asyncio library for application in an Application Server Herd

Terry Ye -- University of California, Los Angeles

Abstract

Asyncio library in Python can support event-driven asynchronous programming and in contrast to one of the most popular event-driven, non-blocking I/O model Node.js, asyncio has its own pros and cons. So it might be a good candidate to be used to develop an Application Server Herd network architecture, which can handle more frequent data updates and access from different protocols better than classical LAMP platform used in Wikimedia Architecture for Wikipedia.

1. Introduction

The key basis of an application server herd is that the message or the data has to be propagated quickly between all servers so everyone is synced with newest information.

This report will talk about the implementation of a simple Server Herd Architecture of a simple proxy of Google Places API model with asyncio library, the features using the library and its comparison with the popular node.js in Java.

2. Server Herd Implementation

The Server Herd in the example here consists of 5 servers and can respond to two kinds of a client's message, IAMAT that updates the location of a client and WHATSAT that request the nearby places of a client's location from Google. It can also handle the AT message from other servers to update information about client.

When a correct server name is called by the server.py, an event loop is created for the server and server is created by create_server function in asyncio, The handling of messages is achieved by inheriting the protocol class in asyncio and passed into the create_server function . Every time a connection is made, a new Protocol instance is created which is running asynchronously with other protocols and coroutines scheduled. The protocol can handle the received data and respond corresponding to it.

Each server has its own dictionary of client information. Every time one of the server receives an info update, it will uses the flooding algorithm to propagate it toa all other servers so every server's dictionary is well-updated-to-time. This achieves the idea of Server Herd.

2.1 IAMAT message handle

IAMAT message has 3 parts of information: clientID , Location, Time. The server will check if the Location and Time are in the right formats. If correct, the server will respond to the client with an AT message that adds the server name and time difference between server time and Time. The (Location, Time) information will be stored into the dictionary of corresponding clientID in the server if there is no current information about it before or the Time is more recent than the one currently stored.

The server, after responding to client, will also flood a message containing the client information to other servers connected with it. The message it propagates is the AT message responded to the client with the server's name at last so that the server it sends message to can identify who sends the message and do not send the same one backward again preventing an infinite loop between two servers. This part is scheduled by ensure_future function as a coroutine function..

2.2 WHATSAT message handle

WHATSAT message has 3 parts of information: clientID, radius, number. After checking these are in right format, the server will respond with the same AT message and a list of nearby places.

The server can get the necessary Time and Location information of client from the dictionary. Then the information is passed to a coroutine function which requests the needed info from Google Places API.

However, Google Places API only accepts HTTP request but the protocol created here is based on TCP. So aiohttp library is used here to create a session, and submit an HTTP request to Google and at last convert the received Json data to compatible string format to respond to the client.

There is no need of flooding information when receiving WHATSAT message because the client only request the information from server rather than updates its own information.

2.3 AT message handle

AT message is the message transported between servers to update the information. It has 7 parts but only 4 are the one that will be needed: clientID, Time, Location, Server Name.

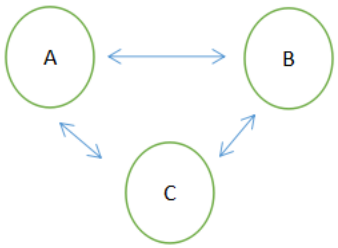
The server will update its information dictionary of clients in the same way as receiving IAMAT message. It will identify the received Server Name and exclude it from possible destinations of propagating messages.It also check the time of the received message, if the time is the same or less than the one stored, then that means the message is already propagated here once, so no more flooding for this server. Othan than the cases above, the server will change the last part of the message to the name itself and flood it to the servers connected with it.

2.4 Problems

The first problem I have is how to actually let the server receive the data from and respond to the client. I initially tried to use a reader and writer to do so, but found out that it is hard to pass all the data around different function so I switched to implement the functions in a single inherited protocol class, so some data can be directly used by functions inside the class.

The second problem I have is that for the flooding function, the server needs to know about the information of the sender so that it will not send the flooding message back to its sender again and result in an infinite loop. I first thought that the transport.get_extra_info will get the port number of the server it makes connection with. Then it turns out the number obtained from the function is not the port number of the server but a number created to represent each connection and has no relation to the other side of the connection. So I has to add the name of the server to the flood message it propagates to others so they can identify who sends the message.

The third problem is also about flooding function. At first I thought that preventing the server from sending the message back to its sender is enough to prevent an infinite loop.



But in situation like above, although C will not propagate the message from A to A, but it will propagate to B and B will propagate to A, which results in an infinite loop A-->B-->C-->A. To solve this problem, I let the server check the message time it received with the already stored message time in the server. If they are equal or the message it gets is actually older, the server will not propagate the flooding message as it must have done before.

The last main problem I have is about getting information from Google.
First, Google's location parameter requirement is different from the one the server gets as it requires a ',' in the middle between latitude and longitude. So I have to find the right place and insert the ',' before calling Google.
Second, after getting a json data from Google, I cannot modify it with default method like json.loads and json.dumps. So I uses aiohttp's json decoding function .json(). This is also a coroutine function, so I have to place 'await' before it and put it into another function as a parameter.

3. Python and Java

Asyncio and node.js can both be used as the module to develop the Server Herd program but there are a lot difference in actual use mostly because the languages they are based on:Python and Java, on the first hand, is a lot different

3.1 type checking

One notable feature of Python language is the dynamic checking (duck typing) wherein the Python interpreter checks the types of variables during runtime rather than compile time.

This makes coding in asyncio a lot easier than coding in static type checking language, which is important because there will be numerous type casting and conversion in the project. Like for example, the server will receive the json data, convert it to a string , parse it to get different substrings and cast the string to float or integer. And when flooding, it has to cast float or int to string again to propagate to other functions.

Type conversion in Java can be achieved as well by calling specific conversion function like parseFloat or ParseInt. But it will be stricter constrained and requires more syntax to write. So asyncio here is more like syntax sugar to save time when coding.

Another advantage is that Python allows a data structure of different types. So when the server saves information to the dictionary, it doesn't have to be the same type. In my program, the information of a client is stored in a tuple which has a float representing Time and a string representing Location.

The disadvantage of this is obvious. With less constraint on type casting and not checking type during compile time, the possibility of bug occurring in the runtime is a lot higher,especially when in real situation where the there are a lot more types to transfer and convert and more data to be handled. And the performance is also worse for Python as there is process of type checking during runtime afterward..

3.2 memory management

Python creates a private heap to manage all objects and data structures and has reference count to all the objects created. So when the reference to an object falls to zero, it will delete that object. This could be troublesome when there is cyclic reference but in my Server Herd project all objects are acyclic so this is not a problem here. And an advantage here is that the objects in the program can be deleted as soon as they are not referenced without using the garbage collector.

On the other hand, in Java, the garbage allocator has to be used to free the memory allocated on the heap. This leads to better performance of Python as Java uses a 'mark and sweep' garbage collector so it marks out the unreferenced objects by traversing through the table of roots. So the performance in this part should be worse in Java as its garbage collector has to go through all the referenced objects to free the memory while Python doesn't have to use garbage collector for this project.

An disadvantage of Python here is that "The heap management is done via the use of python memory manager and is performed by the interpreter itself and that the user has no control over it.". So unlike Java, the server cannot modify the memory heap for use.

3.3 multithreading

Different instances of Python can have different performance with trying to implement multithreading into the program. For example, JPython, developed to actually run on Java, has similar features when using multithreading like Java does. But others like CPython has a bottleneck when trying to implement a multithreading program: Global Interpreter Lock---GIL.

GIL is designed for faster and simpler garbage collection and is used in CPython. It rules that only one thread can run in the interpreter at a time. So even though there can be several threads running simultaneously, only one thread can use the interpreter and all others have to wait for the interpreter to be available again if they want to use it as well. As Python is heavily relied on the interpreter, there will be some interleaving time when thread is not working and this hurts the performance of a multithreading program in Python.

Java, on the other hand, has no such limitations when dealing with multithreading, so it is more free to use and the performance would be better than that in Python.

4. asyncio and node.js

There are several difference between asyncio and node.js themselves.
One difference is that node.js is built to serve HTTP first, designed with streaming and low latency in mind. So node.js is better for developing regular web network asyncio, however, doesn't support the HTTP protocol and has to use other libraries like aiohttp to do an HTTP request which is a bottleneck when serving as a web server development tool.

Both asyncio and node,js, in my mind, are not very suitable for multithreading. Node.js is developed with a single-thread idea to make it more efficient and

easier to use. So it can only achieve multithreading by having multiple child processes , which requires a multiple cores environment to work with.
Asyncio supports multithreading better. It allows an thread to schedule a callback in another thread. It can achieve the threading concurrency in single-core environment so tend to be more efficient in such case. But most asyncio objects are not thread-safe, so there might be problem if access the objects outside the event loop

5.conclusion

It is really hard to come up with a final decision on which one of node.js and asyncio is a better choice. But asyncio is definitely an OK method to implement the Server Herd architecture as it is easy to write, efficient for relatively small amount of data and single-core environment.

References:

[1] Nina Zakharenko, *Memory Management In Python The Basics* https://www.slideshare.net/nnja/memory-management-in-python-the-basics

[2] *Understanding Memory Management,* https://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/geninfo/diagnos/garbage_collect.html

[3]https://github.com/python/cpython/tree/3.6/Lib/asyncio/

[4] https://nodejs.org/en/about/

[5] Sahanda Saba, *Understanding Asynchronous IO With Python 3.4's Asyncio And Node.js*, http://sahandsaba.com/understanding-asyncio-node-js-python-3-4.html

[6] https://docs.python.org/3/library/asyncio.html