**Testing**

**Correctness of dictionary** : client is maintaining its own dictionary, olympus is maintaining a global dictionary for all clients

When a result shuttle comes from tail node, client accepts the result if the result from the tail matches either his own dictionary result or the result in olympus’ dictionary.

**1) Send multiple requests from multiple clients and verify :**

**Config file** : config.txt

**Conditions to run the test case** : No fault injections

**Log files** : client0.log, client1.log, replica0.log, replica1.log, replica2.log, olympus.log

No. of clients -2   
No. of replicas -3  
  
[Workload here is mutually inclusive]

workload[0] = put('movie','star'); append('movie','star'); get('movie')  
workload[1] = put('movie','war'); get('movie')  
  
Sequence in which system runs  
1. Client 0 - sends put('movie','star')  
 Result in Client 0 dictionary - ‘OK’

Result in Global dictionary - ‘OK’  
 Result Status - Accepted  
2. Client 1 - sends put('movie','war')   
 Result in Client 1 dictionary - ‘OK’

Result in Global dictionary - ‘OK’  
 Result Status - Accepted  
3. Client 0 - sends append(‘movie’,’star’)  
 Result in Client 0 dictionary - ‘OK’

Result in Global dictionary - ‘OK’  
 Result Status - Accepted  
4. Client 0 - get(‘movie’)  
 Result in Client 0 dictionary - ‘starstar’

Result in Global dictionary - ‘warstar’  
 Result Status - ‘warstar’  
4. Client 1 - get(‘movie’)  
 Result in Client 1 dictionary - ‘war’

Result in Global dictionary - ‘warstar’  
 Result Status - Accepted

**Outcome** : Pass

**2) Fault Scenarios 1**

**Config file** : config.txt

**Conditions to run the test case** :

**Fault actions used:** change\_operation();change\_result();drop\_result\_stmt()  
**Fault triggers used:** client\_request(0,0) **Functionalities tested:   
 - Detecting inconsistency of order proof and sending reconfig to olympus**

**- Detecting holes in Replica history  
 - Retransmission of request by client  
 -**

**Log files** : client0.log, client1.log, replica0.log, replica1.log, replica2.log, olympus.log

No.of clients - 2  
No. of replicas - 3  
No. of Injections- 3  
  
[Workload here is mutually inclusive]

workload[0] = put('movie','star'); append('movie','star'); get('movie')  
workload[1] = put('movie','war'); get('movie')  
  
failures[0,0] = client\_request(0,0), change\_operation();client\_request(1,1), change\_result();client\_request(0,1), drop\_result\_stmt()  
  
Sequence in which system runs  
1. Client 0 - sends put('movie','star')

2. Head - receives the request

* Detects the failure - client\_request0, 0)
* Failure Action - change\_operation() to get(‘x’)
* Forwards the shuttle to Replica 1

3. Replica 1 - receives the signed statement

* **Detects inconsistency in order proof** - client operation and Head statement have different operations.
* Sends Reconfiguration request to olympus

4. Head - receives append('movie','star') from Client 0

* Performs the operation and send it to next replica

5. Replica 1- receives the shuttle for append(‘movie’,’star’)

* Replica **detects hole in history** since slot for this request is 1 and it didn’t perform the previous operation
* Replica sends a reconfiguration request to Olympus

6. Client 0 - Timer expires() - send a retransmission request to all replicas

7. Replicas

* No result found in cache
* Forwards the request to head and start the timer waiting for result shuttle
* Ultimately time expires and it sends reconfiguration request to olympus

8. Head

* No Result found in cache
* Operation is found in the history
* Start the timer and wait for result shuttle
* Ultimately time expires and it sends reconfiguration request to olympus

**Outcome** : Pass

**3) Fault Scenarios 2**

**Config file** : config.txt

**Fault actions used:** shuttle(0,1)  
**Fault triggers used:**change\_result() **Functionalities tested:   
 - Failure of result validation by client**

**Log files** : client0.log,, replica0.log, replica1.log, replica2.log, olympus.log

No.of clients - 1  
No. of replicas - 3  
No. of Injections- 1

workload[0] = put('movie','star'); get('movie'); append('movie','star')  
  
failures[0,2] = shuttle(0,1), change\_result()  
  
Sequence in which system runs  
1. Replica 2- receives the 1st shuttle for get(‘movie’)

* It changes result to ‘OK’

2. Client is expecting ‘’ as response, but gets OK as result.

* Hence it receives an unexpected result

**Outcome** : Pass

**4) Fault Scenarios 3**

**Config file** : config.txt

**Injections made:  
Fault actions used:**  result\_shuttle(0,1),  
**Fault triggers used:** drop\_result\_stmt() **Functionalities tested:   
 - Detection of dropped result statement**

**Log files** : client0.log,, replica0.log, replica1.log, replica2.log, olympus.log

No.of clients - 1  
No. of replicas - 3  
No. of Injections- 1

workload[0] = put('movie','star'); get('movie'); append('movie','star')  
  
failures[0,1] = result\_shuttle(0,1), drop\_result\_stmt()  
  
Sequence in which system runs  
1. Replica 1- receives the 2nd result shuttle

* It drops the statement

2. Head - upon receiving result shuttle, checks for validity of result proof.

* Since one of the the result statement is missing from the shuttle
* It sends reconfiguration to olympus

**Outcome** : Pass

**5) Stress Testing**

**Config file** : config.txt

**Functionalities tested:   
 - Stress testing with client generating 100 pseudorandom requests**

**Log files** : client0.log,client1.log,client2.log,client3.log,client4.log,client0.log,client5.log,client6.log,client7.log,client8.log,replica0.log, replica1.log, replica2.log, olympus.log

No.of clients - 9  
No. of replicas - 3

Outcome:  
1. Client 0 -  
 - All the operations are passed to head

* All the replicas complete the operation successfully
* Client logs produce evidence of 100 Accepted Results

**Outcome** : Pass