COM SCI 131 - Proxy Herd With asyncio Report

Gabriella Long
University of California, Los Angeles

Abstract

The main goal of this project was to implement Python's asyncio library in an application server herd for a new Wikipedia-style news website. The asyncio library is an asynchronous networking library that is used to process various input and output operations from different sources simultaneously without requiring parallel code execution. This report will explore the possible server implementation of a news site using asyncio. Additionally, the comparison between Node.js and asyncio, the advantages and disadvantages of asyncio, and my recommendations for the hypothetical news site.

1. Introduction

The following report summarizes my research on whether Python, using the asyncio asynchronous networking library, would be appropriate for the proposed "application server herd" application. For this project, students were tasked with building a framework for a new Wikimedia-style service that is primarily going to be used for news.

Currently, websites under Wikipedia use the Wikimedia server platform which is LAMP based; LAMP platforms utilize GNU/Linux, Apache, MariaDB, and PHP+JavaScript. To provide over eighteen (18) million monthly users, Wikipedia uses a virtual router and a web of servers to ensure constant, high performance

This new Wikipedia-esque news site would require three major components: articles to get updates more frequently, access to the server will use various networking protocols to support the traffic, and easy accessibility to smartphones, all three are integral to the success of this new website. Due to the near-constant traffic and updates the website would need to avoid a server framework that will cause bottlenecks. This is where the application proxy herd comes in. To avoid the aforementioned bottlenecks, we could instead use servers that communicate with each other and the associated caches and large databases; this communication between servers allows for quick response to data transmissions, but will still allow servers associated with the given database to handle the less-frequently accessed, more stable, data which reduces some overhead.

The goal to achieve the inter-server communication can be done using Python's asyncio library because of its event-driven nature that would allow fast processing of updates between servers. The application of this library will be explored further in the following sections.

As background, the Python programming language is a popular open source, multi-platform, interpreted, high-level, object oriented language. Python was created and used toward the end of the last century, gaining momentum in applications large and small as interpreters, libraries, and porting efforts have extended its functionality and availability. The asyncio networking library is one example of Python's adaptability to common network applications, improving both the network performance and processing paradigm to match more traditional compiled programming languages yet retaining the ease of use and implementation that have driven Python's popularity [1, 2, 3].

2. Client/Server Implementation

The proposed "application server herd" is composed of five (5) servers named 'Hill' on port 12120, 'Jaquez' on port 12121, 'Smith' on port 12122, 'Campbell' on port 12123, and 'Singleton' on port 12124. The servers are connected to a number of clients, who each connect to their selected server over its assigned port. As each client executes, it sends its GPS location to its assigned server, which in turn transmits the client's GPS location to the other members of the "application server herd". This allows relatively frequent client location updates to all other servers, without the potential performance bottleneck of communicating to the central database server. In addition, the five servers speak to each other in an asymmetric mesh fashion; server Hill communicates only to Jaquez and Smith, server Singleton communicates to all servers except Hill, and server Smith speaks only to Campbell. This network

topology requires that there be multi-hop propagation within the server network to provide all servers with a comprehensive and coherent database.

Clients periodically use the IAMAT message to provide their GPS coordinates. Servers respond with the AT message confirming the server's name as well as any clock skew delta and the client's original coordinates and local POSIX timestamp. Clients can also query locations within a specified range to other clients using the WHATSAT message, including the other client's address as well as radius distance and a maximum count of places to deliver. In response, the server responds with a Google Places message in JSON-format listing as many places as the client requested.

Additionally, the client/server implementation will use a flooding algorithm via AT messages to achieve inter-server communication. A flooding algorithm is a way to propagate data packets from one server to all connected servers. For example, given servers Hill, Jaquez, Smith, Campbell, and Singleton, if Hill receives a data packet, then it will "flood" servers Jaquez, Smith, Campbell, and Singleton; server Hill itself will not receive the transmitted packet again. Since the servers are supporting a site that requires frequent updates, utilizing a flooding algorithm will allow all adjacent servers to receive new data without the need to contact a central server. However, there are downsides to flooding algorithms such as duplicate packets negatively affecting the network bandwidth, but there are ways to limit duplicates. I would assume, given such a large scale proposed project, that there would be safe-guards in place to avoid unnecessarily heavy traffic between the servers.

2.1 IAMAT

The IAMAT command informs the user where the server is, specifically where the server supporting said user is. It takes the client's ID, the client's longitudinal and latitudinal coordinates, and the time at which the client sent the message. It is important to note that the coordinates are required to be expressed in decimal degrees using ISO 6709 notation, and the client's message time must be expressed in POSIX time. The return response to the user will be in the form of an AT message. The message will include the ID of the server that received the client's message, a time stamp, and a copy of the IAMAT command (excluding "IAMAT").

2.2 WHATSAT

The WHATSAT command informs the client of places near other clients' locations. It takes another client's name (as specified by the current client), a given radius, and the amount of information the user wishes to see from the surrounding area within the radius. It is important to note that the radius is given in kilometers, and the amount of information the user enters in the command will act as an upper bound for what they will receive. Currently Google Places API only supports a radius up to fifty (50) kilometers and an upper bound of twenty (20) items of information. As briefly mentioned, the command will return an AT message providing the client's most recently reported location given server communication, and the time at which the communication occurred. Once the AT message is transmitted, a JSON-format message is sent to the client providing the resultant items.

3. Asyncio

As previously mentioned, the server herd was implemented using Python's asyncio library. For a brief recap of the asyncio library: it is an asynchronous networking library that is used to process many I/O operations from varying sources simultaneously without requiring parallel code execution. The proxy server herd was implemented using this library because of the asynchronous client-to-server and server-to-server TCP connections which is incredibly valuable for a task such as a frequently updating news site. asyncio allows servers to complete other operations whilst waiting for user input. If a synchronous approach was taken, the frequent data packets, queries, etc would overload the servers and the service would be quite slow.

3.1 Advantages of asyncio

One of the more important advantages of the asyncio asynchronous networking library is the ability to integrate the ease-of-use that Python offers with the ability to use concurrent programming in a

cooperative multitasking library to efficiently process a thread pool of filesystem, network, and other event-based tasks that under previous versions of Python were difficult to choreograph effectively. The ability to link together Python and C/C++ code opens an expanded opportunity in both code ROI (return on investment) as well as improving performance [6].

3.2 Disadvantages of asyncio

One of the disadvantages of the asyncio asynchronous networking library is since it is a new addition to the Python programming environment is that the amount of testing performed to date may not have identified and corrected any lurking race conditions, timing relative, or priority based edge conditions that may exist. Additionally, asyncio does not support HTTP which is required for Google Places API queries, currently asyncio only supports TCP and SSL. Thankfully Python's aiohttp library fills the gap for Places queries [4, 10].

3.3 asyncio's Suitability for Server Herds

From this brief observance period of Python's asyncio asynchronous networking library, I believe it to a suitable framework to develop and deploy application server herds. The combination of Python's developer base, ease of cooperative tasking for networking applications, and the inherent system performance improvement would be justification to integrate Python's asyncio asynchronous networking library.

It should be relatively easy to develop asyncio-based applications that run and exploit server herds. The only requirement is that developers have a basic understanding of similar thread-based programming paradigms which allow each task to cooperatively time share processing tasks [6].

3.4 Difficulties Implementing asyncio

The only problems I ran into was a definitive mechanism to quantitatively determine performance improvements on multiple machines over realistic network topologies which would have provided more real-world metrics. On a library as new as asyncio, it is important to use the latest version to ensure all bug fixes are present. Overall, the library is relatively programmer-friendly.

3.5 Dependency on Version

Since asyncio is a relatively new library it is reasonable to assume that it works differently according to the version of Python it is operating with. Currently, asyncio works on Python versions 3.3 and later. One of the notable dependencies is the "run" (asyncio.run()) command; prior to Python 3.8 it was not stable. The run command is used to run coroutines in event loops and returns results while actively overseeing other routines. If programmers want stable performance from it then they should ensure Python 3.8 and later are available. However, if there is not an option to update to Python 3.8 and onwards, there are methods to overcome the instability such as having explicit commands to create and execute event loops with little effect on performance [4, 6, 7, 10].

4. Comparison of Java and Python

While Python's asyncio asynchronous networking library and Java's node.js share many similarities, yet Python and Java share some major differences.

4.1 Multithreading

Python's approach to multithreading is also quite different from Java's. Where Python offers a cooperative multitasking paradigm Java offers a more traditional multithreading approach.

More specifically Python cannot have any race conditions when updating references because its approach to garbage collection relies on the reference counts. Unfortunately this means that parallelism is not possible as only one thread can execute at any given time. Rather than losing efficiency with releasing and locking threads, the one thread at a time approach Python is able to execute single-threaded code quite quickly and efficiently.

As previously mentioned, Java supports multithreading to allow for parallelism. Since Java uses locks for multi-threading, the program loses some efficiency due to the overhead of locking and

releasing threads, but gains parallelism. Additionally, you need to be aware of possible race conditions.

4.2 Type checking

One of the aforementioned differences is type checking. Python is a dynamically typed language, meaning checks for type consistency are performed during runtime as opposed to compile time. On the other hand, Java implements static type checking, meaning checks for type consistency are performed during compile time as opposed to runtime. This can be an advantage or disadvantage based on your perception, project requirements, and preference. I prefer static typing for long-term code maintenance, but it comes down to programmer preference and the needs of the project. However, it is worth noting that a dynamically type checked language, such as Python, could potentially miss type errors which will cause errors; this makes debugging a bit more of a challenge, but it allows for more flexibility overall. Like everything in programming, it is a tradeoff.

4.3 Memory Management

While memory management is automatic and scope-based in both Python and Java, there are likely subtle differences in both allocation and garbage collection that may become critical on memory-constrained systems.

For Java the approach is the mark-sweep algorithm. The algorithm first checks all objects to see which ones point to data or other objects, then for the ones that do not point to anything are collected and freed. Since Java does not require a reference count, the count does not need to be continuously updated, and thus is a bit faster of an approach to memory management.

As briefly mentioned previously, Python counts references, it keeps track of the number of times a given object is referenced, and thus it has a more automatic memory management. The garbage collection occurs when there are zero (0) references to an object, so there are no explicit commands required to be included by the programmer. However, if there is a, for lack of a better word, reference loop then some reference counts will never get to zero and will therefore not be removed by the garbage collector. This is a bit of a slower memory management approach.

4.4 asynio vs Node.js

Python's asyncio and Java's node.js libraries occupy similar advantages and positions in their respective languages. With respect to performance and ease-of-development I would expect similar metrics and implementation efforts. Perhaps the Java environment provides a better tool suite or test environment. Or maybe Python's ease of use is more important than Java's. It really comes down to the task at hand.

5. Recommendations

My recommendation on choosing Python or Java as an application development framework would be based on whether there has been any significant development using either language. If there has been significant development, I would recommend pursuing that development to increase the ROI. If there has been little or no development, I would recommend Python for the ease-of-development and the availability of experienced engineers.

6. Conclusion

Given this project's time constraints a proper engineering report can only probe superficially. However, that being said this report can provide topics to drive further investigation to develop test metrics to quantify all aspects of modern software development with regard to project requirements.

Acknowledgments

This research was made possible by Professor Paul Eggert, and the TA team for Com Sci 131. I am grateful for the opportunity to further my knowledge into different libraries which will allow me to gauge performance and the fit of any libraries I implement in the future.

References

[1] *Project. Proxy herd with asyncio*
http://web.cs.ucla.edu/classes/fall20/cs131/hw/pr.html

[2] *Wikipedia at 15: Millions of readers in scores of languages*
https://www.pewresearch.org/fact-tank/2016/01/14/wikipedia-at-15/

[3] *Wikimedia Infrastructure*
https://lugod.org/presentations/Wikimedia_Infrastructure_Utrecht_LUG_2011.pdf

[4] *What's New In Python 3.9*
https://docs.python.org/3/whatsnew/3.9.html#asyncio

[5] *Formatting Submissions for a USENIX Conference: An (Incomplete) Example*
https://www.usenix.org/sites/default/files/usenix-2020-09.pdf

[6] *18.5. asyncio — Asynchronous I/O, event loop, coroutines and tasks*
https://docs.python.org/3.6/library/asyncio.html

[7] *Coroutines and Tasks*
https://docs.python.org/3/library/asyncio-task.html

[8] TA Kimmo Kaerkkaeinen's Week 7 Slides
https://ccle.ucla.edu/mod/folder/view.php?id=3430933

[10] *asyncio 3.4.3*
https://pypi.org/project/asyncio/