# ECE750: Engineering Reliable Software

# Assignment Report

Name: Yanxin Wang
ID: 20439686

## 1. Basic Paxos Algorithm

There are three classes of agents in Paxos: ***proposers, acceptors and learners***. Agents can communicate with one another by sending messages. There are four classes of messages: ***proposal, promise, accept and accepted***.

Agents have local memories, operate at arbitrary speed, may fail and restart. Messages can take arbitrarily long to be delivered, can be duplicated and can be lost, but they are not corrupted.

When a client from outside the Paxos core wants to start a vote, this client tells the proposers that a vote is needed. The proposers then start the following three phases operation (see Fig.1):

***Phase 1***

a) A ***proposer*** sends a ***proposal*** to a majority of ***acceptors***. This ***proposal*** contains a proposal number ***n***, which is also stored at ***proposer's*** memory.

b) If an ***acceptor*** receives a proposal, it compares the proposal number ***n*** in this ***proposal*** with its own memory ***m***. If ***n***>=***m***, this ***acceptor*** responds to the ***proposer*** with a ***promise*** and replaces its memory with ***n***. This message, ***promise***, contains ***rnd*** and ***val***, which are the highest proposal number (***rnd***) it has accepted (if any) and the value associated with ***rnd*** (***val***).

***Phase 2***

a) If the ***proposer*** receives a response message, ***promise***, from a majority of ***acceptors***, then it sends a message, ***accept***, to each of those ***acceptors***. It chooses a value ***v***, which is the ***val*** associated with the highest-numbered ***rnd*** among the responses or any value if the responses reported no ***val***. The message, ***accept***, contains a proposal number **n** and value ***v***.

b) If an ***acceptor*** receives an ***accept*** message with a proposal numbered ***n***, it accepts the message unless it own memory ***m*** is larger than ***n*** now.

***Phase 3***

***Acceptors*** send ***learners*** the ***accepted*** messages, which contain the chosen value ***v***.

```
Client   Proposer      Acceptor     Learner
   |         |          |  |  |       |  |
   X-------->|          |  |  |       |  |  Request
   |         X--------->|->|->|       |  |  Prepare(1)
   |         |<---------X--X--X       |  |  Promise(1, {null,null,null})
   |         X--------->|->|->|       |  |  Accept!(1,V)
   |         |<---------X--X--X------>|->|  Accepted(1,V)
   |<---------------------------------X--X  Response
   |         |          |  |  |       |  |
```

Figure 1. Message Flow Chat of Basic Paxos

There are some optimizations for Paxos:

1) Phase 1 a), proposal could be sent to all acceptors to increase the probability that this message is responded by a majority of acceptors. This is important when links between agents are not highly reliable.
2) Phase 1 b), if n<m, the acceptor should probably inform the proposer to abandon its proposal. Similarly, in Phase 2 b), the acceptor should inform the proposer if an accept message is denied.
3) In phase 3, how to send a chosen value to learners could be flexible. The obvious algorithm is to have each acceptor respond to all learners. Another algorithm is to have the acceptors respond with their acceptances to a distinguished learner, which in turn informs the other learners. More generally, the acceptors could respond with their acceptances to some set of distinguished learners, each of which can then inform all learners.
4) Though safety is ensured, progress is not guaranteed unless a distinguished proposer (called leader) is selected as the only one try issuing proposals.

## 2. Applied Algorithm

Here I simulate basic Paxos with C# in Visual Studio 2010. As mentioned in previous section, several optimizations are applied:

1) A leader is needed as a guarantee of progress. I simply skip the leader election and put in one and only proposer instead.
2) Messages are sent to all responders instead of a majority of them in case that message passing goes wrong.
3) Every acceptor send accepted value to all learners though it may be a heavy traffic.

To simplify the algorithm, some mechanisms are not considered:

1) No retransmission mechanisms. This saves the agents a lot of functions to check and clock.

2) One round vote. The proposer only votes once and terminates the program.
3) No accepted message goes to proposer due to above two reasons.
4) All agents are physically separated. That is to say agents must send messages to communicate with others and there is no safe internal message delivery.

Failure scenarios are focus on message delivery: messages may get lost, be duplicated or delay (out of order).

## 3. Identify elements

### a) Agents

I define a base class Agent which contains these elements:

- Name: e.g. Proposer
- Counter: to denote multiple agents, e.g. (Proposer) #1
- Queue: to store messages in first-in-first-out queue
- Method Enqueue: to add messages to the end of Queue
- Method SendUnreliableMessage: to send messages through unreliable way
- Method ExcuteMultiThread: to start thread and try to get message to process
- Method ConsumeAllMessage: to get messages from the beginning of Queue and process via appropriate methods.
- Method DispatchMessgae: virtual method, override in derived class to process messages

There are three derived classes: Proposer, Acceptor and Learner. Learner is the simplest class. It has:

- AcceptorCount: the number of acceptors from who this learner should get values
- List ProposalNumber: list of received proposal numbers.
- List Value: list of received values.
- List KnownAcceptors: list of acceptors this learner has heard from.
- AcceptedConfirm: to indicate if this learner has received messages from a majority of acceptors.
- Method DispatchMessage: override method, to receive accepted messages from acceptors, write in local memories, judge if vote success (according to AcceptedConfirm) and display accepted value.

Class Proposer has:

- Acceptors: to whom this proposer should propose.
- ProposalNumber: the most important parameter in Paxos.
- MyProposedValue: the value to be proposed.
- List ReceivedNumber: list of received ***rnd*** (see section 1, phase1.b).
- List ReceivedValue: list of received ***val*** (see section 1, phase1.b).
- Method GenerateNextProposalNumber: to generate proposal number

- Method Propose: to send propose messages to all acceptors
- Method DispatchMessage: override method, to receive promise messages from acceptors and to respond with accept messages.

Class Acceptor has:

- Learners: to whom this acceptor should send accepted values
- ProposalNumber: the most important parameter in Paxos.
- PreviousAcceptedNumber: ***rnd*** (see section 1, phase1.b).
- PreviousAcceptedValue: ***val*** (see section 1, phase1.b).
- Method DispatchMessage: override method, to receive proposal and accept messages from proposers, to respond proposers with promises and to send accepted values to learners.

### b) Messages and Values

I define a base class Message which contains these elements:

- Originator: from where this message is sent.
- Destination: to where this message is sent.

There are four derived classes: Proposal, Promise, Accept and Accepted.
Class Proposal has only one item: Proposal Number.
Class Promise has proposal number, ***rnd*** and ***val***.
Class Accept has proposal number and proposal value.
Class Accepted has proposal number and accepted value.
I also define a class MyValue for proposed value in case it may differ from vote to vote.

## 4. Simulation Program

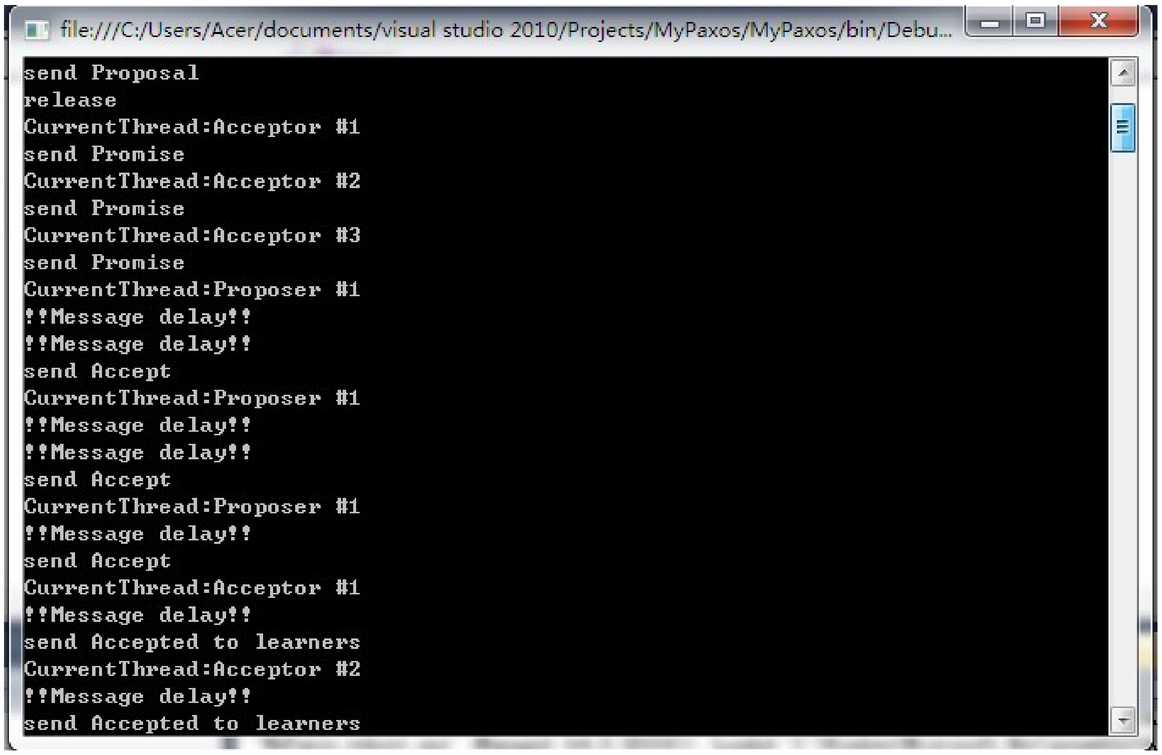
Multiple threads are applied to ensure every agent have equal chance to process messages. Each agent is assigned a thread. Threads are controlled by lock and semaphore which is written in class Agent so that only one thread can get access to message queue at a time. A thread tries to get a message from the beginning of queue without removing it. If this message is for this agent, the thread takes the message and removes it from queue. Otherwise, the thread gives up the right to read queue and let another thread in.
I first set up the system by instantiating agents and then start threads for every agent. Finally I make proposer to propose a value and the algorithm will start automatically.

## 5. Scenarios Study

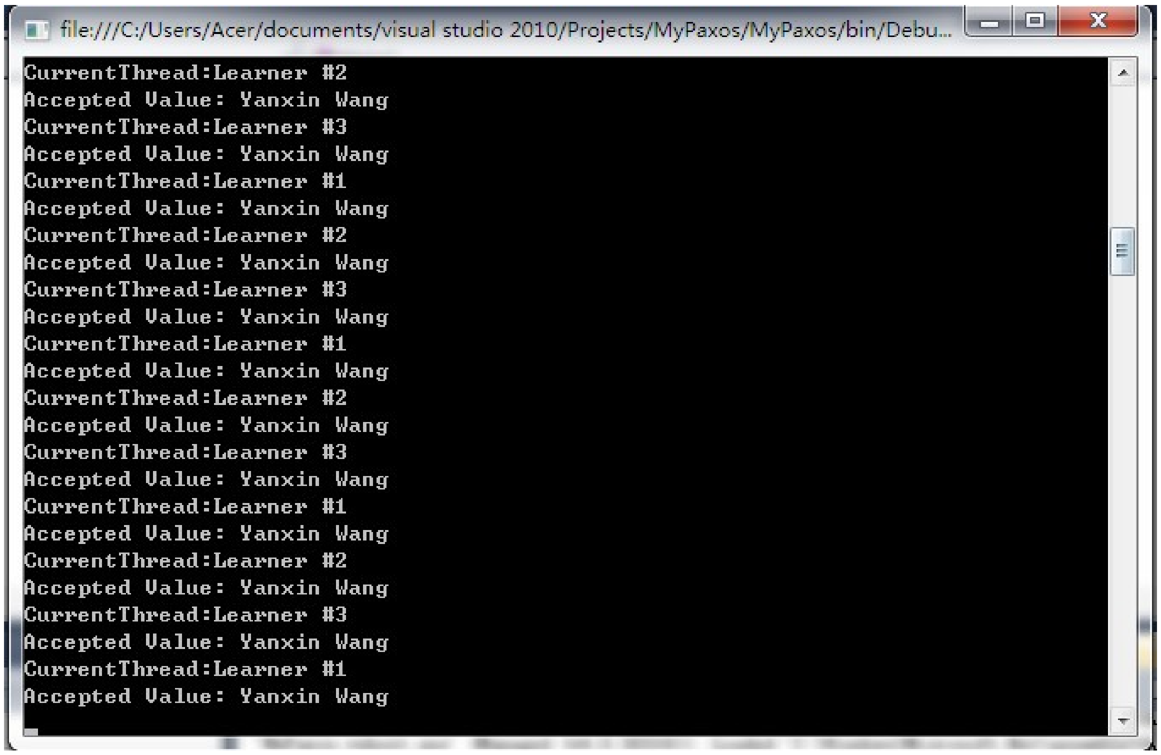
I first try this C# Paxos with 3 acceptors, 3 learners and 1 proposer in safe scenario where messages are sent safely. The program is working well except for termination because I do not write the code to abort threads. Then I try this C# Paxos in scenarios where messages are unreliably sent. The algorithm also works and learners get the right value from a majority of acceptors. I set the probability of message lost to 10%, message duplicated to 10% and out of order to 10%. Out-of-order messages don not disturb the progress because of the queue mechanism. They just consume more time for some agents may be waiting for them. Duplicated messages also have little effect on progress except for more useless messages and processes. Message lost, however, does fail the algorithm but this seldom happens. Paxos fails when 2 or 3 messages are lost between proposer and acceptors either proposals or promises. The probability for this is quite low (10%*10%+10%*10%*10%=0.011 for each). Paxos also fails when all messages are lost from acceptors to a learner. However, this has even lower probability because every acceptor send accepted value. So during my simulation, none of these failures is observed.

## 6. Appendix

Screen snaps for program:


file:///C:/Users/Acer/documents/visual studio 2010/Projects/MyPaxos/MyPaxos/bin/Debu...

```
send Proposal
release
CurrentThread:Acceptor #1
send Promise
CurrentThread:Acceptor #2
send Promise
CurrentThread:Acceptor #3
send Promise
CurrentThread:Proposer #1
!!Message delay!!
!!Message delay!!
send Accept
CurrentThread:Proposer #1
!!Message delay!!
!!Message delay!!
send Accept
CurrentThread:Proposer #1
!!Message delay!!
send Accept
CurrentThread:Acceptor #1
!!Message delay!!
send Accepted to learners
CurrentThread:Acceptor #2
!!Message delay!!
send Accepted to learners
```


file:///C:/Users/Acer/documents/visual studio 2010/Projects/MyPaxos/MyPaxos/bin/Debu...

```
CurrentThread:Learner #2
Accepted Value: Yanxin Wang
CurrentThread:Learner #3
Accepted Value: Yanxin Wang
CurrentThread:Learner #1
Accepted Value: Yanxin Wang
CurrentThread:Learner #2
Accepted Value: Yanxin Wang
CurrentThread:Learner #3
Accepted Value: Yanxin Wang
CurrentThread:Learner #1
Accepted Value: Yanxin Wang
CurrentThread:Learner #2
Accepted Value: Yanxin Wang
CurrentThread:Learner #3
Accepted Value: Yanxin Wang
CurrentThread:Learner #1
Accepted Value: Yanxin Wang
CurrentThread:Learner #2
Accepted Value: Yanxin Wang
CurrentThread:Learner #3
Accepted Value: Yanxin Wang
CurrentThread:Learner #1
Accepted Value: Yanxin Wang
```