Research on Implementation of Server with asyncio Comparing with Java and Node.js

Yuxin Wang 905129084
University of California, Los Angeles

Abstract

In this paper we introduced the server herds application implemented with asyncio library in python3.8. This application is used to experiment concurrent server to server and server to client communication. It is capable of processing I/O data reasonably, as required in the project spec. This paper also illustrate the implementation details, especially with the changes in the newer version of asyncio library in python3.8, as well as a comparison between Java and Python, and a comparison between asyncio and Node.js, in terms of implement concurrent server applications.

1 Introduction

In this project we explored the possibility of implementing a server in python with asyncio asynchronous networking library, in order to realize concurrent communication between servers and clients. “asyncio is used as a foundation for multiple Python asynchronous frameworks that provide high-performance network and web-servers, database connection libraries, distributed task queues, etc.”[1] This implementation consists of five servers with server IDs ‘Hill’, ‘Jaquez’, ‘Smith’, ‘Campbell’, ‘Singleton’ that follow the pattern:

- Hill talks with Jaquez and Smith.
- Singleton talks with everyone else but Hill.
- Smith talks with Campbell.

All of the five servers have the ability to accept TCP connections from clients, communicate and propagate client location to reachable neighbors via TCP connection, and response to proper location query with a response consists of a Nearby Search request provided by Google Places. Communication history with clients and neighbor servers will be recorded in a log file named [SERVER]_logs.txt.

2 Environment

The experiment is conducted with python3 on server lnxsrv09.

3 Implementation

3.1 implementation overview

A server could be estabilished by calling:

python3 server.py [SERVER]

with server names mentioned from previous sections. A asynchronous server is created by calling asyncio.start_server() with some IP address, assigned port and co-routine handle_echo. This co-routine is called whenever a mew client connection is established, and it will be scheduled as an awaitable task, in order to realize multi-connection between the server and several clients. But, according to the library description of asyncio, these tasks are not thread-safe, namely, if there are more than one threads created, then we need to be careful about possible race condition. Here, only one thread will be used to ensure safe concurrency.

3.2 handle_echo

The co-routine handle_echo is responsible for reading data from client, then call another function validation() to process the data, and finally send proper responses

to client in a format required by the spec. In validation() function, firstly the message will be recognized by its type from the leading word of the message. Valid types including WHATSAT, IAMAT and FLOODING will be handled by corresponding handler functions. All other message will be considered as in proper message and send back to client with a leading question mark added in the original message. Each message handler will process the data according to the spec, and return a proper response. Specially, handle_whatsat() will call aiohttp.ClientSession() to send a HTTP GET request to Google servers in order to get locations from Google Places. Because asyncio only supports the TCP and SSL protocols, aiohttp library is used to create and send an http requests.

3.3 Flooding

A simple flooding algorithm is implemented to update client information between servers. Server to server connection is realized by calling asyncio.open_connection(). When a server receive a new IAMAT message from the client, it will send a AT response to the client, meanwhile, it will also flood the AT response to its neighbor servers with its name added to the end and a FLOODING type added to the beginning of the message. When a server receives a FLOODING type message, it will check if this message contain a name of itself, if not the server will use this message to update client information and keep flooding this message to its neighbor servers, otherwise this message will be discarded.

All above functions will participate in keeping a log file of transactions and I/O data.

3.4 asyncio Library

Using asyncio framework to implement server herds is not hard. Each connection between clients and other servers will be created as a coroutine. Async def statement allows the program to use these function as co-routine[3], which could transfer control to others while not active, in order to realize concurrency. Furthermore, using keyword await makes it possible to control the execution between coroutines.

However, as mentioned from previous section, some part of asyncio framework is not thread safe, therefore it is important to notice that multithreading could cause race condition, and synchronization primitives are needed to solve this problem. Another point that is worth to mention is that asyncio only support SSL and TCP connection, but it is compatible with other libraries that support HTTP connection such as aiohttp. Personally, I have not come across a memory issue during test, but it is reported in some other project that large amount of coroutine could cause memory leaking due to waiting for some unreachable resources at the same time.[4]

In Python3.8, lots of the functions in asyncio have deprecated loop parameters, because loop arguments in newer python versions are redundant, due to the increased reliability of get_event_loop(). It is no longer required to pass a global object loop throughout all layers of abstraction. And some functions such as create_task are now available as stand-alone functions, which provide extra simplicity and efficiency.

4 Comparing with Java

Python uses dynamic type checking while Java uses static type checking, this different make it harder to use python to construct server application in larger scale, as run time type checking could be more buggy and more hard to debug. More over, python project may suffer from memory issue such as garbage collector can not resolve circular reference while Java project could perform better on this issue. Also, for the multithreading issue mentioned from previous session, Java could handle it better than python, because Java

support multithreading and provides multiple tools and primitives to ensure a function is thread-safe.

In conclusion, although Java application requires more work, it is more powerful on memory management and multithreading, therefore it is more recommended to build large scale server application in Java.

5 Comparing with Node.js

While both asyncio from python and Node.js support event-driven single thread concurrency, there are some difference. Firstly, asyncio framework does not perform as well as Node,js in terms of memory management. Node,js would run faster on memory intensive project, not to mention that asyncio framework could suffer from memory leaking. On the other hand, in python we explicitly call event loop to run async concurrent block, but in Node.js this process is simplified. In the same level of concurrency programs, python architecture is usually more complicated to write, however, this extra complicated architecture make the implementation simpler and easier to reason when constructing more practical and complicated applications.[5]

Conclusion

In conclusion, asyncio framework in python3 is a handy tool in terms of implementing lightweight server herds. Asyncio library provides the ability to perform concurrent communication between clients and servers via TCP connection, but HTTP connection must require extra library. Comparing to Node.js, asyncio framework makes it easier to build more complicated project reasonably. However, Java is still the better choice for large scale server application comparing to python, because of it is more powerful in multithreading and memory management.

Reference

1.https://docs.python.org/3/library/asyncio.html

2.https://web.cs.ucla.edu/classes/spring20/cs131/hw/pr.html

3.https://www.python.org/dev/peps/pep-0492/

4.https://tech.gadventures.com/hunting-for-memory-leaks-in-asyncio-applications-3614182efaf7

5.https://medium.com/@interfacer/intro-to-async-concurrency-in-python-and-node-js-69315b1e3e36