


Programming Languages 131: Project

Abstract—Python and Asyncio’s suitability for server development excels in its ability for fast development in a well structured, object oriented language that is simple to learn and understand; however, it falters in performance at nearly every end. Another candidate, Java, also allows for well structured and extensible code with greater performance than Python; however, it is not nearly as simple to code in and is much slower to develop with. The last candidate, Node.js, excels at being a lightweight power horse that is easy to develop with but ultimately does not have the same support or object oriented structure that Python and Java has. Thus, I believe Java is best suited candidate for the task of developing the wikimedia server in.

I. INTRODUCTION

 In this Project, in order to better understand the various advantages and disadvantage associated with Python and the many packages the come along with it, we investigated the Asyncio library and created a preliminary network of servers that communicates to each other and Google via asynchronous operations. We were charged with testing the effectiveness of Asyncio to implement server herding, and through this process (and the corresponding research done afterwards on Java and Node.js), we are able to give a well reasoned recommendation for a framework fitted to coordinate server herding on a network.

II. DESIGN

A. Specifications

For this project, we were tasked with creating an asynchronous system of networks using the Python library Asyncio as the main framework to create tasks and eventually execute when appropriate. The servers available to connect to are: Alford, Ball, Hamilton, Holiday, and Welsh

A client should be able to connect to a specified server and issue the following commands to receive the following responses:

1) IAMAT

The IAMAT command will have the following syntax:
IAMAT 'id' 'latitude','longitude' 'Posix Time'

Upon being received, this information will be stored in the server’s local cache. It will then return an AT message with the following syntax:

AT 'server name' 'timeDiff between server receiving and Posix Time' 'latitude','longitude' 'Posix Time'
Additionally, all Servers must be connected unidirectionally in the following way:

- Alford is connected to Hamilton and Welsh

- Ball is connected to Holiday and Welsh
- Hamilton is connected with Holiday

Once a valid IAMAT message is received and the response sent, the new information will be propagated throughout the network via a simple flooding algorithm that uses the connections listed above

2) WHATSAT

The WHATSAT command will have the following syntax:
WHATSAT 'id' 'Radius in km' 'Max number of places to receive'

The server will then find the newest IAMAT entry by that particular person’s id and create a nearby places request to Google Places. If successful, it will return the response message containing up the number of places as requested by the WHATSAT command.

B. Implementation

In order to support these specifications, I created a file server.py which takes as an required argument, the name of the server to create. The valid server names and their corresponding ports are as follows:

- Alford - 15890
- Ball - 15891
- Hamilton - 15892
- Holiday - 15893
- Welsh - 15894

Example server creation: python3 server.py alford

Once given a valid server, a coroutine will be created using the start_server command, passing in the handle_client_msg function as an argument. The coroutine will then be added to the loop and the loop will then perpetually run(unless a keyboard interrupt is received) on the specified port with the IP address localhost '127.0.0.1'.

Once in this loop, the server will do no useful work until a message is received from the client. Once this occurs the handle_client_msg function will act as follows. It will check whether the message is either a valid IAMAT command, WHATSAT command, or FLOOD command (will be explained in detail later). If they do not meet any of these requirements, the server will send the response

? 'Unknown Command'

If a valid IAMAT command is received, the server will extract the useful information including all the arguments included in a IAMAT command along with the current time, and receiving server. It it will then package this into a Location object and add it to the server’s cache of Location entries which is mapped by the client’s id. If

a location by the provider user id is already being used, it will be overwritten by the newly received Location. The server then calls the newly created location's `toATMessage` method which returns a syntactically correct AT message of that location and sends it back to the client to be received. Lastly, the server then floods the connected servers with a FLOOD command.

A valid FLOOD command has the following syntax:
`FLOOD 'id' 'latitude' 'longitude' 'posixTime' 'receivedTime' 'receivedServer'`

If a valid FLOOD command is received, a Location will be created from the information provided and the server will then check if its cache the id already stored. If the id is stored and the posix time of the stored location is greater than that of the flooded Location, then nothing will happen and the command will be ignored. However, if no Location from that id is found or it is a more recent Location, the server will add it to its current cache and flood all connected servers. This will continue until all servers contain the new Location, at which point the FLOOD command will cease to be sent.

Lastly, if a valid WHATSAT command is received, the server will take the following course of action. It will check if the id instead the command is also valid. If that is true, then the server will create a GET request to be sent to Google. The GET request is structured according to HTTP 1.1 protocol. It will then connect to `maps.googleapis.com`. If successful, it will send the GET request and the corresponding header and the body. If the request was valid, it will then decode the body using chunked decoding (necessary due to the HTTP 1.1 protocol). The server will then parse through the given JSON body and return only the amount of locations requested by the client. It will then, finally, send the JSON response back for the command given.

C. Difficulties and Roadblocks

Having not taken a networking class, this project definitely proved fairly difficult to accomplish. Specific, connecting to Google and providing a valid GET request was more easily said than done. Luckily, my TA greatly helped in my understanding of the format needed to accomplish this. Lastly, successfully decoding the chunked encoded response proved to be very research intensive and much more difficult than expected.

III. SUITABILITY OF JAVA AND PYTHON

Java and Python are both incredibly versatile and popular programming languages, but we are interested in which language is most useful to build an functional server. Thus we will be looking at various aspects of the languages and compare their biggest differences in the context of server suitability to help determine which is most appropriate to use.

A. Type Checking

Python is considered to be a dynamically typed languages which means types are checked, not at compile (like Java which is statically typed) but rather during execution. This type checking is done through Duck Typing which is to say, if it acts like a duck, its a duck. Thus if a function during runtime requires that an argument has a certain method, it will cause an exception to occur as it is executing, often crashing the program entirely. This type of type checking has numerous advantages and disadvantages. Using this type checking, Python is able to throw out the necessity for programmers to declare types. This in turn allows for very fast development that is able to bypass much the bureaucracy that statically typed languages such as Java are subjected to, but of course, this doesn't come without consequence. As Python is unable to ensure types are correct during compilation, it is entirely possible for a Python program to crash unexpected after many hours of correct execution. There is no all powerful force to say if a type error will or won't eventually occur in the program. Java, which uses static typing does have this ability. At compile time, you will know whether or not a program will crash due to type errors (excluding dynamic type casting). While this may hinder a programmers ability to write code, it will ensure that once its up, it will continue running, and thats more than Python can ensure.

In the context of servers, it my opinion that since the code could potentially run for months, or years at a time, stability is of more importance than development with less headache.

B. Memory Management

Both Java and Python implement their own special form of garbage collection to manage heap space.

Java uses a form called "mark and sweep" which, when the program feels that its time to take out the trash (maybe if memory is low or nothing important is being done), it will go through all pointer references that is possible given the current state of the system (global pointers, ones in stack frames, pointers pointing to other pointers, etc...) and mark them. It will then go through the entire heap and any unmarked objects are considered garbage and thus will be deallocated. While works perfectly fine and given that one of the most popular languages uses this, it is certainly functional. This algorithm has the downside that once garbage collection is started, it is a fairly lengthy process to finish, and thus is not well suited for real time purposes which we are lucky not doing but it is still something to think about.

Python on the other hand uses reference count on all heap objects to tell whether something is garbage. Anytime a pointer references an object, the object will have a counter at all times to see whether it is being pointed to. Once nothing points at it, the object will be deallocated. This has the advantage of deallocating constantly instead waiting until memory is almost out then taking a very long time just managing its memory; however there is one large flaw. It is entirely possible and that heap objects point to other heap

objects that in turn point to in. In other words, if a cycle can develop in the heap and this style of garbage will never collect that memory and large memory leaks form. Newer versions of Python fix that by calling a mark and sweep algorithm when memory gets real low so it can take care of that pesky issue.

Servers again will be up for large periods of time thus a purely reference count management system wouldn't be great, however the use of mark and sweep with reference count (the kind that modern Python uses) would make this issue vanish. In addition to memory leaks we also want to ensure fast response rates thus, given that Python has all the real time advantages of reference counts and mark and sweep, Python generally has a better memory management system for heap allocated objects.

C. Multithreading

In terms of Multithreading, Python and Java are internally quite different actually. While Python (specifically CPython) might appear to have threading, it suffers from a massive drawback which is that the language is dependent upon a Global Interpreter Lock^{1,2}. This lock actually prevents multi threading from occurring even when multiple cores are available on the system. This in turn means applications will never go above 1 CPU core of processing power. This is mainly done due to the use of C libraries that are not thread safe. On the other hand Java, (besides Java 1.1) does have true multi-threading with native threads³ allowing for much more efficient usage of multi-core systems. This of course makes Java more difficult to program correctly in but if an entire 32 core server only had the processing power of a single core, a lot of potential is wasted. Thus Java's implementation of Multithreading is much more useful for servers than Python's. // Through this analysis, we can clearly see some stark differences in programming approaches that both Python and Java have. Python encourages fast development time, ease of programming, and simplicity while Java opts for more efficient and strict solutions while introducing large programming overhead and complexity. These approaches excel at different scales of server programs but generally, for large and long lasting server implementations, Java seems to be a better choice than Python.

IV. ASYNCIO VS NODE.JS

Node.js and Asyncio are both useful tools to perform server side scripting, however, they differ greatly in area of specialty.

A. Performance

Node.js is a strong performer, its runs on Google Chrome's V8 JIT compiler allowing for code to be run very efficiently and makes great use of low level computer architecture to improve perform⁴. Additionally, it supports multi-threading and with the V8 engine, it is can quite powerful. Python on the other is is straddled in performance as it is forced to be ran on its interpreter, and as previously mentioned, CPython is unable to have true multi-threading thus it simply cannot match Node.js's performance.

B. Ease of Programming

Both Node.js and Python are great in that server and client code can be written in the same language, allowing for less hassle. Python however, is simply an amazing and simple language to code in due to the ease of use. It is incredibly portable and the code can be ran nearly everywhere, it is an object oriented language allowing for large applications to be built in a very intuitive and easy to develop manner, has a massive library support (especially for data science), and has a much larger community of people. Node.js is great as well but is not object oriented, does not nearly have as comprehensive libraries and is fairly new. Both are great, but in terms of development speed and availability of information, Python can simply not be beat.

Both Node.js and Python are great options for server side applications but they truly excel each in their own particular domain. For applications not requiring large server side performance and need to be developed quickly, Python is a great choice; however, powerful and more IO dependent requirements would do much better with an implementation from Node.js than one in Python.

V. CONCLUSION

The asynchronous server network we created benefited greatly from what Python had to offer. Python and the Asyncio library allowed for a very rapid development of a well structured and easy to read server code that also easy to replicate client-side. Alongside Python itself, a host of libraries are also available to allows the ability to easily extend our server program to do some quite amazing tasks such as data mining and quickly learn interesting trends our users exhibit.

Python however is not nearly the only contending language for server side application. Java is a great well developed object oriented language that allows for well structured and easily maintainable code. Java also has a huge assortments of libraries behind it, allowing it also to do amazing things with only a couple of lines of code. The major disadvantage Java has to deal with is the annoyance of it being a strongly typed language. This prevents development from being nearly as quick as the competition (Python or Node.js) but does give a great safety net that looks after potential faults in the code. With a code base as small as the project we developed however, it might be a bit of an overkill.

Aside from Java, Javascript comes bursting out of the gates with Node.js which allows for very fast development of code along with fantastic performance. However, Javascript of course suffers from its age. Being a relatively new programming language, it simply cannot compete with the hordes of support both Java and Python have at their disposal. In addition, Javascript plays the game duck typing, allowing for super quick development, but again, at the cost of potential bugs going unnoticed. For small applications this may be suitable but it might not always be the case that quick development is more important than lurking bugs.

The original task provided to us was to prove the

feasibility of working with Asyncio library to allow for Flood propagated cache updates between servers. While we have shown it is very possible to achieve this, it might not necessarily be the best choice for the task at hand. For the new wikimedia-style service we wish to provide, which will likely have large amounts of traffic (otherwise we would just communicate via a central server but instead of opted for flood controlled caching), have to deal with various amounts of data (GPS locations to ephemeral video data), and need the ability to make the program easily maintainable and extensible, I would cast my vote for either Node.js or Java.

Java will make the program very easily maintainable, extensible, has a large amount of support, has great multi-threading, and is pretty fast in its own merit due to its JIT compiler. Javascript on the other hand allows for fairly quick development speed while still offering little overhead and fantastic performance, however it is much less mature than Java and thus doesn't have nearly the same community and library support as Java does.

Thus, if I were to be safe, and given the requirements of the project at hand, I would recommend Java slightly over Javascript to be the best candidate to build the wiki-media server due to its object oriented nature, allowing for big programs to be constructed, while also offering great performance across the board.

REFERENCES

- [1] <https://docs.python.org/3/library/asyncio-dev.html>
- [2] <https://docs.python.org/2.0/api/threads.html>
- [3] <https://docs.oracle.com/cd/E19455-01/806-3461/6jck06gqe/>
- [4] <https://github.com/v8/v8/wiki/Ignition>