Fault Tolerant Computing Project 3

The design and implementation of my primary-backup e-Voting server.

There are the function list available supplied by my server:

```
RegisterVoter (Voter) returns (Status)
UnregisterVoter (Voter) returns (Status)
PreAuth (VoterName) retuens (Challenge)
Auth (AuthRequest) returns (AuthToken)
CreateElection (Election) returns (Status)
CastVote (Vote) returns (Status)
GetResult (ElectionName) retuens (ElectionResult)
```

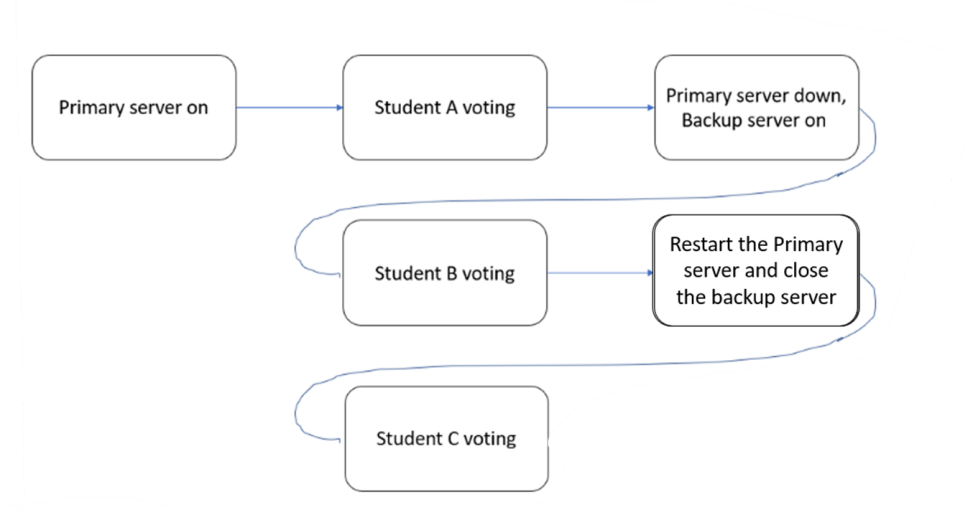
Once one of above operation are used, the `backup_data()` function will be called, write the necessary data to a file named `backup.txt`
If the primary server is restarted, it will load data from `backup.txt` as the initialization.
There is my code,

```
def backup_data(self,):
        ### backup the data to a file, named backup.txt
        """
        need to store: self.Users,      (list)
                       |- self.name              (str)
                       |- self.group             (str)
                       |- self.public_key        (binary)
                       |- self.challenge         (binary)
                       |- self.auth_token        (binary)
                       |- self.auth_token_expire_time (int)
                       |- self.token_expire      (bool)
                       |- self.vote_done         (bool)

                       self.end_date,
                       self.voting_name,

                       self.candidates, (list)
                       |- self.name              (str)
                       |- self.ballot            (int)

                       self.groups,     (list)
                       self.election_done
        """
        with open('backup.txt', mode='w', newline='') as f:
            ### clear contents in the file
            f.truncate()
            f.write(str(len(self.Users)) + '\n')
            for user in self.Users:
                f.write(str(user.name) + '\n')
                f.write(str(user.group) + '\n')
                f.write(base64.b64encode(user.public_key).decode('utf8') + '\n')
                f.write(user.challenge.decode('utf8') + '\n')
                f.write(user.auth_token.decode('utf8') + '\n')
                f.write(str(user.auth_token_expire_time) + '\n')
                f.write(str(user.token_expire) + '\n')
                f.write(str(user.vote_done) + '\n')

            f.write(str(self.end_date) + '\n')
            f.write(self.voting_name + '\n')
            f.write(str(len(self.candidates)) + '\n')
            for candidate in self.candidates:
                candidate_info = str(candidate.name) + ',' + str(candidate.ballot) + '\r
                f.write(candidate_info)
            f.write(str(len(self.groups)) + '\n')
            for group in self.groups:
                f.write(group + '\n')
            f.write(str(self.election_done) + '\n')

        print("Got an action on server, doing the backup operation.")
```

```
def recover_data(self,):

        with open('backup.txt', 'r') as f:
            user_num = f.readline()

            ### reconver all users
            for i in range(int(user_num)):
                name = f.readline()[:-1]
                #print(name)
                #print('read name done')
                #input()
                group = f.readline()[:-1]
                public_key = (f.readline()[:-1]).encode('utf8')
                challenge = (f.readline()[:-1]).encode('utf8')
                auth_token = (f.readline()[:-1]).encode('utf8')
                seconds = f.readline()
                seconds = int(seconds[9:-2])
                nanos = f.readline()
                nanos = int(nanos[7:-1])
                #print(nanos)
                auth_token_expire_time  = Timestamp(seconds=seconds, nanos=nanos)
                dangling = f.readline()[:-1] ### the surplus \n
                str_token_expire = f.readline()
                token_expire = True if str_token_expire == True else False
                #print(type(token_expire))

                str_vote_done = f.readline()
                vote_done = True if str_vote_done == True else False
                #print(vote_done)

                self.Users.append(User(name=name,
                                       group=group,
                                       public_key=public_key))
                self.Users[i].challenge = challenge
                self.Users[i].auth_token = auth_token
                self.Users[i].auth_token_expire_time = auth_token_expire_time
                self.Users[i].vote_done = vote_done

            seconds = f.readline()
            seconds = int(seconds[9:-2])
            nanos = f.readline()
            nanos = int(nanos[7:-1])
            self.end_date = Timestamp(seconds=seconds, nanos=nanos)
            dangling = f.readline() ### the surplus \n
            self.voting_name = f.readline()[:-1]
            candidate_num = f.readline()
            ### recover all candidates
            for i in range(int(candidate_num)):
                candidate_info = f.readline().split(',')
                name = candidate_info[0]
                ballot = candidate_info[1][:-1]
                self.candidates.append(Candidate(name))

                self.candidates[i].ballot = int(ballot)

            groups_num = f.readline()
            self.group = []
            for i in range(int(groups_num)):
                self.groups.append(f.readline()[:-1])

            self.election_done = f.readline()[:-1]
```

The evaluation of my e-Voting server and client. Need to show that the server meets the fault tolerance requirements.

Steps of my evaluation



There, I use the `KeyBoardInterrupt` as the unexpected termination of the server.
Because there are three student, there must be three ballots in the end.

1. Primary server on

```
(base) w311554053@tim7107-BM1AF-BP1AF-BM6AF:~/anaconda3/python_code/FaultTolerantComputing/hw1$ python server.py
server is acting......
```

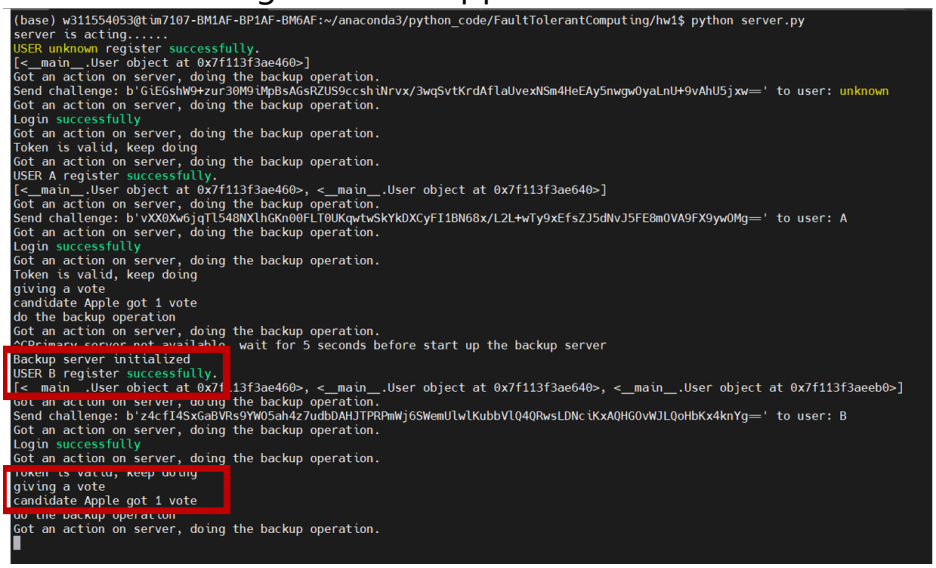
2. Student A voting : vote to Apple

```
(base) w311554053@tim7107-BM1AF-BP1AF-BM6AF:~/anaconda3/python_code/FaultTolerantComputing/hw1$ python server.py
server is acting......
USER unknown register successfully.
[<__main__.User object at 0x7f113f3ae460>]
Got an action on server, doing the backup operation.
Send challenge: b'GiEGshW9+zur30M9iMpBsAGsRZUS9ccshiNrvx/3wqSvtKrdAflaUvexNSm4HeEAy5nwgwOyaLnU+9vAhU5jxw==' to user: unknown
Got an action on server, doing the backup operation.
Login successfully
Got an action on server, doing the backup operation.
Token is valid, keep doing
Got an action on server, doing the backup operation.
USER A register successfully.
[<__main__.User object at 0x7f113f3ae460>, <__main__.User object at 0x7f113f3ae640>]
Got an action on server, doing the backup operation.
Send challenge: b'vXX0Xw6jqTl548NXlhGKn00FLT0UKqwtwSkYkDXCyFI1BN68x/L2L+wTy9xEfsZJ5dNvJ5FE8mOVA9FX9ywOMg==' to user: A
Got an action on server, doing the backup operation.
Login successfully
Got an action on server, doing the backup operation.
Token is valid, keep doing
giving a vote
candidate Apple got 1 vote
do the backup operation
Got an action on server, doing the backup operation.
```

3. Primary Server down, backup server on

```
(base) w311554053@tim7107-BM1AF-BP1AF-BM6AF:~/anaconda3/python_code/FaultTolerantComputing/hw1$ python server.py
server is acting......
USER unknown register successfully.
[<__main__.User object at 0x7f113f3ae460>]
Got an action on server, doing the backup operation.
Send challenge: b'GiEGshW9+zur30M9iMpBsAGsRZUS9ccshiNrvx/3wqSvtKrdAflaUvexNSm4HeEAy5nwgwOyaLnU+9vAhU5jxw==' to user: unknown
Got an action on server, doing the backup operation.
Login successfully
Got an action on server, doing the backup operation.
Token is valid, keep doing
Got an action on server, doing the backup operation.
USER A register successfully.
[<__main__.User object at 0x7f113f3ae460>, <__main__.User object at 0x7f113f3ae640>]
Got an action on server, doing the backup operation.
Send challenge: b'vXX0Xw6jqTl548NXlhGKn00FLT0UKqwtwSkYkDXCyFI1BN68x/L2L+wTy9xEfsZJ5dNvJ5FE8mOVA9FX9ywOMg==' to user: A
Got an action on server, doing the backup operation.
Login successfully
Got an action on server, doing the backup operation.
Token is valid, keep doing
giving a vote
candidate Apple got 1 vote
do the backup operation
Got an action on server, doing the backup operation.
^CPrimary server not available, wait for 5 seconds before start up the backup server
Backup server initialized
```

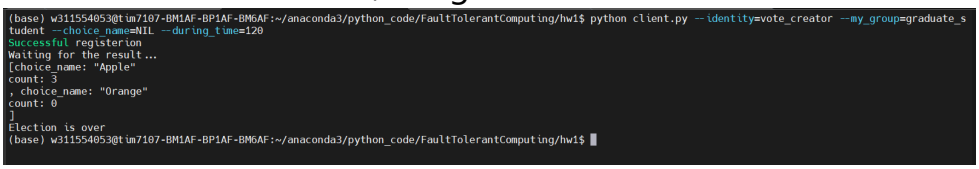
4. Student B voting : vote to Apple



5. Restart the Primary Server and close the backup server

6. Student C voting : voting to Apple

After the election over, we got the result:



As we excepted.

Issue

> If the primary and the backup server are serve at the same time, request from clients will send to one of them randomly.

There comes two solution in my mind.

1. Once the primary server restarted, the backup server should be closed immediately.
2. Forcing the ip address can only used by one server, therefore if we desire to restart the primary server, connect will be build after the backup server is closed.

For this problem, I am only implement the solution 1.