

Proxy Herd Implementation Using Asyncio

Ted Yarmoski

Abstract

In this paper, we will investigate the viability of using the Python library `asyncio` to implement a proxy herd. This herd will consist of various application servers communicating with each other and with other entities such as a database. In this investigation, we will touch upon the suitability of using Java instead of using Python for a similar implementation. Finally, we will also touch briefly upon how the JavaScript runtime Node.js approach compares with the `asyncio` approach.

1 Introduction

Wikipedia uses what is called the “Wikimedia server platform” which uses redundant web servers with two levels of caching proxy servers. For this project, we want to design a service inspired by the Wikimedia infrastructure, but without the PHP + JavaScript which could cause bottlenecks on a “Wikimedia-style service designed for news, where (1) updates to articles will happen far more often, (2) access will be required via various protocols, not just HTTP or HTTPS, and (3) clients will tend to be more mobile.”

We will examine an architecture called an “application server herd” where “the multiple application servers communicate directly to each other as well as via the core database and caches.” This interserver communication will be geared toward rapidly-changing data while the database server will be used for “more-stable” data that is accessed less often. To do this, we will look into Python’s `asyncio` asynchronous networking library as a potential solution to this implementation. More specifically, we will write a simple and parallelizable proxy for the Google Places API.

2 Comparing Python and Java

We will now address the differences between a Python and a Java approach for this problem, specifically examining the type checking, memory management, and multithreading capabilities of the two languages.

Python is a dynamically-typed language as opposed to Java which is a statically-typed language. A dynamically-typed language does not require specific labeling of types on objects and variables in a program. This eases the burden on the programmer since they are free to develop without constantly considering the exact types of objects or variables. For example, when designing a function’s interface, the Python programmer can simply name the inputs and be done with it. In Java, the programmer would need to consider each input’s type and specifically label it. Dynamically-typed languages are especially useful in projects that are smaller in scale and have rapidly impending deadlines since development is made quicker. Additionally, the code is more readable since declarations are inherently shorter due to the exclusion of types. However, static-typing has its own advantages over dynamic-typing. Since typing is checked before runtime as opposed to during runtime, type errors can be caught and fixed earlier. Theoretically, if the programmer makes no mistakes, dynamic typing is better in most cases. However, this is unrealistic and the reduction in bug-fixing time by static type checking is considerable and can make maintaining a large application easier. Since Python support rapid development and is more readable, we prefer Python when choosing one of the languages for this project’s purpose.

Python and Java take different approaches to memory management. While they both have a form of a garbage collector, they are implemented in different ways. Python uses reference counts to detect when memory can be reclaimed. When the reference count for an object reaches 0 (i.e. the object is no longer being used), then the memory is freed. There is an edge case that defeats this method however: circular references. Since circular references will never reach 0, the Python method fails. While the reference count method is fast and relatively simple, it cannot handle cyclical references. However, Python implements a backup garbage collector to fix this issue. Java’s garbage collector on the other hand uses a “mark-and-sweep” algorithm which scans all object references and marks those that are pointed to.

It then frees those that are not marked. While Java's method is more complicated than Python's, it is more effective against circular references. However, since we do not utilize circular references for this project, we opt for Python's simpler method of garbage collection.

Python's garbage collection method, although efficient, bars the possibility of true parallel multi-threading. If there was to be parallelism with the garbage collection method described above, memory leaks would occur easily. Therefore, Python uses the Global Interpreter Lock (GIL) which only one thread can hold at a time (no parallel threads). This allows Python to focus on single-threaded execution or concurrency and to make use of C libraries that are not thread safe. On the other hand, Java supports parallel multi-threading since it does not use the reference count method of garbage collection. This comes with the typical pros and cons that multi-threaded languages come with. While benefits such as utilizing multiple cores and parallel execution of tasks are highlighted, there are also risks of race conditions and complicated implementations using synchronization and locks to contend with. For this project, Python will be adequate since the minor benefits of multi-threading in this case using Java would not justify the increased development complications.

3 Server Herd

Each server in the server herd functions similarly to one another. They all receive a request, make sure it is valid, handle the request, and finally return while flooding information to other servers if necessary. Each command can be thought of as a series of tokens which are separated by spaces.

3.1 IAMAT

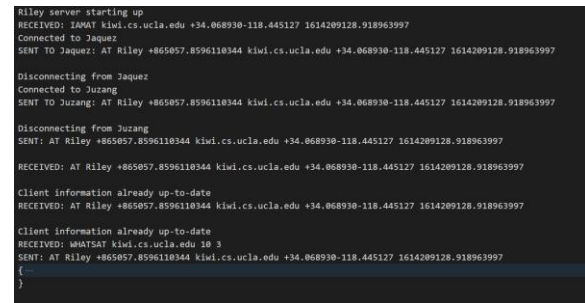
The IAMAT command has the following structure:

IAMAT <client> <coordinates> <timestamp>

When a client sends this command to a server, it lets the server know what the client's location is. This location is then stored and broadcasted to the other servers using a flooding algorithm so that the entire herd is aware of the client's location. The server that received the IAMAT command constructs an AT response with the following structure:

AT <server> <timestamp-diff> <data-from-client>

This response is sent to both the client and to the other servers in the flooding stage.



```
Riley server starting up
RECEIVED: IAMAT kiwi.cs.ucla.edu +34.068930-118.445127 1614289128.918963997
Connected to Jaquez
SENT TO Jaquez: AT Riley +865857.8596110344 kiwi.cs.ucla.edu +34.068930-118.445127 1614289128.918963997

Disconnecting from Jaquez
Connected to Juzang
SENT TO Juzang: AT Riley +865857.8596110344 kiwi.cs.ucla.edu +34.068930-118.445127 1614289128.918963997

Disconnecting from Juzang
SENT: AT Riley +865857.8596110344 kiwi.cs.ucla.edu +34.068930-118.445127 1614289128.918963997
RECEIVED: AT Riley +865857.8596110344 kiwi.cs.ucla.edu +34.068930-118.445127 1614289128.918963997

Client information already up-to-date
RECEIVED: AT Riley +865857.8596110344 kiwi.cs.ucla.edu +34.068930-118.445127 1614289128.918963997

Client information already up-to-date
RECEIVED: WHATSAT kiwi.cs.ucla.edu 10 3
SENT: AT Riley +865857.8596110344 kiwi.cs.ucla.edu +34.068930-118.445127 1614289128.918963997
{
}
```

Figure 1: The log of server "Riley" after running some example commands.

3.2 WHATSAT

The WHATSAT command has the following structure:

WHATSAT <client> <radius> <max-num-results>

This command is used by a client to ask what is near a client. The server uses the Google Places API to find nearby locations using the coordinates stored from the IAMAT command. The response is the same as the IAMAT response (AT) but it has an additional json attached with the Google Places API response data contained.

4 Asyncio

Asyncio is a Python library which allows for concurrent code using the async and await keywords. The python docs described asyncio's use as "a foundation for multiple Python asynchronous frameworks that provide high-performance network and web-servers, database connection libraries, distributed task queues, etc."

4.1 Performance Implications

Since asyncio is a Python library, this means that our implementation will be limited by the GIL model and thus we cannot

4.2 Reliance on Newer Features

Some new features of asyncio were recently added to stable versions of the library. One of these is asyncio.run which can execute a coroutine while managing the event loop. According to the documentation, an equivalent effect can be achieved

by manually creating event loops and executing the loops using functions included in the library.

Another newer feature is an async REPL which can be launched via the command `python -m asyncio`. While this may have applications in other projects using asyncio, it is not necessary for our server herd.

While newer features such as `asyncio.run` and the async REPL can be useful, they are not necessary for an implementation of equivalent behavior. Although the server herd utilizes `asyncio.run`, there are alternative and well-documented ways to achieve the same effects. Another point worth mentioning is that the asyncio library may continue to improve over time and gain new features such as the ones mentioned above that make using asyncio faster or more reliable.

4.3 Node.js Comparisons

Python's asyncio and the JavaScript runtime Node.js are similar in many ways. For example, they both have event loops and the `async/await` syntax. They are also used for similar purposes. An advantage of Node.js is that the client and server will often both be using Javascript which can lower the chance for compatibility issues. Also, Node.js is generally faster than Python so it is well-suited for applications where performance is paramount such as chatting applications. However, asyncio and Python in general is easier to work with because of its simple syntax.

4.4 Suitability

Asyncio is a good choice for this kind of application for the aforementioned reasons such as a simple setup, thorough documentation, adequate performance, and good performance in the server herd test.

References

[Python Docs: asyncio](#).

Anderson, J. [An Intro to Threading in Python](#).

Bhagat, V. [Node.js VS Python: Which is Better?](#)

[Wikimedia servers, Wikimedia Meta-Wiki](#).

[Google Places API Docs](#).