CS131 Project Report

Abstract

In many applications, such as news feed, the server needs to accommodate for frequent updates and mobile clients, and the traditional ‘PHP+JavaScript’ application server has become the bottleneck. To solve this issue, a different type of service architecture called ‘application server herd’ has been developed. In this report, we investigate into the Python ‘Asyncio’ asynchronous networking library, and through developing a simple proxy herd for the Google Places API, we analyze the pros and cons of the framework, make a simple comparison between Python and Java/Node.js, and conclude that Python Asyncio framework is a competent choice for this kind of application.

1 Introduction

Before we describe our results in analyzing the asyncio library, we will explain the structure of our simple proxy herd.
Firstly, our proxy herd consists of five servers that communicates with each other. The following table describes the communication pattern.

Server ID	Communicates With
Hill	Jaquez, Smith
Jaquez	Hill, Singleton
Smith	Hill, Campbell, Singleton
Campbell	Smith, Singleton
Singleton	Jaquez, Smith, Campbell

Table 1: Server Communication Pattern

These five servers are connected to localhost ‘127.0.0.1’ and assigned ports 11670-11674 respectively. Each server accepts TCP connections from clients that emulate mobile devices with IP addresses and DNS names. There are three types of messages sent between the servers and clients.

1.1 IAMAT message

A client can send its location to server by sending a message in the following format:

IAMAT *Client_id* *Coordinates* *Time*

After the server receives the message, it will reply to the client with a message starting with ‘AT’:

AT *Server_id* *Time_diff* *Client_id* *Coordinates* *Time*

and, using a flooding algorithm, communicate this message with other servers.

1.2 WHATSAT message

A client can also query for information about places near other clients’ locations by sending a message starting with ‘WHATSAT’:

WHATSAT *Client_id* *radius* *max_result*

and the server will reply with the most recent location reported by the client, followed by the JSON-format message from Google Places request.

1.3 AT message

Apart from senting ‘AT’ message to client (as explained in part 1.1), a server can also send this message to other servers to flood the information that they receive.

1.4 invalid message

When encountered with an invalid command from the client, a server will reply with a line that contains a question mark, a space, and then a copy of the invalid command.

2 Review of Asyncio Framework

In this section, we will discuss the pros and cons of the Asyncio framework from our implementation of the proxy herd.

2.1 Advantages

Generally, Asyncio performs well in programs that has many IO-bound procedures, since Asyncio takes long waiting periods in which functions would be otherwise blocking and allows other functions to run during that time. For our proxy server herd, servers are constantly waiting for inputs from the client or other servers, hence asyncio enables the program to proceed while waiting for inputs. This greatly improve the performance of our servers.
On top of the benefit of asyncio, Python Asyncio Library has several beneficial features. Firstly, it is easy to use. If you do not have experience with asynchronous programming before, you will still find yourself able to define and run coroutines using 'async/await' keyword. Also, it provides high-level asynchronous API to work with network connections, which is convenient for implementing our proxy herd without considering low-level details. In addition, apart from being easy to use, the asyncio Library also provides developers with low-level APIs to directly manipulate event loops and improve the efficiency of the program.

2.2 Disadvantages

Despite the easiness of implement asynchronous IO using Python Asyncio framework, the resulting code is error prone. As a person who is just got introduced to asynchronous programming, oftentimes I would forget to add 'await' keyword before coroutines. For example, I defined a coroutine that writes message to client and wait for the buffer to drain. However, when I am using this coroutine, I often forget that it is an asynchronous procedure and treat it as a normal function. The result of this is disastrous since this coroutine would not be called at all! When running the program, Python would sometimes give a warning about this, but the program would still be interpreted and run. Python as an interpreted language does not enforce strict checking as other languages such as C do, hence for beginners writing a server program would be error prone, and it is hard to figure out the mistakes.

3 Python vs. Java

In this section, we will compare different language aspects of Python, such as type checking, memory management and multithreading, with a Java-based approach.

3.1 Type Checking

Python is a dynamically typed language, which means that the type of a variable is determined at run-time. This is convenient since we do not need to declare the type of variable when writing code in Python. On the other hand, Java is a statically typed language, which means that the type of a variable is determined at compile-time. Generally, when programming in Java, we need to figure out the type of every variable and declare the type. This is sometimes not beginner-friendly, since Java has complex relationship between objects, especially in class inheritance relations.
However, Python's approach also has its own drawback. Being flexible about typing means that the program would be harder to debug when runtime error occurs. You would have a hard time tracing code to figure out what each variable represents. For example, your code read a string to a variable and you use the variable as a number without explicitly converting it, then the interpreter would complain, and it is often hard to notice such mistake.
Java's static type checking would make the programmer spend more time to determine the type of each variable, but it also means that the code would be more robust since more type errors would be caught at compile time. Such feature would be beneficial when writing large applications.

3.2 Memory Management

Python and Java both uses garbage collection, which recycles unused objects in memory, however, their approaches are different.
Python uses the "refence counting" method, which is basically keeping track of the number of times each object is referenced, and the object is destroyed when its reference counting reaches zero.
On the other hand, Java uses the "mark and sweep" approach, which checks for all reachable

objects in the memory and remove objects that are not reachable.
Either approach has its own merits, for Python, its garbage collection method is simpler and results in faster code. However, it is vulnerable to the problem of "circular referencing", in which the reference relation of objects forms a closed loop. For Java, its method is more robust, but requires more memory to perform the actions.

3.3 Multithreading

Python and Java both supports multithreading. However, in order to support a simpler garbage collection system, Python does not have parallel threading. The Global Interpreter Lock (GIL) in Python ensures that one thread can be executed at a time. Multiple threads could run concurrently using cooperative multitasking, which means one threads runs for some time and yields the CPU to another thread.
On the other hand, in Java multiple threads could run in parallel, and the Java Memory Model controls the interaction of threads through memory. Parallel multithreading in Java has more potential than non-parallel multithreading in Python, since the performance of the program could be improved greatly, if data races are carefully prevented.

3 Asyncio in Python vs. Node.js

As two of the most popular languages among programmers, Python and JavaScript both lack the support for thread-based parallelism. As a result, asynchronous libraries are crucial in speeding up the I/O intensive programs. In this section we will compare the Asyncio Framework in Python and that of Node.js, which is a JavaScript runtime based on Chrome's V8 JavaScript engine.

3.1 Standard library functions

Python's Asyncio module has a diverse of utilities required for async programming, from high level API for coroutines and tasks, to low level API for direct manipulation of the event loop. These standard functions have brought much convenience when we were developing the proxy herd that uses asynchronous network I/O.
On the other hand, JavaScript is a very flexible language, but unfortunately, a lot of basic asynchronous utilities are missing from the standard library. This means that programmer will sometimes have to write their own functions, or to look for little packages outside the standard library to fill the need.

3.2 Ways to write asynchronous functions

In Node.js, the default way of writing asynchronous functions is to use callback functions, which means the async function will take an extra callback function as a parameter, and the callback function would be called when the async function finishes. However, the current most popular method is to use 'Promises', which is similar to Python's coroutines. And like Python's async/await keyword is built upon coroutine, tasks and futures, Node.js has its async/await built upon the promises. However, unlike in Python where we usually write in async/await, in Node.js we will usually work with promises directly. Apart from this, the approach towards asynchronous programming is similar in both languages.

4 Conclusions

Python's Asyncio library is a diverse and efficient library that provides convenience for programmers to write concurrent asynchronous code. Combined with Python's simple syntax, this library is easy to utilize for someone that is not an expert in asynchronous programming. Despite Python's interpreted nature that makes the code hard to debug, with careful checking we can easily write asynchronous programs that are both efficient and robust.

5 References

Python Asyncio Library Documentation:
https://docs.python.org/3/library/asyncio.html

Async IO in Python: A Complete Walkthrough:
https://realpython.com/async-io-python/#async-io-design-patterns

Intro to Async Concurrency in Python vs. Node.js:

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

Async patterns in Node.js: only 5+ different ways to do it:
https://codeburst.io/async-patterns-in-node-js-only-4-different-ways-to-do-it-70186ee83250