CS465 Distributed Systems Problem Set 2

Sherwin Yu

November 3, 2011

1 Asynchronous consensus with delivery notifications

1.1 n=2 and one faulty process

Consider the following algorithm which solves consensus with two processes and tolerates one process failure:

```
Init:
    myValue = inputValue
    send myValue
On receive(value):
    decide value
On delivery confirmation:
    decide myValue
```

The only difficulty of asynchronous consensus is we do not know whose value is delivered first, as both processes try to send their initial values. This algorithm gets around this with delivery notifications because the first value to be delivered is decided upon: the "On Receive" and "On delivery confirmation" steps (for the same message) before any other messages are sent. So we have agreement and termination in the case of no crash failures. same time. Now, in the case of a crash failure, we must only ensure that the surviving process outputs anything at all, as the surviving process will obviously agree with itself. Indeed, the surviving process will either 1) receive the message of the now-dead process, and decide that (in the case that the crashfailure occurs after the other process sends its value) or 2) send its own value and receive confirmation and decide its own value. So we have termination (and therefore agreement) in the case of a crash failure. Finally, this satisfies validity (if both processes have the same inputs, then they must decide on the same input). Thus, this alogrithm solves consensus.

1.2 n=3 and two faulty processes

This is a variation of the FLP impossibility proof for asynchronous consensus with one crash failure; only key differences are discussed. We begin with an

initial bivalent configuration and will show that at each step, there is always another initial bivalent configuration to go to. Follow the FLP proof from the textbook until we are considering Dee' and $De'e$. Because of the existence of delivery notifications, at most two processes can distinguish between Dee' and $De'e$: this occurs e and e' are two messages sent between two processes (p sends a message to p', p' sends to p).

Now, similar to FLP, we consider a finite sequence $e_1e_2e_3...e_k$ in which message sent by either of the two processes who can distinguish Dee' and $De'e$ is delivered, and a process decides in $Dee_1e_2e_3..e_k$ (this must happen because the deciding process cannot wait forever and it is possible that the other two processes failed). Then, this deciding process decides on the same value in Dee' and $De'e$, which contradicts the assumptions made in the FLP proof that one state is 1 valent and the other is 0 valent.

2 Circular failure detector

To solve consensus, we seek strong completeness and weak accuracy. 13.3 shows that weak completeness can be boosted to strong completeness via broadcasting suspects, so we really only need weak accuracy and weak completeness.

2.1 Lowerbound: $\sqrt{n}$

Let m be the number of processes that are permanently suspected, and assume the processes know that there will be at most k failures. For each process suspected, also suspect the $k-m$ processes prior to each suspected process (or until we hit another process suspected originally by a failure detector). (So the worst case is when the original failures are evenly spaced around the circle, as we will have fewer total suspected failures if all of the original failures are lined up one after another). The objective is to suspect as few additional processes while still maintaining weak accuracy. The total number of permanently suspected processes, which has to be less than n in order to maintain weak accuracy, is $m(k-m+1) < n$. We maximize m via differentiation to find the best lower bound:

$$k - 2m + 1 = 0$$
$$m = \frac{k+1}{2}$$

Substituting back we find:

$$\begin{aligned} n &> m(k-m+1) \\ n &> \frac{k+1}{2}(k - \frac{k+1}{2} + 1) \\ n &> (\frac{k+1}{2})^2 \\ k &< 2\sqrt{n} - 1 \end{aligned}$$

We are strongly accurately suspecting the first m processes (they were identified by the original strongly accurate failure detectors), but we then use these m processes as seeds to suspect an additional $k - m$ processes to reach weak completeness. This is because each failed process will either 1) be part of the permanently suspected chains, in which case it is already permanently suspected. Or 2), be outside of the permanently suspected chains, in which case an original strongly accurate failure detector will eventually permanently suspect it. Thus, all failed processes will be eventually permanently suspected by at least some process (weak completeness). As explained earlier, completeness and weak accuracy are sufficient to solve consensus.

2.2 Upperbound

Assume k (number of failures) $> 2\sqrt{n} - 1$. We show that with this, it is impossible to solve consensus. Assume the worst case, with $\sqrt{n}$ original failures distributed evenly around the ring. Then there could be at least $sqrtn - 1$ more failures, which is more than the number of processes separating any two original failures. This is enough to cause the entire "block" to fail. This is indistinguishable from the entire block taking a long time to respond: this reduces to the FLP case with one crash failure (the entire block fails together). As follows with the FLP case, agreement is impossible. So our upper bound is $2\sqrt{n} - 1$.

3 An odd problem

A protocol does not exist for $n > 2$.

1. $n = 1$ the process simply sets off its alarm when its sensor is on (it's the only process). This obviously satisfies termination, there will be no false positives as the single process is the only one sounding the alarm.

2. If $n = 2$, the following algorithm works. When one process's sensor changes state, it notifies the other process of its new state. Upon receipt of a message (which indicates someone's sensor state), the receiver will check his own sensor, and sound the alarm if it is appropriate, because we are guaranteed that the sender's sensor is still in the same state (sensors don't change for at least two time units, and sending message takes one time unit). Example: Process B's sensor turns on, so it sends a message. Process A receives the message; if A is off, then A will sound the alarm (because only B is on). However, if A is on, then A will not sound an alarm because it knows that at that moment, there both processes are on. There will never be a false positive, as only the receiver of a message is sounding the alarm. Termination is satisfied as well, which was shown in the previous example.

3. $n = 3$. Suppose process A's sensor turns on and remains on permanently. Then, for termination, an alarm must be sounded eventually. We will show

that we cannot guarantee this without the possibilty of a false positive. Suppose A eventually sets off the alarm at some time t. But at that moment, B or C's sensor may have went off (there is a one time unit delay in communication). So there is always a possibility of a false positive if A sets off the alarm. Now suppose either B (or C, wihout loss of generality) is the one that eventually sets off the alarm. Similarly, B can never know whether C's sensor goes off at the moment B sounds the alarm. So we also have a possibility of a false positive.

4. $n > 3$. Larger cases reduce to the $n = 3$ case if all sensors excluding the first 3 remain silent. Since the $n = 3$ case is impossible, $n > 3$ must be impossible as well.

Ultimately, the impossibility stems from having a time delay in information. This problem is circumvented when $n = 2$ because we have the receiver sound the alarm when notified of a sensor change from the sender – at that moment, the receiver has full information: it knows its own state as well as the sender's state (guaranteed by the sender's alarm remaining on for two time units). This doesn't work for $n > 2$ because there are additional processes to consider.