SIJIA ZHU

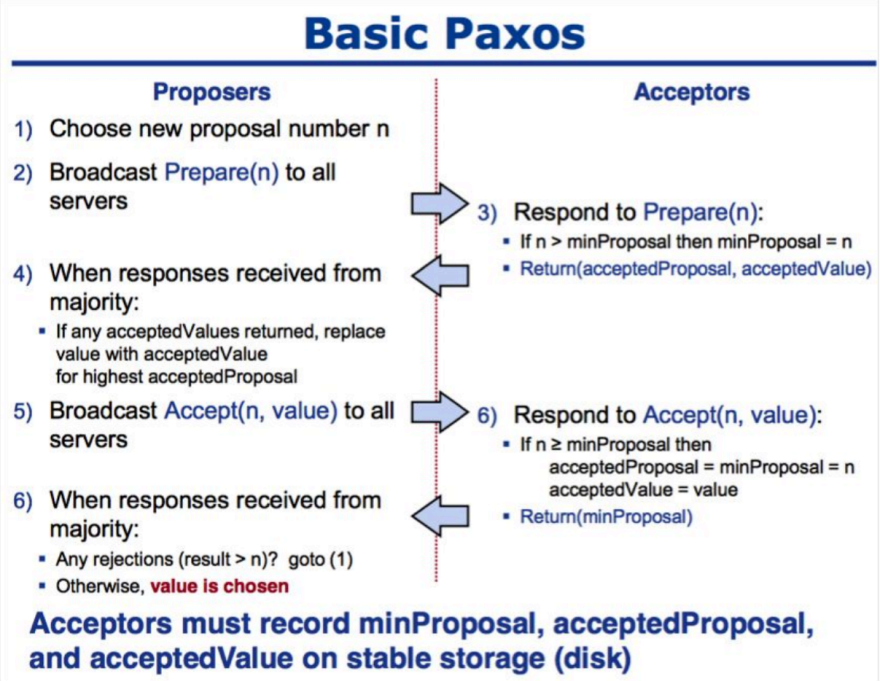
Youtube Link: https://www.youtube.com/watch?v=Wte1KaC2mvY

Assignment4 Overview

For this assignment, we implement the CURD operation for Paxos to realize fault-tolerant consensus through RMI. Through this implementation, we can understand this idea how paxos work and how the Paxos make fault-tolerant important in consensus for web server distributed system through random shut down server practice.

Technical impression

This project is well structured and good practice for us understand Paxos.



First, the paxos algorithm is not easy, before we start implement, we have to understand very well how paxos algorithm works. We have three role in this algorithm, Proposer、Acceptor、Learner. Proposers try to send Id to acceptors and acceptor only accept the highest Id for record, only half of acceptors send back the Id will make the phase two which is send value with Id to acceptor again and only half of acceptors send back the accept ok will go to next pahse which is learning phase. For basic algorithm, we only have one proposer because more proposers will make algorithm more complexity and hard to make consensus in different proposers. And we need odd server as acceptors because we need more than half of acceptors to make consensus. For example for PUT operation, we start from proposer collect 5 acceptors vote prepare and collect 5 acceptors vote accept, and commit.

```
Server 1 is running at port  12345
17:00:24:563 Server 1 Proposer log: Proposer send vote to Acceptor- phase prepare
17:00:24:572 Server 1 Acceptor log: Acceptor receive vote from Proposer: prepare phase
17:00:24:572 Server 1 Acceptor log: Acceptor start to compare ID
17:00:24:573 Server 1 Acceptor log: proposalId 1 is larger than current received Id 1 , accepted
17:00:24:573 Server 1 Proposer log: received 1 servers replied with prepare ok
17:00:24:616 Server 1 Proposer log: received 2 servers replied with prepare ok
17:00:24:659 Server 1 Proposer log: received 3 servers replied with prepare ok
17:00:24:704 Server 1 Proposer log: received 4 servers replied with prepare ok
17:00:24:747 Server 1 Proposer log: received 5 servers replied with prepare ok
17:00:24:747 Server 1 Proposer log: received 5 servers replied with prepare ok, go to next phase-> accept
17:00:24:748 Server 1 Acceptor log: Acceptor receive vote from Proposer: accept phase
17:00:24:749 Server 1 Acceptor log: Acceptor start to compare ID
17:00:24:749 Server 1 Acceptor log: proposalId 1 is larger than current received Id 1 , accepted
17:00:24:749 Server 1 Proposer log: received 1 servers replied with accept ok
17:00:24:751 Server 1 Proposer log: received 2 servers replied with accept ok
17:00:24:753 Server 1 Proposer log: received 3 servers replied with accept ok
17:00:24:754 Server 1 Proposer log: received 4 servers replied with accept ok
17:00:24:756 Server 1 Proposer log: received 5 servers replied with accept ok
17:00:24:756 Server 1 Proposer log: received 5 servers replied with accept ok, go to next phase-> commit
17:00:24:757 Added key : 1000 Value: 101
```

```
Server2 (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_151.jdk/Contents/Home/bin/java (Apr 18, 2020, 5:
Server 2 is running at port  12346
17:00:24:615 Server 2 Acceptor log: Acceptor receive vote from Proposer: prepare phase
17:00:24:616 Server 2 Acceptor log: Acceptor start to compare ID
17:00:24:616 Server 2 Acceptor log: proposalId 1 is larger than current received Id 1 , accepted
17:00:24:750 Server 2 Acceptor log: Acceptor receive vote from Proposer: accept phase
17:00:24:750 Server 2 Acceptor log: Acceptor start to compare ID
17:00:24:750 Server 2 Acceptor log: proposalId 1 is larger than current received Id 1 , accepted
17:00:24:759 Added key : 1000 Value: 101
```

```
Server 3 is running at port  12347
17:00:24:658 Server 3 Acceptor log: Acceptor receive vote from Proposer: prepare phase
17:00:24:659 Server 3 Acceptor log: Acceptor start to compare ID
17:00:24:659 Server 3 Acceptor log: proposalId 1 is larger than current received Id 1 , accepted
17:00:24:752 Server 3 Acceptor log: Acceptor receive vote from Proposer: accept phase
17:00:24:752 Server 3 Acceptor log: Acceptor start to compare ID
17:00:24:752 Server 3 Acceptor log: proposalId 1 is larger than current received Id 1 , accepted
17:00:24:761 Added key : 1000 Value: 101
```

```
Server 4 is running at port  12348
17:00:24:703 Server 4 Acceptor log: Acceptor receive vote from Proposer: prepare phase
17:00:24:704 Server 4 Acceptor log: Acceptor start to compare ID
17:00:24:704 Server 4 Acceptor log: proposalId 1 is larger than current received Id 1 , accepted
17:00:24:754 Server 4 Acceptor log: Acceptor receive vote from Proposer: accept phase
17:00:24:754 Server 4 Acceptor log: Acceptor start to compare ID
17:00:24:754 Server 4 Acceptor log: proposalId 1 is larger than current received Id 1 , accepted
17:00:24:763 Added key : 1000 Value: 101
```

```
Server 5 is running at port  12349
17:00:24:746 Server 5 Acceptor log: Acceptor receive vote from Proposer: prepare phase
17:00:24:746 Server 5 Acceptor log: Acceptor start to compare ID
17:00:24:747 Server 5 Acceptor log: proposalId 1 is larger than current received Id 1 , accepted
17:00:24:756 Server 5 Acceptor log: Acceptor receive vote from Proposer: accept phase
17:00:24:756 Server 5 Acceptor log: Acceptor start to compare ID
17:00:24:756 Server 5 Acceptor log: proposalId 1 is larger than current received Id 1 , accepted
17:00:24:766 Added key : 1000 Value: 101
```

Second, random shut down server is the key test for paxos implementation. However, it is not easy to make server random shut down, First, I compare to random shut down server inside the acceptor class or outside. If we implement inside the acceptor class, it is easy to shut down as we can make it sleep. However, it is fake simulate server shut down, and hard to proposer to collect vote because for RMI it might be wait for result. So I go to second idea that I write function stop() in acceptor, and encapsulation to outer class paxoserver and then server extend paxoserver and will control stop() function for acceptor in server instance. When in server instance, it will sleep block time and then based on flag it run stop() function with parameter of millisecond.

```
public boolean stop(int time) throws InterruptedException {
  if (((int) ((Math.random() * Constants.NUMBER_OF_SERVERS) + 1)) == serverNumber) {
    System.out.println("Server " + serverNumber + " going to sleep");
    active = false;
    Thread.sleep(time);
    System.out.println("Server " + serverNumber + " wake up");
    active = true;

    return true;
  }
  return false;
```

```
public boolean stop(int time) throws InterruptedException {
    acceptor.setActive(false);
    boolean stop = acceptor.stop(time);
    acceptor.setActive(true);
    return stop;
  }
```

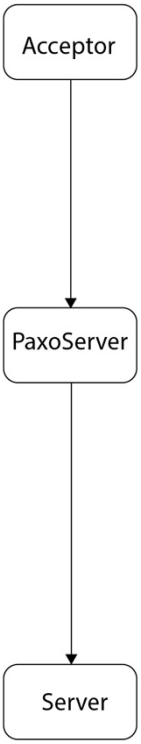
```
  while (true) {
    Thread.sleep(1000);

    if (active) {

      active = server.stop(5000);

    }
    active = !active;

  }
```



Then we can have test based on random stop our acceptor server. We write randomShutDown in client side and run 500 time to do same operation, we found they must be sometime that more than 3 server random shutdown at same time will make consensus less than 3, and Paxos algorithm will make not consensus to CURD operation. Which means paxos algorithm is good to fault-tolerant in server shut down situation even less than half of server is not available.

```
public static void RandomShutDown(Client client) throws IOException {
  int count = 0;
  while (true) {
    count++;
    clientOperation(client, "PUT", "1001", "101");
    if (count == 500)
      break;
  }

}
```

```
Server response time 16:02:31:195 Added key : 1001 Value: 101
Server response time 16:02:31:206 Added key : 1001 Value: 101
Server response time 16:02:31:213 Added key : 1001 Value: 101
Server response time 16:02:31:229 Added key : 1001 Value: 101
Server response time 16:02:31:264 Added key : 1001 Value: 101
Server response time 16:02:31:301 Added key : 1001 Value: 101
Server response time 16:02:31:310 Added key : 1001 Value: 101
Server response time 16:02:31:319 Added key : 1001 Value: 101
Server response time 16:02:31:327 Added key : 1001 Value: 101
Server response time 16:02:31:335 Added key : 1001 Value: 101
Server response time 16:02:31:342 Added key : 1001 Value: 101
Server response time 16:02:31:348 Added key : 1001 Value: 101
Server response time 16:02:31:356 Added key : 1001 Value: 101
Server response time 16:02:31:366 Added key : 1001 Value: 101
Server response time 16:02:31:376 Added key : 1001 Value: 101
Server response time 16:02:31:385 Added key : 1001 Value: 101
Server response time 16:02:31:393 Added key : 1001 Value: 101
Server response time 16:02:31:396 16:02:31:395 Server 1 Consensus could not be reached as only 2 servers replied to the prepare request
Server response time 16:02:31:398 16:02:31:398 Server 1 Consensus could not be reached as only 2 servers replied to the prepare request
Server response time 16:02:31:401 16:02:31:401 Server 1 Consensus could not be reached as only 2 servers replied to the prepare request
Server response time 16:02:31:404 16:02:31:403 Server 1 Consensus could not be reached as only 2 servers replied to the prepare request
Server response time 16:02:31:406 16:02:31:406 Server 1 Consensus could not be reached as only 2 servers replied to the prepare request
Server response time 16:02:31:408 16:02:31:408 Server 1 Consensus could not be reached as only 2 servers replied to the prepare request
Server response time 16:02:31:411 16:02:31:411 Server 1 Consensus could not be reached as only 2 servers replied to the prepare request
Server response time 16:02:31:414 16:02:31:414 Server 1 Consensus could not be reached as only 2 servers replied to the prepare request
Server response time 16:02:31:417 16:02:31:417 Server 1 Consensus could not be reached as only 2 servers replied to the prepare request
Server response time 16:02:31:420 16:02:31:420 Server 1 Consensus could not be reached as only 2 servers replied to the prepare request
Server response time 16:02:31:424 16:02:31:424 Server 1 Consensus could not be reached as only 2 servers replied to the prepare request
Server response time 16:02:31:426 16:02:31:426 Server 1 Consensus could not be reached as only 2 servers replied to the prepare request
Server response time 16:02:31:430 16:02:31:430 Server 1 Consensus could not be reached as only 2 servers replied to the prepare request
```

We can have more details to see how acceptor shutdown effect, when accept sleep, we can find in server 1acceptor will not process for vote, same as server 4, when server is not sleep, it will compare ID and send feedback, and when server is sleep it will not process. Proposer collect 3 acceptor and works for put key and value to store.

Client Side

```
<terminated> Client (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8
Server response time 16:41:09:707 Added key : 1000 Value: 101
```

Server 1 as Proposer and Acceptor

```
Server 1 is running at port  12345
16:41:02:158 Server 1 Acceptor log: going to sleep
16:41:03:824 Server 1 Proposer log: Proposer send vote to Acceptor- phase prepare
16:41:03:834 Server 1 Acceptor log: Acceptor receive vote from Proposer: prepare phase
16:41:03:834 Server 1 Acceptor log: is sleepping, can't process
16:41:03:857 Server 1 Proposer log: received 1 servers replied with prepare ok
16:41:03:925 Server 1 Proposer log: received 2 servers replied with prepare ok
16:41:03:947 Server 1 Proposer log: Consensus could not be reached as only 2 servers replied to the prepare request
16:41:06:574 Server 1 Proposer log: Proposer send vote to Acceptor- phase prepare
16:41:06:577 Server 1 Acceptor log: Acceptor receive vote from Proposer: prepare phase
16:41:06:577 Server 1 Acceptor log: is sleepping, can't process
16:41:06:579 Server 1 Proposer log: received 1 servers replied with prepare ok
16:41:06:583 Server 1 Proposer log: received 2 servers replied with prepare ok
16:41:06:585 Server 1 Proposer log: Consensus could not be reached as only 2 servers replied to the prepare request
16:41:07:162 Server 1 Acceptor log: wake up
16:41:09:660 Server 1 Proposer log: Proposer send vote to Acceptor- phase prepare
16:41:09:661 Server 1 Acceptor log: Acceptor receive vote from Proposer: prepare phase
16:41:09:661 Server 1 Acceptor log: Acceptor start to compare ID
16:41:09:661 Server 1 Acceptor log: proposalId 3 is larger than current received Id 3 , accepted
16:41:09:661 Server 1 Proposer log: received 1 servers replied with prepare ok
16:41:09:663 Server 1 Proposer log: received 2 servers replied with prepare ok
16:41:09:669 Server 1 Proposer log: received 3 servers replied with prepare ok
16:41:09:672 Server 1 Proposer log: received 3 servers replied with prepare ok, go to next phase-> accept
16:41:09:674 Server 1 Acceptor log: Acceptor receive vote from Proposer: accept phase
16:41:09:675 Server 1 Acceptor log: Acceptor start to compare ID
16:41:09:675 Server 1 Acceptor log: proposalId 3 is larger than current received Id 3 , accepted
16:41:09:675 Server 1 Proposer log: received 1 servers replied with accept ok
16:41:09:677 Server 1 Proposer log: received 2 servers replied with accept ok
16:41:09:682 Server 1 Proposer log: received 3 servers replied with accept ok
16:41:09:682 Server 1 Proposer log: received 3 servers replied with accept ok, go to next phase-> commit
16:41:09:683 Added key : 1000 Value: 101
```

Server 4 as Acceptor

```
Server 4 is running at port  12348
16:41:03:925 Server 4 Acceptor log: Acceptor receive vote from Proposer: prepare phase
16:41:03:925 Server 4 Acceptor log: Acceptor start to compare ID
16:41:03:925 Server 4 Acceptor log: proposalId 1 is larger than current received Id 1 , accepted
16:41:06:583 Server 4 Acceptor log: Acceptor receive vote from Proposer: prepare phase
16:41:06:583 Server 4 Acceptor log: Acceptor start to compare ID
16:41:06:583 Server 4 Acceptor log: proposalId 2 is larger than current received Id 2 , accepted
16:41:07:191 Server 4 Acceptor log: going to sleep
16:41:09:667 Server 4 Acceptor log: Acceptor receive vote from Proposer: prepare phase
16:41:09:667 Server 4 Acceptor log: is sleepping, can't process
16:41:09:680 Server 4 Acceptor log: Acceptor receive vote from Proposer: accept phase
16:41:09:680 Server 4 Acceptor log: is sleepping, can't process
16:41:09:689 Added key : 1000 Value: 101
```