CS131 Project Report

Abstract

For this project, we had to create a sort of "application server herd" where the different servers/applications communicate with each other through the database and caches. For the project we are trying to replace part/all the Wikimedia platform by using the asyncio asynchronous networking library. However, we still do not know if this framework if the best after being executed, because even though asyncio is event-driven, we have to keep in mind if it is easily implementable and how reliable these applications will be. There are also many other factors to keep in mind, such as, Python's implementation of type checking, memory management, and multithreading. In this project, after implementing this kind of application using the asyncio library, we try to understand the pros and cons of implementing the application in this manner.

Introduction

Implementing this program had many different components which involved the asyncio library. Our "server herd" consisted of five servers that communicate with each other bidirectionally, asking information about the places near other clients' locations and sending information to the server about where the client is. The servers communicate by accepting TCP connections from clients with IP addresses. Servers also respond to invalid commands, by reporting back the command as is but with a '?' placed at the beginning. The relations of the servers are:

1. Hill talks with Jaquez and Smith.
2. Singleton talks with everyone else but Hill.
3. Smith talks with Campbell.

IAMAT

A client can let the server know of its location through this command:

```
IAMAT kiwi.cs.ucla.edu +34.068930-
118.445127 1520023934.918963997
```

"IAMAT" is the name of the command, the second argument is the client ID, the third argument is the latitude and longitude, and the last the time the client thinks they sent the message. When my program receives this command, it first checks to make sure the command is valid and has the correct arguments. Then, it generates the "AT…" message to send and calls the flooding function with asyncio.create_task.

WHATSAT

Another command the clients can use is "WHATSAT" in which they query information about places near other clients' locations.

```
WHATSAT kiwi.cs.ucla.edu 10 5
```

The arguments of this commands are the name of the other client, a radius, and an upper bound on the number of locations to receive. To query information, we had to use the Google Cloud Platform so we could have access to an API, in URL form, and generated an API key that is specific to us. After checking if the command is valid, my function gets the latitude and longitude from the history dictionary, with the client name passed in as the key. I pass these coordinates, along with the key and radius, into the URL, and get a response in json format, which I attach to the "AT…" message.

Problems during Implementing

One problem I ran into with implementation this application with this framework was trouble with pinpointing where in the async functionality was my program failing. Because we were using the asyncio library, I was running into some issues with implementing the flooding algorithm, specifically with calling my flooding function when receiving a “IAMAT” message. Due to the server loops and my other functions in the class Server being async, my dictionary consisting of the clients and the messages was not being translated to the other server. This was causing the part in my implementation checking if the client is present in the dictionary to fail. After much trial and error, I moved my flooding algorithm outside of my Server class and created a loop in the main, which seemed to fix this problem.

Asyncio

Asyncio is a package and a way to implement concurrent programming rather than parallel programming through a cooperative multitasking library, in which tasks and routines can voluntarily give up their control and allow another task to get control. This idea is similar to the idea of cooperative multitasking.

Advantages

There are many advantages when it comes to working with Asyncio. Even though a threaded design may seem more feasible and easily implementable, there may be underlying errors that can cause fatal errors like race conditions and memory issues. Debugging these race conditions and multithreaded programs is difficult. Async IO is really useful in situations where multiple IO-bound tasks would be dominated by blocking IO-bound wait time, like network IO, serverless designs, and read/write operations. In relation to our application, asyncio event loops and its concurrent behavior make creating a server herd which accepts many requests/commands, easier to implement. It is also convient to be apply to query in Python with aiohttp

Disadvantages

Even though asyncio provides many benefits, especially in the type of application we are implementing, there are a few factors of this library to be wary of. Async’s API has been changing very frequently, which could make it harder to use. In addition, if all the functions in the program use blocking calls, putting async in front of all these functions is not a good idea since there could be a performance cost. In addition, even though multithreading makes debugging a challenge, finding a bug in async functions are not that much easier. One of the biggest problems in my functions was due to one server not closing, so a variable modification was not being set in a different server. Finding this error was really challenging because I could not figure out which part of my program was causing this. Even though this may not apply specifically to our application, another disadvantage of using asyncio would be that await only supports a specific set of objects that define a specific set of methods. You might need specific wrappers to support the await syntax.

Conclusion

Even though asyncio might not be the best in situations of high demand, it is more scalable than a threaded approach, as you can create thousands of async IO tasks but threads are a system resource that only can be created in limited amount. I think the use of asyncio combined with the frameworks and libraries supported by Python, make it a good option for implementing this kind of application. I also could use aspects from previous versions rather than from 3.8, such as the event loops, and it did

not cause problems. There weren't any features that I had to use from Python 3.8 to ensure that my program will run.

Python Vs. Java

Multithreading

Python uses a Global Interpreter Lock, which is a lock that can be held by only one thread at a time. Only the owner of that lock is allowed to execute, and no other thread can run simultaneously. This is due to Python's memory management framework and how it depends on reference counting and possible race conditions that can arise. Python just keeps switching between threads instead of running them at the same time. Therefore, multithreading does not improve the performance of CPU intensive tasks and can make it even slower due to locks. Java on the other hand can utilize multithreading efficiently and can operate on shared memory. Due to its more effective ability to create parallel threads, Java is probably a better option when it comes to building bigger applications. But for our purposes, Python's weakness in multithreading do not seem to affect us very much.

Memory Management

Python and Java both have garbage collection, in which they take care of freeing up memory; however, each of these languages implements it differently. Python takes care of freeing up memory through a process which uses reference counting. The reference count for an object is increased every time it is referenced and decreased when it is dereferenced. The memory for the object is freed if the object's reference count is 0. The problem with this is that Python's garbage collection has a hard time detecting cyclic references and this can create issues. On the other hand, Java's garbage collector does not use the process like Python of reference counting. Java memory management is automatic, and the user need not do anything. In the first sweep, unreferenced objects are found and marked to be garbage collected. Then the second time around, the marked objects are deleted. Similar to multithreading, Java seems to have an edge over Python. But once again for our purposes, Python's method of garbage collection is good enough since there probably will not be any cyclic references in the program.

Type Checking

Python is a dynamic language while Java is a static language, meaning that in Python, type checking is done at runtime, while in Java, type checking is done at compile time. For this reason, it may be harder to debug Python programs, but this also allows the program to be more flexible and easier to write as you can change types of variables throughout the program. Changing types is more difficult and rigid in Java. Since Java requires types to match during compilation, and is more particular, it could be less error prone. However, for this project the flexibility of Python's type checking allowed the program to be more easily implementable.

Asyncio vs. Node.js

Both asyncio and Node.js would be appropriate tools to use to build an application like this. They comprise of many of the same things like the same async function and await keywords, and promises (Node.js's version of coroutines, tasks, and futures). A lot of the comparison between asyncio and Node.js comes from the languages they are based in, asyncio associated with Python, and Node.js associated with Javascript. Python is a lot simpler to use and

more universal. On the other hand, Node.js can use Javascript both frontend and backend Javascript is a fast-growing language. A big advantage of Python is that some of its frameworks are designed to be simpler and they use a more traditional multiprocessing model instead of a more advanced asynchronous paradigm, like Node.js. In addition, if you use Node. Js poorly, there is the risk of ending up with a slow-working product that has low performance. This could be because Node.js is designed to use a small number of workers, but in those scenarios, it performs much better due to not wasting time on context switching between them. Node.js may be able to make our program faster due to the type of application we were creating, but we would need to compare the programs directly to see differences in performance, Furthermore, the asyncio library and using Python made implementing this program more feasible and still gave us good performance.

Resources

[1] Mirowski, Jacek. Python vs. Node.js. Comparing the Pros, Cons, and Use Cases

[2] Notna, Andrei. Intro to Async Concurrency in Python Node.js.

[3] Solomon, Brad Async IO in Python: A Complete Walkthrough