**Final Report**

*Wensong Xiao & Tianyi Wang*

**Code Link**: https://github.com/bladewangpro/Python\_WorkPlace

**Video Link**: https://drive.google.com/file/d/1DIiwVkyRUCIrksuKtfE89CgaDdUSPYn3/view?usp=sharing

Problem motivation:

Nowadays, consensus problems became more and more essential in industry infrastructures. There were a plenty of consensus protocols to solve this kind of problems, such as Zookeeper protocol; however, Paxos algorithm almost represented the whole consensus problem area. So here is a question, what is Three-Army problem? This is the description I borrow from our project proposal:

In a battle, there are three blue army and one red army. The red army is too strong to be defeated by one blue army. Only way to defeat red army is that gathering over one blue army (the majority of all the blue army) to attack red army in the same time. These blue army need to negotiate with each other and get a specific time to attack red one together and they will follow the rules behind.

1. Every army have one staff officer (proposer) to provide a specific time; every army also

have one general (acceptor) to decide which time the blue army (learner) will attack red army; Before one staff officer notify the blue army to attack, this officer need get agreement from at least two generals among these blue armies.

1. Every staff officer will give proposal in two stage (prepare/commit) and the proposal will be given with a proposal ID.
2. If there is any confliction in preparation stage, the general will make decision according to the number id. The bigger, the winner.
3. If the staff officer receives general 's decided time, he or she must change his or her proposal time.

The reason why we choose the Three-Army problem as our topic is that the Three-Army problem can be simulated by the roles in Paxos, proposers, acceptors and learners. We could get further comprehension about consensus problem and Paxos mechanism from this project.

Design goals:

In the final solution, we want to use Dos console to simulate different roles in Three-Army problem. And each army will be represented by an individual computer. On each computer, we will open three console windows to simulate proposer, learner and acceptor individually. After the programs finish their running, each learner will be received the same value by the proposers. And the majority of these proposers should get the same value. We use numbers to represent proposals in Paxos protocol. Furthermore, we want to simulate the Paxos solution with bad network issue. For example, when three computers are running the final solution, we will make one of these acceptors offline. We will shut down the console window of this acceptor. If our code is right and solution is appropriate, we can get the majority of these proposers with the same value. And all the learner can learn the same value from these proposers.

Design architecture:

Till now, we have designed five scripts for our project, they are “LEARNER.py”, “PROPOSER.py”, “ACCEPTOR.py”, “sf.py” and “test.py” separately. Our purpose is to let every single file be able to take the responsibility of one role in Paxos protocol. That means, we use instruction to run “PROPOSER.py” files on the console windows with different IP addresses and ports. And then, these console windows will play the “Proposer” role in the computers they are running on. Same situation to the “ACCEPTOR.py” and “LEARNER.py”. In the “test.py”, we have integrated the instruments to launch the “PROPOSER.py” programs with the ones to launch the “ACCEPTOR.py” programs. Besides, the “test.py” file will invoke GUI function in the “sf.py” file to make front interface. The information of “ACCEPTORs” will be record in the ‘log’ file and the information of “LEARNER.py” will be written into the ‘learnerLog’ file

Description of code and scripts: major data structure and control flow

1. ACCEPTOR.py

We use class to implement this part. The acceptor in Paxos protocol has two function. At first, the acceptor will receive the messages from different proposers and compare the proposal values from different proposers to decide whose proposal value it will adopt. Secondly, it will send messages to the proposers who have sent message to it and direct those proposers with the message it passed.

In the first part, we use an infinite loop to be a listener to monitor the information which will be sent by different proposers. When the acceptor got information from these proposers, it will compare the proposal ID with the one it has. And according to the value in the “phase1Sign”, the acceptor will choose the value in 0, 1, 2, and 3 to reply the proposer who just sent message to it. And ‘0’ means rejection, ‘1’ means that the proposer can pass the first phase, ‘2’ means the acceptor has already took the proposer’s value. ‘3’ means that the acceptor has already took the value of the proposer ID, but it has already decided a value and hope the proposer who just sent message to it to change its value.

1. PROPOSER.py

In this part, we also use class to implement the proposer function. At first, it will read the information in ‘log’ file to get all the information of “ACCEPTORs” and ‘leanerLog’ of “LEARNERs”. We use a progress-oriented programming method in this part. At first, the proposer will send his information to all the acceptors via the list which it got from ‘log’ file. Then, they will begin to wait the information from the acceptors. In the first round, they will count the number of acceptors who give them the first stage permission. When the permission is over half of total number, they will change their status of phase1Sign from 0 to 1. And the 1 means that they have already enter the second stage in Paxos protocol. Then, in the second round, they will start to count the number of acceptors who has accepted their proposal value and who has accepted their proposal ID. Depending to the message from acceptors, they will modify their value automatically and broadcast to all the learners. To avoid the situation where one acceptor is forced to be offline, we let the rejected count of proposer plus one, if the proposer get socket.error message.

1. Other files

We design our layout in the sf.py and design the test function in the test.py file to simulate three army problem in one machine. And we can use instrument in docs to run proposers and acceptors on the same computer and even in different computers. In the ‘LEARNER.py’ file, there exists one socket.recvfrom function to monitor the message which will come from proposers.

Description of difficulty:

The difficulties we met in this project is really a lot. Because, before this project, we did not finish any project with python and we just know the Paxos in very shallow level. Until when we decided to divide this protocol into different individual part, acceptor, proposer and learner. Another challenge is when we try to turn one acceptor to offline, there will come out a plenty of error information and if some conditions cannot be satisfied because one acceptor is offline and cannot send out any information, the program will be stuck. For this problem, in the end, we use timeout function to solve this kind of issue. In a nutshell, in this program, we really get further understanding of Paxos and improve our coding skills. Every stage in this project, we will be faced with countless bugs. Now the final project is done, we have already learned how to use python to code, what the real Paxos is and how to use UDP and json format to transfer information.

Scenario:

1. Single computer, 2 proposers and 3 acceptors

All the proposers in this scenario get the same value 53333 and come to an agreement. (Success)

1. Single computer, 2 proposers and 4 acceptors.

We let one acceptor offline in halfway.

All the proposers get the same value 53332 and come to an agreement. (Success)

1. Two computer, A computer: 3 acceptors, B computer: 3 learners and 3 proposers.

All the learners and proposers get the same value 78. (Success)

1. Two computer, A computer: 3 acceptors, B computer: 3 learners and 3 proposers.

We let one acceptor offline in halfway

All the learners and proposers get the same value 78. (Success)

1. Three computers, each computer: 1 acceptor, 1 proposer, 1 learner (simulate Three-Army problem)

All the learners and the majority of proposers get the same value 23. (Success)

1. Three computers, each computer: 1 acceptor, 1 proposer, 1 learner (simulate Three-Army problem)

We just shut down one acceptor in halfway.

All the learners and all the proposers get the same value 22. (Success)

Future work:

Now our project can only determine one value. In the future, we can consider the number as the serial number of one instrument set. Each instrument can execute one operation. And we also can implement Paxos from deciding one value to multiple value. Furthermore, we can add state machine into this project. Let this project not only can print a set of values with the same order and can let multiple machine running along a set of instruments with the same order.

Reference:

Wensong Xiao && Tianyi Wang ‘s project proposal.