Application Server Design: Python Proxy Herd with asyncio

Arpit Jasapara
XXXXXXXXX
COM SCI 131, UCLA

Abstract
A LAMP architecture may not be performantly effective so we consider another module in this project: asyncio. Asyncio is a module in Python used to implement asynchronous I/O or networking. Using asyncio, we had to demonstrate how we can implement a server herd, while comparing its performance to other alternatives such as a LAMP architecture. We introduce the project design, dive into asyncio, compare it to Node.js, compare it to Java, and ultimately give our recommendation on what the company should do.

1 Introduction

In this project we had to replace a LAMP architecture server with an asynchronous framework to improve performance and scalability. In order to simulate such an asynchronous platform, we used Python's asyncio along with five servers named after the UCLA Basketball team's all stars: Goloman, Hands, Holiday, Welsh, and Wilkes. These servers can only talk to each other based on certain relationships:

- Goloman talks with Hands, Holiday and Wilkes.
- Hands talks with Wilkes.
- Holiday talks with Welsh and Wilkes.

Eventually, the information would be propagated through each server, and even if a server goes down, the other servers would be able to propagate this information. This is in comparison to a LAMP architecture where there is a single server that is responsible for all this information and would easily be affected by crashes.

In order to communicate with these five servers, clients can send the server only two types of messages: IAMAT and WHATSAT. The former tells the receiving server where exactly the client is along with the time on the client's end. The latter sends a query to the server asking for locations of any nearby clients that has previously sent an IAMAT message. The Google Places API is used both by the client and this application to obtain this location information. WHATSAT uses this API to request information about nearby places by using a JSON object and parsing it to get the correct number of results, sending this information back to the client.

2 Asyncio and its Performance in Python

Asyncio as mentioned earlier is a module based in Python and has many notable advantages and disadvantages. These advantages and disadvantages stem both from the module and the fact that it is written in Python. Some of its most notable advantages are:

- The Python documentation for this module, like any other Python documentation is great and easy to follow, so understanding the Protocols API and implementing the functions that come with this was relatively straightforward
- The stream API could also have been chosen instead of the Protocols API, so the module was flexible in this way
- Python is a language that is generally easy to write and subjectively fun, especially due to it being a dynamically typed language so there's less worries about type matching
- Asyncio is a great module and even provides us packages for json that would allow us to dump and load our object instead of having to manually parse it ourselves
- The module was easy to use and was relatively less error-prone as it shows us when we're waiting for something because of await or when something is asynchronous for that matter

While these benefits seem great, asyncio has many notable weaknesses that make it murkier to decide if this is the best tool for this application:

- As previously mentioned, due to dynamic type checking in Python, asyncio would be slow and the interpreter would need to figure out the types of objects during the execution which would increase this slowness.
- The code was hard to understand and the whole library would require some learning beforehand to implement applications using this
- This code is asynchronous and should the code in other applications of our hypothetical company be synchronous, combining the two types of code together and getting them to be consistent would be hard
- Someone who is trying to understand the code may have a hard time following it with the async keywords and the errors that pop up during the implementation, or at least this happened to me and took quite some time to understand and implement

For this application, I would recommend using asyncio if we had a small number of stable, in the sense that they're not mobile and don't send frequent requests, but considering in our hypothetical scenario our boss tells us that we need to take into account these conditions, the weaknesses seem to be more prominent than the strengths so we can consider alternatives as well.

2.1 Issues and solutions of the implementation

I ran into multiple problems during the implementation of the server. Some of were issues with the code itself and my interpretation of the module's capabilities, but some were errors that came up due to infinite looping. At the start of the project, choosing between the stream API and the protocols API was hard because both of them could have been used here. After further research and poring over online forums, I decided that EchoServerClientProtocol would be the best to use for the project along with other functions from asyncio.

Additionally, I got a ton of errors in the beginning because of my understanding of the async keyword and this was fixed by placing async in the correct place. Understanding this was hard however. Also, there were multiple times where I faced infinite looping. The first I mentioned before, with the flooding protocol. This was fixed by a server checking if it's seen the message before and then dropping it should it have seen it. The one that was hard to solve however was the issue with the dictionary that was declared. I could not solve this issue for the longest time and after talking to some peers I understood that making the dictionary global would aid in stopping this infinite looping. The last one was indexing issues within the IAMAT and WHATSAT implementation of the class that was declared. Figuring out this issue with indexing was hard but didn't take as much time as the previous infinite looping issues.

3 Python vs Java

Addressing my hypothetical boss' thoughts about using Python or Java, there are things about each programming language that are different between the two and each could be used to make our server herd implementation better or worse. This comparison will take three factors into account – memory management (which we've learned in class), multithreading (which we've learned in Homework 3), and type checking.

3.1 Storage Management

As we've learned in class, Java uses generational garbage collection, which sweeps through the younger generations of objects more frequently than the older generations of objects. The benefit of this is that it's much more efficient than if we did it the other way around. Python on the other hand uses a different type of garbage collection – it uses reference counts, which is a count of the number of times an object is referenced and when this number hits 0, the object would be dropped.

3.2 Multithreading

As we've seen in our Homework 3, Java has properties of both interpreted and compiled languages and it has great support for multithreading with multiple packages that can be used. Python on the other hand doesn't have a great multithreading system yet and currently has a global interpreter lock which would not work as efficiently. Thus, if our application would have to be very largescaled, we'd probably want it to be implemented in Java rather than Python.

3.3 Type Checking

As mentioned before and as we've learned in class, Python is dynamically typed, and duck typed while Java is strongly typed and statically typed. This application should be able to work with the other applications as it can check what the types of objects are based on the properties.

4 Python vs Node.js

Here are some of the differences and similarities between Python and Node.js for a brief comparison in bullet-form.

- Python handles processor-intensive tasks better.
- Node is natively asynchronous whereas Python has to use asyncio
- Node is used for things like websites whereas Python is used for data analysis.
- Both catch errors very well, but it's relatively easier to deal with Python's errors than Node's.
- Python, or asyncio uses a single-threaded event callback mechanism, as does Node.
- Node creates a single-thread asynchronous architecture with its I/O operations done outside the thread which makes sure that the thread isn't blocked. This allows smooth scalability for smaller applications, that gets worse for larger, more complex applications. Python on the other hand, supports coroutines, like the one used in this project which allows its asynchrony. Because of this, it isn't as scalable.
- In Node.js, all functions are asynchronous unless a we use a promise or a callback from the function whereas in Python we have to specify if a function is asynchronous.
- Node is significantly faster because it runs on the Chrome V8 Engine, an extremely fast and powerful engine, which makes it a top choice for real-time applications like applications that involve chatting for instance]

5 Conclusion

For me, it was a little hard to do this project and write an asyncio-based program. There were a bunch of bugs that I had to take care of, and I ended up spending much of a whole weekend on it. So, in saying this, it'd be even harder for someone else (if they were like me) to exploit this, but it could also be easy for someone who found this project to be easy to exploit a server herd.

As mentioned throughout the whole report, asyncio is quite slow, especially in Python as it's dynamically type checked. But I also outlined the pros of asyncio and the main one is that we don't have to access a central database and instead can just keep talking to other servers through this. Java could be better for a larger-scaled application and Node could be better for an application where speed is of utmost importance.

Overall though, asyncio in Python is very powerful. As seen by the above research and analysis, asyncio is still better than most alternatives and easiest to adapt to. Thus, I would still recommend it to the company for usage in a proxy herd server application, but Node is also a valid alternative.

References

- https://docs.python.org/3/library/asyncio.html
- https://en.wikipedia.org/wiki/Node.js
- https://en.wikipedia.org/wiki/Duck_typing
- https://en.wikipedia.org/wiki/Java_(programming_language)
- https://en.wikipedia.org/wiki/Python_(programming_language)
- https://snarky.ca/how-the-heck-does-async-await-work-in-python-3-5/
- https://da-14.com/blog/python-vs-nodejs-which-better-your-project
- http://sahandsaba.com/understanding-asyncio-node-js-python-3-4.html
- https://medium.com/@mindfiresolutions.usa/key-differences-between-python-and-java-3e5dbbc0bd67