CS223 Project Report

Qi Lin, Ziying Yu, Xiaoying Li

June 9, 2022

1 Introduction

In this project, we implement data replication of transactions based on PostgreSQL. We replicate data to 3 nodes in which one is the leader, and the rest are replicas. A client processes transactions in an optimistic concurrency control way and makes a connection to the leader to send transactions. Section 2 mainly describes how we design the communication between a client and a leader, a leader than two replicas, which both ensure concurrency and maintain conflict serializability. Sections 3 and 4 introduce how we test our program and analyze the results.

2 Design

2.1 Modules

We implement our program in python, and it consists 4 modules: client.py, database.py, node.py and test.py.

We first sudo install postgresql, libpq-dev postgresql-client, postgresql-client-common, and get into the postgres interface. After getting into the interface, we create one user and three tables, tables 0, 1, and 2. Each node performs the transaction on its table, but all three tables should achieve the same results to ensure concurrency.

- In the node.py, the class Handler handles different messages and states, such as *RREQUEST* stands for READ request and *LOG_SYNC* stands for log sync. A leader handles two types of requests from a client, READ and COMMIT.
  - The asynchronous function *get_read_request* handles the read request from the client. This request is stored in a JSON file with the *operation*, *client*, and *transaction_index*. If it is the first time we see the transaction, we initialize a new serial cache database and start to read the data by *read_phase*. *read_phase*, means that we read the data into the serial cache database. Once we finished reading, we stored the transaction index as key value in a dict and returned the result from the serial cache database. If it is not the first time we see the transaction, we produce the read result directly on the related serial cache database, since this serial cache database has been initialized before and linked with this transaction. The return message contains *leader*, *transaction_index*, *result*, *type*, and *operation*. Noticeably, we use the web application to handle all kinds of messages instead of threads to ensure each message is independent of the other.
  - The asynchronous function *get_commit_request* handles the commit request from the client. We will introduce the detail of commit process in the Section 2.2
  - The asynchronous function *log_sync* handles the log synchronize.

- In the client.py, we also use the web application to handle all kinds of messages and different functions to manage reply messages from nodes.

- In the database.py, we defined a simple database to connect with the postgresql database and handle the execution of transactions.

- In the test.py, we defined some test cases, and we will introduce the test result of our system in Section 4.

2.2 Optimistic Concurrency Control

A READ request process:

Upon receiving the read request from the client, we denote it as $\mathbf{Tn}_{Rx}$, **Tn** means transaction n, **Rx** means read operation on variable x.

Here we need to check if this read operation is the first operation in transaction n:

- If yes, we need to construct a new cache database to store every operation in transaction n; mark the start number of this cache database as the latest sequence number; here, the sequence number denotes the number of committed transactions. For example, if we already committed n transactions, the sequence number is n+1, so the start number of this cache database is n+1.
- If no, there existed a cache database storing previous transactions in transaction n. We need to execute this operation in this existed cache database.

When executing the read operation, we first define such read operation as a function callable, then specify a read phase to begin this operation. This style will make every operation (read or write) can have a common execute processing, making our code more readable and easily understood.

After executing the read operation on variable x based on the new cache database or existing cache database, we construct the reply message, which contains the value of variable x, and send it back to the client.

The read request process in the end.

A COMMIT request process:

Upon receiving the commit request from the client, we denote it as $\mathbf{Tn}_C$. **Tn** means transaction n, this request will provide a read set and write set of this transaction, we denote it as $\mathbf{RS}_n$, $\mathbf{WS}_n$.

Here we need to check if this commit operation is the first operation in transaction n:

- If yes, we need to construct a new cache database to execute $\mathbf{WS}_n$ in transaction n, and mark the start number of this cache database as the latest sequence number.
- If no, there existed a cache database that store previous operations in transaction n, we need to execute $\mathbf{WS}_n$ in this existed cache database.

As mentioned in the read request process, we define a common execute processing to execute $\mathbf{WS}_n$ in transaction n, and execute $\mathbf{WS}_n$ on the new cache database or existing cache database according to the previous check result.

After executing the operation on the cache database, we need to determine if this transaction can be committed or aborted. In our design, we used the optimistic concurrent control strategy, we need to check the read set of this transaction $\mathbf{RS}_n$ and write sets of other committed transactions.

Here is the processing:

- First, we should determine which transaction should be compared, as mentioned before, we used the start number of the cache database to denote the start time of each transaction, suppose the start number of transaction n is **tn**, the commit request for transaction n is received at a later time, we denote it as **tnc**, which is the latest sequence number of committed transaction, all committed transactions happened in the scope of **tn** and **tnc** should be compared. Thus, we construct a compare transaction set. For example, if the cache database for transaction n starts from n+1, which means we already have n transactions committed, and the commit request for transaction n is received at n+5. All transactions commit after n and before n+5 should be considered.

- Second, after constructing a compare transaction set, we need to compare the read set of transaction n ($\mathbf{RS}_n$) with the write sets of the compare transaction set, if these two sets disjoint, then we can commit transaction n, otherwise, we should abort transaction n, since some variables that read by transaction n has been changed by other transaction.
- Third, if we can commit transaction n, we should write the cache database into the PostgreSQL database, here we leveraged the write set of transaction n ($\mathbf{WS}_n$) to write the change on the persistence database. If we abort transaction n, we should clear the cache database for transaction n.
- In the end, after finishing the commit or abort operations, we need to construct the reply message and send it back to the client, which contains the commit or the abort information about the transaction n.

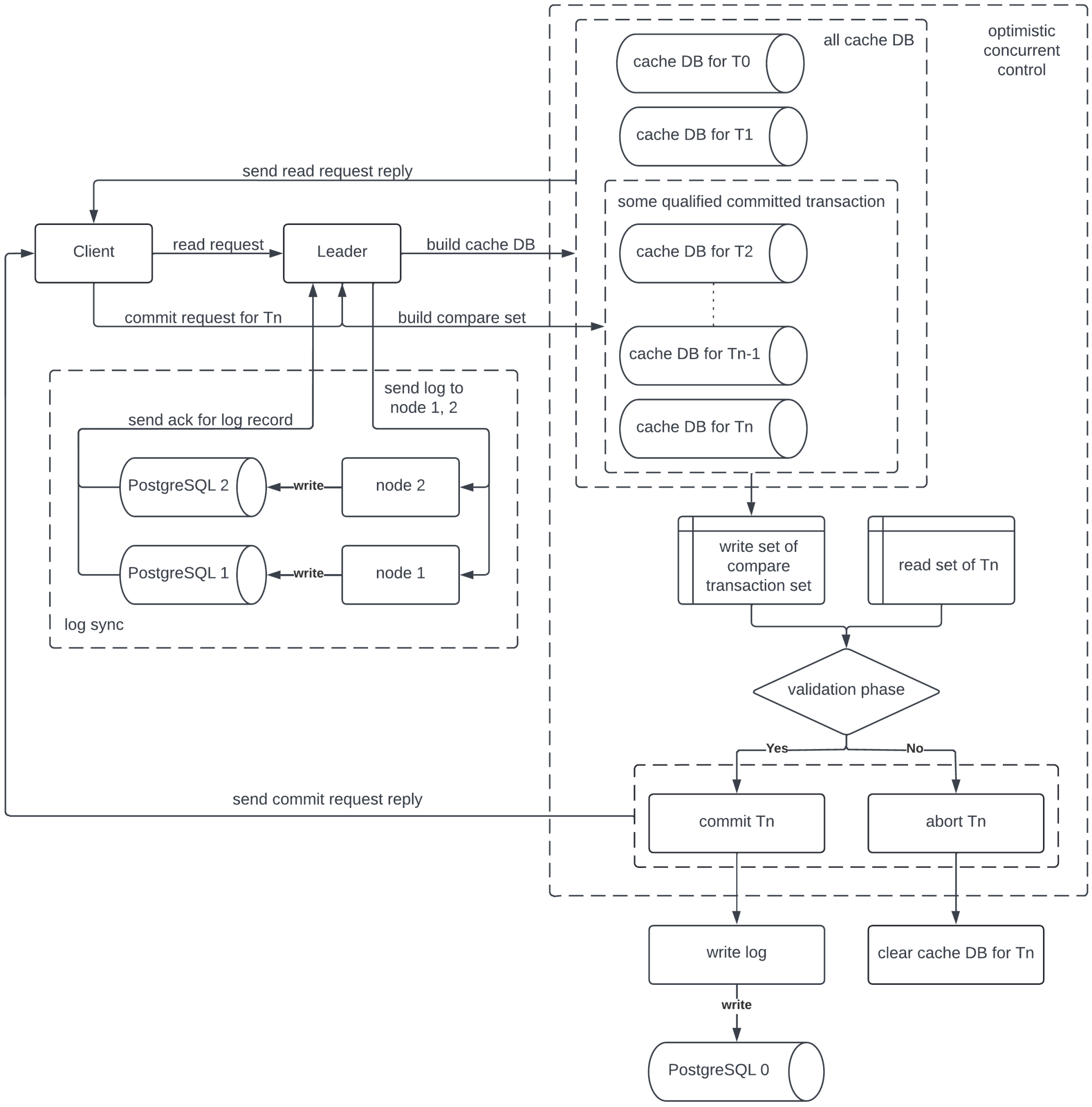


Figure 1: overall design diagram

2.3 Read-Only Transaction

For the leader node, the algorithm to process read-only transactions while maintaining conflict serializability is just like the algorithm to process common transactions. While for secondary node 1 and node 2, things are different. Below is our algorithm.

When a secondary node receives a read-only transaction, its log contains the newest active set showing active transactions' write set and read set. Check whether the write set and read set of the read-only transaction are disjoint. If they are not disjoint, then the read-only transaction is reading something that is being modified now, which may lead to conflict, and the read-only transaction should be aborted. If they are disjoint, then the read-only transaction can be executed on the related serial cache database.

The algorithm in our project uses the validation-based protocol. To prove that conflict serializability is maintained in our system, we need to prove that a schedule generated by validation protocol is conflict serializability. We already know that a schedule allowed by the two phase locking protocol is conflict serializability, so if we can prove every schedule that a validation protocol can generate is allowed by the 2PL protocol, then the schedule generated by the validation protocol is also conflict serializability. Below is the explanation:

Consider a validation schedule S, each transaction T consists of three phases:

- Read phase: transactions read data from database and update local copy.
- Validation phase: check if transactions can apply its updates to database without causing inconsistency.
- Write phase: copy its changes to database.

For each transaction T in S insert locks and unlocks to get S':

- Read phase (when T starts): request S-lock for all read data items of T.
- Validation phase (when T validates): request X-lock for all write data items of T; release S-locks for read-only data items.
- Write phase (when T ends): release all X-locks.

In order to show every schedule that a validation protocol can generate is allowed by 2PL protocol, we need to show that the above schedule S' must be allowed by 2PL protocol. Let's prove it by contradiction.

Suppose S' is not allowed by 2PL protocol due to write-read conflict:

S': ... L1(x) w2(x) r1(x) validate1 U1(x) ...

→ At validation 1: T2 has finished write phase, T1 can observe the change made by T2.
→ T1 cannot validate, since T1 read from some data items written by T2.
→ Contradict to the assumption that schedule S is validation schedule.

Suppose S' is not allowed by 2PL protocol due to write-write conflict:

S': ... validate1 L1(x) w2(x) w1(x) U1(x) ...

→ Suppose T2 validates first.
→ At validation 1: T2 is also at validation phase.
→ T2 has not finished yet, T1 cannot validate, since T1 overwrites some data items written by T2.
→ Contradict to the assumption that schedule S is validation schedule.

Hence, we proved that every schedule that a validation protocol can generate is allowed by 2PL protocol, so the schedule generated by the validation protocol is also conflict serializability. Therefore, conflict serializability is maintained in our system.

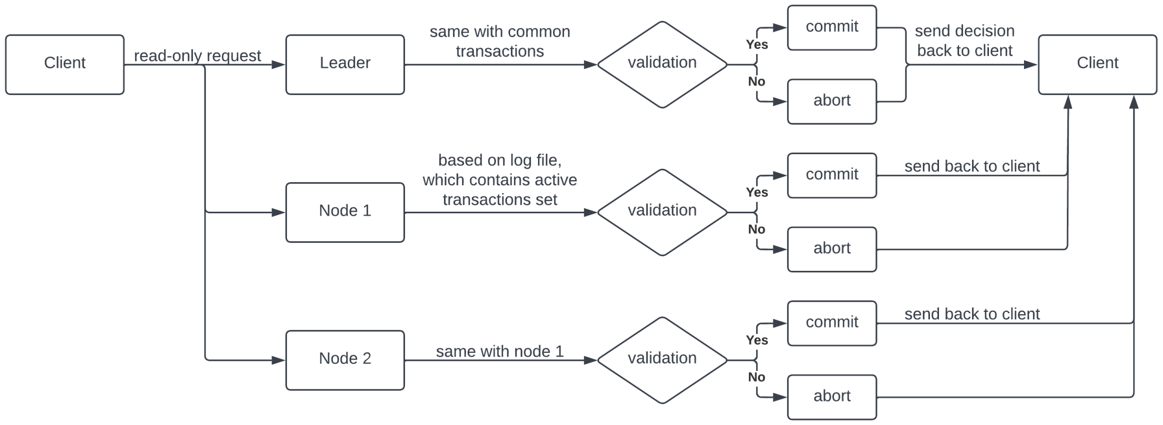


Figure 2: read-only transaction process algorithm design diagram

2.4 Leader Replacement

For the leader replacement, we let the client send a message to node 1 and node 2 to inform them of the leader replacement. Once node 1 and node 2 receive the leader replacement message, they will send each other a message asking about the other node's replication log length and send back its log length to the other node. Compare the two node's log lengths, and the node with the longer log length becomes the new leader; if their log lengths are the same, then set node 1 as the new leader. Then the new leader will send a message to inform the client that it is the new leader. Once the client receives the message from the new leader, it will ask the new leader about the state of transactions on its waitlist to see which transactions have been committed, and transactions that have not been committed should be aborted.

After checking the log file, the new leader can reply to the state of transactions and recover a consistent state by aborting or committing transactions based on the log file.

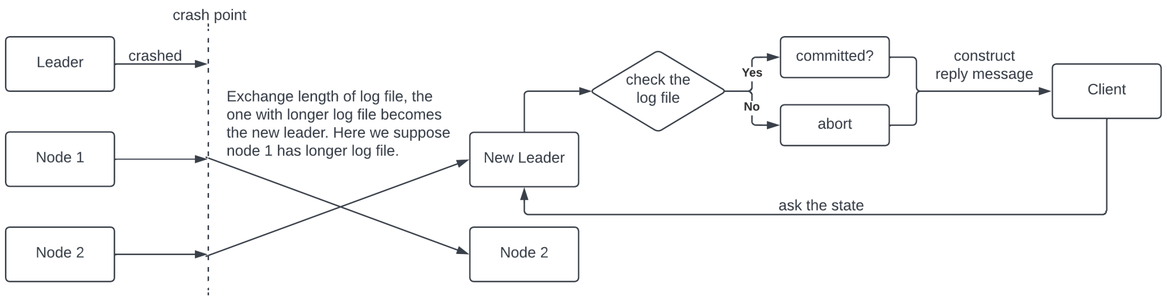


Figure 3: leader replacement algorithm design diagram

A Appendix: Test

We design 6 tests in the test.py module:

- Test1 contains a single transaction test: {T0: read a, write b=1, read c, commit}.
- Test2 contains 2 concurrent disjoint transactions:{T1: read a , write b=2, read c, commit; T2: read a, write d=1, read c, commit}.

- Test3 contains 2 concurrent joint transactions:{T3: read a , write b=3, read c, commit; T4: read c, write a=1, read b, commit}.
- Test4 is the replication log test which contains a single transaction: {read a, write b=1, read c, commit}.
- Test5 contains the read only request: {T5: read a}.
- Test6 is the leader change test. Here we simulate the leader change case by sending the leader change request to node 1 and node 2.

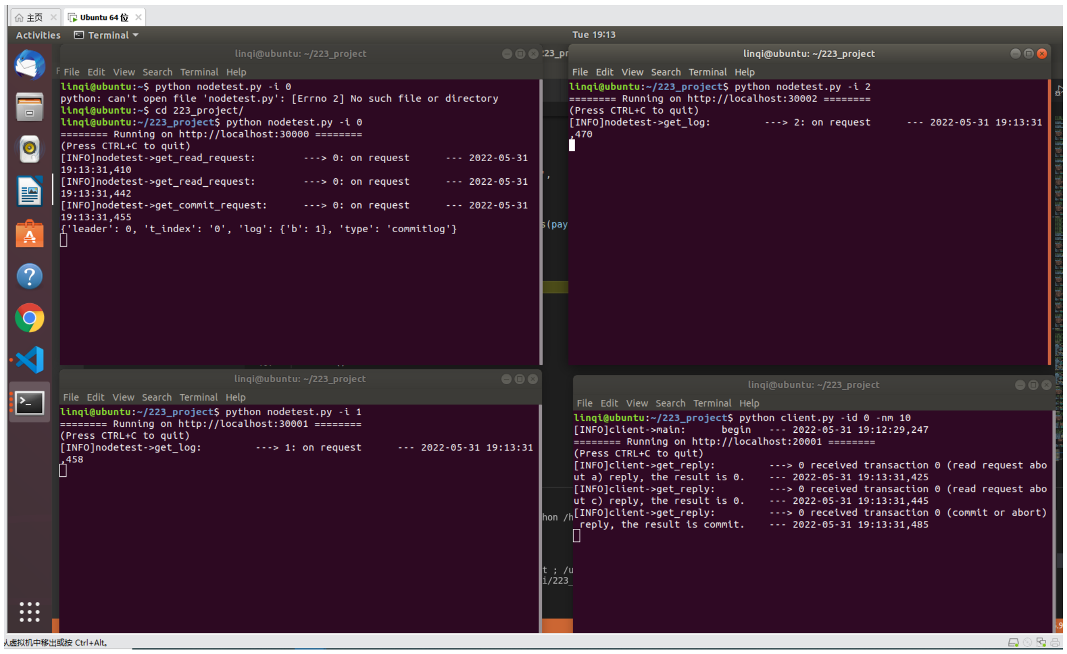
B Appendix: Result

We are going to run client, node 0, node 1, and node 2 on 4 different terminal.

In Test1:

- the client get 3 reply from leader:
  - receive transaction 0(read request about a) reply, the result is 0.
  - receive transaction 0(read request about c) reply, the result is 0.
  - receive transaction 0(commit or abort) reply, the result is commit.
- node 0, the leader, get 3 message from client:
  - get read request: read a.
  - get read request: read c.
  - get commit request: commit.
- both node1 and node 2 get one log from the leader: b = 1

The test ran successfully and met our expectations since all operations within the transaction have been communicated successfully between the client and 3 nodes.



```
linqi@ubuntu:~$ python nodetest.py -i 0
python: can't open file 'nodetest.py': [Errno 2] No such file or directory
linqi@ubuntu:~$ cd 223_project/
linqi@ubuntu:~/223_project$ python nodetest.py -i 0
======== Running on http://localhost:30000 ========
(Press CTRL+C to quit)
[INFO]nodetest->get_read_request:       ---> 0: on request      --- 2022-05-31
19:13:31,410
[INFO]nodetest->get_read_request:       ---> 0: on request      --- 2022-05-31
19:13:31,442
[INFO]nodetest->get_commit_request:     ---> 0: on request      --- 2022-05-31
19:13:31,455
{'leader': 0, 't_index': '0', 'log': {'b': 1}, 'type': 'commitlog'}
```

```
linqi@ubuntu:~/223_project$ python nodetest.py -i 2
======== Running on http://localhost:30002 ========
(Press CTRL+C to quit)
[INFO]nodetest->get_log:         ---> 2: on request      --- 2022-05-31 19:13:31
,470
```

```
linqi@ubuntu:~/223_project$ python nodetest.py -i 1
======== Running on http://localhost:30001 ========
(Press CTRL+C to quit)
[INFO]nodetest->get_log:         ---> 1: on request      --- 2022-05-31 19:13:31
,458
```

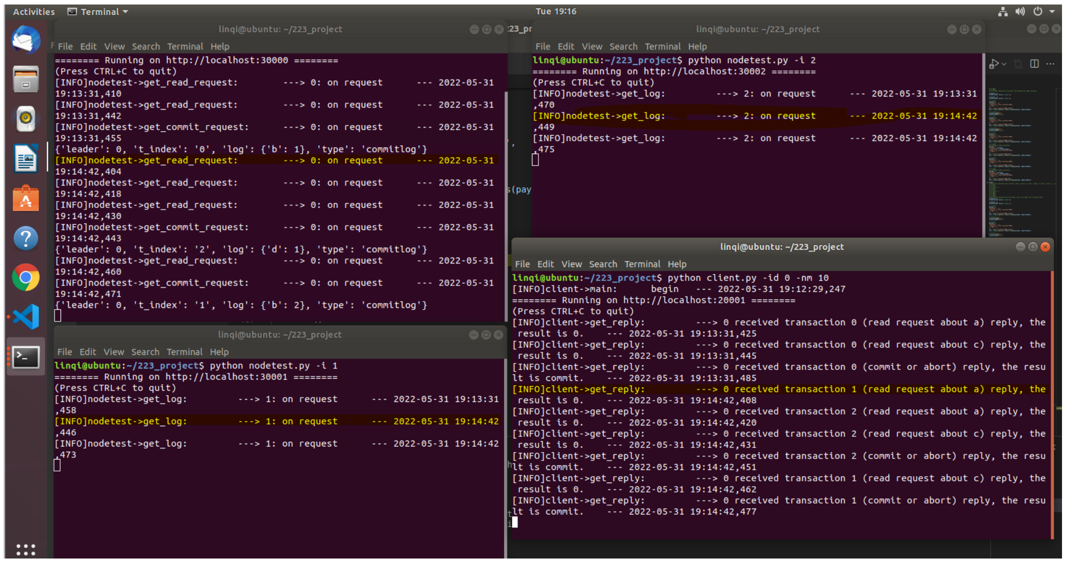
```
linqi@ubuntu:~/223_project$ python client.py -id 0 -nm 10
[INFO]client->main:      begin   --- 2022-05-31 19:12:29,247
======== Running on http://localhost:20001 ========
(Press CTRL+C to quit)
[INFO]client->get_reply:        ---> 0 received transaction 0 (read request abo
ut a) reply, the result is 0.    --- 2022-05-31 19:13:31,425
[INFO]client->get_reply:        ---> 0 received transaction 0 (read request abo
ut c) reply, the result is 0.    --- 2022-05-31 19:13:31,445
[INFO]client->get_reply:        ---> 0 received transaction 0 (commit or abort)
 reply, the result is commit.    --- 2022-05-31 19:13:31,485
```

Figure 4: Test1 Result

In Test2:

- the client get 6 reply from leader:
  - receive transaction 1(read request about a) reply, the result is 0.
  - receive transaction 2(read request about a) reply, the result is 0.
  - receive transaction 2(read request about c) reply, the result is commit.
  - receive transaction 2(commit or abort) reply, the result is commit.
  - receive transaction 1(read request about c) reply, the result is 0.
  - receive transaction 1(commit or abort) reply, the result is commit.
- node 0, the leader, get 6 message from client:
  - get read request: read a.
  - get read request: read a.
  - get read request: read c.
  - get commit request: commit.
  - get read request: read c.
  - get commit request: commit.
- both node1 and node 2 get two log from the leader: d = 1, b = 2

The test ran successfully and met our expectations since all two transactions are disjoint, and all two transactions have been communicated successfully between the client and 3 nodes.



```
======== Running on http://localhost:30000 ========
(Press CTRL+C to quit)
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:13:31,410
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:13:31,442
[INFO]nodetest->get_commit_request:      ---> 0: on request      --- 2022-05-31
19:13:31,455
{'leader': 0, 't_index': '0', 'log': {'b': 1}, 'type': 'commitlog'}
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:14:42,404
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:14:42,418
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:14:42,430
[INFO]nodetest->get_commit_request:      ---> 0: on request      --- 2022-05-31
19:14:42,443
{'leader': 0, 't_index': '2', 'log': {'d': 1}, 'type': 'commitlog'}
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:14:42,460
[INFO]nodetest->get_commit_request:      ---> 0: on request      --- 2022-05-31
19:14:42,471
{'leader': 0, 't_index': '1', 'log': {'b': 2}, 'type': 'commitlog'}
```

```
linqi@ubuntu:~/223_project$ python nodetest.py -i 1
======== Running on http://localhost:30001 ========
(Press CTRL+C to quit)
[INFO]nodetest->get_log:         ---> 1: on request      --- 2022-05-31 19:13:31
,458
[INFO]nodetest->get_log:         ---> 1: on request      --- 2022-05-31 19:14:42
,446
[INFO]nodetest->get_log:         ---> 1: on request      --- 2022-05-31 19:14:42
,473
```

```
linqi@ubuntu:~/223_project$ python nodetest.py -i 2
======== Running on http://localhost:30002 ========
(Press CTRL+C to quit)
[INFO]nodetest->get_log:         ---> 2: on request      --- 2022-05-31 19:13:31
,470
[INFO]nodetest->get_log:         ---> 2: on request      --- 2022-05-31 19:14:42
,449
[INFO]nodetest->get_log:         ---> 2: on request      --- 2022-05-31 19:14:42
,475
```

```
linqi@ubuntu:~/223_project$ python client.py -id 0 -nm 10
[INFO]client->main:      begin   --- 2022-05-31 19:12:29,247
======== Running on http://localhost:20001 ========
(Press CTRL+C to quit)
[INFO]client->get_reply:         ---> 0 received transaction 0 (read request about a) reply, the
 result is 0.    --- 2022-05-31 19:13:31,425
[INFO]client->get_reply:         ---> 0 received transaction 0 (read request about c) reply, the
 result is 0.    --- 2022-05-31 19:13:31,445
[INFO]client->get_reply:         ---> 0 received transaction 0 (commit or abort) reply, the resu
lt is commit.    --- 2022-05-31 19:13:31,485
[INFO]client->get_reply:         ---> 0 received transaction 1 (read request about a) reply, the
 result is 0.    --- 2022-05-31 19:14:42,408
[INFO]client->get_reply:         ---> 0 received transaction 2 (read request about a) reply, the
 result is 0.    --- 2022-05-31 19:14:42,420
[INFO]client->get_reply:         ---> 0 received transaction 2 (read request about c) reply, the
 result is 0.    --- 2022-05-31 19:14:42,431
[INFO]client->get_reply:         ---> 0 received transaction 2 (commit or abort) reply, the resu
lt is commit.    --- 2022-05-31 19:14:42,451
[INFO]client->get_reply:         ---> 0 received transaction 1 (read request about c) reply, the
 result is 0.    --- 2022-05-31 19:14:42,462
[INFO]client->get_reply:         ---> 0 received transaction 1 (commit or abort) reply, the resu
lt is commit.    --- 2022-05-31 19:14:42,477
```

Figure 5: Test2 Result

In Test3:

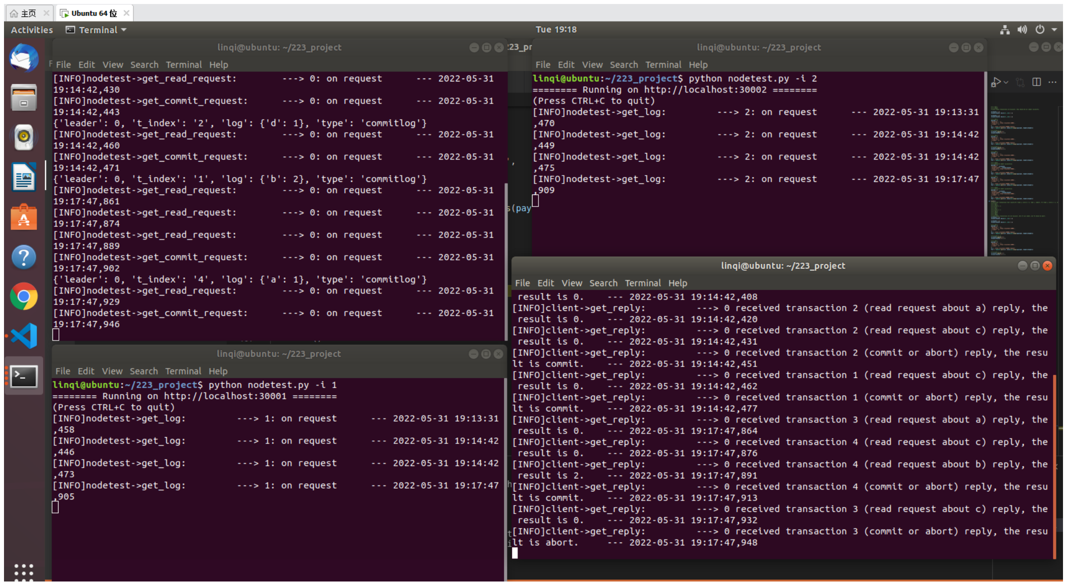
- the client get 6 reply from leader:
  - receive transaction 3(read request about a) reply, the result is 0.
  - receive transaction 4(read request about c) reply, the result is 0.
  - receive transaction 4(read request about b) reply, the result is 2.
  - receive transaction 4(commit or abort) reply, the result is commit.

  - receive transaction 3(read request about c) reply, the result is 0.
  - receive transaction 3(commit or abort) reply, the result is abort.

- node 0, the leader, get 6 message from client:
  - get read request: read a.
  - get read request: read c.
  - get read request: read b.
  - get commit request: commit.
  - get read request: read c.
  - get commit request: abort.

- both node1 and node 2 get one log from the leader: a = 1

The test ran successfully and met our expectations. Even though two transactions are joint, both transactions have been communicated successfully between the client and 3 nodes.



主页
Ubuntu 64 位
Activities
Terminal
Tue 19:18
linqi@ubuntu: ~/223_project
File Edit View Search Terminal Help

```
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:14:42,430
[INFO]nodetest->get_commit_request:      ---> 0: on request      --- 2022-05-31
19:14:42,443
{'leader': 0, 't_index': '2', 'log': {'d': 1}, 'type': 'commitlog'}
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:14:42,460
[INFO]nodetest->get_commit_request:      ---> 0: on request      --- 2022-05-31
19:14:42,471
{'leader': 0, 't_index': '1', 'log': {'b': 2}, 'type': 'commitlog'}
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:17:47,861
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:17:47,874
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:17:47,889
[INFO]nodetest->get_commit_request:      ---> 0: on request      --- 2022-05-31
19:17:47,902
{'leader': 0, 't_index': '4', 'log': {'a': 1}, 'type': 'commitlog'}
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:17:47,929
[INFO]nodetest->get_commit_request:      ---> 0: on request      --- 2022-05-31
19:17:47,946
```

```
linqi@ubuntu:~/223_project$ python nodetest.py -i 1
======== Running on http://localhost:30001 ========
(Press CTRL+C to quit)
[INFO]nodetest->get_log:         ---> 1: on request      --- 2022-05-31 19:13:31
,458
[INFO]nodetest->get_log:         ---> 1: on request      --- 2022-05-31 19:14:42
,446
[INFO]nodetest->get_log:         ---> 1: on request      --- 2022-05-31 19:14:42
,473
[INFO]nodetest->get_log:         ---> 1: on request      --- 2022-05-31 19:17:47
,905
```

```
linqi@ubuntu:~/223_project$ python nodetest.py -i 2
======== Running on http://localhost:30002 ========
(Press CTRL+C to quit)
[INFO]nodetest->get_log:         ---> 2: on request      --- 2022-05-31 19:13:31
,470
[INFO]nodetest->get_log:         ---> 2: on request      --- 2022-05-31 19:14:42
,449
[INFO]nodetest->get_log:         ---> 2: on request      --- 2022-05-31 19:14:42
,475
[INFO]nodetest->get_log:         ---> 2: on request      --- 2022-05-31 19:17:47
,909
```

```
 result is 0.    --- 2022-05-31 19:14:42,408
[INFO]client->get_reply:         ---> 0 received transaction 2 (read request about a) reply, the
 result is 0.    --- 2022-05-31 19:14:42,420
[INFO]client->get_reply:         ---> 0 received transaction 2 (read request about c) reply, the
 result is 0.    --- 2022-05-31 19:14:42,431
[INFO]client->get_reply:         ---> 0 received transaction 2 (commit or abort) reply, the resu
lt is commit.    --- 2022-05-31 19:14:42,451
[INFO]client->get_reply:         ---> 0 received transaction 1 (read request about c) reply, the
 result is 0.    --- 2022-05-31 19:14:42,462
[INFO]client->get_reply:         ---> 0 received transaction 1 (commit or abort) reply, the resu
lt is commit.    --- 2022-05-31 19:14:42,477
[INFO]client->get_reply:         ---> 0 received transaction 3 (read request about a) reply, the
 result is 0.    --- 2022-05-31 19:17:47,864
[INFO]client->get_reply:         ---> 0 received transaction 4 (read request about c) reply, the
 result is 0.    --- 2022-05-31 19:17:47,876
[INFO]client->get_reply:         ---> 0 received transaction 4 (read request about b) reply, the
 result is 2.    --- 2022-05-31 19:17:47,891
[INFO]client->get_reply:         ---> 0 received transaction 4 (commit or abort) reply, the resu
lt is commit.    --- 2022-05-31 19:17:47,913
[INFO]client->get_reply:         ---> 0 received transaction 3 (read request about c) reply, the
 result is 0.    --- 2022-05-31 19:17:47,932
[INFO]client->get_reply:         ---> 0 received transaction 3 (commit or abort) reply, the resu
lt is abort.     --- 2022-05-31 19:17:47,948
```

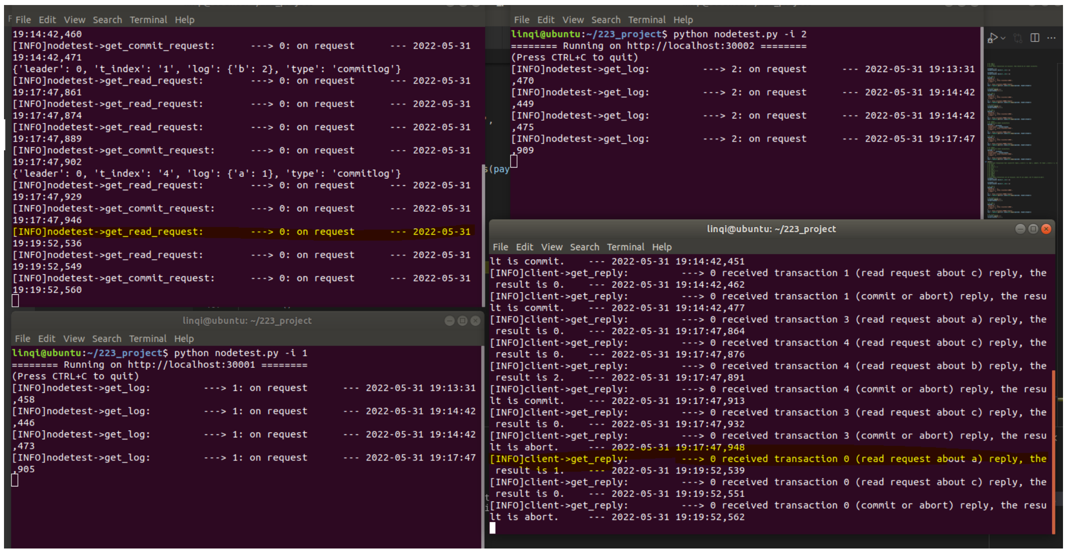
Figure 6: Test3 Result

In Test4:

- the client get 3 reply from leader:
  - receive transaction 0(read request about a) reply, the result is 0.
  - receive transaction 0(read request about c) reply, the result is 0.
  - receive transaction 0(commit or abort) reply, the result is abort.

- node 0, the leader, get 3 message from client:
  - get read request: read a.
  - get read request: read c.
  - get commit request: abort.

- All three table0, 1, 2 print out the same information b 2, proving that the replication log works successfully.



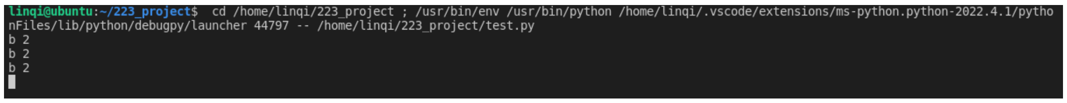
```
19:14:42,460
[INFO]nodetest->get_commit_request:      ---> 0: on request      --- 2022-05-31
19:14:42,471
{'leader': 0, 't_index': '1', 'log': {'b': 2}, 'type': 'commitlog'}
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:17:47,861
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:17:47,874
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:17:47,889
[INFO]nodetest->get_commit_request:      ---> 0: on request      --- 2022-05-31
19:17:47,902
{'leader': 0, 't_index': '4', 'log': {'a': 1}, 'type': 'commitlog'}
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:17:47,929
[INFO]nodetest->get_commit_request:      ---> 0: on request      --- 2022-05-31
19:17:47,946
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:19:52,536
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:19:52,549
[INFO]nodetest->get_commit_request:      ---> 0: on request      --- 2022-05-31
19:19:52,560
```

```
linqi@ubuntu:~/223_project$ python nodetest.py -i 1
======== Running on http://localhost:30001 ========
(Press CTRL+C to quit)
[INFO]nodetest->get_log:         ---> 1: on request      --- 2022-05-31 19:13:31
,458
[INFO]nodetest->get_log:         ---> 1: on request      --- 2022-05-31 19:14:42
,446
[INFO]nodetest->get_log:         ---> 1: on request      --- 2022-05-31 19:14:42
,473
[INFO]nodetest->get_log:         ---> 1: on request      --- 2022-05-31 19:17:47
,905
```

```
linqi@ubuntu:~/223_project$ python nodetest.py -i 2
======== Running on http://localhost:30002 ========
(Press CTRL+C to quit)
[INFO]nodetest->get_log:         ---> 2: on request      --- 2022-05-31 19:13:31
,470
[INFO]nodetest->get_log:         ---> 2: on request      --- 2022-05-31 19:14:42
,449
[INFO]nodetest->get_log:         ---> 2: on request      --- 2022-05-31 19:14:42
,475
[INFO]nodetest->get_log:         ---> 2: on request      --- 2022-05-31 19:17:47
,909
```

```
lt is commit.    --- 2022-05-31 19:14:42,451
[INFO]client->get_reply:         ---> 0 received transaction 1 (read request about c) reply, the
 result is 0.    --- 2022-05-31 19:14:42,462
[INFO]client->get_reply:         ---> 0 received transaction 1 (commit or abort) reply, the resu
lt is commit.    --- 2022-05-31 19:14:42,477
[INFO]client->get_reply:         ---> 0 received transaction 3 (read request about a) reply, the
 result is 0.    --- 2022-05-31 19:17:47,864
[INFO]client->get_reply:         ---> 0 received transaction 4 (read request about c) reply, the
 result is 0.    --- 2022-05-31 19:17:47,876
[INFO]client->get_reply:         ---> 0 received transaction 4 (read request about b) reply, the
 result is 2.    --- 2022-05-31 19:17:47,891
[INFO]client->get_reply:         ---> 0 received transaction 4 (commit or abort) reply, the resu
lt is commit.    --- 2022-05-31 19:17:47,913
[INFO]client->get_reply:         ---> 0 received transaction 3 (read request about c) reply, the
 result is 0.    --- 2022-05-31 19:17:47,932
[INFO]client->get_reply:         ---> 0 received transaction 3 (commit or abort) reply, the resu
lt is abort.     --- 2022-05-31 19:17:47,948
[INFO]client->get_reply:         ---> 0 received transaction 0 (read request about a) reply, the
 result is 1.    --- 2022-05-31 19:19:52,539
[INFO]client->get_reply:         ---> 0 received transaction 0 (read request about c) reply, the
 result is 0.    --- 2022-05-31 19:19:52,551
[INFO]client->get_reply:         ---> 0 received transaction 0 (commit or abort) reply, the resu
lt is abort.     --- 2022-05-31 19:19:52,562
```

Figure 7: Test4 Result



```
linqi@ubuntu:~/223_project$  cd /home/linqi/223_project ; /usr/bin/env /usr/bin/python /home/linqi/.vscode/extensions/ms-python.python-2022.4.1/pytho
nFiles/lib/python/debugpy/launcher 44797 -- /home/linqi/223_project/test.py
b 2
b 2
b 2
```

Figure 8: Test4 Replication Log Result

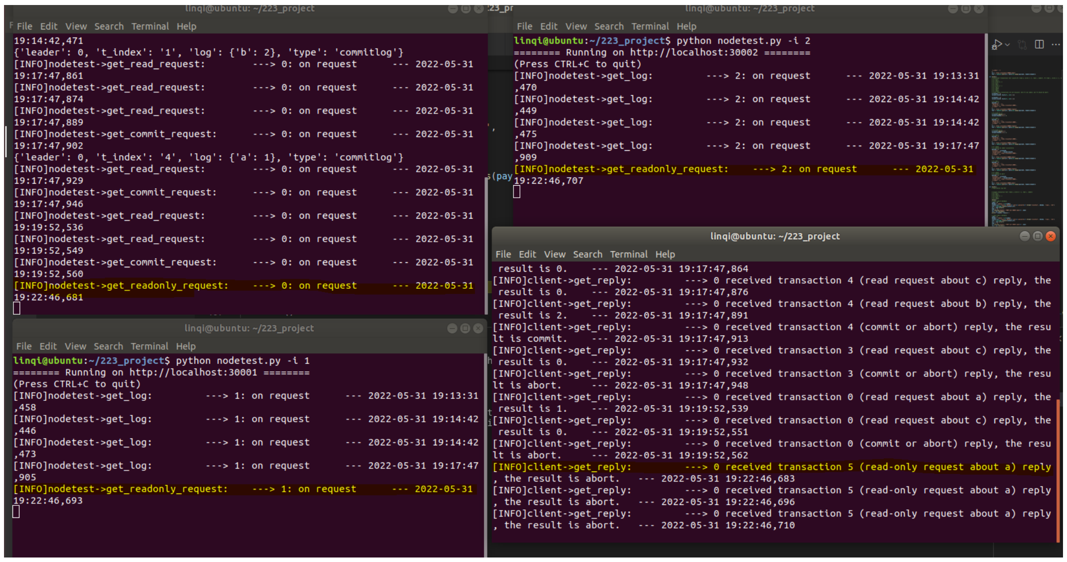
In Test5:

- the client get 3 reply from leader:
  - receive transaction 5(read-only request about a) reply, the result is abort.
  - receive transaction 5(read-only request about a) reply, the result is abort.
  - receive transaction 5(read-only request about a) reply, the result is abort.
- node 0, the leader, get 1 message from client:
  - get read-only request: read a.
- both node1 and node 2 get one read-only request: read a.

The test successfully met our expectations; all three nodes only got the read-only request.

In Test6:

- the client get 3 reply from leader:
  - received leader change message, the new leader is 1.
  - received transaction 0 (state) reply, the result is commit.
  - received transaction 1 (state) reply, the result is commit.
- node 0, now receives nothing.
- node 1, the new leader, get 3 message from client:
  - get leader change request.
  - get leader exchange: become leader.
  - get state request.



```
19:14:42,471
{'leader': 0, 't_index': '1', 'log': {'b': 2}, 'type': 'commitlog'}
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:17:47,861
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:17:47,874
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:17:47,889
[INFO]nodetest->get_commit_request:      ---> 0: on request      --- 2022-05-31
19:17:47,902
{'leader': 0, 't_index': '4', 'log': {'a': 1}, 'type': 'commitlog'}
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:17:47,929
[INFO]nodetest->get_commit_request:      ---> 0: on request      --- 2022-05-31
19:17:47,946
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:19:52,536
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:19:52,549
[INFO]nodetest->get_commit_request:      ---> 0: on request      --- 2022-05-31
19:19:52,560
[INFO]nodetest->get_readonly_request:    ---> 0: on request      --- 2022-05-31
19:22:46,681
```

```
linqi@ubuntu:~/223_project$ python nodetest.py -i 1
======== Running on http://localhost:30001 ========
(Press CTRL+C to quit)
[INFO]nodetest->get_log:          ---> 1: on request      --- 2022-05-31 19:13:31
,458
[INFO]nodetest->get_log:          ---> 1: on request      --- 2022-05-31 19:14:42
,446
[INFO]nodetest->get_log:          ---> 1: on request      --- 2022-05-31 19:14:42
,473
[INFO]nodetest->get_log:          ---> 1: on request      --- 2022-05-31 19:17:47
,905
[INFO]nodetest->get_readonly_request:    ---> 1: on request      --- 2022-05-31
19:22:46,693
```

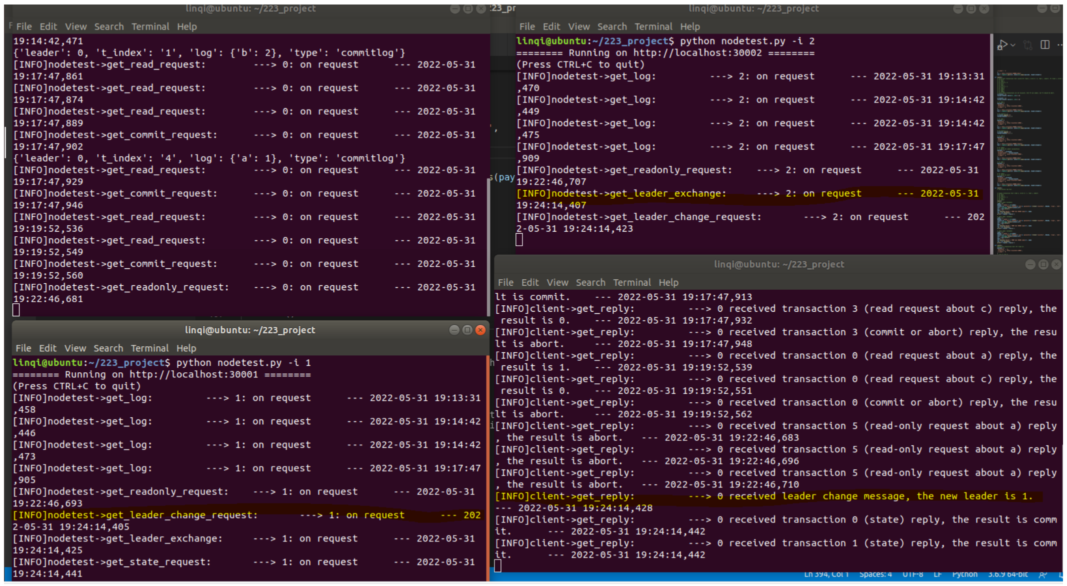
```
linqi@ubuntu:~/223_project$ python nodetest.py -i 2
======== Running on http://localhost:30002 ========
(Press CTRL+C to quit)
[INFO]nodetest->get_log:          ---> 2: on request      --- 2022-05-31 19:13:31
,470
[INFO]nodetest->get_log:          ---> 2: on request      --- 2022-05-31 19:14:42
,449
[INFO]nodetest->get_log:          ---> 2: on request      --- 2022-05-31 19:14:42
,475
[INFO]nodetest->get_log:          ---> 2: on request      --- 2022-05-31 19:17:47
,909
[INFO]nodetest->get_readonly_request:    ---> 2: on request      --- 2022-05-31
19:22:46,707
```

```
 result is 0.    --- 2022-05-31 19:17:47,864
[INFO]client->get_reply:          ---> 0 received transaction 4 (read request about c) reply, the
 result is 0.    --- 2022-05-31 19:17:47,876
[INFO]client->get_reply:          ---> 0 received transaction 4 (read request about b) reply, the
 result is 2.    --- 2022-05-31 19:17:47,891
[INFO]client->get_reply:          ---> 0 received transaction 4 (commit or abort) reply, the resu
lt is commit.    --- 2022-05-31 19:17:47,913
[INFO]client->get_reply:          ---> 0 received transaction 3 (read request about c) reply, the
 result is 0.    --- 2022-05-31 19:17:47,932
[INFO]client->get_reply:          ---> 0 received transaction 3 (commit or abort) reply, the resu
lt is abort.     --- 2022-05-31 19:17:47,948
[INFO]client->get_reply:          ---> 0 received transaction 0 (read request about a) reply, the
 result is 1.    --- 2022-05-31 19:19:52,539
[INFO]client->get_reply:          ---> 0 received transaction 0 (read request about c) reply, the
 result is 0.    --- 2022-05-31 19:19:52,551
[INFO]client->get_reply:          ---> 0 received transaction 0 (commit or abort) reply, the resu
lt is abort.     --- 2022-05-31 19:19:52,562
[INFO]client->get_reply:          ---> 0 received transaction 5 (read-only request about a) reply
, the result is abort.   --- 2022-05-31 19:22:46,683
[INFO]client->get_reply:          ---> 0 received transaction 5 (read-only request about a) reply
, the result is abort.   --- 2022-05-31 19:22:46,696
[INFO]client->get_reply:          ---> 0 received transaction 5 (read-only request about a) reply
, the result is abort.   --- 2022-05-31 19:22:46,710
```

Figure 9: Test5 Result

- node 2, get 2 message from leader:
  - get leader exchange.
  - get leader change request.

The test ran successfully and met our expectations. Now, the leader successfully changes from node 0 to node 1, and node 0 does not receive any message after that change.



```
19:14:42,471
{'leader': 0, 't_index': '1', 'log': {'b': 2}, 'type': 'commitlog'}
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:17:47,861
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:17:47,874
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:17:47,889
[INFO]nodetest->get_commit_request:      ---> 0: on request      --- 2022-05-31
19:17:47,902
{'leader': 0, 't_index': '4', 'log': {'a': 1}, 'type': 'commitlog'}
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:17:47,929
[INFO]nodetest->get_commit_request:      ---> 0: on request      --- 2022-05-31
19:17:47,946
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:19:52,536
[INFO]nodetest->get_read_request:        ---> 0: on request      --- 2022-05-31
19:19:52,549
[INFO]nodetest->get_commit_request:      ---> 0: on request      --- 2022-05-31
19:19:52,560
[INFO]nodetest->get_readonly_request:    ---> 0: on request      --- 2022-05-31
19:22:46,681
```

```
linqi@ubuntu:~/223_project$ python nodetest.py -i 1
======== Running on http://localhost:30001 ========
(Press CTRL+C to quit)
[INFO]nodetest->get_log:          ---> 1: on request      --- 2022-05-31 19:13:31
,458
[INFO]nodetest->get_log:          ---> 1: on request      --- 2022-05-31 19:14:42
,446
[INFO]nodetest->get_log:          ---> 1: on request      --- 2022-05-31 19:14:42
,473
[INFO]nodetest->get_log:          ---> 1: on request      --- 2022-05-31 19:17:47
,905
[INFO]nodetest->get_readonly_request:    ---> 1: on request      --- 2022-05-31
19:22:46,693
[INFO]nodetest->get_leader_change_request:       ---> 1: on request      --- 202
2-05-31 19:24:14,405
[INFO]nodetest->get_leader_exchange:     ---> 1: on request      --- 2022-05-31
19:24:14,425
[INFO]nodetest->get_state_request:       ---> 1: on request      --- 2022-05-31
19:24:14,441
```

```
linqi@ubuntu:~/223_project$ python nodetest.py -i 2
======== Running on http://localhost:30002 ========
(Press CTRL+C to quit)
[INFO]nodetest->get_log:          ---> 2: on request      --- 2022-05-31 19:13:31
,470
[INFO]nodetest->get_log:          ---> 2: on request      --- 2022-05-31 19:14:42
,449
[INFO]nodetest->get_log:          ---> 2: on request      --- 2022-05-31 19:14:42
,475
[INFO]nodetest->get_log:          ---> 2: on request      --- 2022-05-31 19:17:47
,909
[INFO]nodetest->get_readonly_request:    ---> 2: on request      --- 2022-05-31
19:22:46,707
[INFO]nodetest->get_leader_exchange:     ---> 2: on request      --- 2022-05-31
19:24:14,407
[INFO]nodetest->get_leader_change_request:       ---> 2: on request      --- 202
2-05-31 19:24:14,423
```

```
lt is commit.    --- 2022-05-31 19:17:47,913
[INFO]client->get_reply:          ---> 0 received transaction 3 (read request about c) reply, the
 result is 0.    --- 2022-05-31 19:17:47,932
[INFO]client->get_reply:          ---> 0 received transaction 3 (commit or abort) reply, the resu
lt is abort.     --- 2022-05-31 19:17:47,948
[INFO]client->get_reply:          ---> 0 received transaction 0 (read request about a) reply, the
 result is 1.    --- 2022-05-31 19:19:52,539
[INFO]client->get_reply:          ---> 0 received transaction 0 (read request about c) reply, the
 result is 0.    --- 2022-05-31 19:19:52,551
[INFO]client->get_reply:          ---> 0 received transaction 0 (commit or abort) reply, the resu
lt is abort.     --- 2022-05-31 19:19:52,562
[INFO]client->get_reply:          ---> 0 received transaction 5 (read-only request about a) reply
, the result is abort.   --- 2022-05-31 19:22:46,683
[INFO]client->get_reply:          ---> 0 received transaction 5 (read-only request about a) reply
, the result is abort.   --- 2022-05-31 19:22:46,696
[INFO]client->get_reply:          ---> 0 received transaction 5 (read-only request about a) reply
, the result is abort.   --- 2022-05-31 19:22:46,710
[INFO]client->get_reply:          ---> 0 received leader change message, the new leader is 1.
--- 2022-05-31 19:24:14,428
[INFO]client->get_reply:          ---> 0 received transaction 0 (state) reply, the result is comm
it.      --- 2022-05-31 19:24:14,442
[INFO]client->get_reply:          ---> 0 received transaction 1 (state) reply, the result is comm
it.      --- 2022-05-31 19:24:14,442
```

Figure 10: Test6 Result

Ultimately, we ran all 6 tests successfully and satisfied the requirements.