Cs131 Project: Proxy Herd with Python Asyncio

Sokchetra Eung

Abstract

There exist many popular server architectures and designs to implement web based applications through various languages. Popular sites such as Wikipedia utilizes the Wikimedia server platform which is mainly based on PHP and Javascript. While the current Wikimedia style designs provides reliability and performance, it will slow down the response time if updates to articles will happen far more frequently and the clients who update articles are mobile and using various internet protocols. In this project, Python's Asyncio module was used as an alternative to PHP-Javascript node.js. As a result, we will analyze three main components of the Python's Asyncio module, benefits and downside of the Python's Asyncio, the language's features, and the overall approach between the module and Javascript node.js.

1. Introduction

A centralized server using PHP-Javascript node.js can be a bottleneck if there are many clients requesting updates to the servers. Therefore, the project implements a prototype model, a proxy herd with five servers that can communicate with each other with various commands and messages.
The key features of the Asyncio module in python is it provides asyncrosity under a single thread and a single process, and access to sockets, client and server module and other resources. Since it is asynchronous, it can handle multiple requests from clients at the same time by only running a single thread, so it is less overhead to handle update requests to servers various clients than a normal synchronous runtime environment which has many threads.

2. Prototype

There are five servers in the Proxy herd namely: Hill, Jaquez, Smith, Campbell, and Singleton that can communicate with each other bidirectionally with the pattern. Each server is able to accept TCP connections from clients that emulate mobile devices with IP addresses and DNS names. The TCP connection is made with a websocket allocated at its ports.
Hill is assigned with port 12205. Jaquez is assigned with port 12206. Smith is assigned with port 12207. Campbell is assigned with port 12208. Singleton is assigned with port 12209.
API_key='AIzaSyCya3Brm4FSpEY8MVlxMAxpIS0810WiybM
The server also floods to neighbor servers with "AT command" when it receives a "IAMAT command" from a client.
The pattern for bidirectional communication is that:

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

When received command "WHATSAT", the server responds to the client with its positions using Google API_KEY.
The event loop of the server.py will go through the process of accepting clients, read command lines from client, check if command is valid, then process each command accordingly by send messages to clients and log the messages to the log-Servername.txt.

2.1 IAMAT & AT Message

IAMAT message is sent from client to a server to tell the server about client name, latitude and longitude, and time stamp from the command.
The "IAMAT" message has the form:
[IAMAT][client_name][latitude-longitude][time]

IAMAT is the name of the command. Client_name is the name of the client. Latitude-Longitude is the client's latitude concatenated with longitude of the location of the device in ISO 6709 notation, and Time is the time specified by the client in POSIX time, which consists of second and nanoseconds since 1970-01-01 00:00:00 UTC. After verifying the command, it replied back to client an "AT command" with format as bellow:
AT[server_name][time_different][client_name] [lat-lon] [time_sent]

Upon receiving the AT command, the server first determines if the command is redundant or out of date by checking its record whether it has received the command before and whether the command is less recent. If the command is new, it floods the neighbor servers based on specified bidirectional pattern with AT command but with different format as below:
AT[client_name][server_name][latitude][longitude][time_difference][time_sent]

2.3 WHATSAT Message

A client can send a command to a server to request location information of nearby clients around a specified client with a certain radius and a certain number of devices. The WHATSAT command has the following format:
WHATSAT [client_name] [radius] [upper_bound]

The WHATSAT command queries Google Places platform via an HTTP request to the API's endpoint by using a private API_KEY. The query is:
'https://maps.googleapis.com/maps/api/place/nearbysearch/json?location=%s,%s&radius=%d&key=%s'

The url passed in radius, longitude and latitude of the client and an API key. It looks at nearby devices around the client's latitude and longitude within the radius up to a upper bound result. After successfully querying Google places, it replied to the client with "AT message" with the same format except it includes with JSON message:
AT [server_name] [time_difference] [client_name] [latlon] [JSON_msg]

3. Review of Python Asyncio

Since there is no performance measurement of the asyncio and we did not implement Javascript node.js for comparing the performance with, it is difficult to come to the conclusion of the overall performance of Python Asyncio. However, we can discuss other topics such as the usability and the processing ability of the module.

3.1 Pros of Asyncio

Asyncio library provides a simplicity in terms of creating a server and handling requests asynchronously. It allows us to create an event loop and add in coroutines to process many requests at once. As it is processing requests from clients, it can also accept new client connections at the same time which is really good for mobile-centric server-herd architecture. Coding is relatively simple because asyncio provides easy to use APIs that allow the server to establish TCP connection with client via a reliable and compatible web-socket. It also comes with SSL support which ensures security on two parties' connections.
In contrast to JavaScript which demands creating various threads for handling asynchronous events, Asyncio allows the asynchronous event handling with only one thread which makes it generally simpler than node.js.
While HTTP request is missed in Asyncio, Python provides aiohttp library which enables client sessions within the coroutine of the server's event loop. The Json library eases the process of getting information from Google API places.

The simplicity of Asyncio's usage and the provision of Asynchronous events handling

under one thread, the TCP and SSL connection with websocket embedded with the module, and other supported libraries makes Asyncio one of the best approaches to the problem we set.

3.2 Cons of Asyncio

There are three main drawbacks of using the Asyncio. First, it potentially introduces unordering of the way it processes commands and write back to server and log file because it handles them asynchronously. As a result it makes it much harder to debug the process functions I wrote. For example, the “AT messages” and the JSON dump messages. Furthermore, this problem is exacerbated by the many server herd architectures, which creates a reliability concern for the asynchronous models.

The second major problem is the potential hidden bugs that can affect all the servers. Since all the servers share the same code, any bug in one of the servers affects all of the servers. If a fix to a server is introduced, all the server must turn off and rerun again which creates an scalability issue when maintaining many specialized servers.

Although a single thread process is a plus for the python, it creates another issue which is a bottleneck for performance. Unlike Javascript where one can create many threads to handle various clients, Asyncio python only provides one process for each server to handle many requests. Even with the asynchronous processing, it can only linearly run instructions for one async function while pausing on other functions which means it only uses one core on the machine. This creates a bottleneck that is worse than a centralized synchronous server using node.js.

4. Comparison of Python and Java

Which language to choose for a server herd application ? To answer this question, we will analyze memory management, type checking, and Multi-threading features in these two languages.

4.1 Memory Management

Both Python and Java have garbage collection. But they deploy different algorithms to achieve that. Java uses mark and sweep algorithm, while Python uses a simpler approach, a link count method. Each allocated object in python has a count reference that increases or decreases depending on if the object being copied or deleted. When object’s reference count hits 0, it is deallocated by the garbage collector. This provides a simple and efficient approach to deleting unused objects than Java’s approach. However, as mentioned by Prof.Eggert, this approach has an issue with circular references which means the two objects being allocated that point to each other cannot be deallocated by the garbage collector because their reference count will never be zero. However, with the project, there are no circular reference variables and all the variables are allocated frequently and used at most three times. Therefore, this is a plus feature for the project.

Java’s mark and sweep, on the other hand, has to traverse all reachable objects to mark them everytime a new variable or object is allocated or when garbage collectors perform sanitizing. This approach is reliable, yet it adds overhead that impacts the server’s performance because variables or objects are instantiated and removed quite frequently to handle various clients requests.

4.2 Type Checking

Python’s dynamic type checking provides a simple and easier way to write a program because programmers do not need to know the abstract data type of the return variables. For example, socket(), asyncio.open_connection(), asyncio.get_event_loop(), etc. This in turn provides a faster way to create running server with a quick view of documentation. However, it is also a dangerous aspect for a program if

programmers messed up the type of the variables. Since it is dynamic checking, it will not crush at the compile time, but the server can crash in run time while serving clients which causes reliability issues with the program.

Java's type checking, on the hand, is static, meaning the compiler will check the type of all objects and crush if it encounters bugs. This solves the problem of having a runtime type error in python. It also allows more code optimization from the compiler. However, it is a heavy duty for a programmer to keep track of abstract data types for each object, and make it harder to utilize API from libraries because a programmer needs more time to understand the structure of the data and return value for each API.

4.3 Multithreading

As mentioned earlier, while Java fully supports multithreading programs with many useful features to speed up performance, Python does not support multi-threading. The asynchronous in python is achieved by running the current task while pauses on other tasks which translates to linear executions.

Java's multithreading, on the hand, uses many threads to handle clients which exploit multi cores machines. Therefore, Java server is more scalable than Python server.

5. Asyncio vs Node.js

Both frameworks are used for asynchronous event driven code. Asyncio is used in Python whereas Node.js is used in JavaScript. Both of the languages use Duck Typing, a dynamic type checking, which provides simpler and easy to use API's. While python consists of many simple and useful libraries, Javascript is a much more popular language for server/web based application that makes it generally compatible.

5.1 Multithreading

Both asyncio and node.js are asynchronous event code that run on a single thread. Like Node.js, the event loop in python provides promises to complete the tasks in the future. But the Node.js does not have coroutines which make it less flexible and less dynamic than asyncio

5.2 Performance

Node.js is much faster in comparison to Python because it is based upon a V8 engine from Chrome. Python's performance on the other hand only runs on one thread with memory intensive program. This will become a big issue if the project is more intensive than five servers.

Conclusion

The application herd server which receives and replies a lot of requests to and from clients frequently should certainly be using asynchronous framework, as even-based concurrency. After analyzing Python's asyncio is a suitable and robust framework for this server herd applications. Because it provides simple and easy to use APIs with efficient memory management, dynamic type checking, code reusability. Although There was not much time put into looking at Node.js, from the surface of it, it also looks like a reasonable framework for this task.

References

https://snarky.ca/how-the-heck-does-async-awaitwork-in-python-3-5/
https://asyncio.readthedocs.io/en/latest/tcp_echo.html
https://aiohttp.readthedocs.io/en/stable/
https://docs.python.org/3/library/asyncioeventloop.html
https://cloud.google.com/maps-platform/places/
https://realpython.com/async-io-python/
https://asyncio.readthedocs.io/en/latest/tcp_echo.html
https://docs.python.org/3/library/asyncio.html

https://docs.python.org/3/library/asyncio-stream.html