Executive Summary for Project3

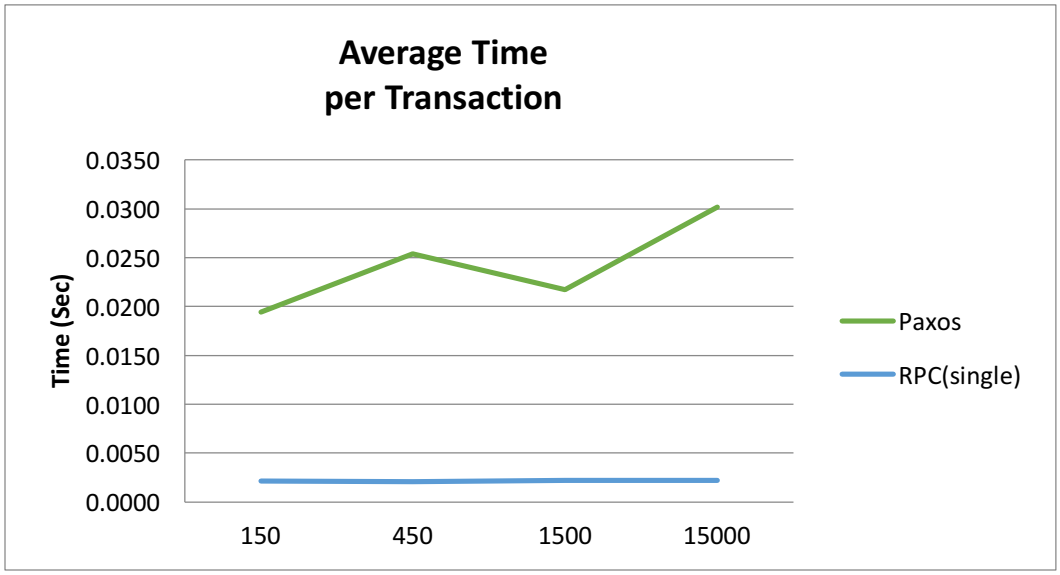
Jiacheng Liu, Ping Han Lei

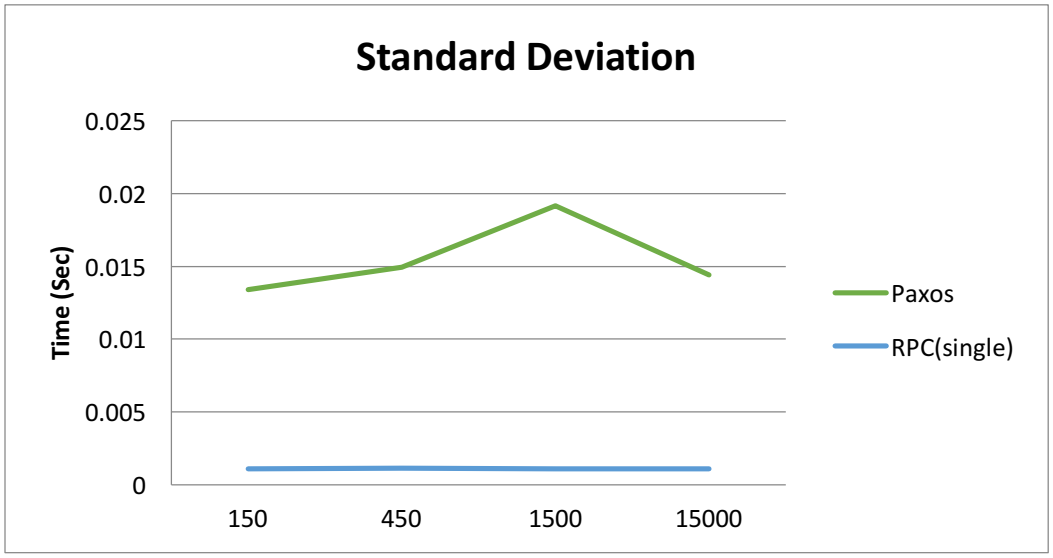
Assignment Overview:

The purpose of this project is to give us a better understanding about Paxos and a taste to handle real world problems like server failures. Although there are many papers about Paxos, few of them specified the implementation details in a real world setting. Among all the variations of Paxos, we choose to implement multi-Paxos with non-collapsed roles deployment. That is to say, three roles (proposers, acceptors and learners) are available on every server in the cluster. According to the requirement, acceptors are configured to fail randomly. We assume the proposer and learners will never fail. Therefore, there is no need to synchronize the database after the restart of an acceptor instance. Also, we assume every learner is able to execute the instructions, so that redo and undo logs are unnecessary. In our settings, there are multiple client instances sending requests to the cluster. We believe that's where Paxos comes in. Upon receiving multiple requests, the cluster servers must agree on which operation to execute. We adopted python2.7.12 and thrift 0.9.0 for this project. We also use pickledb 0.6 as our local key value store on the server. For each server, we placed five identical server.py files and multipaxos.py on five different nodes and have client.py running on a unique node. Then we tested our programs in four data sets with different sizes of 150, 450, 1500 and 150000 commands (basically just duplicate kvp- operations.csv, without malformed commands).

Result Summary

The results show that the Paxos cluster runs much slower than a single server, which is what we're expecting because synchronization takes time. The charts and table give the details.





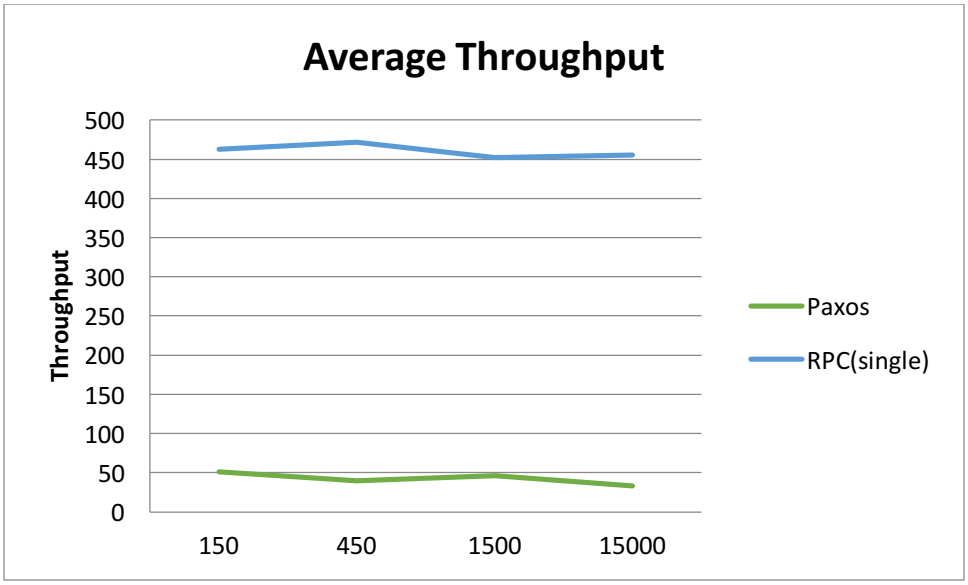


Table in details:

AverageTimePerTransaction		
DataLoad	Paxos	RPC(single)
150	0.0195	0.0022
450	0.0254	0.0021
1500	0.0217	0.0022
15000	0.0302	0.0022
Standard Deviation		
DataLoad	Paxos	RPC(single)
150	0.01339453	0.00111
450	0.01492	0.00113
1500	0.019149511	0.00110
15000	0.014405944	0.00110
AverageThroughPut		
DataLoad	Paxos	RPC(single)
150	51.40310118	462.8916
450	39.41675866	472.0603
1500	46.07453089	451.8850
15000	33.15368483	455.6453

Technical Impression:

i) Server

There are five servers used in the test:

jiachl5@172.22.71.42 (n16) runs thrift_server_42.py

jiachl5@172.22.71.41 (n15) runs thrift_server_41.py

jiachl5@172.22.71.40 (n14) runs thrift_server_40.py

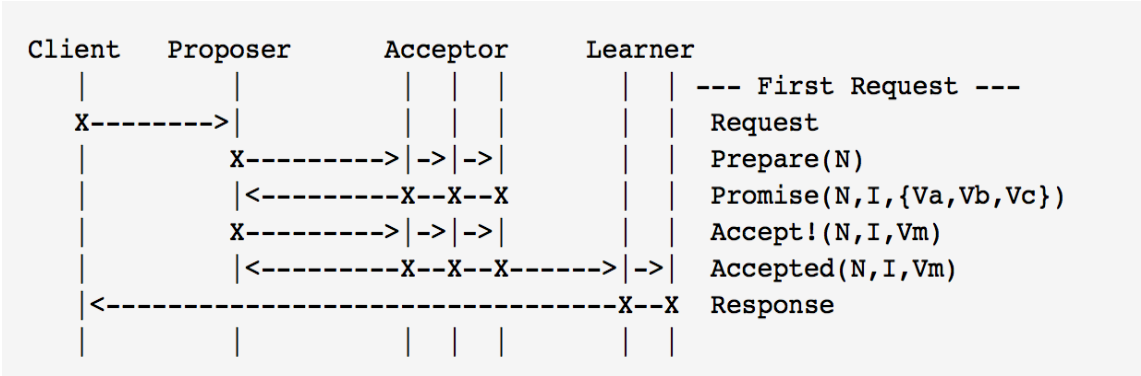
jiachl5@172.22.71.39 (n13) runs thrift_server_39.py

jiachl5@172.22.71.38 (n12) runs thrift_server_38.py

These scripts are identical except the SERVER_IP array is tuned to record the IP addresses of other nodes in the cluster, as the figure below shows.

```
#SERVER_IP=['172.22.71.42','172.22.71.38','172.22.71.40','172.22.71.41']
#SERVER_IP=['172.22.71.42','172.22.71.39','172.22.71.40','172.22.71.41']
#SERVER_IP=['172.22.71.42','172.22.71.38','172.22.71.39','172.22.71.41']
#SERVER_IP=['172.22.71.42','172.22.71.38','172.22.71.40','172.22.71.39']
SERVER_IP=['172.22.71.41','172.22.71.38','172.22.71.40','172.22.71.39']
```

Here is the message flow chart of multi-paxos: (*from Wikipedia*)



In terms of proposer, we make it a rule that whichever server receive the original request form the client will be the proposer in this Paxos instance. And the learner which is on the sane server with proposer is the only thread which is responsible for returning the result to the client. We assume all the learners and proposers will never fail.

ii) Client

jiachl5@172.22.71.37 (n11) is used as a client. In order to test Paxos, we split the input into two commands sets and use two process to execute them in a parallel way. Just as the following picture shows. Each process sends requests to server in the cluster in a round-robin way. In this way, the proposer in Paxos is constantly changing.

```
p1 = mul.Process(target=worker, args=(operations1))
p2 = mul.Process(target=worker, args=(operations2))
p1.start()
p2.start()
p1.join()
p2.join()
```

iii) Other issues

We constantly suffered from a connection error during our experiment. We're contacting Mr. Stephen to try to solve the problem.

```
liujiachengs-MacBook-Pro:~ liujiacheng$ ssh jiachl5@cssgate.insttech.washington.
edu
ssh_exchange_identification: read: Connection reset by peer
liujiachengs-MacBook-Pro:~ liujiacheng$
```

Acknowledgement

Jiacheng and Ping discussed the design options and implement the server together.