CSE 4/586 Distributed Systems: Ben-Or's Randomized Consensus Algorithm

Task 1:

Scenario 1: F <- 0 N <- 4 INPUT <- <1,1,1,1> MAXROUND <-4

In this scenario, all the inputs are assigned to 1 initially and there are no faulty nodes. In this scenario, the model throws no error as the progress property (it means all the nodes eventually end up deciding on a specific value) is satisfied and the model reaches the consensus value, in this case 1. The value **1** is proposed in the phase 1 and is seen as the majority value (as more than N-F nodes agree on this value) which eventually gets decided as the final value in phase 2 (as there would be no fault nodes as well).

Scenario 2: F <- 0 N <- 4 INPUT<- <0,0,0,0> MAXROUND <- 4

This scenario pretty much is the same as Scenario 1. The only difference is that the initial values are set to 0 unlike 1. The reasoning given in Scenario 1 holds for this too and the final value would be decided as 0.

Scenario 3: F <- 0 N <- 4 INPUT<- <0,0,1,1> MAXROUND <- 4

In this scenario, the initial values have a mixture of 0's and 1's. Clearly the majority is not decided in the beginning. In the phase 1, a majority value cannot be decided by the end of the round as there would be no single value that is in majority (> N/2). This will result in no value being decided for phase 2. Hence the default p2v value which is -1 will be proposed for the phase 2. A new assignment of values among 0 and 1 will be proposed for the next round (this can be a prior round value a node has seen or any random value among 0 and 1) thus resulting in a non-consensus state which inturn is violating the progress property. Therefore this randomness can lead the model to not reach a consensus before MAXROUND (round reaches 5). So, the model will fail for the above mentioned case in this scenario.

The following are the last states before the model fails.

```
decided = <<{}, {}, {}, {}>>
p1v = <<1, 0, 0, 0>>
p2v = <<-1, -1, -1, -1>>
 pc = <<"S", "P1", "P1", "P2">>
```

```
r = <<4, 4, 4, 3>>
---------------------------------------------
decided = <<{}, {}, {}, {}>>
p1v = <<1, 0, 0, 1>>
 p2v = <<-1, -1, -1, -1>>
pc = <<"P2", "P2", "P2", "P2">>
r = <<4, 4, 4, 4>>
---------------------------------------------
decided = <<{}, {}, {}, {}>>
p1v = <<1, 1, 0, 1>>
p2v = <<-1, -1, -1, -1>>
pc = <<"Done", "Done", "Done", "Done">>
r = <<5, 5, 5, 5>>
```

Scenario 4: F <- 1 N <- 4 INPUT<- <1,1,1,1> MAXROUND <- 4

In this scenario, the initial values are all set to 1 as in scenario 1, but there can be one faulty node present. As expected, the model reaches consensus and executes without any error. This is because although one of the nodes fail, the majority (>N/2 = 3 in this case) is achieved with the remaining nodes and the value 1 is passed to phase 2. Similarly the phase 2 also decides on this value, as all the remaining nodes propose the same value and also the number of nodes that have seen this value in phase 2 will be greater than F (= 1 in this case).

Scenario 5: F <- 1 N <- 3 INPUT<- <0,1,1> MAXROUND <- 4

In this scenario, there is a mixture of 0's and 1's again. This is similar to Scenario 3 but the number of nodes that can fail (F) is 1 and there are only 3 nodes instead of 4. So there is a possibility that one of the nodes will fail and that would lead to the messages consisting of equal mixture of 0's and 1's. Like we discussed in Scenario 3, this will result in violating the progress property. However, this may not be the case when the node that failed has the minority (in this case, 0 - the first node). So, if the failed node is proposing 1, the consensus will be violated for one of the execution cases due the possibility of random assignment.

```
decided = <<{}, {}, {}>>
p1v = <<1, 0, 1>>
p2v = <<-1, -1, 1>>
pc = <<"P2", "P2", "S">>
r = <<4, 4, 4>>
-----------------------------------------------
decided = <<{}, {}, {}>>
p1v = <<0, 0, 1>>
p2v = <<-1, -1, 1>>
pc = <<"Done", "Done", "P2">>
```

```
r = <<5, 5, 4>>
------------------------------------------------
decided = <<{}, {}, {}>>
p1v = <<0, 0, 1>>
p2v = <<-1, -1, 1>>
pc = <<"Done", "Done", "Done">>
r = <<5, 5, 5>>
```

Task 2:

Scenario 1: F <- 0 N <- 4 INPUT<- <0,1,1,1> MAXROUND <- 4

In this scenario, we introduce an invariant named “MinorityReport” where we claim that when 0 is in minority in the INPUT, it can never be decided as the final value when the consensus is reached. For this scenario, the model runs successfully as there are no failures that can happen and 1 is in majority initially which eventually leads to 1 being decided as the final value. This happens because phase 1 selects 1 as the final value as it is in majority (> N/2 and no failures), also in phase 2 these 1s get passed as that will be decided as the terminal value (> F=0). So, there would be no case where a node gets to randomly select 0 as it 1 is always in majority. Hence the model doesn't fail(i.e it does not violate the Minority Report Invariant)

Scenario 2: F <-1 N <- 4 INPUT<- <0,1,1,1> MAXROUND <- 4

In this scenario, the model fails with the error that states - “Invariant MinorityReport is violated.”. This means that even though clearly 0 is in minority in the beginning, the model can still get 0 as the final/consensus value. In this run, it is observed that the model obtained decided = <<0, 0, 0, 0>> in the 25th state clearly violating the minority report invariant. This is possible when the node that fails contains the value 1, hence violating the majority for phase 1 to decide on the value 1, so a random or prior assignment is done for the consequent round and the 1s present initially maybe flipped to 0, and eventually ending on 0 in the majority. These 0s will pass the phase 1 majority (> N/2 = 3) and phase 2 will also accept (> F = 1) 0s as the final decided value, failing the Minority Report as expected.

```
decided = <<{0}, {0}, {0}, {}>>
p1v = <<0, 0, 0, 1>>
p2v = <<0, 0, 0, 1>>
pc = <<"S", "S", "S", "P2">>
r = <<3, 3, 3, 1>>
------------------------------------------------
decided = <<{0}, {0}, {0}, {}>>
p1v = <<0, 0, 0, 1>>
p2v = <<0, 0, 0, 0>>
```

```
pc = <<"S", "S", "S", "P2">>
r = <<3, 3, 3, 2>>
------------------------------------------------
decided = <<{0}, {0}, {0}, {0}>>
p1v = <<0, 0, 0, 0>>
p2v = <<0, 0, 0, 0>>
pc = <<"S", "S", "S", "S">>
r = <<3, 3, 3, 3>>
```

Invariants:

$$Agreement == \backslash A\, j, k \backslash in\ Procs : Cardinality(decided[j]) = 1 \wedge Cardinality(decided[k]) = 1 \Rightarrow decided[j] = decided[k]$$

Agreement: Agreement property states that for every process, say j,k, the decision values should be the same and the decided set of those processes contains the same number of values.

$$MinorityReport == \tilde{} \backslash A\, j \backslash in\ Procs : decided[j] = \{0\}$$

Minority Report: Minority property states that it is impossible for all the nodes to decide 0 as the final consensus value.

Properties:

$$Progress == <> (\backslash A\, j \backslash in\ Procs : Cardinality(decided[j]) > 0)$$

Progress: This property states that all the process(say j) will come up with a value/or decide something after every round ends. So we check the cardinality of the decided set and we know there is some value decided if it is not an empty set.

Conclusion:

Ben-Or is a leaderless based decentralized algorithm. It uses binary input over arbitrary input value to solve the consensus in case of tie-breaking situations (as there is no leader to solve the tie). Also, it utilizes randomization to achieve progress. The Ben-Or algorithm has two phases in each round and is common to all nodes. A round gets locked until there are at least N-F (N = nodes, F = Maximum number of faulty nodes) nodes that propose a value. In the first phase of a round, every node tries to propose a value supported (among 0 and 1) by a majority of the nodes (to propose it for the next phase). In the second phase, a node finalizes its decision if it observed more than F+1 nodes propose the same value. If it fails to find a majority in a round, the algorithm makes some nodes to change their votes and proceed to the next round which helps the system to tilt towards a decision.

So, the system eventually converges to a consensus value as the round progresses. So, in this project we have seen how the algorithm behaves in various scenarios (with cap on number of rounds), and varying the number of nodes and faulty nodes and when the initial input contains mixture of 0's and 1's. And we were able to model how the algorithm achieved consensus using binary values and randomness.

References:

1. http://disi.unitn.it/~montreso/ds/syllabus/papers/AguileraToeug-CorrecnessBenOr.pdf
2. https://lamport.azurewebsites.net/tla/summary.pdf
3. http://muratbuffalo.blogspot.com/