CS 131 Project: Proxy Herd with Asyncio

1. Introduction

Wikipedia and its related sites are implanted using the Wikimedia server platform, using GNU/Linux, Apache, MariaDB, and PHP+JavaScript also known as the LAMP stack. With new services, this architecture seems to be a bottleneck, due to increased updated articles, additional required protocols, and increased mobile clients. This paper will analyze the utilization of asyncio, an asynchrounous networking library, and whether it will be a good candidate for the new services, as opposed to Node.js and Java.

2. Asyncio

According to the asyncio documents, it is a library used to write concurrent code using async/await syntax. This library provides both higher and lower level APIs for developers to use. With the higher-level APIs, the library allows users to run corutines concurrently, perform network IO and IPC through streams, control subprocesses, distribute tasks via queues, and synchronize concurrent code. As for the low level APIs, users are able to create and manage event loops, implement efficient protocols using transports, and bridge callback-based libraries and code with async/await syntax. Below are functionalities important to the testing of the asyncio library.

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

2.1. Event Loops

A core feature of the asyncio programs is the event loop. Event loops are the managers of asynchronous functions and subprocesses. The event loop furthermore handles network IO operations. With the event loop, programmers are able to define and schedule when coroutines should be called and completed.

2.2. Coroutines

Coroutines, which are generalization of subroutines, are key to the asyncio library because they are allowed to run asynchronously, and thus concurrently. Coroutines are defined by the statement "async def". By defining a coroutine with "async def", the function can only be run as a coroutine, and not as a regular function.

To call a coroutine, there are three options:

- "asyncio.run(coroutine_name)"

  Given the coroutine, the function runs the coroutine, creating a new event loop and closing the event loop after completion of the coroutine. According to the Python documents, this should only be called once in a program and should be the main entry point for the asynchronous program.

- "await coroutine_name()"

  The await keyword tells the program that the following function is a coroutine. "Await" can only be used inside of coroutine. This is because, by using the "await" keyword, the outside coroutine is blocked until the completion of the referenced coroutine. The benefit of awaiting coroutines is that other coroutines inside the same event loop can run concurrently.

- "asyncio.create_task(coroutine_name)"

This function wraps the coroutine into a Task in which it schedules the execution of the coroutine. Similar to await, the Task suspends the current coroutine and waits for the Task to finish before starting the coroutine again.

3. Testing

To test the capabilities of the asyncio library, I implemented a simple application server herd. The server herd consists of five servers: Hill, Jaquez, Smith, Campbell, and Singleton. The servers are connected to each other through the following rules: Hill is connected to both Jaquez and Smith, Singleton talks to everyone except Hill, Smith talks with Campbell. Each of these connections are bidirectional thus, all the servers are connected, even if the server doesn't have a direct connection with another. All of the servers accept TCP connections from clients. Clients are able to send their current location and query the server of places near the client's location. Servers respond to clients with the correct location or a "?" for invalid messages. As for the herd, when a server receives a message, the message is propagated throughout the herd such that every server receives the client's message.

4. Implementation

The program takes one argument to run: the name of the server. After a valid server name is received, the name and port number corresponding to that name are passed into the class Server. The class contains three key functions:

- run_forever()

  This function is called inside the main function of the program. This function handles the creation of the server and the main event loop to handle the following coroutines. The event loop runs until a KeyboardInterrupt is detected. When the server is started, the callback function is the coroutine handle_echo()

- handle_echo()

This is the coroutine that handles messages. This function parses messages received and directs the server to act correspondingly, such as updating which servers are still connected or flooding it's connections with the message.

- flood_neighbors()

This is another coroutine called inside of handle_echo() that handles propagating the message to other servers based on the connections as defined in the previous section. The function loops through each of the server to server connections, opens a connection, sends a message, then closes the connection.

5. Analysis

5.1. Application Suitability

Python and the asyncio library offers a suitable solution for the new services of the Wikimedia servers. For single-threaded applications, the asyncio library allows for programmers to easily create and handle concurrent operations through event loops and coroutines. With both high level and low level APIs, programmers have much freedom to handle client requests. With concurrent programming, messages to multiple servers are able to be handled at the same time, thus increasing speed of message exchanges between servers and clients.

5.2. Type Checking (Python vs. Java)

Python is an interpreted language, thus data is dynamically type checked, as opposed to Java which is statically type checked. This means that the type of data is determined when the program is running for Python. For Java, the type of data is determined when the program is compiled. Because Java requires variables to match types and is checked during compile-time, the typing of variable will always be known, thus there needs to be less error handling. For Python, however, whenever variables are accessed, they always need to be checked for the correct type, such as timestamps and coordinates being floats. Because of the dynamic type checking of Python, there is less protection against invalid variable types and errors can occur more frequently if programmers are not meticulous with passing variables.

5.3. Memory Management (Python vs. Java)

The difference between the memory management of Java and Python is how each of the languages free memory at the end of a program. Python handles memory by having a reference count for every object created. The reference count keeps track of the amount of references to the same object. When the object has zero references to it, then the object is then allowed to be deallocated. This becomes an issue when there are cycles between objects, causing each of the objects to be non-zero and unable to be deallocated. As for Java's memory management, Java checks all referenceable objects and deletes all objects not able to be referenced. Java's memory model is able to handle cyclical object references, and thus has a better memory management system.

5.4. Multithreading (Python vs. Java)

A big difference between Python and Java is the multithreading capabilities. Python, by design, does not handle multithreaded programs well. Python uses a Global Interpreter Lock (GLI) which allows only one thread to access the Python interpreter. Because of this, multithreading is extremely inefficient. As for Java, the Java Memory Model (JMM), is designed to handle multithreading thoroughly. Although the server program above does not need to be multithreaded, if the server were to require multithreading, Python would not be a viable solution.

5.5. Asyncio vs. Node.js

Both asyncio and Node.js provide solutions to asynchronous programming. However, Node.js was designed by default to be asynchronous. The benefits of Node.js being by default asynchronous is that programmers do not need to define functions as "async". For Python, coroutines must be defined as async in order to be used. Besides legibility, the other difference is that when programming with Node.js, the entire program is the event loop, thus programmers do not need to worry about creating the event loop, as Python programmers do.

6. Conclusion

With the testing implementation, Python and the asyncio library provide a suitable solution to client-server communication. In the testing implementation, the program needed only to be single threaded, thus Python is acceptable. In terms of the future, if the Wikimedia servers expanded to a larger scale and required multithreading, Python and the asyncio library would not be viable. However, for what was requested Python was "good enough".