Executive Summary for Project2

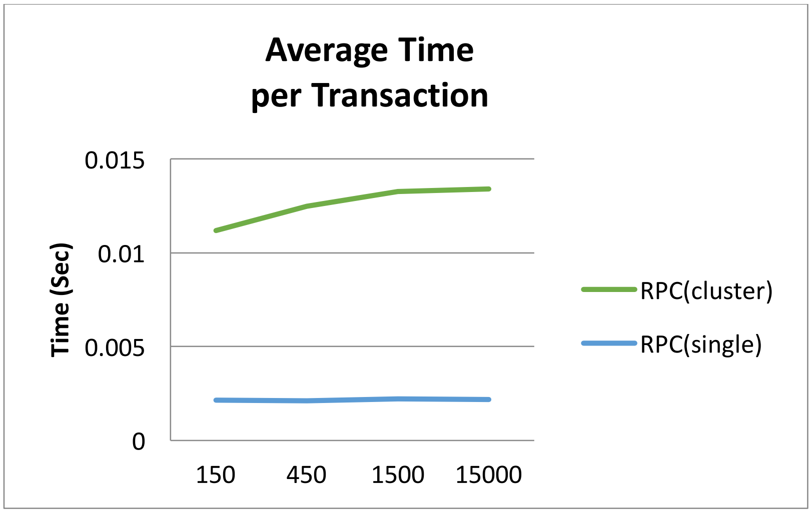
Jiacheng Liu, Ping Han Lei

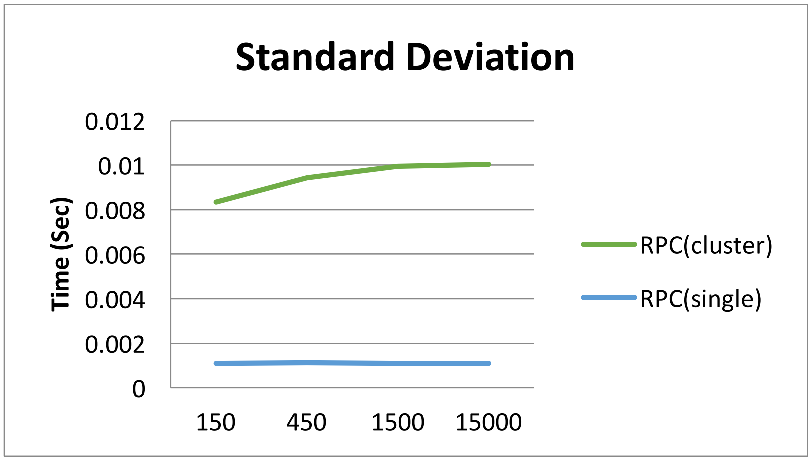
Assignment Overview:

A distributed communication that involves altering data on multiple databases is complicated since servers must coordinate the committing or rolling back of the changes in communication. We use two-phase commit mechanism to ensure the transmitting data. In this assignment, we have five servers to handle the transactions from one client via RPC. 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 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). The execution results indicate that the performance of server cluster is worse than the performance of single server. Single server method is about five times more efficient compare to the server cluster method.

Result Summary

The results show that the server cluster runs much slower than a single server, which is what we're expecting because synchronization takes time. We compute the cluster/single server ratio by dividing the single server running time. The charts and table give the details.


Average Time
per Transaction
Time (Sec)
0.015
0.01
0.005
0
150
450
1500
15000
RPC(cluster)
RPC(single)


Standard Deviation
Time (Sec)
0.012
0.01
0.008
0.006
0.004
0.002
0
150
450
1500
15000
RPC(cluster)
RPC(single)

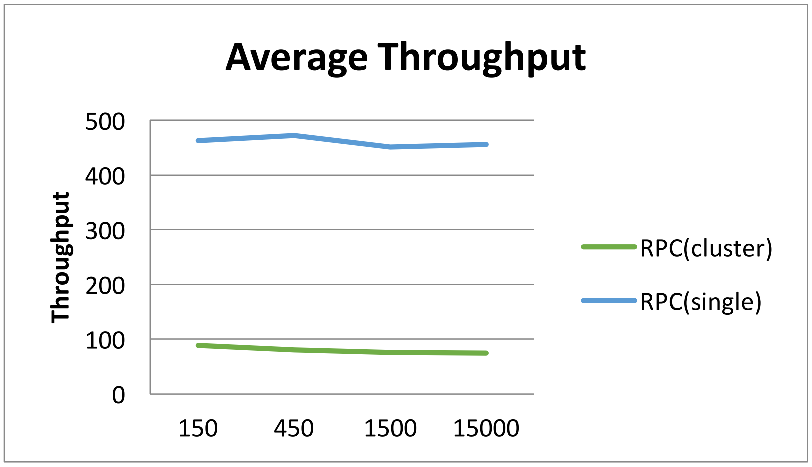

Average Throughput
Throughput
500
400
300
200
100
0
150
450
1500
15000
RPC(cluster)
RPC(single)

Table in details:

AverageTimePerTransaction			
DataLoad	RPC(cluster)	RPC(single)	factor(cluster/single)
150	0.0112	0.0022	5.184385931
450	0.0125	0.0021	5.947619048
1500	0.0133	0.0022	6.001079231
15000	0.0134	0.0022	6.114760751
Standard Deviation			
DataLoad	RPC(cluster)	RPC(single)	factor(cluster/single)
150	0.00833	0.00111	7.512010324
450	0.00942	0.00113	8.339823009
1500	0.00996	0.00110	9.071695743
15000	0.01004	0.00110	9.129945972
AverageThroughPut			
DataLoad	RPC(cluster)	RPC(single)	factor(cluster/single)
150	89.2266	462.8916	0.192759168
450	80.0607	472.0603	0.16959846
1500	75.2891	451.8850	0.166611209
15000	74.51362	455.6453	0.163534284

Technical Impression:

i) Server

There are five servers used in the test:

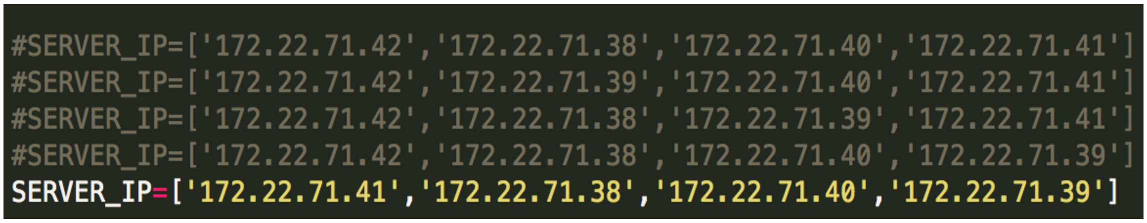
jiachl5@172.22.71.42 (n16) runs thrift_server_42.py

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

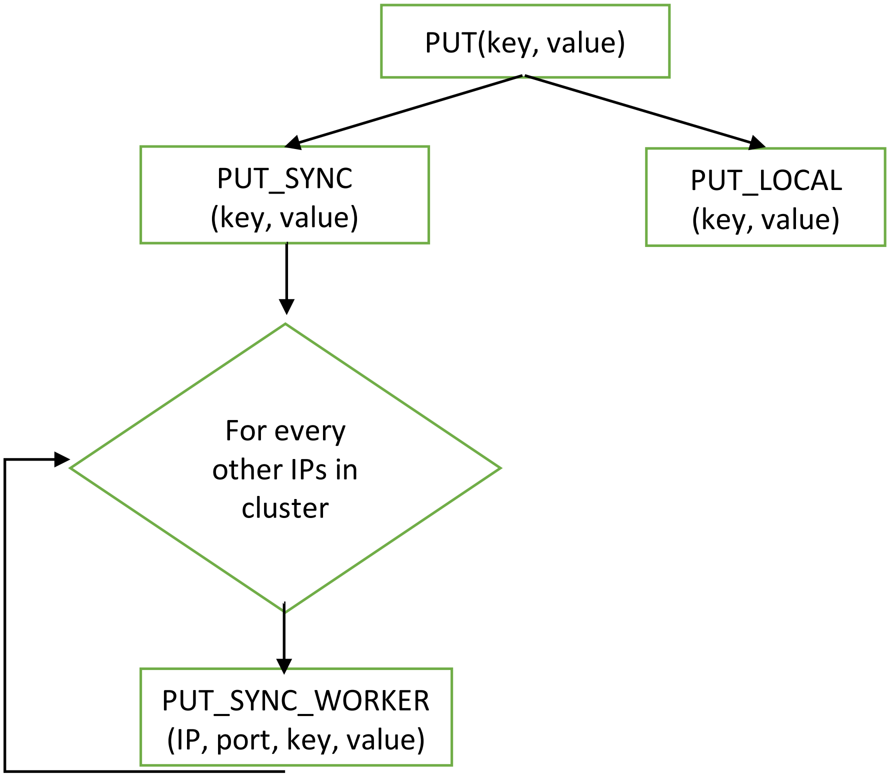
jiachl5@172.22.71.38 (n12) runs thrift_server_38.py

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']
```

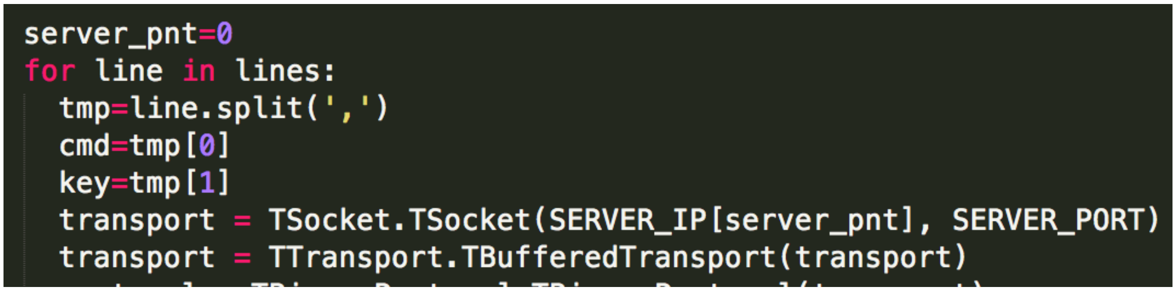
To avoid an endless loop of repeatedly calling PUT function between servers, we create a PUT_LOCAL function which only executes PUT locally. The following flow chart reveals the detail of the PUT function. The structure of DELETE function is the same as the one of PUT. Since presumably every server never fails, no more error handler is implemented.


PUT(key, value)
PUT_SYNC (key, value)
PUT_LOCAL (key, value)
For every other IPs in cluster
PUT_SYNC_WORKER (IP, port, key, value)

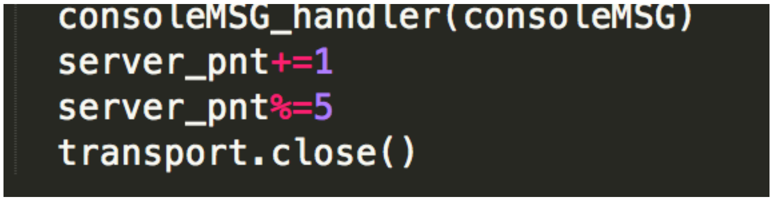
On receiving an RPC PUT request, the server will call two functions: PUT_LOCAL to execute locally and PUT_SYNC to synchronize among other cluster servers in a 2PC way. What PUT_SYNC do is basically sending a PUT_LOCAL RPC request to every other server in the cluster and summarizing those returned messages into a synchronization result. All these work are done by PUT_SYNC_WORKER.

ii) Client

jiachl5@172.22.71.37 (n11) is used as a client. There is no big changes in the client server. We simply send the requests to every server in the cluster in a round-robin way, as the following picture shows.



```
server_pnt=0
for line in lines:
  tmp=line.split(',')
  cmd=tmp[0]
  key=tmp[1]
  transport = TSocket.TSocket(SERVER_IP[server_pnt], SERVER_PORT)
  transport = TTransport.TBufferedTransport(transport)
```



```
  consoleMSG_handler(consoleMSG)
  server_pnt+=1
  server_pnt%=5
  transport.close()
```

Acknowledgement

Jiacheng and Ping discussed the design options and implement the server together. Tests are completed by Jiacheng and Documentations are done by Ping.