Explore asyncio Library for Application Server Herds

Chengyu Wang
University of California, Los Angeles

Abstract

In this report we will explore an "application server herd" architecture for web servers of Wikipedia and its related sites. Different from traditional Wikimedia Architecture, this new architecture allows servers to communicate directly to each other as well as via the core database and caches. Specifically, we will look into the Python aysncio even-driven networking library for implementation of this architecture. We will try writing a simple proxy Google Places API with this library to explore the pros and cons of asyncio framework.

1. My implementation

1.1. Overview

Since asyncio library is designed for the asynchronous I/O, we cannot use the traditional sequential programming method in the implementation. The server will receive events that don't come in with time order. The entire system is based on an event-driven concurrency. The central device of this mechanism is the event loop. It is responsible for creating transports for communication and executing function calls. There are five servers in this case. And each servers needs to deal with three different types of messages. We will discuss them in the next section in detail.

1.2. Implementation details

After the server receives the message, it will split the message based on space and get an array of strings. We will get the first element of this array, which should be one of "AT", "WHATSAT", "IAMAT". Then we handle them based on this keyword.

1.2.1. Handle IAMAT Messages

The first type is "IAMAT" messages. This kind of message is sent by a client with the client name, sending time and client location coordinates. The server that receives this message needs to respond to the client with an "at message" that tells the client this server has received his message.

To process this message, we need to split the message based on the white space. There are many things to check to ensure the message is valid, including the number of arguments in the message, the format of location coordinates, the format of time. After we make sure everything is okay, we send back an "at" message back to client and other servers.

Since asyncio library receives events do not come in particular time order, it's possible that when a client sends information at two different times but the later message arrives first at the server. In this case, when the earlier messages arrived, the server only needs to reply again the "at" message it sent before for the later message and there is no need for propagation.

1.2.2. Handle AT Messages

The second type is "AT" message. A server might receive this type of message from other servers due to propagation, which means that a server shares its own client information with other servers. Every server has a limited list of servers it can talk to, so when a server receives an "at" message, it needs to immediately deliver this message to servers it can talk to. Through the propagation process, the servers share information directly from each other without the need of a central database.

A possible problem with propagation is the infinite loop problem. The servers might propagate forever if there are no constraints. My fix for this problem is that every server has a cache dictionary that records each client with its information. When the server receives a new "at" message, it will first check whether it has records of the client in the message. If there is no record, it will add a new record into the cache and propagate this new message. If there is record, the server needs to check the time of the newly received message and the cache record. If the received message's time is earlier than or equal to the record's time, it means that this message is outdated and there is no need to do the propagation. If the received message's time is most recent, it means that this is the most updated message and we need to propagate this message. When we propagate the message to other servers, we create a coroutine that requires the use of the event_loop.

1.2.3. Handle WHATSAT Messages

The third type is "WHATSAT" message. This kind of message is sent by a client and asks the server the locations within a certain distance around him. The server needs to get the client's location from its past records (from "IAMAT" type messages) and contacts the Google Places to search for locations nearby the client. Finally it returns an "at" message and a json format message including the results from the Google Places.
This part is very difficult because it requires us to send an HTTP request to Google but ayncio only supports the TCP and SSL protocols. So we first need to generate an HTTP request according to HTTP/1.1 request.
The second step is to obtain the json from Google's response. We will need a new protocol for connection to Google. In this protocol, we will deal with chunks of data returned by Google API. If there is more than more chunk of data, there will be problems when we parse the data. So, we need to combine the data into one single chunk to process it.

1.3. Difficulties

The part that takes me most time to debug is that at first I make the cache of the server a local variable of the server. But then I found that the record is not stored into the cache. After I make the dictionary a global variable, it works. The reason is that the server protocol itself is not created as a global variable. Thus when we access the cache, it has already been cleaned.
Another difficulty is that when I try to connect to Google for information, there is no output. At first I thought there is something wrong with the protocol that connects to the Google. However, I found later that actually this protocol is not even called. After adding many debugging output in the code, I found the reason is that the transport for the "AT" message is not closed. Only when one transport is closed can the server start a new transport for data.
Another thing to notice is that when we test our program, we had better let the program sleep for like 0.5 seconds so that the information is ensured to be propagated.

2. Discussion

2.1. Pros and Cons of asyncio

Pros: The asyncio lets you initiate a task (typically an I/O task) that would normally block, and then go on to do other things (like initiating more tasks) while waiting on that task, without having to resort to multiple threads or processes (which have the disadvantages of consuming more system resources as well as introducing the risk of race conditions and deadlocks).
Cons: For beginners, it's kinda hard to write bug-free asyncio code because there are many concepts to understand first and the coding style is not very simple either. Another is that since it's single-threaded after all, its performance is slower than multi-threaded framework. Another possible con is that if we want to trace the progress of the program, it's hard because we cannot get inside the event_loop that controls everything.

2.2. About Python

In this section we will discuss the viability of using Python to write such web server framework. We will focus on aspects including type checking, memory management and multi-threading.

2.2.1. Type checking

Python has dynamic type checking mechanism, which means that the programmer does not need to specify the type beforehand. This is a very useful property especially when we are dealing with unknown or unclear data. In this case, when we receive data from Google API, we are not clear about what type of data it is. With dynamic type checking, we can easily process the data without having to worry about the type problems. Dynamic type checking makes the code shorter and easier to read than the language of static type checking such as Java. Java will require programmer to provide types beforehand and once complier finds unmatched types, the compiler will report errors. So Python provides more flexibility compared to Java. There are also some cons with dynamic type checking. Sometimes we have to check the type manually like isFloat to ensure the data type is correct. Overall, dynamic type checking makes Python more convenient but sometimes might bring ambiguity.

2.2.2. Memory management

Python's memory management is a mixture of reference counting, mark and sweep and garbage collector. In this case, reference counting makes it very convenient for this application server herds architecture. Many transports and coroutines are opened for the communication between servers. Once the communication is over, their reference count will be 0 which means that they are no longer needed anymore. At this time the garbage collector can free them.
In Java, we need to allocate objects using the keyword new and the objects sometimes will remain on heap

even when you don't need them anymore if you forget to make them eligible for garbage collection.
Overall, Python's memory management is more convenient so that you don't need to write any specific code for garbage collection; the complier will do it automatically for you.

2.2.3. Multithreading

Java encourages multithreading and has many libraries for multithreading, but Python mostly supports single thread application. In such an application for server communication, Java's multithreading will bring many problems on data's integrity. It will make the coding extremely unreadable due to the protection of data between threads. Thus, Python is more suitable even though the performance might be a little slower.

2.3. Comparison with Node.js

Node.js is works as a server-side library written in Javascript. Similar to Python, Node.js uses asynchronous events for networking and is single threaded.
The advantage of Node.js is that Javascript can be used for both front end and back end support. It makes Node.js more popular. Another thing to notice is that Javascript is a language writing especially for web, so it's easy to integrate with web applications that need a server.
But Python also has its advantages. Python is a mature language with very details documents and many supported libraries, which makes it easier to debug and less likely to have undefined behaviours.

3. Conclusion

Overall, I will recommend Python's asyncio library for this application server herds architecture. Python's dynamic type checking, single threaded mechanism and memory management make this language a perfect fit for such an application that requires a lot of coroutines and communication. Python's detailed documentation also saves the programmer a lot of time to do the research on the use of library.

References:

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

[2] https://jaxenter.com/top-reasons-to-use-node-js-for-web-application-development-125144.html

[3] https://stackoverflow.com/questions/21934/why-java-and-python-garbage-collection-methods-are-different