Project Report: Proxy herd with asyncio

Zixuan Wang, *University of California, Los Angeles*

Abstract

There exist plenty of web stacks for the web application architecture. Among them, there is a LAMP platform based on GNU/Linux, Apache, MySQL and PHP, the Wikimedia Architecture. We build a new Wikimedia-style application server herd architecture designed especially for news to deal with the bottlenecks concerned with the original Wikimedia Architecture. We compare the use of python with that of Java in terms of type checking, memory management, and multithreading. Also, we consider the comparison between the library asyncio for Python 3.6.4 and the framework Node.js for JavaScript.

1. Introduction

Nowadays, web design is a critical issue in the area of Computer Science. Building appropriate web stack architectures is a challenging and important task for full stack developers. Considering the design of the Wikimedia Architecture, there are several disadvantages: the much more frequent updates to articles; the diverse protocols used to require the access; the more mobile clients. To solve such problems, we design a new kind of application using asyncio as the framework to make the access to the database server become less often while maintaining the communications between clients and multiple distinct servers.

We would examine the benefits of using Python as the programming language to develop such kind of architecture and the advantages as well as the disadvantages of the use of asyncio library compared with the JavaScript Node.js.

2. Design

The "application server herd" we created for the evaluation is a simple and parallelizable proxy making the use of Google Places API we applied online. It is mainly aimed to maintain the communications between multiple application servers both directly and by the central database and caches. It is designed for the purpose of removing the bottleneck of the LAMP working for Wikipedia. The event-driven nature of the asyncio library would improve the performance of processing updates for a server in to other individual servers in the herd.

The prototype we designed for this project is basically consisted of five servers, named "Goloman", "Hands", "Holiday", "Welsh", "Wilkes". According to the specification for the project, the communication condition between these five servers is as follows:

The name of the server	Enable communication with
Goloman	Hands, Holiday, Wilkes
Hands	Wilkes, Goloman
Holiday	Welsh, Wilkes, Goloman
Welsh	Holiday
Wilkes	Hands, Holiday, Goloman

Every server would accept TCP connections from clients that provides IP addresses and DNS names to simulate the mobile device in the real life in an asynchronous approach. We would apply the asyncio library for such implementation, where each client is viewed as a co-routine that adds to the event loop and the event loop processes each co-routine in the queue. As a result, the server is able to process multiple connections and updates of messages at the same time while not becoming too much slower.

We enable the log file and here are messages involved in the application server herd: CHANGELOC Messages, IAMAT Messages, WHATSAT Messages, AT Messages and messages that are invalid.

2.1. CHANGELOC Messages

The CHANGELOC messages are a special type of messages used for the communication between servers, not between the client and the servers. They are mainly used for the servers to update the changes of locations for the clients to avoid further mistakes. There are exactly five kinds of information involved. The message looks like: CHANGELOC <the name of the updated client> <the updated location of the client> <the sending time of the client for this updated location> <the receiving time of the client for this updated location> <the receiving time of the server for IAMAT>. Such CHANGELOC message is enough for our log file to record the activity and also avoid infinite loops.

We would check the validation of the CHANGELOC message in our design of application certainly. More specifically, we will check if the client name is in our client dictionary and if the time stamp of the current location is indeed updated compared with that of the stored location.

2.2. IAMAT Messages

The IAMAT Message is used to tell the application server herd the exact location of the client. The message looks like: IAMAT <the name of client> <the current location> <the sending time of the client. The current location is the latitude and longitude in decimal degrees applying ISO 6709 notation. The sending time of the client is expressed in POSIX time.

2.3. WHATSAT Messages

The WHATSAT Message is used for clients to require information about places near other clients' locations. The message looks like: WHATSAT <the name of another client> <radius in kilometers from the client> <the upper limit of the number of items required>. The radius here is no larger than 50 kilometers and the upper limit is at most 20 for the convenience of the Google Places API.

The server would in addition sends the JSON message using the Google Places API. In this case, the server is using the aiohttp library to send the HTTP request to the Google Places servers.

2.4. AT Messages

The AT Message is used for the server to send the acknowledgement of receiving the message to the client. The message looks like: AT <the name of the server> <difference between the time of the server receiving time and the client sending time> <the name of the client> <the latitude and longitude in decimal degrees applying ISO 6709 notation> <the time of the sending message expressed in POSIX time>. The time difference can either be positive or negative. While it should be positive in normal cases, it can be negative due to the clock skew, meaning the client's time stamp is greater than the serve's time stamp.

2.5. Invalid Messages

The invalid message is used to deal with the invalid commands. This kind of message is a line that consists of a question mark, a space, and the copy that invalid command.

3. Pros and cons of asyncio

The asyncio library we used for this project employs advantages as well as disadvantages.

3.1. Pros

The asyncio library for Python 3.6.4 makes the design easy and convenient to set up our application server herd. The asyncio module satisfies our required functionality in this project. Basically, we are using the event loop in an asynchronous approach and running the co-routines of the event loop asynchronously to handle the requests. Every server employs its own event loop and if a new connection or request occurs, we add the co-routine to the event loop and process the message when the co-routine is in the front of the queue of the event loop. Thus, the request can be guaranteed to be handled later but it will not be processed immediately until it arrives at the front of the queue. Moreover, when we add a new connection, this can be done at the same time of processing other input asynchronously. Technically, the event loop schedules all the co-routines and each server runs the same code for an event loop. Therefore, the asyncio framework is suitable for our new "application server herd". It enables all the basic features of the server herd and makes it simple to add a new server.

3.2. Cons

The main disadvantage of applying the asyncio framework is concerned with the order of the tasks done. It is difficult for us to figure out which task would be processed first because it will not necessarily follow the order of the arrival of the tasks. In our application, for example, if the server does not know the client's location but the client is requesting places nearby because the server processes WHATSAT command first. In this case, even though the client is typing commands in a correct order, the server does not deal with them in the desired orders. This will cause severe problems when we have more servers. Because the order of the tasks is less likely to guaranteed as the number of servers increase. Hence, a synchronous model would perform well in terms of reliability. Also, the asyncio framework is not such scalable like the multithreaded model of servers. The performance would be degraded if the server is not a single thread. The difficulty of closing all servers to debug is also a problem for asyncio.

4. Python and Java

We choose Python over Java as our programming language for this project. Now, we would address concerns with Python over Java in the aspect of type checking, memory management, and multithreading.

4.1. Type Checking

Python is mainly applying dynamic checking, meaning that it checks types at runtime whereas Java is a style of static typed, meaning that it checks types at compile time. The difference of type checking between Python and Java makes Python easier to add a new server because we do not need to know all the types of objects returned when we build up the prototype. The various functions involved, and the types returned need not to be fully understood if we use Python. However, all the types of objects must be declared for Java before we use these arguments and variables for particular functions. This would lead to a big difficulty to build a new server. While it may be beneficial for future full stack developers to evaluate the details of code when we use Java, it is not so friendly for a developer to start the project initially.

4.2. Memory Management

The memory management for Python is mainly related to a private heap used to contain all of the data structures and objects. There exists a difference of garbage collection between Python and Java. The garbage collection for Python is mainly using reference counts. During such implementation, each object has its own field to keep track of the number of references. The number of references would increase or decrease when the object is referenced and a zero of number of references represents a safe status for the deletion. Java employs a traditional garbage collection method called mark-and-sweep method. It would start from the initial block for memory to sweep the heap during regular intervals. It would first mark the block of memory and remove blocks that are not marked during the second time of sweeping. The difference between Python and Java for garbage collection shows that Python is not able to handle circular references and exists an overhead of the performance. However, Python can delete the objects once they are not needed and the circular references are actually not a main issue for our application server herd since we can expect not too much of such cases in the framework. So, the instant actions for memory management make Python a good choice for our project.

4.3. Multithreading

Python's lack of support for multithreading is a main disadvantage compared with Java. Java supports for multicore processors whereas Python is using an inefficient way named multithreading Global Interpreter Lock. The mutex used to control access and synchronize threads. In this way, Python is much less scalable than Java for multithreading server. The effective parallelism achieved by Java increases its throughput and preforms better over Python with respect to multithreading.

5. asyncio and Node.js

We select asyncio as the framework for the "application server herd" project and we would briefly compare the overall approach of this with Node.js.

Basically, the asyncio frame work is similar to the Node.js framework. The Node.js framework uses an event loop and is also event-driven. It also implements in an asynchronous nature, scheduling callbacks in the event loop. The convenience of not using threading but a simplified event-driven model for Node.js is just like the asyncio library for Python 3.6.4. Although Node.js is a relatively new library, the powerful JavaScript engine behind makes Node.js would perform better over the asyncio library. Therefore, the asyncio framework for Python 3.6.4 would not be that fast as the Node.js framework. The asyncio and the Node.js share almost the same characteristics when we look into their architectures, but the asyncio library can deal with the new provided messages more flexible because we are able to guarantee the reentrance of the execution. Moreover, Python and JavaScript are both of a style of dynamically type checking. Node.js can be more popular not only due to its convenience to build up a new server but also due to the popularity of JavaScript over Python in the area of web development.

Overall, the asyncio library is suitable for our implementation of proxy herd. We would recommend the asyncio-based programs for our new Wikimedia-style service.

References

[1] Project. Proxy herd with asyncio

https://web.cs.ucla.edu/classes/fall18/cs131/hw/pr.html

[2] asyncio – Asynchronous I/O

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

[3] Node.js https://nodejs.org/en/