**Byzantine Chain Replication**

**PLATFORM :**

Python version : 3.6.3

DistAlgo : 1.0.9

Hosts : Laptops without VM

OS : MAC/Windows

Platform for multiple hosts : Olympus and Clients - Windows, Replicas - MAC  
  
**INSTRUCTIONS :**   
We have included 5 scenarios of test cases in separate folders.  
1. Place ping.da file inside each folder  
2. Each folder already has csv file needed to test the scenario  
3. Open cmd in folder location and give the following commands  
To run on single host:

OLYMPUS  
python -m da -n olympus -f --logdir ./ --logfilename olympus.log --logfilelevel info ping.da

REPLICA

python -m da -n Replica0 -D -f --logdir ./ --logfilename replica0.log --logfilelevel info ping.da

python -m da -n Replica1 -D -f --logdir ./ --logfilename replica1.log --logfilelevel info ping.da

python -m da -n Replica2 -D -f --logdir ./ --logfilename replica2.log --logfilelevel info ping.da

CLIENT

python -m da -n Client0 -D -f --logdir ./ --logfilename client0.log --logfilelevel info ping.da

python -m da -n Client1 -D -f --logdir ./ --logfilename client1.log --logfilelevel info ping.da

python -m da -n Client2 -D -f --logdir ./ --logfilename client2.log --logfilelevel info ping.da  
  
To run on multi host:

REPLICA  
python -m da -H 172.24.21.17 -R 172.24.18.144 -n Replica0 -D -f --logdir ./ --logfilename replica0.log --logfilelevel info ping.da

python -m da -H 172.24.21.17 -R 172.24.18.144 -n Replica1 -D -f --logdir ./ --logfilename replica1.log --logfilelevel info ping.da

python -m da -H 172.24.21.17 -R 172.24.18.144 -n Replica2 -D -f --logdir ./ --logfilename replica2.log --logfilelevel info ping.da

CLIENT

python -m da -H 172.24.18.144 -n Client0 -D -f --logdir ./ --logfilename client0.log --logfilelevel info ping.da

python -m da -H 172.24.18.144 -n Client1 -D -f --logdir ./ --logfilename client1.log --logfilelevel info ping.da

python -m da -H 172.24.18.144 -n Client2 -D -f --logdir ./ --logfilename client2.log --logfilelevel info ping.da

OLYMPUS

python -m da -H 172.24.18.144 -n olympus -f --logdir ./ --logfilename olympus.log --logfilelevel info ping.da

Here, I’m running 3 replicas on host - 172.24.21.17 (giving 172.24.18.144 as peer node)

olympus and 3 clients on another host - 172.24.18.144 (giving 172.24.21.17 as peer node)  
  
**WORKLOAD GENERATION :**

We have written this function ‘generatePseudoRandomRequests’ which generates the Pseudo Random workload for us. Based on the noOfRequests it picks up the specified number of requests from a set of requests stored in ‘listofRequest’ variable.

def generatePseudoRandomRequests(rSeed,noOfRequests):

output("--------------- Generating Pseudo Random Requests for Client --------------",level=20)

listofRequest = ["put('movie','star')","append('movie',' wars')","get('movie')","put('jedi','luke skywalker')","slice('jedi','0:4')","get('jedi')"]

random.seed(rSeed)

requests = random.sample(listofRequest,k=noOfRequests)

return requests

For seed value = 233 and no of requests = 5, as specified in the sample config file given to us.

The code picks up the same set of 5 requests from the list of requests everytime we set the seed value to 233 before retrieving those 5 requests into the ‘requests’ variable.

The we return the requests stored in the ‘request’ variable to the client. It then sequentially sends those requests to replicas.

**BUGS AND LIMITATIONS** **:**

We have implemented of fault\_injection:- forward\_request but were unable to test it due to lack of scenario to be able to test it.

**CONTRIBUTIONS :**

CLIENT

Generate pseudorandom workload with good diversity using specified seed - **Selina**

Generate request sequence specified in config file - **Dinesh**

Handle result: check signatures and hashes in result proof - **Dinesh and Selina**

Timeout and send request to all replicas if timely response not received - **Selina**

Check that dictionary contains expected content at end of test case - **Dinesh**

OLYMPUS

Create initial configuration: create keys, create, setup, and start processes - **Selina**

REPLICA

Dictionary object: support put, get, slice, append - **Dinesh**

Head: handle new request: assign slot, sign order stmt & result stmt, send shuttle - **Dinesh and Selina**

Head: handle retransmitted request as described in paper - **Dinesh**

Handle shuttle: check validity of order proof (incl. signatures), add signed order

statement and signed result statement, send updated shuttle tail: send result to client; send result shuttle to predecessor - **Dinesh and Selina**

Handle result shuttle: validate, save, and forward it - **Dinesh and Selina**

Non-head: handle request: send cached result, send error, or forward request - **Dinesh and Selina**

Fault-injection: required triggers - **Dinesh**

Fault-injection: required failures - **Dinesh**

MULTI-HOST EXECUTION

Processes are spread across multiple hosts - **Selina**

CONFIGURATION FILES

Support configuration files specified in project.txt - **Dinesh**

LOGS

Detailed and readable logs - **Dinesh**

DOCUMENTATION

README and testing.txt contain all information specified in project.txt - **Selina**

For the tasks that say - Dinesh and Selina. We have sat together and figured out the code.

**MAIN FILES :** our main file ping.da in ./src/ contains all the code for olympus client and replica  
  
**CODE SIZE :**

(1a) Algorithm (LOC) :710

Other (LOC) :190

Total (LOC) :900

(1b) We used CLOC (<https://github.com/AlDanial/cloc>) to count the number of blanks lines.

Then we used the find feature of sublime to calculate the number of comments (lines beginning with #) in it.  
 To calculate other functionalities: - We searched for keyword ‘output’ the total of which was 175.  
 To calculate fault injections: - We searched for all the fault actions(changeOperation(), dropStatement() and changeResult()). The total came out to be 17. For each fault action there are approximately 5 lines, bringing the total to 85

(2) Algorithm code (LOC):710

Other functionality interleaved with it (LOC) :260

**LANGUAGE FEATURE USAGE :**  
Numbers of list comprehensions : 5  
Dictionary comprehensions : 18  
Set comprehensions : 0   
Aggregations : 0  
Quantifications : 5  
  
**OTHER COMMENTS :**