**Proxy herd with Asyncio in implementing TCP connection**

**CS 131 UCLA**

Asyncio – Asynchronous I/O, event loop, coroutines and tasks is a great library to build a simple client server based connection because it equipped with great built in functions, most of which are single threaded and has capacity to prevent blocking because of asynchronous nature. Also, in contrast to other library, this one doesn’t have a data race. However, for the multithreading programming this library is not safe because it is not a thread-safe. It also seems slows because of python dynamic programming as it types check at the run time.

1. **Introduction**

Asyncio library is a new framework and only work on python 3.4 and later version. This library is considered the replacement for the Twisted framework. Comparing Twisted framework, there is places where to using asyncio is very convenient because of its higher functionality built in functions. It is also considered fast because it is a part of Python 3 standard library.

1. **Client and Server Setup**

**Servers**

Goloman

Hands

Client Holiday

Google

Wilkes

Welsh

We are going to have five different servers namely Goloman, Hands, Holiday, Wikes and Welsh. They are set up in such a way that everyone can communicate with its neighbor and

and the communication is bidirectional. Same way, client can talk to any of the given server and request any info they want. On the other side of the servers, there is going to be a google server. Our servers can directly talk to the google server and can request the data to get fetched. Once the data is fetched from the google server, it then sends the fetched data to the client that requested it. In the project, our servers will request the query using the API to google. Furthermore, client can talk to server using two command IAMAT and WHATSAT.

1. **Implementation**

Client can talk to server using two different command IAMAT and WHATSAT. However, WHATSAT query need an extra information in order to talk to the server. It can be send via using IAMAT command. Being said that, client need to IAMAT query first, then only WHATSAT will work.

Once the server gets the information from the client, it will parse the information and store each token. It will first check what kind of command is requesting, based on the instruction command, it will talk to google server if it need to, to get an extra information from it. Hence, in order to talk to google, servers use google API

***3.1 Network***

When client request servers, it uses 3 different formats, two of those are command namely IAMAT and WHATSAT, the third one is AT. Using IAMAT, client can request to server. Using WHATSAT, client can request the places around the location that already sent by using IAMAT. AT, facilitate propagation information from server to client. It is possible that client command would be invalid incase doesn’t follow the right procedure. For example, without requesting IAMAT first, if request WHATSAT, server would echo ‘?<message>’, that means command is invalid. Those implementations are done by using loop that asyncio library has provided.

***3.2 Propagation***

In Asyncio library, I have used the method like BaseEvenetLoop and Transport to propagate information. But I found that, these functions behave differently different where it has been used. Thus, it is good in a sense that it gives more flexibility of modifying the library in a way that fits the requirements, and can also use to achieve desired event loop. As an example, semantic type-checks can be removed to ensure reliability. But, for the simple program like this project, it is unnecessary to modify any method.

***3.3 Protocols and Event loops***

Asyncio base class is defined by the basic interface, Protocol, namely asyncio.Protocol. Base class also has a basic method like connection\_made, data\_received, and connection\_lost. These three methods automatically called when define event loop using the method like create\_connection. To define event loop we called get\_event\_loop() from ayncio library. A new Protocol instance is created every time make a connection. The Protocol received data via connection and then abstract is using transport class. Hence, Protocol created every time connection is made. Reading data is asynchronous, hence first reads all the data before it calls next event. Protocol then synchronously call next available event. When Protocol receive the instruction, it schedule those in a patterns or iterations. It sends events signal to the scheduler when the current task finishes to execute next instruction of the scheduler. The inner implementation for the loop is designed in such a way that it records the scheduled callback, it then collects all the all finished call backs. Doing so it knows that which function needs to execute after current instruction finishes as it knows all the information regarding currently executing instruction.

1. **IAMAT**

This is called command. It has given total of 3 parameters with it.

IAMAT <ClientName><latitude/longitude><Time>

For the client, it is the first sept to interact with the client. Command ‘IAMAT’ must be specified for the server to read the query. First parameter is client name i.e. the name of the client. Second parameter is location coordinate in the form of latitude and longitude in degree. The third parameter is time in ISO 6709 format expressed in nanoseconds since 1970-01 00:00:00 UTC. Time is a record as when the client sent the message to the server.

Implementing IAMAT, it helped to understand the basics behind synchronous callback functions. Because of the synchronous nature of the callback, after client sends data to the server, data\_received function execute that reads the data sends from server. If the IAMAT is not valid, server will reply with the <? ……> format. If data is valid, server will response the query with ‘AT’ in the front of the data. IAMAT command is first steps to communicate to the server in this project for the client.

1. **WHATSAT**

‘WAHTSAT’ is also a command like ‘IAMAT’. In the information, the command needs to be specified at the very first of the query to consider it a valid query. It also has 3 parameters and the format is given below.

WHATSAT <ClientName><Radius><Places data within>

Here ClientName is the same as IAMAT. Second parameter is a radius in kilometers from the client and third parameter is the places data within that radius of client. In contrast to IAMAT, it first verifies the client name that stores in the servers when called IAMAT. If the client name is verifying then it echoes error in the form of with a line contained with question mark(?) i.e. <?.......>

Once it verifies, it then sends info to google to get the places in the specified address using JSON. It uses API to fetch the information. The result then sends in the format of ‘AT’ beginning at the very first of the line, then bunch of information after it. To connect with google, used Async.open\_connection method with API and JSON. When called this method it returns with reader and writer pair. It will be slow sometimes since it has to communicate with google and fetch data from it. It causes to block the execution loop if it’s synchronous execution. To get around this problem is to parse the data during call back in the data\_received method. To finish call back, the alternative way is to call ensure\_future() that helps to finish\_call back function completely without blocking in the event loop. I have also used

await that helps with the waiting problem for the Google places in the API connection.

Couple of method I have used are coroutine like open\_connection and reading methods. They are called using ‘yield’. Using yield helps to transfer control of await to the coroutine. For the event loop, this point is used as a callback point. So, event loop yields and transfer program execution to its callee. Hence once the result creates, coroutines executes in the even loop asynchronously and call the next segment of the coroutines. Doing so, there prevent the dead lock/blocking because reads, writes will perform asynchronously. HTTP connection is being made with Google places API using ‘yield’ and information’s are parse between connection and connection established with Google API. On the other hand, coroutine do extra work when connection is made between client and server by closing the connection because Protocol unable to detect the time coroutine finishes.

1. **AT**

‘AT’ is different than other message the discussed above. This is not sent by the client, but it this message is propagated to all the neighbor servers. This message is created by the server upon request received from the client. This message is sent after only when serer performs validation based on the cache or rules mentioned in the spec.

‘AT’ is being inserted into the string of message implies the return request from the server. The format of AT is as follows

AT<Server Name that reply><Client Name><Places Latitude/ Longitude> <Time>

For IAMAT message, the recipient server is the server that receive the IAMAT query, Latitude/Longitude are the one that sent with the IAMAT query and Time fields is the one reported by IAMAT. But for WHATSAT, Server name with AT is the one that received the IAMAT from the request server, client name is the client who requested WHATSAT and location is the location of the client that searches the places.

1. **Python Vs Java**

As previously mentioned in several places that Python is dock typing, and dynamically type i.e. type checking taking place in the run time. Whereas, Java is a static type checking, i.e. type checking is taking place in the compile time. So, the why would we use asyncio? It is because this library helps to write asynchronous application. It gives an easy way to handle server having many concurrent users. It also handles race condition differently. Especially with the lightweight server, parallel computing is very hard because of race condition. Using asyncio, I realized that the code is more clean and readable that helps to write a less error-prone code.

Java is using garbage collection, as well as Python does too. However, the method using garbage collection is different in each language. Python using the reference count method to handle the objects whereas Java uses the GC(garbage collector) that destroy the objects that are no longer in use in a specific time. There is a drawback of using the reference count garbage collector. It can have a circular reference sometimes and cause the memory leaks. For example, A reference B, B reference C and C Reference B, it is a circular reference and memory will never be freed and hence memory leak. Another drawback is that, reference counting will make execution slower. It is because, the interpreter/VM needs to see if the counts have been incremented or decremented as program referenced or differenced but in garbage collection doesn’t need to. Also, it is possible that garbage collection can be done in separate thread as well.

1. **Conclusion/Summary**

To build the simple program like this, asyncio library is very useful and efficient. Making an infrastructure like this is a simple lightweight server within an application server heard which has bidirectional connection and each server propagate client’s information in the way server implemented. Asycio library is best for the application because ‘loop’ is very powerful tool and it is independent from the others and performs asynchronous task using coroutine yields from. In the server Protocol, it is considered a very good expandable as new message can be add easily by calling the method called data\_received. On the other hand, aysncio is consider faster compare to other as python features native supports for coroutines.

When it comes with the memory, python has a very clear memory management system. It has garbage collection like Java, that keep track of unused objects by the reference count and make memory immediately available for next object to use. In Asyncio it is helpful because the Protocol instance are creating and destroyed constantly. Garbage collection lower the overhead and hence newly created Protocol instances doesn’t require to waste time to search for memory and hence execution is faster.

Being said that, there is also possibilities that things can be wrong when no precaution taken. As we know python is dock type and thus asyncio library comes with the semantic checking techniques that check for the correct argument types. Also, Asyncio doesn’t support multithreading, the event loop, read and write are not thread safe. Hence, coroutines would not writing and reading global.

As part of the server herd, we can use Node.js JavaScript framework as an alternative to implement server. But Node.js is used mostly to implement web application as opposed to asyncio which is used for scheduling the local task that are asynchronous. Other different node.js has it that it is an event driven program that has callback functionality where callbacks occur right after event is called.

**Reference:**

[1]. <https://docs.python.org/3/library/asyncio.html>

[2]. <https://ehmatthes.github.io/pcc/cheatsheets/README.html>

[3]. <https://console.developers.google.com/cloud-resource-manager>

[4]. <https://developers.google.com/places/web-service/search>

[5]. <https://github.com/python/cpython/tree/3.6/Lib/asyncio/>

[6]. <https://en.wikipedia.org/wiki/Transmission_Control_Protocol>

[7]. <http://www.json.org>

[8]. <http://asyncio.readthedocs.io/en/latest/why_asyncio.html>

[9]. <https://stackoverflow.com/questions/21934/why-java-and-python-garbage-collection-methods-are-different>