Distributed Algorithms
Lecture 13

Lecture by Dr. Gadi Taubenfeld
Typeset by Steven Karas

2017-07-05
Last edited 17:22:58 2017-07-05

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

- Timing Based Algorithms
- Known Delays
- Unknown Delays

HW2 will be graded by tomorrow evening, and HW3 by next week.

1 Exam

The exam will be similar to homework questions. 4 questions of equal weight (25%). At least 1 question on mutual exclusion. At least 1 question on leader election. At least 1 question on consensus. There will not be a question on snapshot algorithm. The final question may be on any of the other material.

Gadi's recommendation is to go over all the homework and practice all the questions.

The exam will be with open material, but laptops are not allowed due to a general rule of IDC.

Gadi will not be at the exam on the 30th, but Dr. Tami Tamir will be present to answer questions.

2 Timing Based Algorithms

Formal model

- Asynchronous timing: no bound on how much time it takes to access a shared register.
- Known delays: such a bound exists and it is known.
- Unknown delays: such a bound exists but it is not known.

Δ is the time it takes the slowest process to complete a single access to a shared register. $delay(d)$ is a function for waiting d time units.

Note that delays are delays between operations, and operations are instantaneous.

3 Known delays

3.1 Fischer's Algorithm

Invented by Mark Fischer, but only presented by Nancy Lynch and Nir Shavit in 1992.

Mutual exclusion for known delays.

For process i in 1...n

```
while x != i:
  await(x = 0)
  x = i
  delay($\Delta$)
CRITICAL SECTION
x = 0
```

3.1.1 A faster variant

A fast variant constructed by combining Lamport's fast bakery with Fischer's framework:

For process i in 1...n

```
start: x = 1;
       await(y = 0)
       y = i
       if x != i:
         delay(2$\Delta$)
         goto start if y != i
         await(z=0)
       else:
         z = 1
       CRITICAL SECTION
       z = 0
       y = 0 if y = i
```

3.2 Consensus with 1 atomic register

Let x be a shared atomic register with a starting value of $\perp$.

For process i in 1...n

```
x = in.i if x = $\perp$
delay($\Delta$)
decide(x)
```

3.2.1 Fast variant under noncontention

Let $x[0]$, $x[1]$, and y be shared atomic registers.

The general approach here is to flag that 0 or 1 are potential values by writing them to the x registers, and skipping the delay if there is no contention, since it means they must observe y with the consensus value.

For process i in 1...n

```
x[in.i] = 1
y = in.i if y = $\perp$
delay($\Delta$) if x[1 - in.i] = 1
decide(y)
```

4 Unknown Delays

4.1 Fast consensus

Let x[round][0,1], y[round], and out be a shared atomic registers. Let v, round be local registers.

For process i in 1...n

```
v = in.i
while out = ⊥:
  x[round,v] = 1
  if y[round] = ⊥:
    y[round] = v
  if x[round, 1−v] = 0:
    out = v
  else:
    delay(round)
    v = y[round]
    round += 1

decide(out)
# delay(r) ~ r.times { skip }
```

If we grow the round too fast, then we may end up passing the actual value of Δ. If we grow it too slowly, then we may waste time before we pass it (and our delays become functional).

Note that we don't actually grow the round, we just grow the delay value.

delay()	rounds	time
round	Δ	$O(\Delta^2)$
round!	$O\left(\frac{\log\Delta}{\log\log\Delta}\right)$	$O\left(\frac{\Delta\log\Delta}{\log\log\Delta}\right)$

This also happens to be a lower bound for the problem in general.