CSC2221: Introduction to Distributed Computing **Fall 2017**

Lecture 10: Miscellaneous

Lecturer: Faith Ellen *Scribe: Lily Li*

10.1 Randomized Algorithms

Definition 10.1 *An algorithm for randomized wait-free consensus must satisfy agreement and validity and before, but also randomized wait-free termination: for a non-faulty process, the expected number of steps is finite (regardless of the behavior of other processes).*

Where the expectation is over the sequence of random coin tosses. Let R be a sufficiently long sequences of $\{0,1\}^*$ where the probability of 0 is $\frac{1}{2}$ and R_i is the i-th toss.

Let A be an algorithm, S a scheduler, and R a random sequence R. Fixing these three values yields and execution $\alpha_A(S,R)$. Then

$$\textit{expected step complexity of } A = \max_S \max_{p\in\{0,\ldots,n\}} E_R[\textit{number of steps by } p \textit{ in } \alpha_A(S,R)]$$

$$\textit{expected work of } A = \max_S E_R[|\alpha_A(S,R)|]$$

***Adaptive/strong adversary**: can see everything that has happened so far, including coin flips, but cannot predict future coin flips (coin flips are separate steps). It was shown that we have tight bounds: $\Theta(n^2)$ expected work complexity and $\Theta(n)$ expected step complexity for randomized consensus.*

***Oblivious adversary**: fixes a sequence of process ids in advance and at each step, the next process in the sequence takes a step.*

*In between these two extremes you have many different types of **weak adversaries**.*

10.1.1 End-Zone Algorithm

This algorithm achieves $O(n^4)$ expected work against adaptive adversaries using $n+2$ registers. But it more natural to do expected $O(n^2)$ work against an adaptive adversary using 2 registers and a counter with increment, decrement, and read.

Just do binary consensus for the time being. Let R_0 and R_1 be flags indicating the existence of the input value. Both flags are initialized to 0. The counter C is also initialized to 0. See pseudo-code 1.

Observe that validity is quite easy to show. Suppose all the inputs are 0. Then R_1 will always be 0. Thus the third else-if statement will always be true and all the processes will push the counter past $-2n$. This execution has length $O(n)$.

Lemma 10.2 *If any process reads $\geq 2n$ from C, then all subsequent reads will never read $\leq n$.*

Proof: By contradiction. Consider an execution α that contains a $read(C)$ which returns a value $\geq 2n$. Let q be the process which took this step. Let α_j be the first $read(C)$ which reads $\leq n$ following α_j. Since the value of counter C went from $\geq 2n$ down to $\leq n$, there must have been $> n$ decrement operations. This means that there exists a process which decrements twice. Take a look at the code. Observe that between any two decrements by the same process, there must be a $read(C)$. Such a $read(C)$ must return a value $\geq n+1$ so the process should have incremented. ■

The lemma holds if you swap $\geq 2n$ for $\leq -2n$ and $\leq n$ for $\geq n$.

Theorem 10.3 *The expected work is $O(n^2)$.*

Proof: In order to make this proof work, we need to define the following value:

$$U = C + (\#\text{ processes poised to perform } inc(C)) - (\#\text{ processes poised to perform } dec(C)).$$

Consider the steps in the algorithm that affects U: (1) if $read(C)$ return a value between $[n, 2n)$ and $(-2n, n]$ then we will get an $inc(C)$ and $dec(C)$ respectively. (2)$read(R_0)$ in line 8 and $read(R_1)$ in line 10 causes an $inc(C)$ and a $dec(C)$ respectively. (3) with probability $\frac{1}{2}$ the coin flip on line 13 performs $inc(C)$ or $dec(C)$.

Also observe that there is a small constant K such that among any sequence of k consecutive steps by one process, at least one changes the value of U (this is true since there are quite a few steps which affect U). ■

Lemma 10.4 *If $U \geq 3n$ at some configuration S, then every process terminates when it next performs $read(C)$ if it has not yet terminated. This implies that the algorithm terminates in $O(n)$ additional steps.*

Proof: Let m be the number of processes poised to do $inc(C)$. Then $C \geq 3n - m$ in S. Observe that the m processes will not decrement C before next reading C. Further the other $n - m$ processes many each decrement C at most once before reading C. Consider the first $read(C)$ after S. This read returns a values

$$\geq (3n - m) - (n - m) \geq 2n.$$

From Lemma 10.2, all processes will only increment. In total we will have at most $n - m$ decrements after S. Thus all reads performed after S will return a values $\geq 2n$ and the process will terminate. ■

Lemma 10.5 *The expected number of steps in an execution until $|U| \geq 3n$ is in $O(n^2)$.*

Proof: Observe that U starts at 0 and the execution is essentially a random walk on the value of U (generally it is better than a random walk because some steps deterministically increase the magnitude of C). There $\frac{4}{5}$ steps which don't modify U, but don't worry about those. Finally, by a result from probability theory U reaches $3n$ or $-3n$ in expected $O(n^2)$ steps. ■

10.1.2 Algorithm Against an Oblivious Adversary

The structure of the algorithm is as follows: we take the inputs and check for agreement. If we are successful, then output the value. Otherwise we will conciliate and check agreement again.

To facilitate the process we will define the Adopt-Commit object. This object supports operation $AdoptCOmmit(v)$. The result of this operation is either $(commit, v)$ or $(adopt, v)$ where v is an input value. If we get $(commit, v)$ then we decide v immediately. If we get $(adopt, v)$ then use use v as the input to the next round.

Observe that the following properties are satisfied:

Algorithm 1 End-zone algorithm: code for p_i.

```
1: R_{x_1} ← 1
2: c_i ← read(C)
3: while |c_i| < 2n do
4:     if c_i ≥ n then
5:         inc(C)
6:     else if c_i ≤ −n then
7:         dec(C)
8:     else if R_0 = 0 then
9:         inc(C)
10:    else if R_1 = 0 then
11:        dec(C)
12:    else
13:        f_i ← flips a coin
14:        if f_i = 1 then
15:            inc(C)
16:        else
17:            dec(C)
18:        end if
19:    end ifc_i ← read(C)
20: end while
21: return c_i ≥ 2n
```

Coherence: if the output of some AdoptCommit operation is $(commit, v)$, then every output is either $(commit, v)$ or $(adopt, v)$.

Convergence: if all inputs are v then all outputs are $(commit, v)$.

Wait-free Termination

The conciliation task requires: (1) validity, (2) probabilistic agreement - all outputs are the some with probability $\delta > 0$, and (3) wait-free termination.

If follows that if the Adopt-Commit operation takes A steps and conciliate takes C steps then the expected step complexity is $O\left(\frac{C+A}{\delta}\right)$.

Lets implement the Adopt-Commit object! We need three shared registers $a[0]$, $a[1]$ (both initialized to 0) and proposal (initialized to $\perp$). These will act as flags to ensure validity. Again, we are only going to concern ourselves with binary consensus.

You should check that Algorithm 2 satisfies the three conditions above.

How about conciliation? This time we will use one shared register R initially set to $\perp$, and we assume an oblivious adversary. See the pseudo-code in Algorithm 3.

Lets do the analysis: we claim that once some process p writes to R, the chance that any of the other $n-1$ processes write to R before noticing that $R \neq \perp$ is at most $\frac{(n-1)}{2n}$. The easiest way to see that this is the case by considering the first write to R by some process p_i. There can be at most $n-1$ other processes which can write to R, but they must all flip a coin first. By union bound the chance that some coin flips a head is just $\frac{n-1}{2n}$. So δ for conciliation is constant (that is good). Further the expected work and expected step complexity is $\Theta(n)$.

We can improve this algorithm to $\Theta(\log n)$ step complexity by making the probability of flipping a head $\frac{2^k}{2n}$

Algorithm 2 Adopt-Commit object: code for p_i.

```
1: AdoptCommit(v_i) by process p_i
2: a[v_i] ← 1.
3: if proposal = ⊥ then
4:     proposal ← v_i
5: else
6:     v_i ← proposal
7: end if
8: if a[1 − v_i] = 0 then
9:     return (commit, v_i)
10: else
11:    return (adopt, v_i)
12: end if
```

Algorithm 3 Conciliation: code for p_i.

```
1: r_i ← read(R)
2: while while r_i = ⊥ do
3:     f_i ← flip biased coin (1/2n heads)
4:     if f_i gets heads then
5:         R ← x_i
6:         return x_i
7:     else
8:         r_i ← read(R_i)
9:     end if
10: end while
11: return r_i
```

where $k = 0$ at the beginning and increments by one every time we flipped a tail.