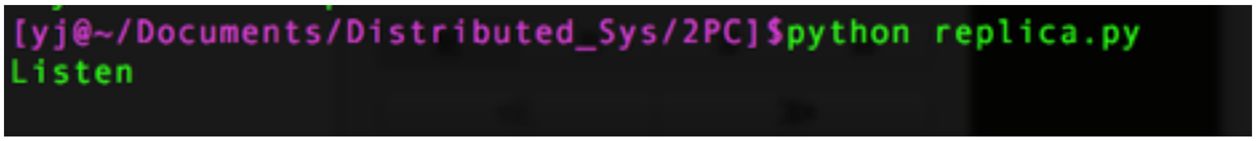
Here is a screenshot of the final project of Distributed Computing.
The project is to implement a classic protocol that has been widely adopted in Distributed world to make consensus between multiple machines - Two phase commit.

Execution Sequence

1. Run replica.py

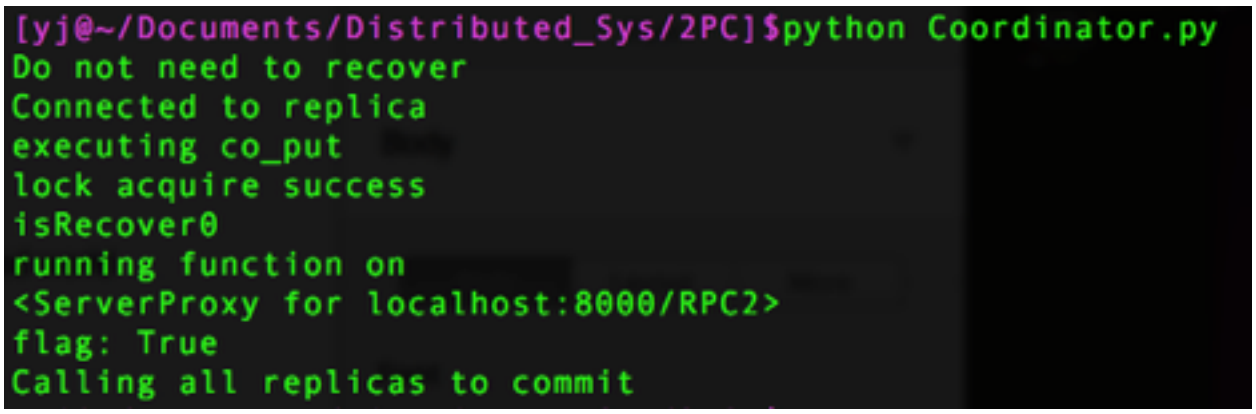


```
[yj@~/Documents/Distributed_Sys/2PC]$python replica.py
Listen
```

2. Run Coordinator.py

3. Execute put function in Coordinator

4. A few outputs on coordinator side to show progress



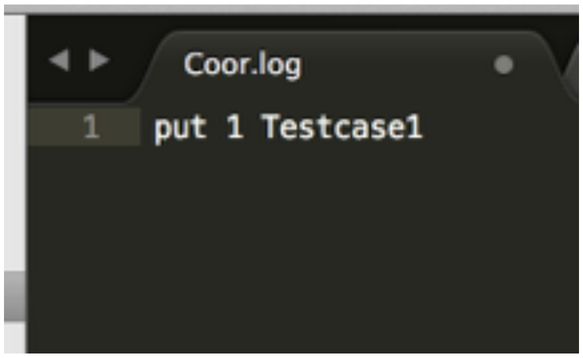
```
[yj@~/Documents/Distributed_Sys/2PC]$python Coordinator.py
Do not need to recover
Connected to replica
executing co_put
lock acquire success
isRecover0
running function on
<ServerProxy for localhost:8000/RPC2>
flag: True
Calling all replicas to commit
```

The coordinator first open Coor.log and look into the content. Since this is the first operation, the file is blank then it decided no recover is needed. And then it connects to replica in replica_add. In the first case, I test the system on localhost only, making sure the syntax and logic is reasonable before distributing the file to multiple machines.

Now execute the put function. As there might be more than one request to put key value to replicas at a time, lock is introduced to make sure the resource is exclusive and system is running in a first-come-first-serve manner. The system is then tested again whether there is need to recover from failure by looking into log file. The statement ”isRecover 0” means no failure so far.

After verifying all replicas’s status, the coordinator starts to run put function on replicas. Like I said, this case only includes one replica, which is running on local machine. The statement

“<ServerProxy for localhost: 8000/RPC2>” verify the address of connected machine. This is the

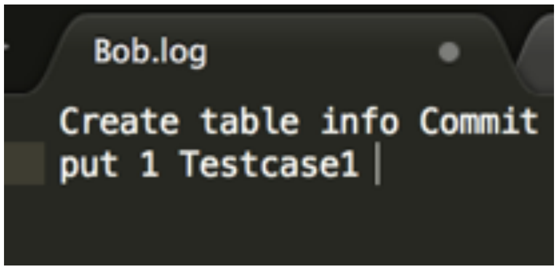


```
put 1 Testcase1
```

voting phase in Two-phase-commit. Coordinator logs the request sent by client in coor.log file.

Noted that the entry written in the file indicates: this is a “put” operation with key = 1, and value = “Testcase1”.

Coordinator asks all of replicas to execute the request and replicas would reply with a yes or no answer. The fact that flag equals to true means individual answer from each replicas. In this case, replica on localhost replies with a yes, setting the flag to be true. At the time, each replica execute the transaction to the point where a commit or abort command is needed to complete the transaction. They log the status the same way as Coordinator.

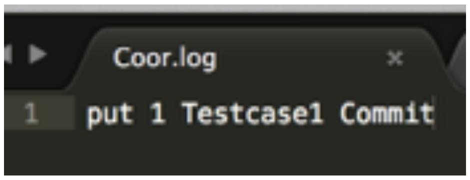


```
Create table info Commit
put 1 Testcase1
```

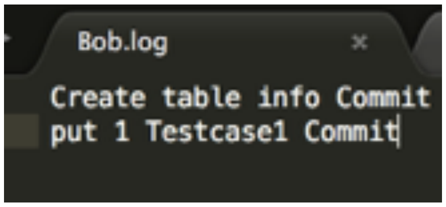
The replica is identified by the name Bob, so it creates a Bob.log file. While setting up the replica class, the system connects to sqlite db and creates a table named info, which will store key value pair. The last keyword “Commit” indicates that the transaction have completed, since it doesn't require consensus from other replicas, commit action is executed right after creating table. And the statement “put 1 Testcase1” is explained above in coordinator section.

After leveraging every reply from replicas, coordinator makes a decision to commit. Here comes into the second phase - commit phase. Coordinator notifies all the replicas of the decision and request them all to commit the transaction.

Now both coor.log and Bob.log will add “Commit” keyword at the end of line. This specifies the completion of transaction, and also indicates the commit phase is completed. Shown as below:

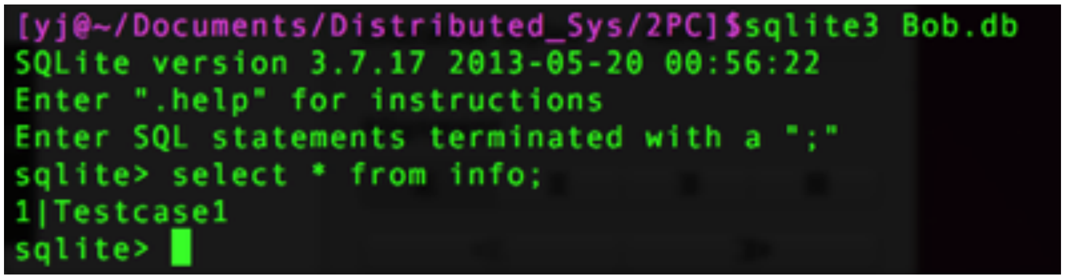


```
put 1 Testcase1 Commit
```



```
Create table info Commit
put 1 Testcase1 Commit
```

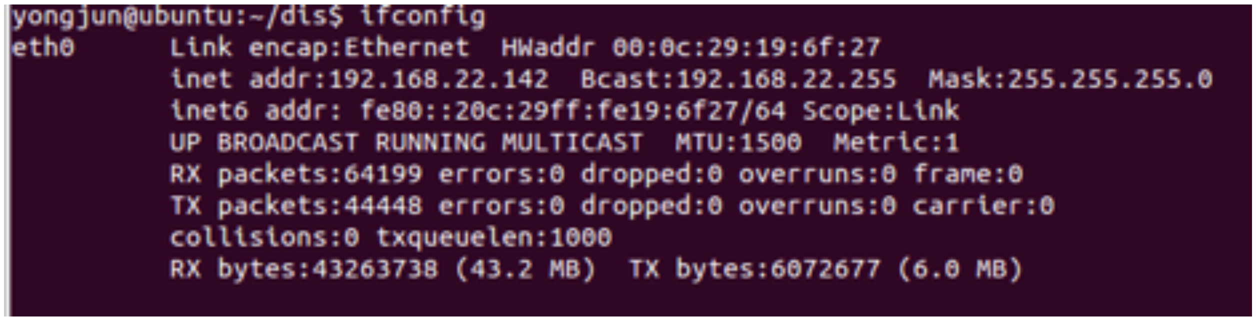
Now let us take a look into sqlite3 database to see if the record has been inserted.



```
[yj@~/Documents/Distributed_Sys/2PC]$sqlite3 Bob.db
SQLite version 3.7.17 2013-05-20 00:56:22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> select * from info;
1|Testcase1
sqlite> 
```

The next case is to try to connect to multiple replica in the same time. I have tried the gee nodes before but it failed all the time. So I used virtual machine on my local machine.

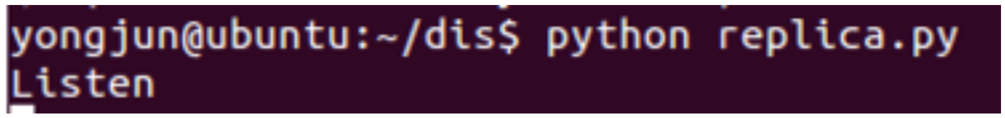
At first you need to upload the replica.py to vmware, and type in linux command to fetch IP address.



```
yongjun@ubuntu:~/dis$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:19:6f:27
          inet addr:192.168.22.142  Bcast:192.168.22.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe19:6f27/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:64199 errors:0 dropped:0 overruns:0 frame:0
          TX packets:44448 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:43263738 (43.2 MB)  TX bytes:6072677 (6.0 MB)
```

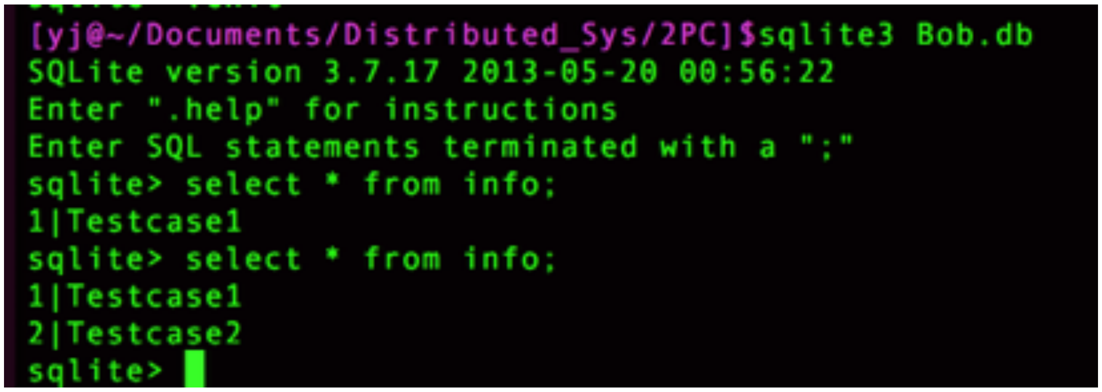
And then update replica address in fabfile.py to IP above. And put replica.py to a set of nodes.

Build up rpc server.

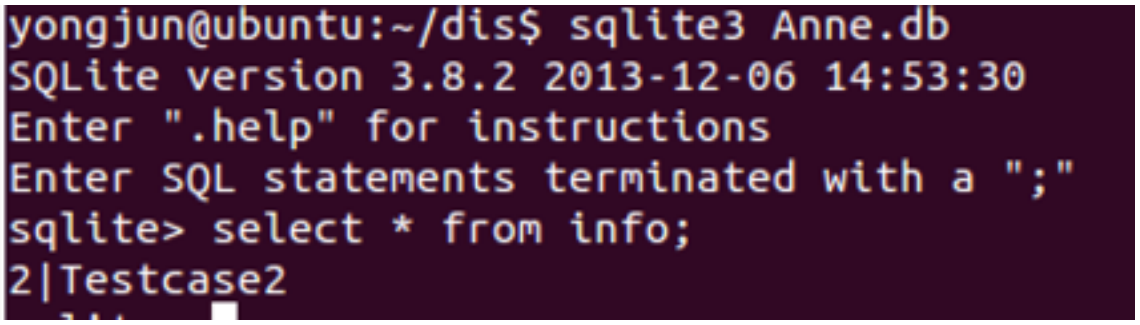


```
yongjun@ubuntu:~/dis$ python replica.py
Listen
```

Execute put operation with key = 2, value = “Testcase2” in Coordinator. All process is executed as above. See sqlite3 for both Bob.db and Anne.db(in VMware)



```
[yj@~/Documents/Distributed_Sys/2PC]$sqlite3 Bob.db
SQLite version 3.7.17 2013-05-20 00:56:22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> select * from info;
1|Testcase1
sqlite> select * from info;
1|Testcase1
2|Testcase2
sqlite> 
```



```
yongjun@ubuntu:~/dis$ sqlite3 Anne.db
SQLite version 3.8.2 2013-12-06 14:53:30
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> select * from info;
2|Testcase2
```

As shown above, the delete function and get function is the same as put function.

How to handle failure:

1. A log file is created both on Coordinator and Replica machine. They will keep track of the ongoing transactions as well as the history transaction. Both coordinator and replica will look into the files before any new requests are handled. If they discover uncompleted transactions, they will go ahead to deal with them accordingly.

2. There is a chance that requests are sent simultaneously. This will definitely affect [put] and [del] function. So I introduce Lock module from built-in python to make sure exclusive use for the resource. Wrap critical code around lock.acquire() statement and lock.release() statement.

3. Check data validation before fetching/deleting. Keep in mind return value could be None. Make the best use of try-catch block. Wrap the code that fires up remote procedure call around try-catch blocks. This is useful to traceroute issues.

Test Cases:

A. Coordinator Recover

Basically if the coordinator restores and check in files, and discover there is a transaction that hasn't been completed(remain in the voting phase). It finished the uncompleted transaction by calling all replicas to commit/abort.

1. A request to insert a key-value (3, "Testcase3") in sqlite3.
2. In the Coor.log, an entry "put 3 Testcase3 Commit" will be inserted upon the completion of transaction.
3. Manually delete the "Commit" keyword, this means the coordinator crashes in the voting phase.
4. Restart coordinator process.
5. It will run the recover function and discovers a missing action "Commit/Abort" at the end of one row.
6. Continue with the uncompleted action and asks the replicas to commit/abort. [Recover process]
7. After recovering, coordinator goes ahead to do the request as normal.

B. Replica Recover

Replica Recover is similar to Coordinator Recover, except replica process will not interact with other replicas like Coordinator, it only updates its own database record.

1. Write a entry(put “1” “Testcase1” Commit/Abort) in Log kept by replica. e.g Anne.log for Anne replica and Bob.log for Bob replica.
2. Manually delete the keyword “Commit/Abort” in log file.
3. Restart the replica process.
4. A missing keyword of either Commit or Abort will trigger the replica to complete the transaction again.

C. Test Null Value

1. Input key = 3 into multiple replicas database.
2. Del key = 3
3. Initial a request to fetch the value of key = 3.

1. Input key = 4 into multiple replicas database.
2. Del key = 4
3. Del key = 4 again. -> Note that the transaction try to delete something doesn't exist.