CS 131 Project: Analysis of Asyncio through Implementation of Proxy Herd

Ryan Wakefield
University of California, Los Angeles

Abstract

The goal of this project was to implement a Proxy herd using the *asyncio* asynchronous python library. We built this prototype to determine if using the *asyncio* library to create the herd is an effective and efficient library to use in order to construct an "application server herd" where multiple application servers communicate directly to each other as well as through the core database and caches. Finally, we compared this implementation to a Java-based approach and to using Node.js.

Testing Proxy Herd

Implementation and testing occurred on my local machine as well as the UCLA SEASnet Server, using Python version 3.9.0.

1. Introduction

Wikipedia and its related sites are based on the Wikimedia server platform, which uses GNU/Linux, Apache, MariaDB, and PHP+Javascript, using a load-balancing virtual router and multiple redundant web servers. We want to build a new service that is designed for news, and, because of these changes: (1) updates to articles will happen more regularly, (2) access will be required via various protocols, not just HTTP, and (3) will be more mobile. Because of these features, the PHP+Javascript application server will create a bottleneck and needs to be changed to accommodate the new service. The current software makes it too difficult to add newer servers, so we need to change the design. The "application server herd" and its interserver communication is designed for rapidly-evolving data whereas database server will be used to store more stable data that is less-often accessed. The ***asyncio*** library is a potential candidate because of its event-driven nature that should allow an update to be processed and propagate to other servers in a timely manner.

2. Comparison of Java and Python

Type Checking

Python is a dynamically typed language. The Python interpreter checks the code only as it being run and determines the type of the variable at runtime. Thus, the type of this variable can change over the lifetime of the program. This allows for more freedom in code by allowing the types of variables to change as needed. However, this also creates the possibility for more errors, as you will not know if you are comparing or using compatible types with other variables until your code is being run. Additionally, not having to declare types allows the code to be more readable, as you don't have to write the additional code that takes away from the core logic.

Java is a statically typed language much like C and C++, in which variables are typically not allowed to change types (unless casting between these types are allowed). At the time the variable is declared in the code, the type must be declared along with it, and the variable will be bound at compile time to whatever type the variable was declared with. This allows less freedom with your code, but it will help prevent bugs that occur from TypeErrors.

Type checking during runtime slows down the code, as the code will have to look at the data and interpret it on the spot as opposed to knowing the variable type before runtime and have the memory allocated for storage. Overall, type checking would be more beneficial in this instance as it would speed up the program to allow for propagation across servers to happen quicker.

Memory Management

Python uses dynamic memory allocation which means that the memory is allocated at runtime due to the fact that variable types aren't determined until runtime. A heap is used for implementing dynamic allocation in python. Memory in the heap is garbage collected, meaning that unused memory will be automatically deallocated when it is no longer being used by the program.

Java uses both Heap and Non-Heap memory. The Java Virtual Machine creates default non-heap memory that stores runtime constants and code for

methods and constructors, as well as internal Strings (JMM for JVM, Gesso 2017). Heap memory for Java is used for all java class instances and array is allocated.

The local stack is faster to allocate, deallocate, and access memory from given the nature of how it works, making Java the better choice in terms of speed for accessing memory, and both are good for memory management given that they both use garbage collectors.

Multithreading

Multithreading uses shared memory from a process and allows a single process to have multiple code segments (threads). These threads perform different tasks but share the heap memory with the other threads.

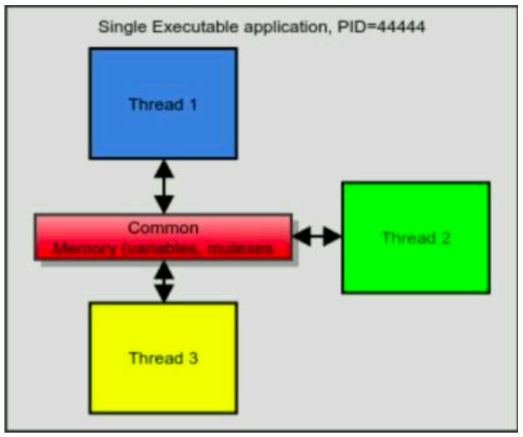

Figure 1: Chart of a Single Process with Multiple Threads, Sharing Memory (Taken from Kimmo Karkkainen's slides).

Python uses a Global Interpreter Lock (GIL) to implement multithreading. Only the thread that owns the GIL is allowed to execute, all other threads must wait to obtain the lock. Python uses GIL because of the way that Python performs memory management. Memory management in Python depends on reference counting. Objects in Python are deallocated when there is no reference to the program (i.e. the reference count is 0). Because of this, possible race conditions can occur in multithreaded programs with Python. However, GIL creates many advantages as well. Because of the simple memory management implementation, the single threaded code will execute faster compared to other types of garbage collection.

3. Asyncio Library Python

Asyncio was introduced in Python 3.4. It is a single-threaded approach for concurrent programming, and thus does not use parallelization. With this cooperative multitasking library, tasks in the program are allowed to voluntarily take breaks to let other tasks run. Thus, while other tasks are waiting for a response, another task can run, which will make the program more efficient.

The basic idea of asyncio is the use of the event loop.

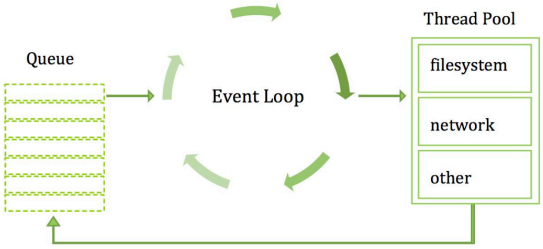

Figure 2: Event loop used to implement asycnio (Taken from Kimmo Karkkainen's slides).

The event loop runs task that are waiting to be executed. When the thread encounters an await keyword in the program, it will break to let other tasks run. It will be added to the queue and called later when the function is ready to resume after it has the information it was waiting on.

There are two key components to asyncio: the *async* keyword and the *await* keyword. The *async* keyword is used when defining the function, it defines the function as a coroutine. This allows the function to suspend its execution and give control to a waiting coroutine. The *await* keyword suspends the execution of the current coroutine until the function that is being awaited upon is finished, at which point it can continue.

4. Implementation of Proxy Herd Assignment

Our prototype consists of five servers: Hill, Jaquez, Smith, Singleton, and Campbell. These servers communicate bidirectionally with each other, in the following pattern: Hill speaks with Jaquez and Smith, Singleton speaks with everyone but Hill, and Smith speaks with Campbell.

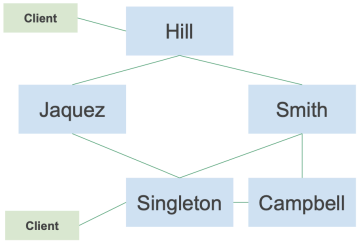

Figure 3: Architecture of the Proxy Herd implemented. Shows bidirectional connections between servers.

These servers accept TCP connections from clients using the following commands:

IAMAT [ClientID] [Location] [POSIX Time]

WHATSAT [ClientID] [Radius (in kilometers)] [Maximum Number of Places Returned]

(Location is in ISO 6709 notation and POSIX time is number of seconds and nanoseconds since 1/1/1970 00:00:00 UTC).

When a client sends a valid "IAMAT" command to a server, the server will store the client's information and then flood the client's information to the servers that it communicates with, in the form of the following:

AT [ServerName that received IAMAT] [timeDifference] [ClientID] [location] [startTime]

These servers will save and propagate the information to all the servers it is in contact with until all the connected servers have received the new message and updated their information. When a client sends a "WHATSAT" message to a server, the server will establish an HTTP connection with Google Places, which will process the information you send it and return back to the server in JSON format a list of places near the location you send (an API key is needed to use Google Places). The server will then return the given "AT" message shown above, followed by a list of places that was sent to the server from Google's servers in JSON format. If a client sends an invalid command, then the server will send an error message in the form:

? [Original Message]

And terminate the connection with the client.

4. Difficulties with Proxy Herd

A few difficulties were encountered while implementing the proxy herd. One of the ones was learning how to use the *asyncio* library. After reading the documentation and looking at some examples of asyncio, it was fairly easy to integrate into code. Another issue encountered while doing this project was properly establishing connections between servers when implementing the flooding algorithm, as well as preventing the flooding algorithm from looping infinitely. I was able to use asyncio open_connection function to establish a connection with another server, and then relay the message to the server that it established a connection with. Finally, I created a dictionary that holds the most recent message received from each client. If the most recent message matches the one being received currently, then it will not propagate the message to the other servers.

5. Writing Application with Asycnio

Once you see some example uses of asyncio, it is fairly easy to implement concurrent programming with this library. Overall, this would be a valid option to use in order to write a server.

6. Asyncio vs. Node.js

Node.js is an asynchronous event-driven JavaScript runtime used to build scalable network applications. Node.js will establish a connection and then sleep until there is work to be done. Node.js uses an event model and uses an event loop, like asyncio does as well. It will exit the event loop when there are no more coroutines. Node.js uses a callback (calling a user back when the task is complete) in its event loop instead of coroutines. Node.js doesn't directly perform I/O, while asyncio does. Thus, because of the abundance of I/O operations in our proxy herd, it will be beneficial to use asyncio over Node.js for this project, however the two are very similar to each other overall.

7. Asyncio HTTP Requests

One problem with *asyncio* is its inability to perform HTTP requests, specifically to send requests to Google's servers to receive the list of places to Google places. This can be fixed, however, using a library such as *aiohttp.* Using this library along with *asyncio* will resolve the lack of functions that handles HTTP request in *asyncio.*

8. Performance Implications Using asyncio

Using an asynchronous library for concurrent programming is very advantageous for implementing a proxy herd because it allows for multiple messages to be processed simultaneously, which will prevent one message from backing up all of the servers. Even though it isn't multithreaded, multithreading doesn't always improve for CPU intensive tasks, especially when factoring in locking and context switches. Thus, this isn't much of a disadvantage, as at most you will be gaining little increase in performance by using multithreading.

9. Dependence of asyncio Features in Python3.9

Most of the new features of *asyncio* in Python3.9 weren't used to implement to implement the Proxy Herd. Asyncio.run() was updated to use the new coroutine of asyncio, however it was introduced in

version 3.7 and thus older versions of Python can be used to implement the Proxy Herd using asynico. Additionally, in version 3.9, the function “python -m asyncio” launches a REPL that can be beneficial, however this was not really needed for this assignment, and this feature was also available in Python 3.8, so you can still use it in some older versions of Python. Overall, there is not a huge dependence of asyncio features introduced in Python version 3.9.

10. Recommendation

It is my recommendation that asyncio is used to implement a proxy herd. In terms of readability, ease of use, and asynchronization which allows for multiple messages to be processed simultaneously, asyncio would be a very strong method of implementation. Asyncio is the optimal balance between implementation and performance, and I would recommend this for the creation of a proxy herd.

11. Conclusion

After researching asyncio, reading the documentation, creating a prototype of the proxy herd using asyncio, and researching possible alternatives, I have found that using asyncio to update the Wikimedia server platform would be a strong candidate. This would allow for efficient updates to Wikipedia’s news articles, and other libraries can be used for access to various protocols, such as *aiohttp* for HTTP requests.

References

[1] Type-checking Python, https://realpython.com/python-type-checking/

[2] Memory management in Python, https://towardsdatascience.com/memory-management-in-python-6bea0c8aecc9

[3] Memeory management in Java, https://www.betsol.com/blog/java-memory-management-for-java-virtual-machine-jvm/

[4] About Node.js, https://nodejs.org/en/about/

[5] TA Kimmo Karkkainen’s Discussion Slides