TESTING

- failures(C,r), where C is the configuration number, r is the replica number
- Failure triggers like client_request(c, m), where c is the client number and m is the request number (mth request)
- Config and log files are present scenario number wise in the folder

CLIENT :

- **Scenario** 14
  Ask Olympus whether configuration has changed periodically.

  Config file:

  Running the test case:

1. Set periodic check for latest config timer smaller than the retransmission timer or ask olympus before retransmission.
2. Put head replica to sleep when the first client request arrives.
3. failures(0,0) ; failure trigger: client_request(0, 1) ; action: sleep()

1. **Scenario:**
   Dictionary contains expected content at the end of the test case.

   Running the test case:

OLYMPUS :

- **Scenario: 2**
  Reconfiguration. Sending of wedge requests. Validation of wedge requests.

  Running the test case:

1. Olympus will first send wedge request to replica 1 for replica 1 is at fault,
2. failures (0,0) ; failure trigger: client_request(0,2) ; action: change_operation()

- **Scenario: 3**
  Testing reconfiguration and checkpointing

  Running the test case:

1. Running the code to test checkpointing and reconfiguration together
2. failures(0,0) ; failure trigger: (0,4) ; action: change_operation()

- **Scenario: 4**

Wedge requests from a different quorum (Reconfiguration). Replica catch up.

Running the test case:

1. Olympus will send wedge requests to the first quorum (replicas 0 and 1). They reply with history and checkpoint proofs. Using the histories obtained olympus calculates the longest history and sends catch up messages to the quorum (0,1). They reply with running states and result proofs as part of the catch up messages. Olympus now when validates the result proofs and the running states, detects the misbehavior and sends wedges requests again to a different quorum.
2. failures (0,1) ; failure trigger: shuttle(0,2) ; action: drop_result_stmt() ; wedge_request(1): truncate_history(2)

- **Scenario: 5**
  Get_running_state from a different replica (Reconfiguration)

Running the test case:

1. When olympus chooses replicas 0 and sends get_running_state message to it. It crashes and therefore olympus receives no reply and it send the get_running_state request to a different replica.
2. failures (0,1) ; failure_trigger: shuttle(0,2) ; action: drop_result_stmt()
3. failures (0,0) ; failure trigger: get_running_state(1) ; action: extra_op()

REPLICA:

- **Scenario: 1**
  Normal checkpointing

Running the test case:

1. Running the code without any failures to observe checkpointing (1 client 3 replicas)

- **Scenario: 6**
  Validation of checkpoint statements inside the checkpoint shuttle. Multiple configurations.

Running the test case:

1. When checkpoint shuttle goes from head to the next replica. The next replica executes an extra_op(). When the checkpoint shuttle shuttle reaches the next replica. It detects the misbehavior from the incorrect running state that is passed onto it. It calls for reconfiguration. Olympus sends wedge requests to replica 0 and 1. The history has no error so the validation of histories shows no error so olympus sends catch up messages

to replica 0 and 1. The replicas reply with the running states and this time olympus detects misbehavior in the running states and thus sends wedge requests to a different quorum (0 and 2 replica). Everything happens correctly and new configuration is set up. In the new configuration a new failure occurs when the 2nd shuttle is received by replica number 1 (drop_result_stmt). Replica 2 then calls for reconfiguration which happens smoothly

2. failure(0,1) ; failure trigger: checkpoint(1) ; action: extra_op()
3. failure(1,1) ; failure trigger: shuttle(0,2) ; drop_result_stmt()

- **Scenario: 7**
  Detection and handling of inconsistent(mutated) complete checkpoint shuttle

  Running the test case:

1. When tail sends the completed checkpoint shuttle back in the chain. Replica 1 drops first t+1 checkpoint statements from the completed checkpoint shuttle. Replica 0 matches the length of the completed checkpoint shuttle it receives with the total number of replicas in the system. If the length of the completed checkpoint shuttle is lesser, it detects the misbehavior and calls for reconfiguration. Olympus then sends wedge requests to replica 0 and 1 and receives completed checkpoint shuttles as a part of the wedge messages. It then compares the length of the completed checkpoint shuttles with the number of replicas in the system. If the length of any of the replica does not match, it send wedge requests to a different quorum.
2. failures(0,1) ; failure trigger: completed_checkpoint(1) ; action: drop_checkpt_stmts()

- **Scenario: 8**
  Validating the order proof signatures

  Running the test case:

1. When replica number 1 receives the order shuttle from client number 1, it signs the order statement with an incorrect signature and forwards the order shuttle. The next replica detects this and calls for a reconfiguration.
2. failure(0,1) ; failure trigger: shuttle(0,1) ; action: invalid_order_sig()

- **Scenario: 9**
  Validating the result proof signatures

  Running the test case:

1. When replica number 1 receives the shuttle from client number 1, it signs the result statement with an incorrect signature and forwards the shuttle. The next replica detects this and calls for a reconfiguration.
2. failure(0,1) ; failure trigger: shuttle(0,1) ; action: invalid_result_sig()

- **Scenario: 10**

Detection of holes by the next replica and by olympus

Running the test case:

1. Head on receiving the client request introduces a failure and increments the slot number. The next replica (replica number 1) detects this and calls for a reconfiguration. Olympus sends wedge request to replica 0 and 1 and detects the misbehavior of replica 0 while validating the histories. It then sends the wedge requests to a different quorum.
2. failures(0,0) ; failure trigger: client_request(0,2) ; action: increment_slot()

- **Scenario: 11**
  Send reconfiguration request if replicas time out while waiting for the result shuttle after client has sent the retransmission request.
  Send reconfiguration request if head time out while waiting for the result shuttle after client has sent the retransmission request.

  Running the test case:

1. When the client sends the first request, head crashes. Now clients keeps waiting and sends a retransmission request to all the replicas and the crashed head. The replicas other than the head, start their timer and forward the request to head. The head being in crashed state, doesn't respond, eventually the timers of the replicas expire and they send a reconfiguration request to olympus.
   After reconfiguration, the client sends a retransmission again. This time head is alive and it checks if the result cache has the result stored for this retransmitted request. The Result cache won't have the result, therefore it starts the timer and starts the operation from scratch. When the shuttle reaches the tail, the tail drops the incoming message. head does not receive the result, the timer expires and it sends a reconfig request to Olympus.
2. failures (0,0) ; failure trigger: client_request(0,1) ; action: crash()
3. failures (1,2) ; failure trigger: shuttle(0,1) ; action: drop()

- **Scenario: 12**
  Picking a different quorum in case catch up fails or doesn't arrive

  Running the test case:

1. When client request 3 arrives at head, it changes the operation. The next replica calls for reconfiguration. When the catch up messages arrive as a part of reconfiguration, replica number 1 drops the request. In this scenario olympus timeout after a while when it does not hear from replica 1. After timing out, olympus sends wedge requests to the next quorum and the process happens smoothly after this.
2. failure(0,0) ; failure trigger: client_request(0,3) ; action: change_operation()
   failure(0,1) ; failure trigger: catch_up(1) ; action: drop()

- **Scenario: 13**
  Detection of errors introduced when a new configuration arrives.

  Running the test case:

1. When a new configuration is received, replica executes drop() action. This gets detected when the client sends retransmission to replicas and they timeout. The replicas then call for a reconfig.
2. failure(0,0) failure trigger: client_request(0,2) ; action: change_operation()
   failure(1,1) failure trigger: new_configuration(1), drop()

- **Scenario: 15 MULTIHOST**
  Failure free scenario

- **Scenario: 16 MULTIHOST**
  Failure scenario (crash() failure)

  Running the test case:

2. When a new configuration is received, replica executes drop() action. This gets detected when the client sends retransmission to replicas and they timeout. The replicas then call for a reconfig.
2. failure(0,0) failure trigger: client_request(0,2) ; action: change_operation()
   failure(1,1) failure trigger: new_configuration(1), drop()

PERFORMANCE EVALUATION
(These test cases use 5 servers, 3 clients, and 300 requests/client.)

Raft (elapsed time - 10sec)

Running the program with configuration file perform900.txt in a single-host configuration.
(elapsed time - 52sec)

Running program with configuration file perform900.txt in a multiple-host configuration with replicas spread across multiple hosts, i.e., at least two different hosts contain replicas (clients and Olympus can be anywhere).
(elapsed time - 55sec)