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

Lecture 3: Byzantine Fault Consensus

Lecturer: Faith Ellen *Scribe: Lily Li*

3.1 From Last Week

We looked at synchronous model with crash failures. Now we will look at asynchronous model with crash failures. It is a bit more difficult for us to tell when a processor crashed as opposed to it running really slow. Generally we say a processor **crashed in an infinite execution** if it only have finitely many execution steps and the last step is not is terminating step. As before, when a process crashed a subset of its sent messages get put on the out buffer.

Theorem 3.1 *(FLP) It is impossible to solve* 1*-resilient binary consensus in an asynchronous message passing system.*

Proof: (This follows closely the original proof in the paper.) The theorem follows from Lemma 3.2 and Lemma 3.5. Use the first lemma to start a bivalent configuration. Repeated apply the second lemma to the longest pending process. This gives an infinite bivalent execution. ■

Lemma 3.2 *A binary consensus algorithm has an initial bivalent configuration.*

Definition 3.3 *A* ***critical configuration*** *is a multivalent configuration where all next step moves turns the system into a univalent configuration.*

Lemma 3.4 *No binary consensus algorithm has a critical configuration.*

Proof: By contradiction. Suppose C is a critical configuration. Then there is some s_0 that leads to a 0-valent config. and s_1 which leads to a 1-univalent configuration. Suppose that s_0 and s_1 are both computation events by different processes then the processes commute and we have a contradiction. This is the same thing that happens when you have two delivery events or one comp and one deliver but the message gets delivered to someone else. You only need to consider a $\mathsf{comp}(i)$ event and a delivery event to process p_i.

Welp, we are 1-resilient so you just sort of *crash* process i (I say sort of since the adversary doesn't really say any process crashed, it just slows p_i down). The algorithm must still terminate so s_0 followed by s_1 needs to output 0 (remember s_0 is one univalent). Since this is indistinguishable to all other processors as s_1 followed by s_0 that must output 0 as well. But s_1 is 1-valent, contradiction. ■

Lemma 3.5 *Let C be a multivalent configuration and let s be a step that can be performed in C then there is a finite execution starting at C and ends at s. That ends in a multivalent configuration.*

Proof: Note that the above lemma is not strong enough to prove the theorem (not the one we are currently proving)! We need *finite* executions! Suppose all executions ending in s are univalent. WLOG Cs is 0-univalent. Since C is bivalent there must be some other execution which ends up being 1-univalent. If this execution contains s then one branch in the tree is 1 univalent. If not in tree just tack on a s and then add it into the tree. Consider the shortest execution γ in the tree which ends in 1. On the path from C to s via γ we have s' followed by s. If there is another execution p from s' then s' followed by p must be 0-univalent (why?). There is a contradiction as before if process s and s' commute. Suppose s and s' do not commute since they are computing and delivering for some processor p_i. ■

3.2 Byzantine Agreement

3.2.1 Formal Model

Definition 3.6 *In the execution of an* f***-resilient Byzantine system****, there exists a subset of at most* f *processors (known as the* ***faulty*** *or* ***traitor*** *processors). For each computation step of a faulty processor, the new state and the content of the message sent are unconstrained.*

3.2.2 Consensus Again

The definition of consensus under the Byzantine failure model is the same as that of the crash failure model. Note that in the case of validity we need that if all processors get the same input v then the output of all non-faulty processors is v.

First it is a good idea to show a lower bound on the ratio of faulty to non-faulty processors. This makes sense intuitively since too many traitors can sway the vote considerable.

Theorem 3.7 *In a system with three processors and one Byzantine processor, there is no algorithm that solves the consensus problem.*

Proof: The proof is by contradiction. Suppose there is an algorithm for three processes p_0, p_1, and p_2. In order to see that this doesn't work we will look another situation: consider the cycle $p_0, p_1, p_2, p'_0, p'_1, p'_2$ where the unprimed processors get 0 and the primed processors get input 1. Consider an execution α in this hexagon (which is the same algorithm for the three processor case, we don't expect that it will work). For each processor, the local picture is the same. Consider p_0 and p_1 with a Byzantine processor q. By validity, p_0, p_1 must output 0. q will behave like p'_2 to p_0 and p_2 to p_1. Similarly for p'_2 and p'_1 with another Byzantine processor q'. Now consider the pair p_0 and p'_2 with a third Byzantine processor q". What happens? ■

We can generalize the above to the following:

Theorem 3.8 *In a system with* n *processors and* f *Byzantine processors, there is no algorithm that solves the consensus problem if* $n \leq 3f$.

Proof: We will reduce to Lemma 3.7. Again proof by contradiction. We are first going to divide the n processes into three groups each with $\leq f$ processes. We will have p_0, p_1, p_2 simulate one of these groups each. At most one of these simulated big nodes is faulty. Well the algorithm still has to work so it has to work in the three node case which we know is impossible. ■

Thus in the forgoing we can assume that fewer than one third of the processors are Byzantine. There are two different algorithms which solve the f-resilient Byzantine system for consensus. One has optimal round complexity but exponential bit complexity while the other has twice the round complexity but constant bit complexity.

3.2.2.1 Exponential Bit Complexity

Theorem 3.9 *There is an algorithm that solves the Byzantine Agreement (BA) problem in a complete network in $f + 2$ rounds when $f < n/3$ processors are Byzantine.*

Proof: Not going to go through this in detail. In the analysis you see the bit complexity is quite bad. ■

Algorithm 1 Algorithm for BA in a complete network: code for processor p_i.

```
1: Broadcasts input value $u_i$
2: for rounds $2, ..., f + 1$ do
3:     Broadcasts all the information it receives in that round
4: end for
5: Compute $u_i$ output from all the messages it receives using something like a recursive majority function.
```

3.2.2.2 Constant Bit Complexity

Theorem 3.10 *There is a BA algorithm for a complete network that runs in $2f + 3$ rounds where each message consists of only one input value and tolerates $f < n/4$ Byzantine processors.*

Termination should be pretty obvious. Use Lemma 3.11 to show validity and Lemma 3.12 to show agreement.

Algorithm 2 Phase King Algorithm: code for processor i.

```
1: Initialize array $\mathsf{pref}_i$ of length $n - 1$.
2: $\forall j \neq i$ set $\mathsf{pref}_i[j] \leftarrow 0$.
3: for rounds $1, ..., f + 1$ do
4:     do Phase $k$:
5:     Round 1:
6:     Broadcast $\mathsf{pref}_i[i]$
7:     $majority \leftarrow$ majority value in $\mathsf{pref}_i$
8:     $multiplicity \leftarrow$ number of times $majority$ appears in $\mathsf{pref}_i$
9:     if $i = k$ then
10:        $p_i$ is the King of phase $k$
11:        send $majority$ to all processors
12:    else
13:        $kingmajority \leftarrow$ value received from King
14:        if $multiplicity > n/2 + f$ then
15:            $\mathsf{pref}_i[i] \leftarrow majority$
16:        else
17:            $\mathsf{pref}_[i] \rightarrow kingmajority$
18:        end if
19:    end if
20: end for
21: output $\mathsf{pref}_i[i]$
```

Lemma 3.11 *If all non-faulty processors prefer v at the beginning of phase k, then they all prefer v at the end of phase k for all $1 \leq k \leq f + 1$.*

Proof: During the first part of k phases each non-faulty process gets $\geq n - f$ copies of v. This is greater than $3n/4 > f + n/2$ so the multiplicity is greater than $3/4$. Thus all the non-faulty processors set $\mathsf{pref}_i[i]$ to $majority$. ■

Lemma 3.12 *Let g be a phase whose king p_g is non-faulty. Then all non-faulty processors finish phase g with the same preference.*

Proof: Since p_g is non-faulty it will send $majority_k$ to all other processors during the second round of phase k. We will show that for every non-faulty processor p_i, $\mathsf{pref}_i[i] = kingmajority_k$. Suppose there exists some non-faulty processor p_i for which $\mathsf{pref}_i[i] = v \neq kingmajority_k$. Then p_i must have gotten $n/2$ messages for v from *non-faulty* processors. But p_g must have gotten more than $n/2$ messages of v as well. Then $kingmajority_k = v$. This is a contradiction. ■

3.3 Asynchronous Shared Memory System

Definition 3.13 *The **shared memory** communication model has processors which communicate via a common memory area containing a set of **shared variables**. Here we only consider **asynchronous shared memory systems***

There are a variety of different types of shared variables including: read/write registers, read-modify-write, test & set, or compare & swap. There is further classification according to the number of processors allowed to access a specific variable with a particular operation.

3.4 Homework

There is a Byzantine Agreement algorithm that tolerates $f < n/3$ Byzantine faults in a network if and only if the network is $2f + 1$ connected.