Implementing Python’s Asyncio Module on Application Server Herd Architecture

Sung Min Ahn - University of California, Los Angeles

Abstract

We are designing a new system architecture for an application where there are frequent updates to servers, servers are accessed via different protocols, and clients are highly mobile. The Wikimedia architecture that uses LAMP platform is not suitable for this application because its application server is a central bottleneck for the response time and adding new servers. Thus, we are coming up with application server herd architecture where Python’s asyncio asynchronous network library will be used in replacement of Wikimedia platform. In this paper, we will examine the suitability and maintainability of asyncio method and briefly make comparison with Node.js and Java.

Introduction

Wikimedia architecture, which uses LAMP platform based on GNU/Linux, Apache, MySQL, ad PHP, is implemented on Wikipedia and its related websites. The architecture and the platform are suitable for the purpose that Wikipedia and its related websites serve. However, in the environment where there are far more frequent updates to server, servers are accessed through various, different protocols, and clients tend to be mobile, the Wikimedia becomes a bottleneck. Thus, it requires different implementation from of Wikimedia.

New application server herd architecture uses Python’s asynchronous network library to allow multiple servers to communicate directly to each other, and its infrastructure allows faster data transfer through taking advantage of asynchronization and flooding information.

This report will highlight both advantages and disadvantages of Python implementation, Python’s type checking, memory management, and multithreading, compared to a Java-based approach to this problem, and the comparison made between the approached take by asyncio and by Node.js.

1. Application Design

1.1 Server Design

The implemented application includes five different servers: Goloman, Hands, Holiday, Welsh, and Wilkes. Bidirectional communication is configured between the servers. Figure 1 shows the communication pattern established for this implementation.

Server	Servers to which connected
Goloman	Hands, Holiday, and Wilkes.
Hands	Goloman and Wilkes
Holiday	Goloman, Welsh, and Wilkes.
Welsh	Holiday
Wilkes	Goloman, Hands, Holiday

Figure 1. Table of server communication pattern

Each server establishes Transmission Control Protocol (TCP) from clients, and the messages from clients are handled as coroutines that are scheduled in the event loop. Awaitable tasks and callbacks are arranged in the event loop to execute in an asynchronous manner.

Servers expect to received IAMAT and WHATSAT commands from their clients and UPDATE command from other servers. Those commands will be briefly explained in next few sections.

Additional to replying back to clients, the servers will also store clients’ information in order to process WHATSAT and UPDATE accurately.

1.2 IAMAT Command

Messages with IAMAT command allow server to recognize the client’s identification, location, and time when the message was sent. Each field of the message must not contain whitespace.

The location of client is represented with latitude and longitude in decimal degree using ISO 6709 notation, and the time is expressed in POSIX time. The following is the format of the message:

IAMAT [client] [location] [time]

Upon receiving the IAMAT command, the server must reply with AT command message using the following format:

AT [server] [time difference] [client] [location] [time]

The [server] field takes in the name of the server that replies to the LAMAT command, the [time difference] is the difference between the time server received the message and the time client sent the message, and the last three fields are the same arguments as the last three fields of IAMAT command. Additionally, the server floods other server with clients' information upon receiving message with IAMAT command about clients whose location were unknown.

1.3 WHATSAT Command

Messages with WHATSAT command allow clients to query for information about nearby places of the client who is asking for. It takes in the identification of client, radius from the client in kilometer, and the number of information on places to receive. The maximum radius a client could request is 50 kilometers, and the maximum information bound is 20 items. The following is the format of WHATSAT message:

WHATSAT [client] [radius] [info_bound]

Upon receiving the message, servers respond to the client with the same AT message as from the IAMAT command and add the JSON filtered result from Google Places API. Before retrieving the information from Google Place API, servers must know the location of the client who sent the WHATSAT command. Client's location can be known through client's IAMAT command or the UPDATE command received from other servers.

1.4 UPDATE Command

Messages with UPDATE command are the only message type transmitted between servers. It consists of client's identification, location, time when the original message was sent, label number that indicates the level of importance, and the names of the server that updates were made. The format of the UPDATE command message follows:

UPDATE [client] [location] [time] [label] [server_sent]

Label and the name of the sender are used for updating client information and flooding. When a server already has client from UPDATE message but recognizes the label being higher than of stored information, server changes the client's information and floods neighboring servers. There is no need to update client information and flood neighbors when its information is the latest update. Server name is used for flooding purpose. It avoids sending the same message back to where the original UPDATE message came from.

1.5 Invalid Message

When servers receive invalid message from the sender, they will respond with message that starts with question mark followed by the received message. The following is the example of message sent by a sender and the message sent by the receiver respectfully:

FOO i_am_client 10 32

? FOO i_am_client 10 32

2 Implementing Asyncio on the Application

The asyncio platform is not a perfect platform where there is only benefit of using it. It comes with both benefits and losses. It is important to know both aspects before making a decision whether asyncio is better platform than others for a particular application.

2.1 Advantages

The first advantage of using asyncio is that it is simple to code and maintain. The same code is used to run different servers. To run different configured server, it only requires different server name in the argument. It is also easy to add new server. With port number, name of server, and names of server to which the new server is connected, developers can add new servers without difficulty.

Asyncio allows asynchronous operations using the event loop that schedules coroutine and gives back a future object that is guaranteed to run in the future. It does not need to wait for one operation to finish before running the other tasks. Thus, the event loop runs different tasks while waiting for other operation to finish. This concurrency on a single thread becomes a benefit when an application needs to run multiple requests that are independent from each other, especially when they are I/O requests. The performance on speed is better than of synchronous implementation.

Concurrency in a single thread allows another advantage on sharing data. When the scheduler switches from one task to another, it would be safer and easier to share data. Compared to multithreading mechanism, it neglects the possibility of operation on a same section of code. Additionally, because asyncio runs in a single thread, the program in which implements asyncio does not need to consider context switching and allocating memories for other threads.

2.2 Disadvantages

The advantage of using single thread can be seen as a disadvantage. The single thread allows the implementation to have concurrent code, but parallelism becomes an issue. Parallelism allows multiple processes to create multiple threads and run different sections of code simultaneously. Although there is a safety issue with the multithreaded environment, parallelism still utilizes more processor unlike asyncio that utilizes only one process. Thus, asyncio is not suitable for applications where parallelism is more important than concurrency.

Another disadvantage is the characteristic of asynchronous coding. Unlike a synchronous coding scheme, where the sequence of codes is easily recognizable by observing the code, asynchronous coding scheme makes it hard to know the order of operations. This becomes especially important for debugging and when the order of operation matter.

The disadvantage of having an asynchronous behavior is that it doesn't guarantee sequential order of the code. Developers of the server herd may expect that future objects will be executed in the order of when they were called. However, without using "yield from" keyword, the sequential order is not guaranteed.

Finally, asyncio allows servers to run from a same code, but this becomes a problem when it comes to maintaining and changing a code for servers. To add or change a feature for one server requires every server to be shut down. It is possible to write a program to run all the servers, but it both adds another file to manage and opens occasionally unwanted servers. Additionally, clients will have no connection to any of the server while applying a modification of code for servers.

3 Comparison between Python and Java

3.1 Type Checking

Python is a strongly, dynamically typed while Java is not. This means that the name is bound to strongly typed object in runtime. The behavior of the object is what defines the type in runtime. This type of type checking is also known as Duck Typing, which means "if it walks like a duck or talks like a duck, it is a duck". As long as a function supports the operations required by the object, any typed object can be passed in as the function's argument.

The duck typing eases the writing and reading of code, but it makes it hard for the developer to understand real type information on any given point.

Meanwhile, Java is statically typed, which does the type checking in compiled time. Although all the types of variables must be specified and the type of argument for functions must be consistent, all the type bugs can be caught through compiler. Thus, it provides less hassle for developers to catch trivial type checking bugs, which can be found in Python.

3.2 Memory Management

Python and Java takes different approaches to collect garbage or objects from the heap memory. Python heavily depends on reference counters embedded with each object while Java Virtual Machine runs garbage collection.

Python's garbage collection uses reference counter to decide whether to deallocate an object from heap or to keep it. Every object contains reference counter, and every time an object is referenced, its reference counter is incremented by one. Otherwise, when a pointer to an object is deleted, the object's reference counter is decremented by one. When reference counter reaches zero, interpreter deallocates the object and decrements reference counter of objects to which the deallocated object pointed to. However, this method does not recognize reference cycle, so Python allows optional, manual implementation general cyclic garbage collector to deal with cyclic reference issue.

The benefit of using reference counter is that objects can be deallocated as soon as they are no longer needed. Additionally, programmers do not need to stress about freeing an object and memory leaks.

Java's garbage collection in the other hand is operated by Java Virtual Machine, and usually has two phases: marking and sweeping. Marking phase scans through objects in heap and marks unreferenced objects to deallocate them in sweeping phase. There are two different approaches in sweeping phase. The first approach simply deallocates all the marked, unreferenced objects and allow new objects to fill in the space. The second approach deallocates all the marked, unreferenced objects and compact them in order to create larger chunk of memory. This process can be understood as defragmentation.

Additionally, Java's heap is divided into generations, which simply are chunks of memory. There are objects that are long-lived while most general application objects are short-lived. Thus, Java allows long-lived objects to dwell in order generation and short-lived objects in younger generation. The newest generation is called nurture, and it is where most of the garbage collection spends its time. When minor garbage collection is done in newer generations, all the reference object will move down to older generations. This is the basic idea of sweep phase where it deallocates and compacts.

3.3 Multithreading

Python has less support on multithreading compared to Java. CPython uses global interpreter lock (GIL), which is a mutex that allows only one thread to hold control, to avoid multiple threads to work on same bytecode. Multithread allows parallelism, but Python's limitation does not take a full advantage of multiple processes to run different threads simultaneously. Although Python supports concurrency, it may not be the best fit for parallel programming. In the other hand, Java doesn't have such restriction due to its finer lock granularity.

4 Comparison between Python Asyncio and Node.js

4.1 Similarities

Python's asyncio and Node.js are very similar in providing fast solution for servers. Node.js also uses event-based architecture and non-blocking I/O, and this allows concurrency. With asynchronous behavior, Node.js maximizes the use of CPU and memory like asyncio. However, Node.js also have less efficiency with operations that are heavily depended on CPU.

Another similarity between two is the ease of writing code. One of Python's strength is it allows easier read and write coding. Similarly, Node.js is purely based on JavaScript, so it is easy to learn. Additionally, Python asyncio and Node.js both are capable of developing both servers and clients. It is an advantage for developers to come up with simple server and client implementation.

4.2 Differences

Node.js is built on the V8 JavaScript engine which translates JavaScript code into more efficient code than using an interpreter. In contrast, Python asyncio is built on interpreter.

Additionally, as mentioned above, Node.js is not suitable for CPU-intensive applications where frequently operates on graphics and data. However, it is suitable and its performance outperforms Python when used in data-intensive application.

Finally, the same language can be used for front-end and back-end with Node.js. Having a same language allows more power and flexibility when creating an application.

Conclusion

From the investigation, we can learn that Python's asyncio is suitable for the server herd application. Although there wasn't a test for Java based server implementation and Node.js, asyncio seems promising in performance for the given application description as it handles tasks through event-loop asynchronously.

References

[1] "18.5.9. Develop with Asyncio¶." *16.2. Threading - Higher-Level Threading Interface - Python 2.7.15 Documentation*, Python Software Foundation, 29 Nov. 2018, docs.python.org/3.6/library/asyncio-dev.html.

[2] Artem Golubin. "Garbage Collection in Python: Things You Need to Know." *Artem Golubin*, Artem Golubin, 5 Apr. 2018, rushter.com/blog/python-garbage-collector/.

[3] Driscoll , Mike. "Python 3 – An Intro to Asyncio." *The Mouse Vs. The Python*, 26 July 2016, www.blog.pythonlibrary.org/2016/07/26/python-3-an-intro-to-asyncio/.

[4] "Memory Management¶." *16.2. Threading - Higher-Level Threading Interface - Python 2.7.15 Documentation*, Python Software Foundation, 29 Nov. 2018, docs.python.org/3/c-api/memory.html.

[5] "Node.js vs Python Comparison: Which Solution to Choose for Your Next Project?" *Netguru Blog on Machine Learning*, 30 Jan. 2018, www.netguru.co/blog/node.js-vs-python-comparison-which-solution-to-choose-for-your-next-project.

[6] Radcliffe, Tom. "Python vs. Java: Duck Typing, Parsing on Whitespace and Other Cool Differences." *ActiveState*, 15 Nov. 2018, www.activestate.com/blog/python-vs-java-duck-typing-parsing-whitespace-and-other-cool-differences/.

[7] Stinner, Victor. "TCP Echo Client and Server¶." *TCP Echo Client and Server - Asyncio Documentation 0.0 Documentation*, asyncio.readthedocs.io/en/latest/tcp_echo.html.

[8] Wouter, Thomas. "Global Interpreter Lock." *TcpCommunication - Python Wiki*, Python, 2 Aug. 2017, wiki.python.org/moin/GlobalInterpreterLock.