Lecture 4

Reaching Consensus

4.1 The Problem

In the previous lecture, we've seen that consensus can't be solved in asynchronous systems. That means asking "how can we solve consensus?" is pointless. But it doesn't mean we should give up! Impossibility results and lower bounds tell us what questions we should not ask, but at the same time, we learn a lot about the questions we *should* ask!

For starters, you saw in the exercises that the question "Can we solve consensus *in synchronous systems*?" makes a lot of sense. And having a positive answer to this already implies something very important: In the presence of faults, synchronous and asynchronous systems are fundamentally different. Not just in terms of message or time complexity, but in terms of what type of problems can be solved! In particular, we cannot hope for a convenient tool like synchronizers.

This immediately raises more important questions. Given that synchrony is a strong (read: often unrealistic) assumption, where exactly does the boundary between consensus being solvable and unsolvable lie? This has been studied quite extensively, but today we're going to focus on something else. We're asking the question

"Can consensus be solved *almost* certainly despite asynchrony?"

Put differently, instead of making stronger assumptions on the system, we make our problem easier. We permit randomization[1] and relax the requirements of the binary consensus problem:

Agreement Correct nodes i output the same value $o = o_i$.

Validity If all correct nodes have the same input b, then $o = b$.

Termination *With probability 1,* all correct nodes decide on an output and terminate.

Wait a second – doesn't termination with probability 1 mean that it is *certain* that the algorithm will eventually terminate? This is not true, as seen by the

[1]Ok, ok. Depending on your point of view, we *do* make the system more powerful.

following simple example. We take an *unbiased* coin and flip it. If the result is "heads," we stop. Otherwise, we repeat this. In principle, it is possible that this goes on forever. However, the probability for this to happen is, for each $r \in \mathbb{N}$, bounded by the probability of getting "heads" r times in a row. This probability is 2^{-r}, which goes to 0 for $r \to \infty$. Hence, with probability 0, we never stop, but it is not impossible that we never stop!

Ok, so the above definition may make some sense. However, it *does* open a new can of worms. What do we even *mean* by probability here? An algorithm is not just a sequence of coin flips! Changing the system model such that the next node that makes a step or that crashes is random might be too optimistic (there could be systematic faults), and it's not clear what the respective probability distributions should look like.

4.2 The Model

The simplest way to resolve the above conundrum is to interpret randomized algorithms as deterministic ones with additional random input. Each node gets a separate input, which is an infinite string of independent, unbiased random bits (the node will generate the finite prefix of bits it uses during the course of the algorithm). The nodes now execute a "deterministic" algorithm that may, at any time, use some of these random bits to decide on its next step. Our probability space now consists of all the possible combinations of such strings with the probability measure induced by each individual bit being 0 with independent probability $1/2$.

Formally, we can now say what "termination with probability 1" means as follows. We fix our randomized algorithm (i.e., our "deterministic" algorithm that is given the random input strings), draw the random strings from the distribution, and then check whether there is *any* execution for which termination is violated. The latter must happen with probability 0 in terms of the distribution of random input strings. Of course, we will also have to show that agreement and validity are *always* satisfied.

Moreover, we get rid of another, often unrealistic, assumption, namely that failing nodes "crash" nicely. In fact, we're going all the way, to so-called *Byzantine* failures. This means that failing nodes can behave in *any* way: They can crash, they can send erroneous and conflicting messages to different nodes, and they can violate the protocol in an arbitrary way. They can even stick to the protocol (for a while), making it hard or impossible to figure out that they are faulty. If you think this sounds crazy, you are not *completely* wrong. However, if we can handle this scenario, we don't have to worry about the type of faults that can actually happen!

Let's say we have f faulty nodes. Faulty nodes can pretend to be correct, but with different input. By the validity condition, this implies that in a fault-free execution, we must decide on 0 (or 1), if $n - f$ nodes have input 0 (or 1), respectively. This immediately implies that consensus cannot be solved if $f \geq n/2$! In fact, more careful reasoning shows that $f \geq n/3$ also breaks our necks: If we have two sets of f correct nodes with inputs 0 and 1, respectively, the faulty nodes can play to each set the role of correct nodes with their input; the correct nodes in each set then cannot figure out whether the respective other set of nodes is faulty or the real baddies, and thus cannot distinguish from $n - f$

correct nodes having input 0 respectively 1. The surprise is that, indeed, it is possible to handle any number f of faults strictly smaller than $n/3$! Throughout this lecture, we will hence assume that $f < n/3$.

Finally, we will not design our algorithms in the shared memory model from the previous lecture, but in the asynchronous message passing model. However, we assume that "everyone can talk to everyone," i.e., the communication graph is complete.

Remarks:

- One way to visualize the probability space we're using is to think of the input string of each node as encoding a uniformly random real number from $[0, 1]$. The first bit says whether it's from $[0, 0.5]$ or $[0.5, 1]$, etc.[2] The complete space is then the product of all the individual ones (for correct nodes), i.e., a hypercube of dimension $n - f$; the bit strings determine a uniformly random point in this hypercube.

- A subset of the hypercube with measure 0 is e.g. given by a side of the cube, which is equivalent to fixing a single node's string to be all 0s or all 1s. As we already saw, this happens with probability 0!

- Since this hypercube has exactly volume 1, the probability to draw bit strings corresponding to a given subset of the hypercube is exactly the volume of the subset. For instance, requiring that the first bit of some fixed node is 0 defines a "half-cube" and happens with probability 1/2.

4.3 First Thoughts and a Key Ingredient

Let's start with some basic observations:

- No matter how the algorithm is going to look like, we cannot trust that any information that comes from fewer than $f + 1$ nodes is definitely correct (unless there's a way to be sure that not all faulty nodes are among them).

- Since the system is asynchronous, waiting until this many messages arrived is the only way of doing this.

- However, one must not wait to hear from everyone! Byzantine nodes may decide to not send anything, so a node cannot be sure that more than $n - f$ messages arrive (if every node is supposed to send one)!

- We are going to think about our algorithms in terms of rounds again, where in each round, each node is supposed to send one and only one message to each other node. But as described above, each node can only expect to receive $n - f$ of these messages. You can view this as a "deteriorated" version of the α-synchronizer.

- It's important to understand that if the faulty nodes *do* send messages, it may be the good guys' messages that get discarded while waiting for the end of the round. Only $n - 2f$ of the messages a node receives for a round are certainly legit!

[2]This way we have several encodings for the same number, as, e.g., $0.1\overline{0} = 0.0\overline{1}$. Throwing some measure theory at this shows that this is ok, though.

- That means if actually all correct nodes agree on something (e.g., they have the same input), a node will "observe" at least $n - 2f$ being in agreement.
- Since the algorithm does not know f, it will have to use an upper bound on f whenever counting messages as above. We'll slightly abuse notation and still use the variable f in algorithms, but remember that in fact our algorithms should use a parameter t (instead of f) and the corresponding assumption in the analysis is that $f \leq t < n/3$.

Ok, so let's get started. Think about validity first. We need to make sure that if everyone has input 0, this is also the output – deterministically. So, all nodes announce their inputs; for simplicity, we assume that nodes also send messages "to themselves." If out of the $n - f$ messages received by a node, $n - 2f$ say "my input is 0," it's possible that all nodes had input 0. Say a node in this situation decides that it will output 0, tells the others, and terminates. This means that if *not* all inputs were 0, we need to make sure that agreement holds, i.e., everyone else decides on 0, too!

Given this insight, consider now the case that some correct node following the above rule decides on 0 and terminates. It must have received at least $n - 3f > 0$ messages saying "my input is 0" from correct nodes. That means that at least one node indeed had input 0, so this is alright. But a single node having input 0 is not enough to convince anyone else, as other nodes have no way of being certain that this node is not faulty and in fact all nodes had input 1!

To resolve this, let us be a bit more lenient and make the stronger assumption that $f < n/5$. Now, a correct node seeing $n - 2f$ times "0" means that $n - 3f$ correct nodes indeed have input 0. And since each node waits for all but f messages, this means each correct node receives $n - 4f \geq f + 1$ times "0." Now we let nodes that see $f + 1$ times "0" change their "opinion" to 0, even if their input was 1.

Algorithm 9 Subroutine for trying to decide on 0, code at node $i \in [n]$. Each node i is given an input "opinion" $op_i \in \{0, 1\}$, which initially will be the input of i.

```
1: send "$op_i$" to all nodes // also to yourself
2: wait until received "$op_j$" messages from $n - f$ nodes // drop all but one
   message per node
3: **if** received $\geq n - 2f$ "0" messages **then**
4:     decide(0)
5: **else if** received $\geq n - 4f$ "0" messages **then**
6:     $op_i := 0$
7: **end if**
```

Let's summarize the properties of this basic procedure, under the assumption that $f < n/5$.

Lemma 4.1. *If a correct node decides on 0 in Algorithm 9, then all correct nodes have decided 0 or have opinion 0 at the end of the algorithm.*

Proof. The node received $n - 2f$ times "0," $n - 3f$ of which are from correct nodes. Since each node drops up to f messages, all nodes receive at least $n - 4f$ times "0" and hence execute either the IF or the ELSEIF statement. □

Lemma 4.2. *If all correct nodes have opinion 0 at the beginning of Algorithm 9, then all correct nodes decide 0.*

Proof. If all (at least $n - f$) correct nodes have opinion 0, each node receives at least $n - 2f$ times "0" and executes the IF statement. □

Lemma 4.3. *If all correct nodes have opinion 1 at the beginning of Algorithm 9 and $f < n/5$, then they all keep this opinion and none of them decide.*

Proof. If all correct nodes have opinion 1, no node will receive more than f times "0." As $n > 5f$, we have $n - 4f > f$, implying that no correct node executes the IF statement or the ELSEIF statement. □

Remarks:

- This is nice: we can achieve termination if all nodes agree on 0 without destroying an existing agreement on 1! And if some node decides, we can be sure that all nodes agree afterwards and will decide on 0 when running the subroutine again.
- Alternating with the "twin" algorithm which tries to agree on 1 makes sure that validity is satisfied. Since neither of the twins destroys an existing agreement, we have ensured agreement and validity.
- Unfortunately, it is possible that the algorithm doesn't terminate. We know from the last lecture that this cannot be avoided, since so far we haven't used randomness!

4.4 Shared Coins

We now have a way of deciding in a safe way. The issue is that we can guarantee that nodes decide only if all of them already have the same opinion. Hence, we need a mechanism to establish this common opinion if it's not initially present.

Definition 4.4 (Shared Coin)**.** *A* shared coin with defiance $\delta > 0$ *is a subroutine that generates at each node a bit such that the probability that the bit is 0 at all correct nodes is at least δ and the probability that it is 1 at all correct is at least δ. The coin is* strong*, if it is guaranteed that all nodes output the same bit. Otherwise it is* weak*.*

Remarks:

- A strong shared coin essentially generates a common "random" bit which may be influenced by the faulty nodes, but not too badly. One can interpret it as a random experiment in which with independent probability $1 - 2\delta$ an "adversary" determines the outcome, but with probability 2δ an unbiased coin is flipped to determine the result.
- If the coin is weak, the adversary can even make it so that if the first case applies, different correct nodes observe different outcomes. Of course that's less useful, but easier to achieve.

- Surprisingly, strong shared coins with constant defiance exist. Unfortunately, they all need some additional assumptions: states of and communication between correct nodes is secret (i.e., the faulty nodes may act based on the inputs and communication they have seen only), cryptography can be used to prevent messages to be understood until the sender reveals the respective key, etc.
- If one makes such additional assumptions, one must be more careful in defining the probability space over which one argues. We dodge this bullet today, by assuming that the adversary is *oblivious* to the random choices of the algorithm.
- If the adversary knows the random decisions of the correct nodes in advance, can you see how to *deterministically* prevent termination?
- You will show how to obtain a better shared coin in the exercises.

Lemma 4.5. *There is a weak shared coin with defiance 2^{-n} that requires no communication.*

Proof. All nodes flip an unbiased coin independently, i.e., output a fresh bit from their random input strings. The probability that all these bits are 0 (respectively 1) is 2^{-n}. □

Given this coin, we can solve the problem, can't we? Let's look at Algorithm 10. Once we used Algorithm 9 and its counterpart to make sure that we terminate with the right output if all inputs agreed, we simply flip the weak shared coin to obtain new opinions and repeat. Eventually, all nodes have the same opinion and will decide on the same output. Almost, except that we can get into trouble if some node already decided and then we mess up the opinions using the coin flips. We make sure that this doesn't happen by checking this with another voting step.

Lemma 4.6. *The last IF statement in the WHILE loop of Algorithm 10 does not change any opinions if all correct nodes had the same opinion. If the coin has defiance δ and $n > 5f$, all correct nodes have the same opinion after execution of the WHILE loop with probability at least δ.*

Proof. If all correct nodes agree, each of them receives at least $n-2f$ messages with the same value; hence previous agreement is not destroyed.

For the second statement of the lemma, assume for contradiction that two correct nodes with different opinions ignore the shared coin.[3] This entails that each of them received $n-3f$ messages with 0 respectively 1 from correct nodes. However,

$$2(n-3f) = n - f + (n-5f) > n - f,$$

i.e., there are not sufficiently many correct nodes to make this possible. We conclude that for at most one opinion value correct nodes may ignore the shared coin. With probability at least δ, the value of the shared coin at all correct nodes is this opinion value (if no opinion is fixed, having the same value at all nodes is good enough), yielding the second statement of the lemma. □

[3]Here we exploit that the adversary is oblivious of the nodes' randomness, and hence the outcome of the coin matches any predetermined opinion with probability δ!

Algorithm 10 Consensus algorithm using weak shared coin. Note that messages must be numbered, so receivers can figure out to which "round" a message belongs; for readability, this is omitted from the code.

```
$op_i := b_i$ // set opinion to input
while not decided do
    send "$op_i$" to all nodes
    wait until received "$op_j$" messages from $n - f$ nodes
    if received $\geq n - 2f$ "0" messages then
        decide(0)
    else if received $\geq n - 4f$ "0" messages then
        $op_i := 0$
    end if
    send "$op_i$" to all nodes
    wait until received "$op_j$" messages from $n - f$ nodes
    if received $\geq n - 2f$ "1" messages then
        decide(1)
    else if received $\geq n - 4f$ "1" messages then
        $op_i := 1$
    end if
    compute a (weak) shared coin $c_i$
    send "$op_i$" to all nodes
    wait until received "$op_j$" messages from $n - f$ nodes
    if received $< n - 2f$ "$op_i$" messages then
        $op_i := c_i$ // $c_i \in \{0, 1\}$ uniformly random
    end if
end while
send the messages for the next iteration of the loop, where the opinion
  remains fixed to the decided value (i.e., do not wait)
terminate and output decided value
```

Theorem 4.7. *Given a weak shared coin and that $f < n/5$, Algorithm 10 solves consensus. All nodes terminate in expected time $\mathcal{O}(1/\delta)$, where δ is the defiance of the shared coin.*

Proof. Let us verify validity, agreement, and termination one by one.

Validity If all correct nodes have input 0, by Lemma 4.2, in the first "round" of the first loop iteration all nodes decide on 0. If all correct nodes have input 1, by Lemma 4.3, no opinions change and no node decides in the first round. Applying Lemma 4.2, all correct nodes then decide on 1 in the second round.

Agreement By Lemma 4.1, once a correct node decides on value b, all correct nodes adopt opinion b. By Lemmas 4.3 and 4.6, no correct node will change its opinion any more after this happens. Hence, no correct node can observe more than $f < n - 2f$ messages $1 - b$ in any future "round." Thus, no correct node will decide $1 - b$.

Termination First, note that no correct node gets "stuck" waiting for $n - f$ messages of a "round" provided that all correct nodes send a message for that round. By the previous observations, if some correct node decides, all nodes will adopt that opinion and in the next iteration of the loop they will decide on the respective value by Lemma 4.2. As deciding nodes will complete the current iteration of the loop and then send the (known) messages for the next iteration, the algorithm will thus terminate provided that some node decides. By Lemma 4.6, with probability δ all nodes have the same opinion at the end of the loop, irrespectively of previous iterations. Once this happens, all nodes decide in the next loop iteration by Lemmas 4.2 and 4.3. Thus, as the probability that an infinite number of shared coins "fail" $\lim_{k\to\infty}(1-\delta)^k = 0$, the algorithm terminates with probability 1.

Finally, we bound the expected time complexity. In each iteration, every correct node constantly often sends a message to every node and waits for $n-f$ responses. Since messages are sent in parallel, it takes $\mathcal{O}(1)$ time per round and thus $\mathcal{O}(k)$ time for k rounds. The probability that "the algorithm decides" in round k (i.e., all nodes obtain the same opinion) is $\delta(1-\delta)^{k-1}$; once this happens, all nodes terminate within $\mathcal{O}(1)$ additional rounds. All of this means that the expected time until termination is bounded by

$$\sum_{k=1}^{\infty} \mathcal{O}(k)\cdot\delta(1-\delta)^{k-1} = \mathcal{O}\left(\frac{\delta}{\delta^2}\right) = \mathcal{O}\left(\frac{1}{\delta}\right). \qquad \square$$

Corollary 4.8. *Using randomness and given that $f < n/5$, consensus can be solved in asynchronous systems with Byzantine faults with probability 1.*

Proof. Plug the weak shared coin from Lemma 4.5 into Theorem 4.7. □

Remarks:

- Two things are not satisfying here. One is that using the simplistic shared coin from Lemma 4.5, the expected running time is astronomically large: for many inputs, it's $\Theta(2^n)$. The other is that we can tolerate only $f < n/5$ faults, but it might be possible to handle $f < n/3$.
- We will now see how to get down to $f < n/4$ using a new subroutine, safe broadcast. Combining the various voting ideas and a strong shared coin in a more clever way, one can indeed get down to $f < n/3$.
- There was some mild cheating: I silently omitted discussing what happens if the weak shared coin communicates and expects correct nodes to send messages. This is easy to resolve by making sure that the shared coin can somehow terminate using default messages from correct nodes (without maintaining any guarantees on the output), as it becomes irrelevant as soon as some node terminated.

4.5 Safe Broadcast

One of the issues we had in Algorithm 10 was that faulty nodes could announce different values to different nodes. Let's take this power away! This is essentially the following task, named *safe broadcast*:

- The source node s is globally known.
- s is given a message M.
- Each correct node i outputs at most one message M.
- If s is correct, all correct nodes eventually output M.
- If any correct node outputs a message M', all correct nodes eventually output M'.

Using this subroutine as a wrapper for each message sent, we can, in effect, force each faulty node to either send the same message to all nodes in a given "round," or just stay silent. Let's see how we can use this to improve Algorithm 10.

Algorithm 11 Consensus algorithm using weak shared coin and safe broadcasts.

```
1:  $op_i := b_i$ // set opinion to input
2:  while not decided do
3:    safely broadcast "$op_i$"
4:    wait until received "$op_j$" messages from $n - f$ nodes
5:    if received $\geq n - 2f$ "0" messages then
6:      decide(0)
7:    else if received $\geq n - 3f$ "0" messages then
8:      $op_i := 0$
9:    end if
10:   safely broadcast "$op_i$"
11:   wait until received "$op_j$" messages from $n - f$ nodes
12:   if received $\geq n - 2f$ "1" messages then
13:     decide(1)
14:   else if received $\geq n - 3f$ "1" messages then
15:     $op_i := 1$
16:   end if
17:   compute a (weak) shared coin $c_i$
18:   safely broadcast "$op_i$"
19:   wait until received "$op_j$" messages from $n - f$ nodes
20:   if received $< n - 2f$ "$op_i$" messages then
21:     $op_i := c_i$
22:   end if
23: end while
24: safely broadcast the messages for the next iteration of the loop, where the
    opinion remains fixed to the decided value (i.e., do not wait)
25: terminate and output decided value
```

Corollary 4.9. *Given a weak shared coin, safe broadcast, and that $f < n/4$, Algorithm 11 solves consensus. All nodes terminate in expected time $\mathcal{O}(1/\delta)$, where δ is the defiance of the shared coin.*

Proof. The reasoning is the same as before, so we need to revisit only the steps that required that $f < n/5$ in the previous proofs, as well as check the effect of the $n-3f$ thresholds that replaced $n-4f$ thresholds. This concerns Lemmas 4.1, 4.3, and Lemma 4.6.

Regarding Lemma 4.1, observe that if a node receives $n-2f$ times "0," then at most $2f$ nodes – including faulty ones! – broadcast "1." Hence, any $n-f$ received values contain $n-3f$ times "0," i.e., all correct nodes adopt opinion 0. For Lemma 4.3, we just need to note that $n-3f > f$ holds, so the faulty nodes will not be able to change anyone's opinion if all correct nodes have opinion 1. Finally, Lemma 4.6 still holds because two nodes with different opinions ignoring the shared coin would mean that there have been at least $2(n-f) = n + (n-4f) > n$ many distinct broadcast messages in the respective round. □

Remarks:

- Although the broadcast problem is very similar to consensus (the input is send to nodes by the source), there is a crucial difference: it is ok that no correct node "terminates," i.e., no correct node outputs a message. That's all it takes to make it much simpler.
- In the exercises, you will solve the safe broadcast problem.
- By adding the round number and source's identifier to the messages sent by the safe broadcast algorithm, we can tell the instances apart. Any "excess" messages for a given round number/source ID combination are simply discarded. The same holds for communication that is obviously not conform with the algorithm.
- By introducing the safe broadcast statements into the code, I also introduced a bug: the algorithm may not terminate anymore! Can you see why?
- Can you come up with a fix? This is not too difficult, but again underlines how careful one needs to when reasoning about asynchronous algorithms.

What to take Home

- Randomization can work miracles in distributed systems. Maybe even more so than in case of "classic" centralized algorithms!
- Impossibility results and lower bounds usually can and will be circumvented by changing some details – often those that we do not mind in practice or that are the smallest burdens. Only sometimes one hits hard walls, as with the $\log^*$ lower bound for coloring (but this one we don't mind in practice either).
- The running times we get here are terrible. However, strong shared coins with constant defiance and large resilience exist! This yields algorithms that can solve consensus in constant expected time, and with overwhelming probability in time $\mathcal{O}(\log n)$!
- Dealing with Byzantine faults always involves voting using thresholds requiring, e.g., $n-f$ or $n-2f$ nodes claiming a certain opinion.

- What you've seen here are some core ideas for handling Byzantine faults. There are myriads of variations and generalizations of the consensus problem or other fault-tolerance primitives. The basic ideas and the approach to reasoning about Byzantine faults you've seen today typically resurface in one way or another when considering these.

- One needs to be extremely careful in terms of how randomness and the adversary may interact. For instance, even if the adversary cannot magically predict future coin flips, node may reveal the outcomes of these coin flips prematurely due to asynchrony. This is especially important when designing systems that are supposed to be resilient against deliberate attacks.

Bibliographic Notes

The consensus algorithm presented in this lecture is a variant of Bracha's algorithm [Bra87]. However, Bracha mistakenly claimed that the algorithm tolerates $f < n/3$ faults. The issue is that in Bracha's algorithm nodes terminate only when receiving at least $2f+1$ messages supporting the decided value in a certain step, but for $n = 3f + 1$ this is exactly $n - f$; given that Byzantine nodes can always claim a different value and only $n - f$ messages are evaluated (otherwise the algorithm might deadlock), termination can never be guaranteed.

For a solution that tolerates $f < n/3$ faults using a strong shared coin, see Mostéfaoui et al. [MMR14]. A cryptographically safe, optimally resilient, constant expected time strong shared coin is given by Cachin et al. [CKS05]. Together this yields a constant expected time optimally resilient solution to consensus. Both algorithms are also very efficient in terms of messages. The caveat is that a trusted dealer is needed in a setup phase for the shared coin.

Bibliography

[Bra87] Gabriel Bracha. Asynchronous Byzantine Agreement Protocols. *Information and Computation*, 75(2):130–143, November 1987.

[CKS05] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography. *Journal of Cryptology*, 18(3):219–246, 2005.

[MMR14] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free Asynchronous Byzantine Consensus with $t < n/3$ and $\mathcal{O}(n^2)$ Messages. In *Proc. 34th Symposium on Principles of Distributed Computing (PODC)*, pages 2–9, 2014.