CSE535: Distributed Systems

Project: DiemBFT v4 Consensus Algorithm Phase 2

Team Name: Loyal Byzantine Generals

Vivek Neppalli

Manish Adkar

Shubham Sahu

Pseudocode

1. Validity checks for cryptographic values

```
//Signing Message using private key
Function sign_message(message, private_key)
    return private_key.sign(message)

//Verifying Message using public key of sender
Function verify_signature(message, signer)
    data = public_key[signer].verify(message)
    if data is not None:
        return True
    else:
        return False

//Verifying author signatures for QC, tmo_signatures for TC, and author signature
for Block
Function valid_signatures(message):
   all_signatures_valid <- True

   if message is Timeout Certificate then
       signatures <- message.tmo_signatures
       signers <- message.tmo_signers
       for i from 0 to |signatures| do
           is_signature_valid <- verify_signature(signatures[i], signers[i])
           all_signatures_valid <- all_signatures_valid and is_signature_valid
```

```
        return all_signatures_valid

    if message is Quorum Certificate then
        signatures <- message.signatures
        signers <- message.signers
        for i from 0 to |signatures| do
            is_signature_valid <- verify_signature(signatures[i], signers[i])
            all_signatures_valid <- all_signatures_valid and is_signature_valid
        return all_signatures_valid

    if message is Block then
        return verify_signature(message.author_signature, message.author)
```

2. "sync up" replicas that got behind

```
//How to know if replica is behind?
Procedure process_proposal_msg(P)
    if P.block.round >> Pacemaker.current_round then
        Broadcast SyncRequestMsg(P.block.round)

//Send data to replica that is behind
Procedure process_sync_request_msg(S)
    if S.block << Pacemaker.current_round then
        send SyncData(Ledger, BlockTree)

//Handle the sync data received from the replicas
Procedure process_sync_data(S)
    pending_sync_data ← pending_sync_data ∪ (S.Ledger, S.BlockTree)
    if |pending_sync_data| = 2f + 1 then
        update_block_tree(S.BlockTree) // Make a local copy of pending blocks
        update_ledger_state(S.Ledger)  // Copy the txns from the last locally
                                       // committed txn

//As the replica which is behind does not have an updated copy of the MemPool,  it
will keep on receiving new requests and adding into it's MemPool. Meanwhile, it
won't be able to create new proposals with transactions i.e. it would just propose
empty blocks for liveness.
```

3. client requests: de-duplication; include appropriate requests in proposals

```
At Validator:

request_cache  // Maintains a cache containing committed requests
can_start_new_round = true //Flag to check if new round is to be started
                           // request from client
replicas                // List containing all the validators
validator_id            // contains current validator's id
last_tc                 // stores the information about the last_tc
recently_commited_blocks // Contains blocks recently committed by LedgerModule

Event Listener: Wait for a request R

Procedure wait_for_request_message(R)
    if R is a request_message then process_request_message(request)

Procedure process_request_message(request)
    ...
    request_id  = request['request_id']
    // Return response if already exists in cache
    if request_id  in request_cache then
        response = request_cache[request_id ]
        return response to client
    else
    // Add to mempool start new round
        mempool.add_transaction(request)
        if can_start_new_round then
                process_new_round(last_tc)
                can_start_new_round = false
    ...

Procedure process_certificate_qc(qc)
    ...
    BlockTree.process_qc(qc)
    If |recently_commited_blocks| > 0 then
            foreach block in recently_commited_blocks
    foreach transaction in block.payload
                    request_id = transaction['request_id]
    response = {request_id : "success"}
    // Add response to request_cache
    request_cache['request_id '] = response
                recently_commited_blocks.clear()
    ...
```

```
BlockTree:
Procedure process_qc(qc)
    ...
    if qc.ledger commit info.commit state id != ⊥ then
        Ledger.commit(qc.vote info.parent id)
    ...

Ledger:

Recently_commited_blocks; // contains information of all committed blocks in a
Ledger.commit call

Procedure commit(block._id)
        ...
        Appends all the blocks committed during this call to
recently_commited_blocks which is in turn used by the validator to keep those in
its cache and respond according for a request to client
        ...


MemPool:

pending_transactions; // Stores all the pending requests that are to be executed

Procedure add_transaction(request)
    request_id = request['request_id']
    If request_id not in pending_transactions then
        pending_transactions[request_id] = request
```

4. client pseudocode: verify that a submitted command was committed to the ledger

```
At Client:
Event listener: wait for a Message M

Procedure wait_for_Response_Message(M)
        If M is a response_message then process_response_from_validator(response)
        If M is a timeout_message then wait_and_retransmit_request(request)

Procedure process_response_from_validator(response)
        Request_id <-- response[‘Request_id ’] // Clients gets the request_id for
which validator sends a response
        Req_Status <-- response['Response_status'] // status of the received
response either success or failure
        if Req_Status = "success" then
            pending_success_results[Request_id].add(response) // collected
responses for a request
                if |pending_success_results[req_id]| = f + 1 then
                        Received sufficient amount of responses back to confirm
that the request sent by client is executed
        else
                pending_fail_results[Request_id].append(response)
                if |pending_fail_results[Request_id]| = f + 1 then
                         Received sufficient amount of responses back to confirm
that the request sent by client is failed to execute

Procedure wait_and_retransmit_request(request)
        If |pending_success_results[Request_id ]| < f + 1 and
|pending_fail_results[Request_id]| < f + 1 then
                retransmit_request(request) // Retransmit the request again
                pending_success_results[Request_id ].clear() // Clear the list so
that it repopulate again
                pending_fail_results[Request_id].clear()
```