Massachusetts Institute of Technology — Lecture 13
6.895: Advanced Distributed Algorithms — October 22, 2002
Professor Nancy Lynch

Consensus; Generic Models for Distributed Computing

Readings:
*** Keidar, Rajsbaum. On the cost of fault-tolerant consensus when there are no faults—a tutorial.
*** Gafni, Round-by-round fault detectors: Unifying synchrony and asynchrony

We didn't get to the following; will cover them at a later date:
*** Attiya, Welch
Ladin, Liskov, Shrira, Ghemawat
*** Fekete, Gupta, Luchangco, Lynch, Shvartsman

Next time: Performing work in distributed systems (Alex Shvartsman)
Dwork, Halpern, Waarts. Performing work efficiently in the presence of faults. Sections 1, 2.1, 2.2. Browse the rest.
Chlebus, De Prisco, Shvartsman. Performing tasks... Sections 1, 2, 3.1, 4.1.
*** Georgiou, Russell, Shvartsman. The complexity of synchronous iterative do-all with crashes. Sections 1-4.
*** Malewicz, Russell, Shvartsman. Distributed cooperation in the absence of communication. Sections 1-4.
Georgiou, Shvartsman. Cooperative computing with fragmentable and mergeable groups. Sections 1-4.

1 Keidar, Rajsbaum

1.1 Overview

On the cost of fault-tolerant consensus when there are no faults—a tutorial.
Overviews many results about time bounds on consensus, and presents some new results and new proofs of old results.
I'll focus on Section 4.

Their motivation starts with partially synchronous models (models with timing assumptions, which may hold from some point on), and asynchronous models with unreliable failure detectors like diamond-S.
But the actual technical results they get are for the round-based synchronous model. They then use those to get results about the other models as corollaries.

So, I'll focus on the results about the synchronous model.
Process stopping failures.
Assume a worst-case bound of t on the number of failures, but they also consider the number f that occur in particular executions.

The results I will describe deal with uniform consensus:

1. A 2-round lower bound for the time for all nonfaulty processes to decide, in executions with no failures.
   This matches the bound for Mostefaoui, Raynal's algorithm, when executed in the synchronous model (implement your own reliable failure detector by observing when messages don't arrive).
2. An $f+2$-round lower bound for the time for every nonfaulty process to decide, in executions with at most f failures.
   Charron-Bost and Schiper have an earlier version of this, slightly less general (assumes $f \geq 1$).

3. Unsolvable in the "mobile failures" model, which simply allows messages from at most one process to be lost at each round. (Can change from round to round.).
   Re-proves a result previously proved by Santoro, Widmayer and re-proved earlier by Moses, Rajsbaum.

Compare: Dolev, Reischuk, Strong 90 algorithm: min(f+2,t+1) rounds (?).
That is, if $f \leq t-2$ then it decides within $f+2$ rounds (else within the max, $t+1$).

KR emphasize their technique. It's pretty simple, like that of Aguilera, Toueg's. But it doesn't use bivalence. In fact, they claim that a bivalency argument can't be used to get the $f+2$-round lower bound. Not exactly clear how one would state/prove that precisely.

1.2 Layering, similarity, and connectivity

Everything in this section applies to general algorithms in the round-based synchronous model—not just consensus.

I don't see a careful definition of "layering" here. I'll try to fill in.
Restrict attention to executions in which at most one process fails for the first time at each round.
Moreover, when it fails, it fails to send to some "prefix" of the set of processes, $[k] = p_1, p_2, \ldots, p_k$ for some k.

An execution is determined by an assignment I of initial states to all the processes plus a sequence E of "environment actions" of the form (i,k), where p_i is a process that fails to send to all of the processes in $[k]$.
If $k=0$, then p_i sends to everyone. In this case, no failure occurs at this step.

It's clear how such an initialization and sequence of actions entirely determines an execution of the standard process-stopping-failure model.
The process designated for each round indicates a new failure, except when $k=0$.
The convention is that a failed process doesn't send any messages at later rounds either.

However, they "overload" their notation a bit by using it also for the "mobile failure" model.
Here, the convention is different: process failures are for only one round.
By this convention, their notation determines an execution for the mobile failures model.

Let's restrict attention to the stopping failure model, to avoid having to go back and forth thinking about the two different cases.

Then we can check that the results apply in the mobile model.

Define an environment sequence E to be *well-formed* if it includes at most t process failures and the same process fails at most once in the sequence.

I don't understand exactly what a "layer" is. I think their ideas are better understood in terms of executions and extensions of those executions, as usual.

Recast the L^k definitions slightly:
Let $\mathcal{E}$ be a set of well-formed finite environment sequences of the same length.
Then $L^k(\mathcal{E})$ is the set of well-formed environment sequences that are k-step extensions of sequences in $\mathcal{E}$.

Define a *run* to be a pair (I, E): initialization and well-formed environment sequence.

Similarity: Two runs (I, E) and (I', E') of the same length are *similar* provided that there are two processes, p_i and p_j, such that:
(a) p_i and p_j are both non-failed in both of the executions generated by these runs, and
(b) These two executions look identical to all processes except p_j.
Thus, if two runs are similar, at least one nonfaulty process p_i can't distinguish them.

Q: Why should we care if p_j is faulty or not?
Maybe we could just say that these runs look the same to all but one process, and some process is non-failed in both executions. ???

A set Φ of runs is said to be *connected* if every pair is in the reflexive transitive closure of the similarity relation.

Lemma 4.1: The set of "initial" runs, in which the envt-seqs have length 0, is connected.
Proof: Change the initial state of one process at a time.

1.3 The 2-round lower bound for failure-free executions

Now assume that the algorithm is solving uniform consensus.

Coloring: For any run $\rho = (I, E)$, we define $val(\rho)$ to be the unique decision value that results in the extension of ρ in which no failures occur.

Key Lemma, 4.2: Let Φ be a connected set of runs. Suppose ρ and ρ' are two runs in the set such that $val(\rho) \neq val(\rho')$.
(Also assume there are "enough" failures left to use.)
Then there is a run $\sigma \in \Phi$ such that, in the failure-free extension of σ, some non-failed process has not decided within one additional round.

This immediately implies the first main theorem: the 2-round lower bound for uniform consensus when there are no failures.
Uses the fact, in Lemma 4.1, that the set of initial states is connected.

Proof of Lemma 4.2: WLOG, we can assume that ρ and ρ' are actually similar, not just connected (by the usual argument—the two assumed runs are connected by some chain, so we have a difference at some point in the chain).
Then we know that ρ and ρ' look identical to everyone except for a particular p_j. And there are enough failures left over for us to play around with.
Choose p_m to be some other process that is nonfaulty in both runs—in fact, the one with the largest index that isn't faulty in both runs.
(We seem to be using the fact here that the same processes are faulty in both. That seems to follow...)

Assume for the sake of contradiction that everyone decides within one round in the failure-free extensions of ρ and ρ'.
Assume wlog that $val(\rho) = 1$; thus, all processes in the failure-free extension of ρ decide 1, within one more step.
And $val(\rho') = 0$.

Consider the execution constructed by extending ρ with $(j, m-1)$, that is, j (who hasn't failed yet) now fails and misses sending to everyone up to and not including m.
To process m, this looks the same as the failure-free extension of ρ, so m must decide 1 after one round.
Then everyone else must also decide 1 in *any extension* of this extended run, though because they know this is not a failure-free extension (they can see missing messages), they are allowed to take more rounds to do it.

Likewise, extend ρ' with $(j, m-1)$, conclude that m must decide 0 after one round, so everyone else decides 0 in any extension of that extended run.

But, among the extensions, there are two—one extending $\rho \cdot (j, m-1)$ and one extending $\rho' \cdot (j, m-1)$— in which m fails immediately after the decision round and no one else ever fails.
No remaining nonfaulty process can tell the difference between these two runs.
But they are forced to decide differently in these runs.
Contradiction.

1.4 The $f+2$-round lower bound for f-failure executions

The previous result is like the special-case result used in the Aguilera-Toueg proof. This paper also has an analog of the earlier-rounds argument (bivalence preserved for one round when a failure occurs) of AT.
Stated in terms of preserving similarity.
This result doesn't say anything about consensus.

Lemma 4.4: Let Φ be a connected set of states. (Assume there are enough failures left to use.) Then $L(\Phi)$ (the set of one-round extensions of Φ) is also connected.

Proof: This isn't hard—add lots of extra intermediate "links" in the chain, removing one message at a time. LTTR.

Let Φ_0 be the set of initializations for the processes.

Lemma 4.5, part 1:
$L^k(\Phi_0)$ (the set of k-round runs) is connected for all k, $0 \leq k \leq t$.

This follows because Φ_0 is connected and the previous lemma says this is preserved (since we have "enough" failures to play with).

Now consider uniform consensus again.
Lemma 4.5, part 2:
For every $k \geq 0$, there exist ρ, ρ' in $L^k(\Phi_0)$ such that $val(\rho) \neq val(\rho')$.

Just take non-failing runs that start with all 0 or all 1.

Finally, we can prove the $f+2$-round lower bound on uniform consensus with f actual failures (out of a possible t):

Theorem 4.9: ...
Proof: Fix f, $0 \leq f \leq t-2$.
Use Lemma 4.5 to obtain runs ρ, ρ' in $L^f(\Phi_0)$ such that $val(\rho) \neq val(\rho')$.
Also by Lemma 4.5, $L^f(\Phi_0)$ is connected.
Then we use Lemma 4.2 (noting that there are "enough" failures left), and conclude that there is a run $\sigma \in L^f(\Phi_0)$ such that in the failure-free extension of σ, some non-failed process hasn't decided within one additional round.
That is, it takes at least $f+2$ rounds to decide.

1.5 Mobile failures

They use the same results to show impossibility of ever deciding in the (uniform or nonuniform—the difference isn't meaningful here) consensus problem, in the mobile failures model.

This uses Lemma 4.5 again, but with no upper bound t—so after any finite number of rounds, the set of runs is connected.
Since each such set has runs with distinct vals, a simple argument shows there's another run with no decision.
Use Konig's lemma to show that there's an infinite branch in which some nonfaulty process doesn't decide.

1.6 Discussion

Results have topological flavor.

The definitions of similarity, layers, etc., might be polished a bit.

Q: What's the minimum time for consensus to terminate after stabilization time, in synchronous models where:
(a) For any amount of time processes may exhibit any number of omission failures, and may stop and restart.

(b) After stabilization time, no more processes stop or restart, and no messages are lost.
Since the failure model is worse-behaved than the one considered by Keidar and Rajsbaum, it might be that a larger bound applies.

2 Gafni

Round-by-Round Fault Detectors: Unifying Synchrony and Asynchrony
PODC 98

Gafni sets up a simple, general, round-based modeling framework that can be specialized to describe variants of many of the models used elsewhere, including the usual synchronous models, the usual asynchronous models, models with failure detectors; message-passing and shared-memory models; models with different kinds of failures.

Casting the different models as special cases of a common general model allows him to compare different models, e.g., describe transformations from one to another.
This allows him to carry over impossibility results from one model to another, e.g., to show that an impossibility result for a consensus problem in an asynchronous model implies a number-of-rounds lower bound in a synchronous model.

2.1 Introduction

Assume a fixed set of processes $S = \{1, \ldots, n\}$.
The general framework is a round-based model, with rounds numbered $1, \ldots$.
In each round r, every process i broadcasts a single message to all processes, including itself.
Then it receives messages from some subset $S(i,r)$ of the processes (possibly not including itself), and gets an indication that another subset $D(i,r)$ may be faulty.
He assumes that:

1. $S(i,r) \cup D(i,r) = S$, the set of all processes, that is, at each round, each process either hears from each other process or has some explicit indication that the other process might be faulty.
2. $D(i,r) \neq S$, that is, a process can't hear that everyone has failed.

He doesn't rule out $S(i,r) \cap D(i,r)$ being nonempty.
He does allow $i \in D(i,r)$.
Given these two sets, process i makes a state transition and goes on to the next round.

This modeling framework gets specialized to a particular "computation model" by the use of a particular "predicate" P defining the allowable sets of functions $D(i,r)$.
It gets further specialized to a particular "system" by fixing the processes' sets of states, initial states, and transition relation.
The same system admits different inputs, represented by different initial process states.
Once all this is fixed, the system can execute. The execution can be nondeterministic, based on which allowable $D(*,*)$ sets the "adversary" contributes.

Define a task T to be a set of (input-vector, output-vector pairs).

E.g., the uniform consensus task is the set of pairs (V, W) where $W(i) = W(j)$ for all j, and $W(i) \in \{V(j) | j \in S\}$.

A particular system "solves" a task T provided that, if the processes in the system are started with inputs represented by V, any outputs they produce are portions of a vector W such that $(V, W) \in T$.
That expresses safety.
Moreover, he says that "eventually, after enough rounds, processes commit to outputs".

But this isn't completely clear. It seems that we would want to say that "nonfaulty" processes eventually decide.
But he hasn't actually modeled process failures! ???

I'll read on, see later if this omission causes confusion...

2.2 Examples of RRFD models

He shows how to view many previously-defined computational models within his framework, by particular "predicates" that capture appropriate restrictions on the D functions.

In each case, he makes some argument that his representation is faithful to the intention of the original model.
These arguments say that the two models "implement" each other: for any particular system A in one, there is a system B expressed in the other, such that B "simulates" A. Warning: He doesn't define what "simulates" means. Surely that notion would have to say something about process failures, which aren't modeled here...

Anyway, read on...

Example 1: Round-based synchronous systems in which processes broadcast just one message at each round, and in which at most f processes fail by send-omission.

Express this by the following restrictions on the D functions:

1. $i \notin D(i, r)$, that is, no one is ever told that he might be faulty.
2. There is a single set D with $|D| \leq f$, such that every $D(i, r) \subseteq D$.

This is essentially the usual formulation, so no special "implementation" argument has to be made.

Example 2: Round-based synchronous systems...in which at most f processes fail by stopping.

Express by using the restrictions of Example 1 plus one other restriction:

1. $D(i, r) \subseteq D(j, r')$ for all i, j, r, r' with $r < r'$.

Again, this is essentially the usual model.

Example 3: Asynchronous system with at most f stopping failures.

Express simply by:

1. For every i and r, $|D(i,r)| \leq f$.

That is, independently at each round, and for each recipient, at most f nodes are designated as possibly faulty.

He makes an informal argument that this representation implements the usual asynchronous model, and vice versa.

(a) Asynch implements the RRFD version:
Assume A is an algorithm in this RRDF model; construct B in the asynch model:
Use asynchronous rounds.
In each round r, each process i of B waits to hear from all but f processes. (It can do that in the asynch model because it eventually hears from all but f.)
Then process i of B simulates the transition of i in A by letting $D(i,r)$ be the set of processes that i doesn't wait for at round r.

It should be reasonably clear that B "simulates" A pretty directly, though this isn't formally defined (stating it carefully would require modeling the actual failure events).

This simulation corresponds to a very common strategy in constructing asynchronous algorithms: Asynchronous rounds, wait for all but f.
Ignore any of the remaining messages if they arrive "too late", when the recipient has moved on to the next round.

(b) The new RRFD model implements the usual asynch model:
Now assume A is an algorithm in the standard asynch model; construct B in the RRFD version:
Now the problem is that the RRFD model seems to give the adversary more options than the asynch model does. In the RRFD model, different collections of f processes might appear to have failed to different other processes, and at different rounds. In the asynchronous model, the adversary is more restricted: there are only a particular set of f processes that ever fail. The implementation result has to show that the apparent freedom of this model isn't really any greater than what's allowed in the standard asynch model.

Gafni shows this by letting the processes of B turn missed messages into late arrivals.
The sketch is extremely rough; I'll try to fill in.
Assume A uses bcast messages, with eventual delivery (no ordering assumptions).
And assume that every process in A is always enabled to bcast (perhaps dummy messages).

Each process i of B starts by sending, at round 1, a bcast message that it is enabled to send in the initial state of i in A.
In the RRFD model, it receives $n - f$ messages; it then does local state transitions corresponding to the $n - f$ receive events in the asynch model, in any order. (The order doesn't matter—the asynch model allows any order.)
Then it sends its second round message—a bcast message that it is enabled to send in its simulated

A-state.

Well, not quite.
If we kept doing rounds like this, we might not be correctly simulating a fair execution of the asynch algorithm, because we're not capturing the liveness assumption that the channels eventually deliver all messages from the real nonfaulty processes. We are throwing out some messages.

To get around this problem, we use a trick, by running the processes in what he calls "full information mode". This means that each process keeps track of all messages it has ever received from everyone (disambiguated somehow), and sends this history on every message. A little more precisely, what it sends in full info mode is a "super-message" containing its actual message plus a history of all the actual messages it has ever received from anyone in the past.
When a process receives a super-message, it unpacks all the messages contained within, figures out which ones it hasn't previously seen, and simulates their receipt.
Thus, in one round, the process in B simulates the receipt of, not just the new messages sent by everyone, but also any old messages that those processes knew about.
This is simulating delayed arrival of those old messages, but this is fine because the asynch model allows those messages to arrive at any time!
(I'm assuming that the channels between processes in the asynch model don't need to be FIFO; if they do, we can further constrain the order in which we simulate delivery using sequence numbers. LTTR.)

Example 4: Asynchronous shared memory model, at most f stopping failures, reliable memory. The shared objects are single-writer multi-reader r/w registers.

He models this in his framework by using the restriction of the asynch network model, given above.
This isn't enough: the asynch shared memory model is stronger than the asynch network model in that the objects don't fail.
So he has to assume something is reliable. What he assumes (not self-evident) is the extra condition:

1. In each round r, there is some process j such that $j \notin D(i,r)$ for all i.
   That is, at each round separately, some process is not suspected by anyone.

He gives some very sketchy arguments—too sketchy for me to work out quickly—for why this is actually equivalent to the standard shared-memory model.

Example 5: Atomic snapshot shared memory (where a process can read everyone's local variable all at once, atomically), with at most f process stopping failures.

He equates this with the RRFD that uses the restrictions of the asynch network model, plus:

1. $i \notin D(i,r)$
2. For each round r and each pair of processes i and j, either $D(i,r) \subseteq D(j,r)$ or vice versa.

No equivalence argument is given.

Example 6: Asynchronous message-passing system with failure detector S (some nonfaulty process is never suspected)

He views this as the RRFD satisfying (the conditions of the asynch network model) plus:

1. There is some process j that does not appear in any $D(i,r)$ set.

2.3 RRFDs and k-agreement

What can we do with these models? And what does this all have to do with consensus, which is today's main topic?

Viewing all these models in a uniform framework makes it somewhat easier to relate different models. Gafni relates synchronous and asynchronous models in this way (next section), and uses this to get corollaries about impossibility and lower bounds for consensus.
In this section, he characterizes solvability of k-consensus in terms of RRFDs.

Basically, he defines a particular RRFD and claims that it captures the ability of a model to solve the k-consensus problem. The one he uses is:

1. For each round r, there are at most $k-1$ processes that appear in some $D(i,r)$ but not in all $D(i,r)$.

That is, $k-1$ bounds the "uncertainty" of the fault detector—the amount of disagreement in each round.

Theorem 3.1: With this fault detector, can solve k-consensus.
Proof: Trivial. Use one round. Everyone sends around their initial values, choose the value from the process with the smallest id.

Then he claims that this yields Chaudhuri's result asserting solvability of k-consensus in asynch shared memory systems with at most $k-1$ stopping failures.
But it takes a bit of work: It first requires showing that shared memory with $k-1$ failures can implement snapshot memory with $k-1$ failures. That's a known result, from the Borowsky paper. The proof could presumably be redone in the setting of this paper. But this isn't exactly self-contained.
Once we have the snapshot memory, it's easier because the predicate defining the snapshot memory immediately implies the predicate given here for k-consensus.

Theorem 3.3: They also give a result—the statement is a little loose—that says that this predicate isn't too strong to model k-consensus.
In particular, having a solution to the standard k-consensus problem in some form and having SMRW asynch shared memory, it's possible to compute $D(i,r)$ sets satisfying the condition above. The precise statement and proof aren't so clear to me, esp. when he's talking about the standard models and when about the RRFD framework.

2.4 Relating synchronous and asynchronous systems

He uses his framework to show that that the asynchronous shared memory model with 1 stopping failure implements a model in which the first f rounds satisfy the restrictions of a synchronous system with f omission or stopping failure.

This implies that a consensus algorithm that works in the synchronous model within f rounds also runs in the asynchronous model with 1 stopping failure. But that's impossible, by FLP.

The result is given in a more general form: for k stopping failures in the asynch model, and $\lfloor f/k \rfloor$ rounds of the synchronous system.
That means that the unsolvability result for k-consensus in the asynch model with k failures (proved by three different groups of researchers in 1993) implies the lower bound result of Chaudhuri, Herlihy, Lynch, Tuttle, also 1993, for k-consensus.
(Both of these proofs are pretty hard, and use similar topological techniques.)

I'll leave the stopping failure results to you. Let's just look at the key theorem for send-omission:

Theorem 4.1: The asynchronous atomic-snapshot shared-memory model with k stopping failures (example 5) implements the first $\lfloor f/k \rfloor$ rounds of the synchronous message-passing model with f send-omission failures.

(He seems to want a special notion of "implements the first r rounds of". Remains to be cleared up. Maybe we can talk alternatively about a model that satisfies the needed predicate for r rounds but not necessarily afterwards. ???)

He uses snapshot, but that's not supposed to be significantly different from using ordinary shared-memory, because of snapshot algorithm results.

Proof: The snapshot memory's predicates imply that, at each round, the total size of its D sets is at most k.
(Follows from the predicates in example 5).
Over f/k rounds, that gives a total of at most f failures.
That yields the predicate for the RRFD for a synchronous system with at most f omission failures.
(Well, just for the first f/k rounds.)

This is very rough. But the idea seems right, and interesting.
The cited paper by Herlihy, Rajsbaum, and Tuttle, called "Unifying synchronous and asynchronous message-passing models", sounds related and should be looked at.