CS131 Project Report

Abstract

Testing the practicality of implementing a server herd using asyncio with flooding algorithm to populate servers with data representing client geolocations. Ability to use Google Places API to receive information on geolocation.

1. Asyncio Practicallity

The goal of this project is to see how well an asyncio based server herd implemented in python will function as a replacement to a PHP + JavaScript based Wikimedia server. The current server acts as a bottleneck in the Wikimedia server platform and is put up with due to relatively few updates and mostly immobile clients. The python server is an attempt to alleviate these problems as well as the current server's ability to only handle HTTP transports.

Asyncio can be implemented through the use of two main keywords: async and await. Async is used to denote a coroutine function which can be put on hold and returned to at a later time if an await is used within. Await is used before a function call and instructs the program to save the current async coroutine's state and execute another piece of code while it waits on the await'ed function call to return. Overall an asyncio program operates by creating an event loop of code to execute. Every time an await function is called, that async coroutine is moved to the end of the event loop and the next waiting in the loop resumes execution. This allows for slow returning I/O operations such as server connections or reads and writes to run in the background and not cause the program to hang up while waiting on a return, achieving concurrency.

1.1 Suitability

Asyncio is well suited to run a server herd like this because of its simple interface and event driven nature. With the goal in mind of increasing server herd traffic and amount of location updates it is important to have the herd run efficiently and ideally have work increase linearly with amount of traffic received. The event loop nature of the program allows for this to be almost achieved with the only nonlinear increase coming from context switches in the event loop. Its ability to create new network connections quickly and easily is also very convenient for implementing a server flooding method. To prevent over-transmitting a received location I had each server check its data base for the same message it receives. If it's already there it stops flooding, otherwise it relays the message to its own server connections. Asyncio works very well for creating and breaking these connections continually for each new location added to the server. It also handles servers going down well by throwing an exception which can be caught by try, except in python. Overall it works well for all features necessary to the server herd.

1.2 Problem Run Into

Many of the problems I faced were due to being fairly unfamiliar with python and finding ways to store and communicate pertinent data effectively. However, I found that python and asyncio had efficient ways of handling every problem that I faced. The first problem had to do with maintaining the locations of every client that was connected and having an easy way to update them with new information. Python's dictionary feature was very useful because at each client entry I could store a corresponding array of all necessary client data. I also was having trouble parsing the information passed through TCP connections as a string and converting it to a list of accessible data. Python's slice and join functions were very useful in converting between lists and strings for communication. Slice also had the added bonus of removing extra whitespace from the input to deal with oddly formatted, but valid, inputs. I did not face any major problems in creating the server.

1.3 Performance Implications

Asyncio implies that concurrency can be achieved. Its utilization of coroutines and an event loop allow for it to execute other code while waiting on long I/O like reads, writes and network connections. This allows for the single processor that a server is running on to utilize as much computation time as possible. It also means that servers can be added or dropped at any time and unless the herd map becomes disjointed the flood algorithm with still operate and performance will be affected very little. While python's global interpreter lock prevents any of the server's from being truly parallel, they operate efficiently with ideally only context switches between events causing computational downtime. Also, each server is running as a separate process meaning the servers can be accessed in parallel. Performance is efficient for the purposes of creating a server herd.

1.4 Ease of Use

Asyncio is very easy to use for most purposes, including this one, due to its high level and understandable API. Asynchronous programming can be achieved through the use of async and await keywords exclusively. On top of that it provides simple functions for creating and connecting to servers that are async compatible for efficient networking. It can be implemented very quickly and easily and can improve efficiency immediately. It requires very little additional education and most people already familiar in network I/O will adapt to its interface quickly. While the high-level API is often all that is needed in an asynchronous program, asyncio also offer lower level APIs to do things like manage event loops and alter behavior to OS signals. Its ease of use is definitely a strength.

2 Comparing Asyncio

2.1 Java

Java could be another possible language for this assignment. It is partially interpreted through the JVM as python is interpreted and has strong exception handling to deal with downed servers and other problem. However, there are many differences between the two as well.

First is the difference in variable type checking. Python uses dynamic type checking while Java is statically typed. This means that python doesn't know the true type of a variable until runtime whereas Java variable types are linked during compile time and won't compile if there's a problem. This could lead to problems in Java where a server receives an input that it cannot handle and must throw an exception while python can deal with the improper variable during runtime and respond appropriately.

Another issue comes from memory management which leads to a difference in ability to produce multithreaded programs. The Java memory model allows for all parts of a process to be visible in code at all times. It is up to the programmer to add protection to prevent data races or unintended corruption of data. Locks can be placed around key pieces of code to prevent these data races. In python however, all processes use a global interpreter lock memory management protocol to ensure that only one thread ever has access to process data. This means that while both can create multithreaded programs, only Java can produce parallelizable programs. So Java may be able be able to produce a faster server herd, but at the same time statically typed variables would make the server more cumbersome to work with.

2.2 Node.js

Node.js is very similar to asyncio. Both are single threaded, asynchronous, event driven networking applications. They both use an event loop to maintain efficient execution and have no potential deadlocking worries or blocking concerns.

They do differ however in a few regards. Node.js uses a callback function to complete necessary asynchronous computations while asyncio organizes its events into coroutines and return to that entire routine after a function returns instead of having a separate callback function. A benefit of Node.js is their increased importance placed on HTTP communication and have spent time on optimizing transport through Node.js. Asyncio on the other hand doesn't handle HTTP and aiohttp must be used to resolve this issue. One benefit asyncio has over Node.js is lower lever APIs to access the event loop directly and create performance improvements. Node.js handles all event protocol internally and doesn't provide any access to developers.

Both provide excellent and robust access to building asynchronous networking applications.

2.3 Python 3.8

The run() method added in the newest version of python is very nice because it can call an async function to run at the start of the program and also create and manage the entire event loop in one function call. This allows for the start of asyncio programs to appear much cleaner and also reduce the chance of programmer messing up the creation of the event loop as it is done automatically through run.

The command 'python -m asyncio' is a nice addition to the language and provides new functionality. It creates an asynchronous REPL for asyncio based programming and allows for a new event loop to be created on every invocation, eliminating the need for run() to be called. This allows for a more rapid way to test new async code and understand its behavior quicker.

Neither of these things are strictly necessary or are things that couldn't be done before, but they advance asyncio's goal of being an easy to use, high level networking application by creating new, simple and easy to use improvements.

3 Conclusion

Asyncio has many benefits that come with its use. It is very powerful, easy to learn, and adaptable to many networking uses. While creating a server herd to manage client location information asyncio was very straightforward to use and easy to produce accurate and robust asynchronous code with. While it does limit parallelization of execution, the numerous benefits of ease of use along with worry free memory management make it a solid choice for this and many other network applications.