processes can fail. This proof is modeled on the one in [Lyn96, §6.7] and works backwards from the final state; for a proof of the same result that works in the opposite direction, see [AW04, §5.1.4]. The original result (stated for Byzantine failures) is due to Dolev and Strong [DS83], based on a more complicated proof due to Fischer and Lynch [FL82]; see the chapter notes for Chapter 5 of [AW04] for more discussion of the history.

Note that unlike the algorithms in the preceding and following sections, which provide validity, the lower bound applies even if we only demand non-triviality.

Like the similar proof for coordinated attack (§8.2), the proof uses an indistinguishability argument. But we have to construct a more complicated chain of intermediate executions.

A **crash failure** at process i means that (a) in some round r, some or all of the messages sent by i are not delivered, and (b) in subsequent rounds, no messages sent by i are delivered. The intuition is that i keels over dead in the middle of generating its outgoing messages for a round. Otherwise i behaves perfectly correctly. A process that crashes at some point during an execution is called **faulty**

We will show that if up to f processes can crash, and there are at least $f + 2$ processes,[1] then at least $f + 1$ rounds are needed (in some execution)

[1] With only $f + 1$ processes, we can solve agreement in f rounds using flooding. The

for any algorithm that satisfies agreement, termination, and non-triviality. In particular, we will show that if all executions run in f or fewer rounds, then the indistinguishability graph is connected; this implies non-triviality doesn't hold, because (as in §8.2), two adjacent states must decide the same value because of the agreement property.[2]

Now for the proof. To simplify the argument, let's assume that all executions terminate in exactly f rounds (we can always have processes send pointless chitchat to pad out short executions) and that every processes sends a message to every other process in every round where it has not crashed (more pointless chitchat). Formally, this means we have a sequence of rounds $0, 1, 2, \ldots, f-1$ where each process sends a message to every other process (assuming no crashes), and a final round f where all processes decide on a value (without sending any additional messages).

We now want to take any two executions A and B and show that both produce the same output. To do this, we'll transform A's inputs into B's inputs one process at a time, crashing processes to hide the changes. The problem is that just crashing the process whose input changed might change the decision value—so we have to crash later witnesses carefully to maintain indistinguishability all the way across the chain.

Let's say that a process p **crashes fully** in round r if it crashes in round r and no round-r messages from p are delivered. The **communication pattern** of an execution describes which messages are delivered between processes without considering their contents—in particular, it tells us which processes crash and what other processes they manage to talk to in the round in which they crash.

With these definitions, we can state and prove a rather complicated induction hypothesis:

Lemma 9.3.1. *For any f-round protocol with $n \geq f+2$ processes permitting up to f crash failures; any process p; and any execution A in which at most one process crashes per round in rounds $0 \ldots r-1$, p crashes fully in round $r+1$, and no other processes crash; there is a sequence of executions $A = A_0 A_1 \ldots A_k$ such that each A_i is indistinguishable from A_{i+1} by some*

idea is that either (a) at most $f-1$ processes crash, in which case the flooding algorithm guarantees agreement; or (b) exactly f processes crash, in which case the one remaining non-faulty process agrees with itself. So $f+2$ processes are needed for the lower bound to work, and we should be suspicious of any lower bound proof that does not use this assumption.

[2]The same argument works with even a weaker version of non-triviality that omits the requirement that A and B are failure-free, but we'll keep things simple.

process, each A_i has at most one crash per round, and the communication pattern in A_k is identical to A except that p crashes fully in round r.

Proof. By induction on $f - r$. If $r = f$, we just crash p in round r and nobody else notices. For $r < f$, first crash p in round r instead of $r + 1$, but deliver all of its round-r messages anyway (this is needed to make space for some other process to crash in round $r + 1$). Then choose some message m sent by p in round r, and let p' be the recipient of m. We will show that we can produce a chain of indistinguishable executions between any execution in which m is delivered and the corresponding execution in which it is not.

If $r = f - 1$, this is easy; only p' knows whether m has been delivered, and since $n \geq f + 2$, there exists another non-faulty p'' that can't distinguish between these two executions, since p' sends no messages in round f or later. If $r < f - 1$, we have to make sure p' doesn't tell anybody about the missing message.

By the induction hypothesis, there is a sequence of executions starting with A and ending with p' crashing fully in round $r + 1$, such that each execution is indistinguishable from its predecessor. Now construct the sequence

$$\begin{aligned} A &\to (A \text{ with } p' \text{ crashing fully in } r+1) \\ &\to (A \text{ with } p' \text{ crashing fully in } r+1 \text{ and } m \text{ lost}) \\ &\to (A \text{ with } m \text{ lost and } p' \text{ not crashing}). \end{aligned}$$

The first and last step apply the induction hypothesis; the middle one yields indistinguishable executions since only p' can tell the difference between m arriving or not and its lips are sealed.

We've shown that we can remove one message through a sequence of executions where each pair of adjacent executions is indistinguishable to some process. Now paste together $n - 1$ such sequences (one per message) to prove the lemma. □

The rest of the proof: Crash some process fully in round 0 and then change its input. Repeat until all inputs are changed.

9.4 Variants

So far we have described **binary consensus**, since all inputs are 0 or 1. We can also allow larger input sets. With crash failures, this allows a stronger validity condition: the output must be equal to some input. Note that this

stronger condition doesn't work if we have Byzantine failures. (Exercise: why not?)