Distributed Algorithms
Lecture 8

Lecture by Dr. Gadi Taubenfeld
Typeset by Steven Karas

2017-05-17
Last edited 18:13:31 2017-05-24

Disclaimer These lecture notes are based on the lecture for the course Distributed Algorithms, taught by Dr. Gadi Taubenfeld at IDC Herzliyah in the spring semester of 2017. Sections may be based on the lecture slides and accompanying book written by Dr. Gadi Taubenfeld.

Agenda

- Impossibility of Consensus - full proof
- Universality of Consensus

This lecture is based on chapter 9 in the book.

1 Impossibility of Consensus with One Faulty Process

Proven in 1985 by Fischer, Lynch, and Patterson.

Theorem: There is no (asynchronous) algorithm that solves the consensus problem using atomic read/write registers in the presence of a single faulty process.

1.1 Equivalence of Atomic Registers and Message Passing

This also holds for message passing systems, because there is a reduction from message passing to atomic registers.

Proof Assume that we do have an algorithm that solves consensus for message passing in the face of a single fault. For every pair of processes i, j, define a list $m_{i,j}$ of the messages between i and j. When i sends a message to j, it writes the message to $m_{i,j}$, and increments its pointer into the list. When j reads a message from i, it reads the message pointed to by its pointer and increments it.

As such, we can simulate consensus in the model of atomic variables such that we contradict the above theorem.

1.2 Notations

A run is an alternating sequence of states and events. An empty run is a run consisting of only an initial state.

We can build a tree of decisions where each state has n children states caused by each process succeeding in executing an instruction. This tree represents the full space of possible future runs of the algorithm.

Univalent states are states such that all its terminal descendants decide on a single value.

Bivalent states are states such that there are terminal descendants which decide different values.

1.3 Lemma 1

Any consensus algorithm that can tolerate one crash failure has a bivalent empty run (a bivalent initial state).

Let us examine four processes deciding on the values 0, 1. Consider five combinations of inputs: 0000, 0001, 0011, 0111, 1111.

0000: Trivially univalent to 0.

0001: Assume it is univalent to 0. Otherwise we've already finished. This implies that if the failure is a "0" process, then a fault-free run of 001 is univalent to 0.

1111: Trivially univalent to 1.

0111: As in 0001, this is univalent to 1. Similarly, it implies a fault-free run of 011 is univalent to 1.

0011: There must be a fault-free run of this state to 0 if one of the "1" processes fails at the start. Similarly, there must be a fault-free run of this state to 1 if one of the "0" processes fails. Therefore, 0011 is bivalent.

1.4 Proof

Assume the contrary. Let CONS be a consensus algorithm that can tolerate one crash failure.

Define an ambivalent step as some partial run of CONS such that both the starting state is bivalent, the ending state is bivalent, and process i has performed at least 1 step.

If we repeat ambivalent steps in a round robin fashion for each process, we must eventually reach a state x where for some process p, for every possible run that includes a step by p, results in a univalent state.

Consider the shortest extension of x such that it decides a value different from an immediate step by p. Either a step by p results in the the state becoming univalent, or a step by another process results in such a state, at which point, p must take another step eventually. Mark the bivalent state immediately prior to this as y. At some point between x and y where the next step by p is univalent for different values. Mark the process that takes this next step as q.

There are 4 cases from this point w.r.t the actions of p and q at this point:

case	p	q
1	read	read
2	write	read
3	read	write
4	write	write

If q only reads something, then if it were to fail, the information it had gathered from reading would be lost. As such, the run would be equivalent to that of the other step of p, and therefore it cannot be that q reads. Therefore, case 1 and 2 cannot happen.

In case 3, the two orderings are equivalent, as p does not have sufficient time to propagate such information to the other processes. As such, if we consider the case where p crashes after its read, it follows that in both orderings are equivalent, and therefore it cannot be that p reads.

In case 4, there are two sub-cases: writing to different registers, and writing to the same register. If they write to the same register, then p would overwrite what q wrote, and the runs are equivalent. If they write to different registers, then the information that p adds is insufficient to decide on any single value, and therefore case 4 cannot happen.

1.5 Notes

If we allow infinite runs, then we call this model Eventual Consistency, and the result is not impossible.

Also, if we consider a synchronous network, the result is also not impossible.

2 Relative power of Synchronization Primitives

Consensus Number The consensus number of an object of type o denoted $CN(o)$ is the largest n for which it is possible to solve consensus for n processes using any number of objects of type o and any number of atomic registers. If no largest n exists, the consensus number of o is infinite.

- $CN(\text{Atomic Registers}) = CN(\text{safe}^m) = CN(\text{regular}^m) = 1$
- $CN(\text{RMW bits}) = CN(\text{Queue}) = CN(\text{Stack}) = CN(\text{Test and Set}) = 2$
- $CN(\text{3-valued RMW bits}) = CN(\text{CAS}) = \infty$

Where o^m is an object in which a process is allowed to atomically access m objects of type o.

The equivalence between the various classes of consensus strength is a mostly open question, in particular w.r.t. composing several together.

3 Universality

We discussed this very briefly, but did not go into details, and Gadi chose to continue next week with this.

4 Next week

- More detail on universality