CSE535: Distributed Systems

Project: DiemBFT v4 Consensus Algorithm Phase 2

Team Name: Loyal Byzantine Generals

Vivek Neppalli

Manish Adkar

Shubham Sahu

Test Report

All the test cases follow the same format for the config file; ledger files for validators; and log files for all validators as well as clients mentioned in the following format.

Config File:

'*config/config.da'* contains a list of all configurations to be executed.

Ledger File:

For each configuration at index *'$c'* mentioned in the configuration file, each validator with index *'$v'* creates its own ledger file under '*ledgers/config$c/validator_$v.ledger'*

Log File:

For each configuration at index *'$c'* mentioned in the configuration file, each validator with index *'$v'* creates its own log file under '*logs/config$c/validator_$v.log'.*

For each configuration at index *'$c'* mentioned in the configuration file, each client with index *'$r'* creates its own log file under '*logs/config$c/client_$r.log'*

The Ledger and Log files are not shown in the test cases as it clutters the report. The same can be seen by setting the configuration file and running the program as mentioned in the User-Manual.

Note: All the test cases included have probability set to 1 to deterministically demonstrate the test scenario. The program can take any value between 0 and 1 to have probabilistic behavior

Explanation of all the variable names mentioned in the config file:

'nvalidators': Number of Validators/Replicas,
'nfaulty': Number of Faulty Validators,
'nclients': Number of Clients,
'nclientops': Number of operations each client performs,
'sleeptime': Delay between consecutive operations for the same client in seconds,
'clienttimeout': Amount of time the client waits in seconds to receive the response. If no response is received, it retransmits that request
'delta': Amount of time in seconds used to decide the pacemaker timer timeout time,
'window_size': Window size used for Leader Election,
'exclude_size': Exclude size used for Leader Election,
'failure_config': FailureConfig object with a list of Failures to be injected

For each Failure object, the parameters are
src: The source of failure injection for the given message Example(0 for 0th validator, _ for all validators; Same for dest parameter),
dest: The destination of failure injection for the given message,
msg_type: Type of the message Example(MsgType.Proposal, MsgType.Vote),
round: Round number where the fault is to be injected,
prob: Probability of injecting the fault,
fail_type: Type of failure Example(FailType.MsgLoss, FailType.Delay, FailType.SetAttr),
val: Fault value to be injected Example(For FailType.Delay val is the delay in seconds, For FailType.SetAttr val is the value to be set for the given attribute in attr),
attr: Attribute to be set for FailType.SetAttr,
seed: Seed for the pseudorandom number generator used to determine outcomes of probabilistic message losses. If the seed is None, a seed is generated and written to a log.

Case 1:

Fault Injection: None

Outcome: The system follows a “Happy Path” with 4 Validators. There are no faulty validators. All the ledger files have consistent transactions written by their corresponding validator. The clients and validators terminate gracefully after all the commands by clients have been executed in the ledger.

```
{
    'nvalidators': 4,
    'nfaulty': 0,
    'nclients': 2,
    'nclientops': 2,
    'sleeptime': 1,
    'clienttimeout': 10,
    'delta': 2,
    'window_size': 5,
    'exclude_size': 0,
    'failure_config': FailureConfig(
        failures=[
        ],
        seed=12345678
    )
}
```

Case 2:

Fault Injection: Message Loss for Vote Message

Outcome: During the 3rd round, all the Vote messages get dropped. This causes a timeout. It is handled by generating Timeout messages which bring back the validators to recovery. This results in the abandonment of the block (and its transaction) generated in the proposal for 4th round. The client does not receive the response for the corresponding request and retransmits the request after the specified ‘clienttimeout’. The validators respond to this new (retransmitted) request by adding the transaction in the mempool and committing it in future rounds.

All the ledger files have consistent transactions written by their corresponding validator. The clients and validators terminate gracefully after all the commands by clients have been executed in the ledger.

```
{
   'nvalidators': 4,
   'nfaulty': 1,
   'nclients': 5,
   'nclientops': 5,
   'sleeptime': 1,
   'clienttimeout': 10,
   'delta': 5,
   'window_size': 5,
   'exclude_size': 1,
   'failure_config': FailureConfig(
       failures=[
           Failure(src='_', dest='leader', msg_type=MsgType.Vote,
round=3, prob=1, fail_type=FailType.MsgLoss, val=None, attr=None),],
       seed=12345678
   )
}
```

Case 3:

Fault Injection: Message Loss for Proposal Message.

Outcome: During the 4th round, the Proposal message from the round's leader is dropped. Now, no replica votes for this round. This causes a timeout. It is handled by generating Timeout messages which bring back the validators to recovery.

All the ledger files have consistent transactions written by their corresponding validator. The clients and validators terminate gracefully after all the commands by clients have been executed in the ledger.

```
{
   'nvalidators': 4,
   'nfaulty': 1,
   'nclients': 5,
   'nclientops': 5,
   'sleeptime': 1,
   'clienttimeout': 10,
   'delta': 2,
   'window_size': 5,
   'exclude_size': 1,
   'failure_config': FailureConfig(
       failures=[
           Failure(src='leader', dest='_', msg_type=MsgType.Proposal,
                   round=4, prob=1, fail_type=FailType.MsgLoss,
val=None, attr=None),
       ],
       seed=637713655
   )
}
```

Case 4:

Fault Injection: Message Delay for Vote Message.

Outcome: During the 5th round, the second and third validators delay their Vote messages. As the next round's leader does not receive 2f + 1 votes, it cannot form a QC. This causes timeout. It is handled by generating Timeout messages which bring back the validators to recovery. This results in the abandonment of the block (and its transaction) generated in the proposal for 4th round. The client does not receive the response for the corresponding request and retransmits the request after the specified 'clienttimeout'. The validators respond to this new (retransmitted) request by adding the transaction in the mempool and committing it in future rounds.

All the ledger files have consistent transactions written by their corresponding validator. The clients and validators terminate gracefully after all the commands by clients have been executed in the ledger.

```
{
   'nvalidators': 4,
   'nfaulty': 1,
   'nclients': 3,
   'nclientops': 6,
   'sleeptime': 1,
   'clienttimeout': 10,
   'delta': 5,
   'window_size': 5,
   'exclude_size': 1,
   'failure_config': FailureConfig(
       failures=[
           Failure(src=2, dest='_', msg_type=MsgType.Vote,
                   round=5, prob=1, fail_type=FailType.Delay, val=25,
attr=None),
           Failure(src=1, dest='_', msg_type=MsgType.Vote,
                   round=5, prob=1, fail_type=FailType.Delay, val=25,
attr=None)
       ],
       seed=637713655
   )
}
```

Case 5:

Fault Injection: SetAttr.

Outcome: In this configuration, we introduce a byzantine failure in round 8. The 4th validator while voting overwrites the current_round attribute to a lower value of 2.

All the ledger files have consistent transactions written by their corresponding validator. The clients and validators terminate gracefully after all the commands by clients have been executed in the ledger.

```
{
   'nvalidators': 4,
   'nfaulty': 1,
   'nclients': 3,
   'nclientops': 6,
   'sleeptime': 1,
   'clienttimeout': 10,
   'delta': 3,
   'window_size': 5,
   'exclude_size': 1,
   'failure_config': FailureConfig(
       failures=[
           Failure(src=3, dest='_', msg_type=MsgType.Vote,
                   round=8, prob=1, fail_type=FailType.SetAttr, val=2,
attr='current_round')
       ],
       seed=637713655
   )
}
```

Case 6:

Fault Injection: None

Outcome: The system follows a “Happy Path” with 7 Validators including 2 Faullty Nodes

All the ledger files have consistent transactions written by their corresponding validator. The clients and validators terminate gracefully after all the commands by clients have been executed in the ledger.

```
{
    'nvalidators': 7,
    'nfaulty': 2,
    'nclients': 6,
    'nclientops': 9,
    'sleeptime': 1,
    'clienttimeout': 10,
    'delta': 2,
    'window_size': 5,
    'exclude_size': 3,
    'failure_config': FailureConfig(
        failures=[],
        seed=637713655
    )
}
```

Case 7:

Fault Injection: VoteMsg loss, Proposal Msg delay, SetAttr

Outcome: Here, we have 7 validators out of which 2 are faulty. We introduce multiple faults in this config. Firstly, in the 4th round, we drop all the vote messages which cause a Timeout for that round. Next, in the 5th round, we delay the Proposal message sent by the leader which again will cause a timeout. In the same round, we introduce a byzantine fault by setting the current In round 8, we introduce two byzantine faults with the two faulty validators by setting the highest_vote_round to a higher value and the current_round to a lower value. Lastly, in round 10, we drop the votes sent by these two faulty validators, but now it won't cause a timeout as the leader will receive at least 2f + 1 votes which will lead to QC and commit of a block in the ledger advancing the round.

All the ledger files have consistent transactions written by their corresponding validator. The clients and validators terminate gracefully after all the commands by clients have been executed in the ledger.

```
{
   'nvalidators': 7,
   'nfaulty': 2,
   'nclients': 6,
   'nclientops': 9,
   'sleeptime': 1,
   'clienttimeout': 10,
   'delta': 2,
   'window_size': 5,
   'exclude_size': 3,
   'failure_config': FailureConfig(
       failures=[
           Failure(src='_', dest='leader', msg_type=MsgType.Vote,
                   round=4, prob=1, fail_type=FailType.MsgLoss,
val=None, attr=None),
           Failure(src='leader', dest='_', msg_type=MsgType.Proposal,
                   round=5, prob=1, fail_type=FailType.Delay, val=40,
attr=None),
           Failure(src=5, dest='_', msg_type=MsgType.Vote,
                   round=8, prob=1, fail_type=FailType.SetAttr, val=2,
```

```
attr='current_round'),
            Failure(src=6, dest='_', msg_type=MsgType.Vote,
                    round=8, prob=1, fail_type=FailType.SetAttr, val=100,
attr='highest_vote_round'),
            Failure(src=5, dest='leader', msg_type=MsgType.Vote,
                    round=10, prob=1, fail_type=FailType.MsgLoss,
val=None, attr=None),
            Failure(src=6, dest='leader', msg_type=MsgType.Vote,
                    round=10, prob=1, fail_type=FailType.MsgLoss,
val=None, attr=None)
        ],
        seed=637713655
    )
}
```

Case 8:

Fault Injection: VoteMsg Loss and Timeout Msg Loss

Outcome: Here, we have 7 validators out of which 2 are faulty. Firstly, in the 2nd round, we drop all the vote messages which cause a Timeout for that round. Next, as the round did not advance, we now drop the Timeout messages in round 2 with 0.25 probability. So now the validators restart the timer for the same round and then retry sending their Timeout broadcasts after it times out again. The clients and validators terminate gracefully after all the commands by clients have been executed in the ledger.

```
{
   'nvalidators': 7,
   'nfaulty': 2,
   'nclients': 6,
   'nclientops': 9,
   'sleeptime': 1,
   'clienttimeout': 10,
   'delta': 4,
   'window_size': 5,
   'exclude_size': 3,
   'failure_config': FailureConfig(
       failures=[
           Failure(src='_', dest='leader', msg_type=MsgType.Vote,
                   round=2, prob=1, fail_type=FailType.MsgLoss,
val=None, attr=None),
           Failure(src='_', dest='_', msg_type=MsgType.Timeout,
                   round=2, prob=0.25, fail_type=FailType.MsgLoss,
val=None, attr=None)
       ],
       seed=637713655
   )
}
```