Executive Summary

Jiacheng Liu, Ping Han Lei

Assignment Overview:

A client can communicate with a remote key-value database in three ways: through UDP, TCP or RPC. We adopted python2.7.12 and thrift 0.9.0 for this project. Node leiph@n01, leiph@n02, jiachl5@n14, jiachl5@n15, jiachl5@n16 are used in the experiments. We also use pickledb 0.6 as our local key value store on the server. For each approach, we write a server.py on one node and have client.py running on another 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). Also, the same commands are executed locally on a MacBook to provide a baseline of the database performance. The execution results indicate that TCP is the fastest and RPC is the slowest in all experimental settings. We're expecting the RPC to be the slowest one and UDP the fastest, however, surprisingly, UDP is slower than TCP in most cases. Performance Analysis has more detailed discussions. We believe the purpose of this project is to gain better understanding about the mechanisms and features of these three typical network communication protocols. We also had a taste of synchronization and many implementation and design details as we delved into the field.

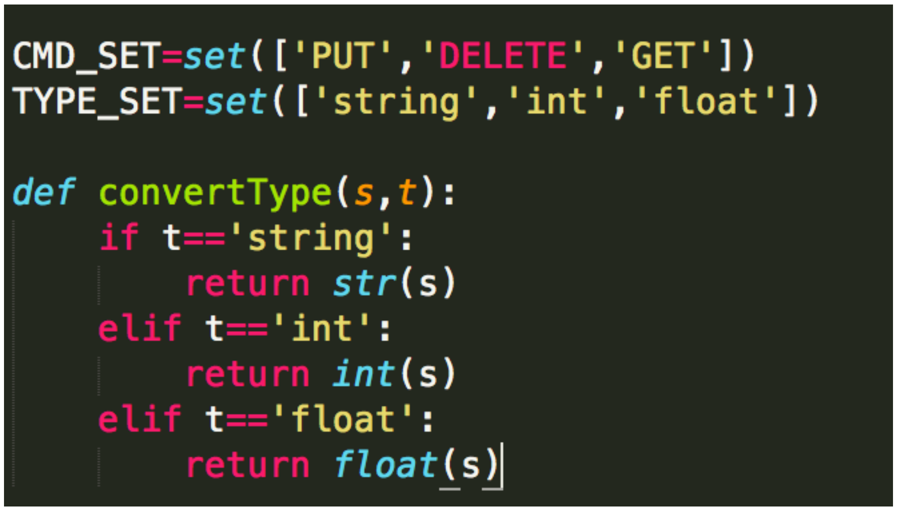
Technical Impression:

i) **Data format for TCP/UDP**

We need to consider this problem as a matter of application layer. We adopted a simple format to specify the data type of the key, value and probably other arguments transported through TCP/UDP, which means, for example, '12' and 12 are two different keys. The format is like:

[arg1, type_of_arg1, arg2, type_of_arg2, ….]

Of course, we can bypass this problem if we just simply consider all the keys to be string. The point is that different languages have different data types, you need to specify it and convert string to that desired data type on the server. Similarly, the return data type also needs to be specified. While in RPC, we explicitly specified the data type in '***.thrift file***'. We have implement three basic data types for the moment, and maybe more to come in the future.



```python
CMD_SET=set(['PUT','DELETE','GET'])
TYPE_SET=set(['string','int','float'])

def convertType(s,t):
    if t=='string':
        return str(s)
    elif t=='int':
        return int(s)
    elif t=='float':
        return float(s)
```

ii) **Return Data type in RPC / Data type binding**

This is our first time to work on thrift. We have to say that it is amazingly handy! Meanwhile, it seems like not many original python data types are supported. May be C/C++ or Java are the best working partners with thrift or maybe we just need more time to get to know about Thrift. Definitely, we're interested in learning more about Thrift in the future.

iii) **Exception Handler**

If one runs into the detail of the whole process of TCP/UDP transportation, one will find himself/herself surrounded by all kinds of exceptions which will probably appear. We handled input errors, data type errors and some of the database errors. Still, there're a lot to be done to make our programs more robust.

iv) **Mutex, Multithread test / Python Multithread**

We have implemented multithread and mutex on both UDP and TCP servers, but haven't got the chance to test this trait yet. Part of the reason is because, as we all know, python's multithread is a joke, it's pseudo multithreads. ☺

v) **TCP/UDP Mystery**

Why UDP is outperformed? Due to unknown network conditions, we can't name a specific reason to account for the TCP/UDP Mystery. Maybe it's just because everyone run their programs at the last day before due. It would be interesting to understand this phenomenon.