- Platform:
  - DistAlgo Version: 1.1.0b15
  - Python Implementation: CPython
  - Python Version: 3.7.10
  - Operating system name: MACOS Big Sur
  - Operating System Version: 11.5.2
  - Type of Host: Laptop
- Workload Generation:
  - The main.da reads config.da file to get configurations for the run.
  - Based on the “nclients” value in the config given, required number of client processes are generated. The implementation for this is in driver.da
  - Requests are generated and sent to validators in driver.da
  - In driver.da, the client requests are generated and sent to all the validators
  - These requests are used to populate mempool.da and the validators take the required transactions from mempool.da.
  - Design: We have provided configs in a configs.da file for various test cases which help to set up the clients and validators in the testcase. We inject a failureconfig to each of these test cases and override the send message and check if any particular failure config is valid for the sending of the message. The various failure messages are in the configs file, and help to test our system.
- Timeouts:
  o Choice of timeout formula: 4 * Δ where Δ=2. After several iterations, we decided that Δ=2 gives the best results.
  o Timeout value for replicas (get_round_timer): 4*2= 8 secs
  o Timeout value for client (get_round_timer) : for client we give additional 2 secs of leeway, which makes client timeout value=10 secs
- Bugs and Limitations:
  - no bugs and limitation
- Main Files:
  - librabft/main.da
  - librabft/driver.da
  - librabft/blockTree.da
  - librabft/leaderElection.da
  - librabft/ledger.da
  - librabft/verificationHelper.da
  - librabft/config.da
  - librabft/safety.da

- Code Size:
  - Non-blank non-comment lines of code: total lines: 933
    - main.da- 47
    - driver.da- 430
    - blocktree.da- 106
    - leaderelection -61
    - ledger- 56
    - verificationHelper- 19
    - configs.da – 17
    - safety.da - 84

    - Algorithm: replica algorithm lines,other interleaved functionality like logging,instrumentation: 550
    - Other: clients,configuration,test drivers: 70
    - Total - 620
    - Out of the algorithm code, rough % of code actually for algorithm: 80%
    - Out of the algorithm code, rough % of code for interleaved and other process: 20%

- Language Feature Usage:
  - No. of list comprehension (python feature): 2 in driver.da
  - No. of dictionary comprehension (python feature): 0
  - Set comprehension (distalgo feature): 2 in driver.da
  - Aggregations (distalgo feature): 0
  - Quantifications (distalgo feature): 2 in driver.da
  - Await statements (distalgo feature): 2 in driver.da
  - Receive handlers (distalgo feature): 8 in driver.da, 1 in fault_injection.da,
- Contributors: [Equal Contributions]
  - Shivam- Timeout, slow replica sync-up, Safety, BlockTree, Main, Pacemaker, Verification, Happy Path
  - Abhinav – Fault Injection, Configuration, Ledger, BlockTree, Mempool, Cryptography, Verification, Happy Path
  - Ashutosh- LeaderElection, Logging, Happy Path, Configuration, Verification
- Other Comments: (optional)

- **Pseudocode:**

"sync up" replicas that got behind

Procedure start_event_processing(M):

```
// Everything else remains same …
...
If M is a initiateRecovery message then process_recovery_message(M)
If M is a recoveryMsg message then process_recovery_response(M)
```

Procedure process_proposal_msg(P):

```
If (P.block.round > pacemaker.current_round+1):
        initiateRecoveryProcess()
// Everything else remains same
....
```

Procedure initiateRecoveryProcess():

```
// construct taken from VR by Liskov
Send <initiateRecovery, nonce> to all other validators
```

Procedure process_recovery_message(M):

```
safetyState <- SafetyState(
                safety.get_highest_vote_round(),
                safety.get_highest_qc_round()
                )
blockTreeState <- BlockTreeState(
                self.blockTree.pending_block_tree,
                self.blockTree.pending_votes,
                self.blockTree.high_qc,
                self.blockTree.high_commit_qc,
                self.blockTree.signers
                )
ledgerState <- LedgerState(
                self.ledger.block_id_to_ledger_state_map,
                self.ledger.ledger_id_to_ledger_state_map,
                self.ledger.block_id_to_block_map
                )
pacemakerState <- PacemakerState(
                pacemaker_current_round,
                pacemaker_last_round_tc,
                Pacemaker_pending_timeouts
                )
leaderElectionState <- LeaderElectionState(
                self.leaderElection.reputation_leaders
                )
recMsg <- RecoveryMsg(ledgerState)
Send <recMsg, hash(rec), n), to= sender>
```

Procedure process_recovery_response(M):

```
// construct taken from VR by Liskov
Wait for f+1 same hash(recoveryMsg) from distinct replicas for given nonce
// set states to new state
setLedgerState(M.ledgerState)
setLeaderElectionState(M.leaderElectionState)
setPacemakerState(M.pacemakerState)
setBlockTreeState(M.blockTreeState)
setSafetyState(M.safetyState)
Set Status = “Normal” // construct taken from VR by Liskov
```

VALIDITY CHECKS FOR CRYPTOGRAPHIC VALUES

We validate the quorum of signatures by using the public keys of the respective signers by calling the verify function in the form below. We need to verify all the signatures and even if one of them is not verified, we reject the QC. The pseudocode is similar for a tc

```
for signer, signature in (qc.signers, qc.signatures):
        if not verify(public_keys[signer], message, signature):
                return False
 return True
```

We use hash values to create block ids and ledger commit id. Due to the consensus, everyone arrive on the same round, and receive the same qc and proposal, hence creating same block ids and commit ids. We don’t compare hashes directly, but indirectly we ensure same hash values are created as we are using same values.

VERIFYING THE CLIENT RESPONSE

The client will wait for f+1 responses from the replicas, which ensure at least one valid replica will respond with the correct response.
Pseudocode is given below

```
wait for received messages until length of received messages matching with the same commit id and the request number > f+1:
      Increase request number
else timeout():
   Retransmit message
```

CLIENT DE-DUPLICATION HANDLING

We handle the client de duplication by creating a hashmap in the mempool, and updating the status of the transaction over the protocol. Whenever we receive a transaction, we add it to the map and mark it status as “UNTOUCHED”. When a replica chooses the transaction in order to add it to the proposal block, we change its status to “PENDING”. So now when a client tries to retransmit, we ignore it. Once the transaction is committed, we change its status to “COMMITTED”. During the timeouts, there is a possibility, some messages are lost. While pruning the state tree, we will change the status of those transactions if they are marked as “PENDING” to “UNTOUCHED”, so that validators can pick it up if required.

```
add_transaction(transaction):
  if transaction not in mempool_map:

        mempool_map[transaction]=”UNTOUCHED”

mark_transaction_pending(transaction): //Called when block is received or created
        mempool_map[transaction)=”PENDING”

mark_transaction_committed(transaction):
        mempool_map[transaction]=”COMMITTED”
```