EE 382N: Distributed Systems **Fall 2017**

Lecture : Failures - November 3

Lecturer: Vijay Garg *Scribe: Eric Addison*

0.1 Introduction

In this lecture, we continue our discussion of process failures. We know from the FLP result that in an asynchronous network, there is no algorithm that can solve the binary consensus problem in the presence of even one unannounced failure. We discuss here algorithms that can solve the consensus problem when we restrict ourselves to a synchronous network. Once that result is established, we broaden the failure model to allow arbitrary, or Byzantine, behavior.

0.2 Synchronous Systems

A synchronous distributed system is one in which there is some mechanism by which messages can time out. If a message takes longer than some specified time interval, the corresponding process can be considered as failed. A first attempt at solving the binary consensus problem in a synchronous network might be:

```
$P_i$ ::
        $v_i -$ initial value
        $V -$ array of values received from others

    do:
        send $v_i$ to all
        receive $v_j$ from others, set $V[j] = v_j$
        decide on majority value in $V$
```

Unfortunately, this algorithm does not work under certain failure conditions. For instance, consider three processes P_1, P_2, and P_3, which propose values 1, 1, and 0 respectively, and agree to break ties by choosing 0. Imagine that P_1 successfully sends its value to P_2, but fails before it can send to P_3. The resulting V arrays for P_2 and P_3 the look like:

$$P_2 : \ [1, 1, 0] \ \Rightarrow \text{choose } 1$$
$$P_3 : \ [x, 1, 0] \ \Rightarrow \text{choose } 0$$

This is clearly an incorrect state, as both remaining processes have confidently decided on different values. We can fix this issue by introducing the concept of *rounds* that can last a finite duration.

0.2.1 Binary Consensus

We can think of performing the same algorithm stub above multiple times, where instead of broadcasting only your own value, you broadcast your entire vector V. By repeating this process a number of times, and merging a received vector with our own, we can ensure that eventually all processes will see the same vector of values and come to the same decision. This process must be done $f+1$ times in order to tolerate f failures. The algorithm looks like:

```
$P_i$ ::
        $V -$ array of known values

   for $round = 1..(f+1)$ :
        send $V$ to all
        for $j = 1..(N)$ :
           receive $V_j$ from $P_j$
           $V = V \cup V_j$

decide on majority value in $V$
```

0.3 Byzantine Failures

We can extend the failure model to encompass not just crashed processes, but faulty or even malicious processes by allowing a “bad” process to generate arbitrary responses. In this so called Byzantine model, a faulty process can do whatever it wants, for example send an incorrect value, skip messages, etc.

0.3.1 The Byzantine General Agreement Problem

Consider the canonical allegory of three battalions of an army surrounding an enemy army. The battalions are led by Generals who must decide whether they will attack or retreat by coordinating with the other Generals. If the Generals all vote to attack, then they attack. If even one votes to retreat, then they all retreat. This problem is well modeled by the binary consensus problem, where instead of deciding on the majority value, the Generals decide on the minimum value. This is known as the Byzantine General Agreement Problem (BGA).

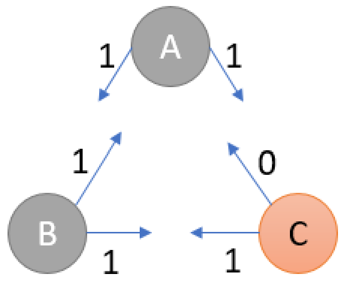

Figure 0.1: Faulty process C causes the correct processes to disagree.

Let the processes (Generals) be labeled A, B, and C, where C is a faulty process (treacherous General). The faulty process can cause the two correct processes to disagree by sending different information to each. For example, imagine that A and B both choose value 1 (attack), but the faulty process sends a 1 to one correct process, and a 0 to the other. This will cause the two correct processes to disagree, which in the context of the allegory, means that only one of the two battalions will attack, and presumably be defeated.

What if we try to add multiple rounds to the process as with the simple failure model? It turns out that BGA cannot be solved, after any number of rounds, for $N = 3$ and $f = 1$, where f is the number of faulty processes. This is a theorem:

Theorem: There is **no** protocol to solve BGA for $N = 3$ and $f = 1$.

If $N = 4$ and $f = 1$, then it is solvable. This result can be proven, and generalized to the stronger result that:

Theorem: There is **no** protocol to solve BGA for $N \leq 3f$.

0.3.2 The Queen Algorithm

In 1989, Berman and Garay [1, 2] described two algorithms for solving the BGA problem, known as the King and Queen algorithms. The Queen algorithm is simpler and faster (requires fewer rounds), but the King algorithm can tolerate more failures. The queen algorithm is described below.

The Queen algorithm, also known as the rotating coordinator algorithm, can tolerate failures when $N > 4f$ and takes $f + 1$ rounds. Initially, all processes store their value x in their vector V such that $V[i] = x$, and stores a default value $V[j] = v_\perp$ for all $j \neq i$. The algorithm now proceeds with $f + 1$ rounds, each with two phases. In the first phase, all processes exchange their values $V[i]$ with all other processes. If there are no failures, all processes will now know the correct value so each process now computes what would be the correct value and stores it as $v_i = majority(V)$, or $v_i = v_\perp$ if no majority is found. However we think there may be up to f failures. In phase 2, a "Queen", or coordinator, is assigned based on the round number, who sends her value out to all other processes. If a process's own value v_i has an *overwhelming majority*, that is, $tally(V, v_i) > N/2 + f$, then the process P_i (re)sets the value $V[i] = v_i$. Otherwise, P_i will set $V[i] = queenValue$.

Once an overwhelming majority is found, it persists (as shown in the text book). It can also be shown that a majority will be valid, and that agreement will be reached within the $f + 1$ rounds. The algorithm is shown below.

```
$P_i$ ::
    var
        $V$ : array of known values, initially $(V[i] = x)$ $[j] = v_\perp, j \neq i$

    for $Q = 1..(f + 1)$ :
        phase 1:
            send $V[i]$ to all other processes
            set $V[j]$ to the value received from all other $P_j$
            $v_i = majority(V)$ or $v_\perp$ if no majority

        phase 2:
            if($Q == i$) :
                send $v_i$ to all other processes
            receive $queenValue$ from $P_Q$
            if($tally(V, v_i) > N/2 + f$ :
                $V[i] = v_i$
            else $V[i] = queenValue$
decide $y = V[i]$
```

References

[1] Piotr Berman and Juan A. Garay. Asymptotically optimal distributed consensus (extended abstract). In *Proceedings of the 16th International Colloquium on Automata, Languages and Programming*, ICALP '89, pages 80–94, London, UK, UK, 1989. Springer-Verlag.

[2] P. Berman, J. A. Garay, and K. J. Perry. Towards optimal distributed consensus. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, SFCS '89, pages 410–415, Washington, DC, USA, 1989. IEEE Computer Society.