Solving Binary Consensus Problem with Synchronous Paxos Algorithm

CS403/534 - Distributed Systems

14 January 2024

PA3 Solution Report

Ataollah Hosseinzadeh Fard
ID: 28610

1) At first have imported the needed libraries and defined the default IP address and first PORT. Also have defined the context of zmq as global so all of the processes will have it by default. Inside the main part, first I check if the number of arguments passed are correct or not. If less or more arguments given display an error and guide for correct usage. If the arguments passed are correct numbered and typed, it will create “numProc” many processes with “PaxosNode” function as target. And lastly, program will wait for all of the processes to finish.

```python
import sys
from multiprocessing import Process
import random
import zmq

IP = "127.0.0.1"
PORT = 5577
context = zmq.Context()

> def PaxosNode(ID, prob, N, val, numRounds):···

if __name__ == "__main__":
    if len(sys.argv) != 4:
        print("Usage: python paxos.py [NUM_NODES] [CRASH_PROB] [NUM_ROUNDS]")
        sys.exit(1)

    numProc = int(sys.argv[1])
    prob = float(sys.argv[2])
    numRounds = int(sys.argv[3])
    print(f"NUM NODES: {numProc}, CRASH PROB: {prob}, NUM ROUNDS: {numRounds}")

    nodes = []
    for i in range(numProc):
        initVal = random.choice([0, 1])
        newNode = Process(target=PaxosNode, args=(i, prob, numProc, initVal, numRounds,))
        newNode.start()
        nodes.append(newNode)

    for i in range(numProc):
        nodes[i].join()
```

2) “PaxosNode” function has 3 parts, first is the part that common functions that will do network operations. “send”, “sendFailure”, and “broadcastFailure”. Other than these functions, in this part the needed PULL and PUSH sockets are created using zmq. Lastly, the variables that will store the history of decisions are defined with default values. The important function among those 3, is the “send” because eventually other 2 functions use this function. The difference of this with others is that in others depending on the probability, randomly instead of message, a crash message may be sent.

```python
def PaxosNode(ID, prob, N, val, numRounds):
    def send(msg, sender, destination):
        pushes[destination].send_json({'msg': msg, 'id': sender})

    def sendFailure(msg, proposer, i, prob):
        crash_status = random.choices([True, False], weights=(prob, 1-prob), k=1)[0]

        if crash_status:
            send(f"CRASH {proposer}", proposer, i)
        else:
            send(msg, proposer, i)

    def broadcastFailure(msg, proposer, N, prob):
        for i in range(N):
            sendFailure(msg, proposer, i, prob)

    pull = context.socket(zmq.PULL)
    pull.bind(f"tcp://{IP}:{PORT+ID}")

    pushes = []
    for i in range(N):
        push = context.socket(zmq.PUSH)
        push.connect(f"tcp://{IP}:{PORT+i}")
        pushes.append(push)

    maxVotedRound = -1
    maxVotedVal = None
    proposeVal = None
    decision = None

    for currRound in range(numRounds):···
```

3) The part 2 is the part when the node is Leader of that round will run. Basically, at first it will "broadcastFailure(START)" and will wait for N responses. If number of responses is not more than "N/2" if will broadcast "ROUNDCHANGE" to every other node. If number of responses are more than "N/2" then it will propose the voted value of latest voted round unless this is the first round. If this is first round then it will propose its own value. And again, will wait for N responses. If number of successful votes are more than "N/2" then it will announce that he has decided has his proposal as decision. And the round ends whether it has changed round or decided on a value or not.

```
    for currRound in range(numRounds):
        if currRound % N == ID:
            print(f"ROUND {currRound} STARTED WITH INITIAL VALUE {val}")
            broadcastFailure("START", ID, N, prob)
            tmp_maxVotedRound = -1
            tmp_maxVotedVal = -1

            join_count = 0
            seenJoin = set()
            while len(seenJoin) != N:
                tmp = pull.recv_json()
                if not tmp["id"] in seenJoin:
                    seenJoin.add(tmp["id"])
                    newMsg = tmp["msg"]
                    print(f"LEADER OF {currRound} RECEIVED IN JOIN PHASE: {newMsg}")
                    if not "CRASH" in newMsg:
                        join_count += 1
                        if "JOIN" in newMsg:
                            parsedMsg = newMsg.split()
                            acc_maxVotedRound = int(parsedMsg[1])
                            if acc_maxVotedRound > -1:
                                acc_maxVotedVal = int(parsedMsg[2])
                                if tmp_maxVotedRound < acc_maxVotedRound:
                                    tmp_maxVotedRound = acc_maxVotedRound
                                    tmp_maxVotedVal = acc_maxVotedVal
                        else:
                            if tmp_maxVotedRound < maxVotedRound:
                                tmp_maxVotedRound = maxVotedRound
                                tmp_maxVotedVal = maxVotedVal

            if join_count <= N / 2:
                print(f"LEADER OF ROUND {currRound} CHANGED ROUND")
                for i in range(N):
                    if i != ID:
                        send("ROUNDCHANGE", ID, i)
            else:
                maxVotedRound = tmp_maxVotedRound
                maxVotedVal = tmp_maxVotedVal
                if maxVotedRound == -1:
                    maxVotedVal = val
                proposeVal = maxVotedVal
                print(f"LEADER OF ROUND {currRound} PROPOSES {proposeVal}")
                broadcastFailure(f"PROPOSE {proposeVal}", ID, N, prob)

                vote_count = 0
                seenVotes = set()
                while len(seenVotes) != N:
                    tmp = pull.recv_json()
                    if not tmp["id"] in seenVotes:
                        seenVotes.add(tmp["id"])
                        newMsg = tmp["msg"]
                        print(f"LEADER OF {currRound} RECEIVED IN JOIN VOTE: {newMsg}")
                        if "PROPOSE" in newMsg or "VOTE" in newMsg:
                            vote_count += 1

                if vote_count > N / 2:
                    decision = proposeVal
                    maxVotedRound = currRound
                    maxVotedVal = proposeVal
                    print(f"LEADER OF {currRound} DECIDED ON VALUE {decision}")
        else:
```

4) Last part is the code for node when it is an Acceptor. Basically, acceptor is inside an infinite loop until it receives a message. After receiving “START” it will send its own join message with “sendFailure” but if it receives a crash message it will send crash message back. Then it

waits for propose phase message, which can be either PROPOSE or CRASH or ROUNDCHANGE. If it is ROUNDCHANGE, nothing will happen and it will move to net round. If it is crash, it will send it back to Leader. If it is PROPOSE, then it will update its local history variables and “sendFailure(VOTE)”. In any of 3 cases now the round has ended.

```
100         else:
101             while True:
102                 startMsg = pull.recv_json()['msg']
103                 if len(startMsg) > 0:
104                     print(f"ACCEPTOR {ID} RECEIVED IN VOTE JOIN: {startMsg}")
105 
106                     if "CRASH" in startMsg:
107                         send(f"CRASH {currRound % N}", ID, currRound % N)
108                     else:
109                         sendFailure(f"JOIN {maxVotedRound} {maxVotedVal}", ID, currRound % N, prob)
110 
111                     while True:
112                         proposeMsg = pull.recv_json()["msg"]
113                         if len(proposeMsg) > 0:
114                             print(f"ACCEPTOR {ID} RECEIVED IN VOTE PHASE: {proposeMsg}")
115 
116                             if "PROPOSE" in proposeMsg:
117                                 maxVotedRound = currRound
118                                 maxVotedVal = int(proposeMsg.split()[1])
119                                 sendFailure("VOTE", ID, currRound % N, prob)
120                             elif "CRASH" in proposeMsg:
121                                 send(f"CRASH {currRound % N}", ID, currRound % N)
122                             break
123                     break
```