Distributed Algorithms
Lecture 7

Lecture by Dr. Gadi Taubenfeld
Typeset by Steven Karas

2017-05-17
Last edited 18:11:52 2017-05-17

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

- Consensus

1 Consensus

Our model is:

Given a set of processes, where each process p_i has an input value in_i. Processes may crash (fail-stop).

1.1 What is consensus?

The processes reach consensus when all non-faulty processes decide on the same value v.

A valid consensus is one in which $v \in \{in_1, \dots, in_n\}$.

There are variants for binary and multivalued consensus.

1.2 Impossibility of Consensus

There is no asynchronous algorithm that solves consensus using atomic read/write registers in the presence of a single faulty process.[1]

We can work around this with stronger primitives[2], timing assumptions, or randomization.

1.3 3-state RMW register

Given a 3-state RMW register z with an initial value of $\perp$. The first process p_i to access z sets it to in_i. All other processes adopt this value as the decision value.

This algorithm is wait-free, which means it can tolerate the failure of any number of processes.

1.4 2 processes, 3 bits

Given 2 atomic RW registers r_i and a single RMW bit z. Each process sets $r_i := in_i$. Each process tries to set z from 0 to 1. If the process p_j succeeds, the decision value is set to in_j.

1.5 2 processes, 2 RMW bits

Given 2 RMW bits x and y with an initial value of 0. If $x = 1$, then p_i decides 1 and halts. Otherwise, p_i sets x to in_i and continues. If $y = 0$, then p_i sets y to 1, decides in_i, and halts. Otherwise, p_i continues. if $x = 0$, then p_i decides 0 and halts. Otherwise, p_i decides $1 - in_i$.

This works because it uses x to communicate the value of in_i that should be decided, and y is used to decide who decides[3].

[1] A formal proof may be found in Chapter 9.1 of the book, and next week

[2] We will use the shared memory model for the next few lectures until we discuss message passing

[3] A full proof is just case analysis

1.6 Impossible Consensus

3 processes Impossible for any number of RMW and atomic registers.

There is no consensus algorithm for n processes that can tolerate 2 faults for any n.

2 processes, any atomic registers Impossible to solve.

There is no consensus algorithm for n processes that can tolerate 1 fault for any n.

n processes, 4 RMW bits, at most 1 fault Non-wait free. Two processes run the above algorithm and the decider writes the result to a

Crash-safe semaphores We cannot implement a semaphore S for three or more processes using RMW bits and atomic. Proof by contradiction: given such a semaphore, we can implement a consensus algorithm that is fault tolerant.

1.7 Consensus without memory initialization

Assume the following model:

- A single RMW register
- the initial value of the register is not known
- Processes are asynchronous
- stop-failure

A solution for a fault-free model Increment the register n times, and wait until the register reaches n more than the initial value. Then set the register to $in_i + n + start + 1$. If the other processes see the register has grown by more, then they extract in_i from the register and decide on it.

Bounded register size At least $2n + \max(in_i) + 1$.

Impossibility for $n/2$ failures If half the processes fail, then it is impossible to solve this problem. Proof by contradiction: assume that there is such an algorithm. Run this algorithm on half the processes with the input 0. Run the algorithm again on the other half with their input set to 1. Both groups have decided on different values, and therefore there cannot be such an algorithm.

An algorithm for $n/2 - 1$ failures Using a single RMW register with $3n$ values.

Use the MSB of the register as a separate bit, such that we can construct two rings with $1.5n$ values each.

Each process remembers its location on the current ring and increments the register on the ring. It then waits for the location on the ring to be at least $n/2$ from its location. This process becomes the master. The master flips the MSB (switching the active ring), and sets the value to 0 if the consensus value is 0, and $n/2$ if it's 1. If another process switched rings, then we decide on 0 or 1 based on the location on the ring. If the location on the current ring has moved by more than n, then we decide based on the location on the ring. The master keeps setting the location back to 0 or $n/2$.

Fault-free solution using 5 values Based on a See-saw barrier. Details on page 215 of the book.

Lower bound on number of values When there are faults, the lower bound on the number of values is $\Omega(n^{0.63})$

1.8 Consensus using a shared queue

Solution must be wait-free. Can use many queues and atomic registers.

Observations

- With a peek operation: trivial for many operations
- Three processes - Impossible without peek
- 2 processes with an initialized queue - trivial without peek
- 2 processes with an empty queue - nontrivial without peek

2 processes with an initialized queue Start with 2 values in the queue: win and lose. Each process dequeues a value. The winner sets the deciding value.

2 processes with an empty queue Algorithm designed by Roi Ramon in approx 2007.

Given a queue Q, initially empty, and an atomic bit R with an initial value of 0.

For processes $i = 1 \ldots n$

```
if $in_i = 0$:
  enqueue($Q$, 0)
  if $R = 0$:
    decide(0)
  else:
    if dequeue($Q$) = empty:
      decide(0)
    else:
      decide(1)
else:
  $R = 1$
  if dequeue($Q$) = empty:
    decide(1)
  else:
    decide(0)
```

2 Next week

- Impossibility of Consensus - full proof
- Universality of Consensus