Implementation of Server Herd Architecture with asyncio

Tianyu Zhang, University of California, Los Angeles

Abstract

There are many architectures for building a web server such as the LAMP platform of Wikimedia. However, with mainly mobile users and frequent updates, LAMP's highly centralized system is not very efficient. Therefore, a different approach is required to support this need. In this project, we will implement a proxy server herd architecture using Python's asyncio module and analyze the pros and cons of such an approach.

Introduction

A server herd is a group of servers that can communicate with each other. Every server is able to handle requests from client without a central database. With this model, the lookup time per request made to each server is minimized, since they all have the same information. Additionally, the use of asynchronous programming via asyncio will improve performance in the face of large number of client connections, network requests to the Google Places API from multiple clients, as well as constant I/O between server and client and server and other servers.

We consider the advantages of Python with other languages such as Java in terms of designing a server herd with asyncio, focusing on the differences in typing, memory management, and multithreading between these languages. Finally, we consider the benefits and drawbacks of using Python's asyncio framework compared to JavaScript's Node.js framework

1. Design

1.1. Structure of Server Herd

There are five servers in the server herd, named Goloman, Hands, Holiday, Welsh, and Wilkes. Each server communicates only with a subset of these servers, but all client locations are propagated to all of the servers using a flood-fill algorithm. Each server accepts TCP connections from client asynchronously, and messages to the server are processed and handled asynchronously using co-routines. Each co-routine is added to an event loop, which processes each co-routine in its queue. In this way, the server can accept many connections or messages at the same time without slowing down significantly.

Additionally, any information passed to one server via IAMAT message must be propogated to the other servers by way of a flooding algorithm and AT message. The bi-directional connections between servers is in the chart. Clients can also send WHATSAT messages, requesting information about clients/users that have communicated with the server herd and ask about nearby locations using an HTTP request to Google Cloud API's Google Places Platform

Server	Communicate
Goloman	Hands, Holiday, Wilkes
Hands	Goloman, Wilkes
Holiday	Goloman, Welsh, Wilkes
Welsh	Holiday
Wilkes	Goloman, Hands, Holiday

A typical loop of a server will go through the following: accept clients and input, determine the command's validity, then write/update the information, send the correct response message to clients, and continue until a keyboard interruption.

1.2. IAMAT Messages

IAMAT messages are sent from client to a server and contain the information about the reported location of the client and the time this information was sent. The messages have the following form:

IAMAT <client > <location> <timesent>

Specifically, <client> is the name of the client which sends this message. <location> is the reported latitude and longitude in ISO 6709 notation. <timesent> is mainly the time when the client sent the message and it is in Posix format. Time is in seconds and nanoseconds since 1970-01-01 00:00:00 UTC.

1.3 AT Messages

A valid IAMAT message with updated data will return this to the client device:

AT <server_name> <time_diff> <client_name> <location > <time_sent>

Specifically, <server_name> is the name of the server that the client communicates with. <time_diff> is the time between the client send the message and the time server receive the message. <client_name> is the name of the client which send the message. <location> is the

the location with ISO 6709 notation. <time_sent> is the time when the server sent this message in Posix format.

In addition to sending the client an AT message, the server also communicates the newly received information to the rest of the servers through a flood-fill algorithm. The server sends a CHANGELOC message to each of the servers it communicates with, allowing each server to store the most recent client location. The process is described in more detail in section 1.5

1.4 WHATSAT Messages

WHATSAT messages are sent from the client to any of the five servers and contain information about which client to search around and how much information to return. The messages have the following form:

WHATSAT <client> <radius> <num_entries>

Specifically, <client> is the name of the client, <radius > is the radius around the client to search. <num_entries> is the number of points to send back to clients. In order to make the WHATSAT message valid, <client> must be one of the clients which sent the message to servers before. <radius> should be a number that can be casted to float between 0-50. <num_entries> should be a number that is less than 20.

After received the message, the server will send a request to Google places API to find points with number <num_entries> and with radius <radius>. The server uses aiohttp to make an asynchronous call to the Google Places API and get the information in JSON format. The server returns to the client an AT message along with the <*num_entries*> entries of the results.

1.5 CHANGELOC Messages

CHANGELOC messages is sent from server to server. It is used to update changes of client locations. The form looks like this:

CHANGELOC <client> <new_location> <time_sent> <time_received> <server_received>

Specifically, <client> is the name of the client, <new_location> is the location with latitude and longitude combined. <time_sent> is the time when the client send this location. <time_received> is the time when the server received the message sent from the client. <server_received> is the name of the server which received the new location.

After receiving the CHANGELOC message, the server will first look at the client. If the client is the first time received, the server will store the information and send the same message to other server. If the server has already have a location of this client, then the server will check the time, if the time is later than time stamp of the old location, the location will be updated. Otherwise, the location will not be updated.

2. Implementation using asyncio

2.1 Advantage of asyncio

First advantage of asyncio is that it helps server to asynchronously handle multiple requests. It enables server to process multiple requests at a time. Every time a new request come in, the server create a co-routine and add it to the event loop. Adding a new connection could be done at the same time as processing a new message.

Another advantage of asyncio is that it helps code simple. Since the server in the herd runs the same code, it will be simple to add a new server. Rather than write code to build a server from scratch, we can simply use command line to initialize the server.

2.2 Disadvantage of asyncio

One disadvantage of asyncio is that the tasks are not processed in order that they arrive. For example, in synchronous mode, when a WHATSAT was sent after a IAMAT was sent. The client can make sure that the server has already stored the location. Because IAMAT is processed before the WHATSAT message. However, with asynchronous mode, the IAMAT could not be processed even when WHATSAT message is processed and returned to the client. In that case, server would have no knowledge about the location of client and possibly return error even though the client has already sent its location. Similarly, when there are more servers, the problem could occur more often. When the client send the IAMAT message to Hands and send a WHATSAT message to Goloman. The update from Hands to Goloman may need more time, which means the client could easily receive error message.

Another problem is that asyncio framework does not have enough support for multithread server. The server frame work that asyncio run is single thread. If the number of requests grows large, the performance costs of asyncio may outweigh this ease of use, as the

framework is not as scalable as a multithreaded server model due to the fact that it can only process tasks one at a time.

3. Comparing Python and Java

A large part of choosing the most suitable framework for the server herd is choosing the language itself. In particular, the language's implementation of type checking, memory management, and multithreading could affect the implementation of server herd. Therefore, we compare Python's implementation and of these areas to Java's implementation.

3.1 Memory Management

Python handles automatic memory management by utilizing reference counting and garbage collection. Reference counting contains the number of times that an object is referred by other objects in the system. When the reference count of an object hits zero, the object is de-allocated. In addition, garbage collection occurs at scheduled intervals to remove objects with reference counts of zero from memory.

Java implements automatic memory management as well, but doesn't use reference counting. It employs garbage collection: (1) sweep through objects, and mark objects that are unreferenced, and (2) sweep through again, and delete objects that were marked in the previous step.

Comparing these two languages, Java is a bit better than python in terms of memory management. The first reason is that python's reference counting could result in a problem called reference cycle, which means count for a object can never reach zero. For example, if two objects are referring each other, then they will never be de-allocated. Another reason is that the overhead of memory management. Python's implementation executes slower because it needs to update the reference count. However, Java's garbage collection could be done on another thread while the program is running.

3.2 Multi-threading

A big disadvantage of Python compared to Java is Python's lack of multithreading support. Python uses a global interpreter lock, meaning that the synchronization of threads forces a one at a time execution. In contrast, Java is able to utilize multi-core processors in its threading applications to get more work done in the same amount of real time. Consequently, a server running Python is less scalable than a server running Java, as the Java server has multithreading capability and would be able to process more requests and have higher throughput.

3.3 Type Checking

Python's type checking is dynamic which means it will check at runtime, and type is allowed to change at runtime. In that case, the server could be setup without knowing the types of all objects returned by function call. You can get a running server with a quick review of the documentation.

In contrast, for Java, the types of arguments and variables must be declared before use them. It requires much more research and reading documentation just to start the project. It is hard to start the project but as you learn more and more about the types, it becomes a very useful skill for debugging and, ultimately, makes the code more understandable and documentable. Therefore, although Python helps developers who are unfamiliar with the framework start project development, the language's dynamic typing is not benefit for future developers trying to understand the details of the project.

4. Comparing asyncio and Node.js

Both asyncio and Node.js are frameworks for writing server-side code, written in Python and JavaScript respectively. To compare them, we will look into performance and concurrency of the two frameworks.

4.1. Performance

In this part, Node.js is significantly faster than Python. Node.js is based on Chrome's V8 engine, which is extremely fast and powerful. On the other side, Python's performance suffers with memory intensive programs. This might not be a major problem because server herds are not memory intensive.

4.2 Concurrency

Both asyncio and Node.js run the same single-threaded event-driven asynchronous architecture. Therefore, many of their features are common. However, python utilizes the concepts of the event loop and co-routine, it execution could be halted and reentered with new information, which makes asyncio more dynamic and flexible.

5. Conclusion

From our analysis of asyncio and Python as well as our implementation of the proxy server herd, we can tell that the asyncio framework is suitable both theoretically and practically for the task of designing a server herd. The feature of asyncio provides ability to handle process simultaneously. Although we did not investigate Node.js and JavaScript in as much detail as we did asyncio and Python, a preliminary look shows that Node.js may be also appropriate for our server herd, with its increased performance. However, before we can conclude that Node.js is the most suitable frame-

work to use for the server herd, we need to conduct more test regarding Node.js implementation.

References

https://aiohttp.readthedocs.io/en/stable/

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

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

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

https://cloud.google.com/maps-platform/places/

https://docs.python.org/3/c-api/memory.html