Proxy Herd with Asyncio

Tianyang Zhang 404-743-024

Abstract

The purpose of this project is get the performance and writing easiness of Asyncio by write an example proxy herd. Asyncio is a powerful tool to build interprocess communication and networking programs, it allows task or requests processing asynchronously, which could efficiently reduce the response time of the server.

In this project, we built the source code for the server, and the source code contains 5 fixed server names which bounded with 5 different port number. Those 5 servers would propagate the location information received from clients, and clients could send commands to servers to update their location or get other clients' location who are near by.

1 Introduction:

The implementation by using Asyncio is based on 'event_loop' and 'Tasks'(coroutines). The event loop is similar as a while loop, which continue running and will response when tasks detected.

The 'ServerProtocal' class is the base class for server.py, I use the AbstractEventLoop. create_server() method from Asyncio to create the server, and using 'ServerProtocal' as it's protocal. There are two other classes helping the ServerProtocal, which are 'ServerLog' and 'PropagateProtocol'.

The 'ServerLog' class is used to open or create the log file for the server, and it provide a file descriptor for 'ServerProtocal' to do the write operation to log file. The 'PropagateProtocol' is used to help the inter-server communication. This protocol will be used by the AbstractEventLoop. create_connection() method which is called in Propagate() method in 'ServerProtocal'. PropagateProtocol' contains the tools to send data to other server.

The event loop is assigned globally(which means asyncio.get_event_loop() is not called in main()), the reason is that I need to use the same event loop object inside the 'ServerProtocol' for creating new tasks, such as propagate information to other server, or making request to Google API. The main() method would read in the argument passed in by command line, and use it as the server name which the user want to open. Then it will create the server as a co-routine and start the event loop until keyboard interrupt detected.

Moreover, there are several helper function declared globally, in order to help other methods inside protocol classes.

2 Pros and Cons:

2.1 Pros:

2.1.1 Pros on Structure:

From the introduction, we could know that the structure of a server built with Asyncio is very simple and clean. The most works I done in the project are methods that process client's data, or say add features into the server. By using Asyncio, we don't need to spend lot of time and effort on building up the server itself, such as creating connection, connection listening, or data transporting etc.

2.1.2 Pros on Efficiency:

The core idea of Asyncio is same as it's name, do tasks such as I/O asynchronously. We could use 'async' key word or '@asyncio.coroutine' to define a method as a 'coroutine', which the method will return a coroutine object. Then we could run this coroutine asynchronously by using key word 'await' or 'yield from'. This would make the response time to client much less than synchronously process the requests.

For example, there are 2 clients and both of them are trying to download a million files from server. If the function which handling downloading is blocking(synchronously), then client #2 has to wait a long time until client #1 done it's downloading, and that would be terrible when multiple clients making request at the same time. By using Asyncio coroutine to handle the downloading, all of the requests would be responded and started at the same time.

Therefore, proxy herd with Asyncio is much more efficient than regular synchronously running proxy herd.

2.1.3 Pros on Easiness:

It is not very hard for a programmer who never use Asyncio to understand the basic concept of Asyncio, and it is not very hard to do the basic implementation.

As long as the programmer understand the idea of event loop, tasks and coroutine, he/she can easily change a fully synchronize server program to core-asynchronously server program since Asyncio packed up the methods into 'yield from'/'await' key words.

2.1.4 Pros on Errors/Exceptions/Type Checking:

The idea of event loop provide a strong protection on program crashing when error occur. If one method has error occur, it would not affect the whole program or crash the server. This is also related to python's type checking. Since python uses dynamic type checking, the program won't report error until the program runs into specific line.

Therefore, even if there is a buggy method, the server could still run and other methods and features could still be use. When error occur, the event loop would not stop but will only terminate the current running method, clients could still make request to server to use other features. This identity makes the program written by Asyncio much more stable, and thus python's dynamic type checking would not be a problem but becomes a advantage.

2.2 Cons:

2.2.1 Cons on Low Level Implementation:

Asyncio is highly packaged and highly encapsulated, it provides lot of APIs to do high level implementations.

However, this would become a problem when we want to use Asyncio to develop low level implementation, such as making request in TCP connection. Asyncio does not provide request method itself. Therefore, if we not using other module, it would not be easy to write low level methods.

2.2.2 Cons on Unity

If we use Asyncio to write an asynchronously method and we want the get the return value of this method, then both of the caller and callee has to be asynchronously. Therefore, it is hard to use Asyncio to write a program that partial asynchronously.

3 Problems Ran Into:

There are three critical (or say hard) point when I was building the proxy herd.

First is the identification of each server. Since there is no such API to get and set a server's name, I have to add an attribute of server name inside the protocol, and I have to create a global dictionary to bound the port number and server name together.

Second one is building up the propagation between servers. Since Asyncio only allow one event loop running at the same time in single program, we cannot use the same way as client sending message to server on server to server communication. Therefore, I spend a lot of time on correctly create coroutines and adding tasks into current event loop.

Third is build up a method to do HTTP request by using Asyncio only. There is almost no resources about building up HTTP GET feature by using Asyncio. Also, since the encapsulation of Asyncio, there is not much API I could use for the GET request method, and hence it needs the programmer to understand Asyncio deeply to complete the implementation.

4 Python-Base vs. Java-Base:

4.1 Type Checking:

As I mentioned in 2.1.4, the dynamic type checking of Python is an advantage vs. Java's in this case since the most important ability of proxy herd is stable. And because of the todo-list and the identity of event loop, the program would not crash easily when one of the methods is buggy.

4.2 Memory Management:

In Java, garbage collection is relevantly slow comparing to Python when dealing with large amount with data throughput. In proxy herb, the servers would continually run 24/7 in a long time, most of the proxy herb would deal with lot of data.

In this case, the advantage of Python's garbage collect would be bigger, and hence Python is a better choice for the proxy herb than Java.

4.3 Multithreading:

For multi-threading, Python is much worse than Java because of the GIL(Global Interpreter Lock) of CPython.

Python cannot run multiple number of threads at the same time when using a multi-core processor. Every thread needs to get the authorization from the interpreter, and other threads would be waiting at the mean time. Java does not have this issue. Java could use the fully resource of the multi-core processor and hence the multi-threading program in Java is much faster than Python.

But this is not a problem for using Python is this project, because we could make up with coroutines or multi-processing to solve the asynchronies problem but not using multi-threading.

5 Asyncio vs. Node.js

Node.js is a JavaScript framework designed for web based on Chrome's V8 JavaScript engine, it has different performance when dealing with different kinds of tasks comparing to Asyncio.

Programs written by JavaScript using Node.js framework is harder to read and understand because of it's anonymous function and closure. Comparing to Asyncio using Python, Node.js is better for simple logic programs such as Web, and Asyncio is better for complex logic programs such as server backend.

6 Conclusion:

Asyncio is a very good framework for Python to build asynchronously programs. It welly makes up the disadvantage of Python's multi-threading problem, and it speed up the synchronize programs a lot.

Because of the encapsulation and packaging, Asyncio is a framework that easy to learn for programmers, and it provides lot's of powerful APIs to use, which could makes the lines of code of programs more clear and simple to read.

The performance of proxy herb written by Asyncio is also good, it could nicely handle the works and request made by clients.

As an overall conclusion, Asyncio is a suitable framework for this kind of application.

7 References:

[1] Python3.6.3 Documentation - 18.5. asyncio — Asynchronous I/O, event loop, coroutines and tasks, https://docs.python.org/3/library/asyncio.html

[2] Google Place API, https://developers.google.com/places/web-service/search

[3]Aiohttp Client Example, https://aiohttp.readthedocs.io/en/stable/

[4]Python3.6.3 Documentation - 16.3. time — Time access and conversions, https://docs.python.org/3/library/time.html

[5]Node.js.org, https://nodejs.org/en/about/

[6]Introducing JSON, http://www.json.org/