# Project Report

Saman Hashemipour
*University of California, Los Angeles*

## Abstract

In our research, we analyze the suitability of creating a proxy for the Google Places API using the Python asyncio networking library. This application is used to evaluate asyncio in replacing a Wikimedia Architecture like LAMP as described in the specification. Wikimedia Architecture requires the ability to make frequent updates using various protocols, many of which are described in detail throughout this report. When compared to Java, Python has much less strict type checking, which will require less typecasting for many of the values that we will check. We also compare Python's asyncio library to the increasingly popular Node.js for Javascript. This comparison will show that due to multiple reasons, which will be discussed in further detail throughout this report, Python and its asyncio library will be recommended for our assignment.

## 1. Introduction

We are presented with many different architectures when deciding to develop an application similar to the one developed here. Each of these comes with different tradeoffs and have different benefits for particular projects. As described in our spec, the Wikimedia sites use a LAMP architecture which is based on Linux, Apache, MySQL, and PHP. This architecture uses web servers and the bottleneck is primarily at server application. To improve (and possibly eliminate) this bottleneck, we look to alternative architectures such as the asyncio library in this project.

For this project, we analyze the reliability and how maintainable applications using asyncio are. Our servers will make HTTP requests and respond to clients, it will also flood data between the server and the client. Servers should maintain a level of independence that allows them to continue performing their operations even when their neighboring servers don't respond due to a shutdown or a crash. Throughout this assignment, we will compare the advantages of using asyncio to alternative architectures like Java and Node.js. Some points of comparison will be attributes such as ease of use, reliability, maintainability, and performance. Although we expect to have our Python asyncio library to be preferable to the others, it is important to take a look at alternatives in order to see why this library is best suited for the task at hand.

## 2. Implementation and Design

This implementation of the Proxy Herd Application uses asynchronous event loops which are a critical part of the asyncio library.

Each connection to a client to the application server herd is treated like a TCP connection, and this is then added onto the event loop as its callback. The event loop processes each callback on its queue one by one while waiting for an input or a response from a server request. Once a client has made a connection the server awaits a message that follows one of the following formats: an IAMAT or an AT message, or a WHATSAT request. Each of these will be discussed in greater detail later in the report, but for now, it is important to understand that they are used by the client to communicate with the server and that they are what the server expects from the client.

## 2.1 The IAMAT Command

The IAMAT message has the duty of informing the application server herd of the location of a given client. The IAMAT message contains the client id and the latitude and longitude position of the client as well as the time that the message was sent. Each of these must be some non-whitespace string. The time is then calculated to determine the difference in the time that the user send the message to the time that the server actually received it.

## 2.2 The WHATSAT Command

The WHATSAT message is useful in giving the client a response that informs them of the places close to their current location. Much like other location-based applications, the WHATSAT message will give the user a list of locations based on their own given a particular radius. The WHATSAT message contains the client's ID, the radius to do the search and the number of results they wish to see. The radius must be within a range of 0-50 (representing miles) and the number of results must be a number in the range of 0-20 (representing the number of results the user would like to receive). The response is very similar to the IAMAT response, with the minor difference that after a newline the JSON response from Google Places that is associated with the query is appended at the end, which is limited to the number of results specified by the request. If the client has not provided their location using a IAMAT message, however, the server lacks the information about that particular client the server will instead respond to the client with an invalid message response, containing a ‘?’ followed by the message sent by the client. This is to be expected considering that if the server does not know where the client is there is no way for it to return any locations that the client would like to see, because the client has not provided where they are currently.

## 2.3 The AT Command

The AT message is similar to the IAMAT message, but it contains the name of the server that received the original IAMAT message. The message from the original IAMAT message is appended to the AT message. If we see that the client exists on the server already and the saved time is older than the newly received time, then our server returns control to the event loop. If not, the server will process our coordinates, time and message from the AT and will save and associate them with the client’s name. The server will then use its flooding algorithm which will write to each of the neighboring servers, we call this writing to multiple servers *flooding*.

## 2.4 Unrecognized Requests

We don’t expect our users to be perfect, but as a programmer, user error is one of the many pitfalls that we must look out for. If the user were to give a request that we cannot recognize we respond to the client with a ‘?’ followed by the message that they sent the server. These requests can be either an unrecognized message type, one that is not: WHATSAT, AT, or IAMAT, or even a correct message type but incorrect formatting (likely a white space one of the message fields). Regardless of the type of request, we simply send back the message to the user and hope that they will send a correct message next, if they do not, we will simply resend a ‘?’ followed their message until they do so.

## 3. Using asyncio

### 3.1 Advantages

Using the asyncio framework makes running servers that asynchronously handle requests easy to implement. Each server has an event loop and you can queue co-routines to the event loop whenever you recieve a new message or connection. Since the server is asynchronous it will not process the particular request until it has made it to the front of the queue. This is important when it comes to dealing with multiple requests at the same time (or at least very close to one another). By using the feature of asynchronicity you can make a connection at the same time that you are processing an incoming message. This asyncio library is fantastic because it contains all the components that we would need from the server implemented in the framework.

Some other advatantages to asyncio is that creating a new server instance is easy by simply running your server code with the name of the server as the first parameter. Lastly, the final advantage is that there are a number of libraries that work with asyncio to create HTTP GET requests and other web interactions. Some of these libraries include the aiohttp library which we have used in this project.

## 3.2 Disadvantages

One of the biggest drawbacks from using the asyncio library is that not all tasks are dealt with in the order that they are received. This is an issue because for our program, in particular, the user must send their information before we can respond with useful data, this has the clear issue of causing bugs in our code and difficulties on the user side. Another major issue is if we have a server herd processing messages in the incorrect order will become much more likely and we will have to look out for servers sending messages to each other in the “wrong” order due to the lack of synchronicity. Lastly, a drawback of the asyncio framework is that is doesn’t allow for the use of multithreaded servers. The server framework that asyncio runs is on one thread only. For a small scale project such as our own asyncio may work perfectly, but we may look for a performance increase if we decided to scale our project up in size, and asyncio is not the solution for that larger project.

## 4. Comparing Python and Java

### 4.1 Checking Types

Python is unlike Java in that it uses dynamic type-checking. In Python, we are only concerned with whether or not the object in question is available. If we have an object in Python that uses a certain variable or method we use it. There is no restriction on type since type checking is dynamic. In Java, however, we use static type-checking. This means that in order to use a variable with a particular method we need to ensure that the variable type is the same as the parameter type specified for that particular method. This static type checking is very common among programming languages but removes a certain element of flexibility. When a piece of Java code is compiled using the Java compiler, it knows all the different types that have been declared at compile time. It then uses this information to determine each of the object’s available behaviors and will give us an error if we attempt to call upon one of these behaviors that a particular object does not have.

In Python, we follow a different set of rules. If we would like to make sure that a particular instance of a certain type of variable or function is correct we can use the *type()* function which will return to use the particular type of a variable or function. Python’s relaxed type checking allows for many less noticeable errors, however, this feature of being relaxed is what makes Python such a flexible language with simple interfaces.

### 4.2 Type Casting

In Java, type-casting is a necessary part of making your code acceptable to compile and run. Many different Java codes are littered with type-casted variables that indicate the correct behavior of an object. Python lacks this excessive type-casting due to the dynamic type-checking that Python is so popular for. In Java, if we wanted to parse a message, for instance, the code becomes messy and filled with a multitude of different type casts. Some for numbers we want to find, some for strings, etc… but in Python, we can quickly go through a string and find all the values we need without the clutter of type casts. This feature is one of the many things that makes Python so preferable in the Proxy Herd Application.

### 4.3 Memory Management

Python manages its memory during execution by using something called reference counting. This makes it so that all the objects that are created during the execution of a particular Python program has a certain reference associated with it, once we create another different object we increase the reference counter by one and assign it to the new object. If we delete an object we simply decrement the reference count and move on.

Java, on the other hand, is a lot different. The Java garbage collecter from the JVM uses a “trace algorithm” which goes through all the objects and searches for which ones are “live” and which ones are “dead”. Whichever objects are still dead after the next garbage collection cycle. Will be deleted.

### 3.4 Multithreading

A major disadvantage of Python, when compared to Java, is the lack of multithreading support. Python uses a global interpreter lock to synchronize threads such that only one thread is being executed at one time. Java, however, uses multiple core processors to complete more processes in the same amount of time. This means that a server built in Java is more scalable than a one in Python.

## 5. Node.js vs asyncio

Node.js is the framework for Javascript for writing server-code while asyncio is the framework for Python. Each comes with pros and cons that make them good for different types of applications.

### 5.1 Similarities

Neither Node.js or asyncio supports multithreaded applications. Both have instead have taken advantage of event loops. These event loops have been mentioned in detail throughout this report.

### 5.2 Differences

In Node.js we use callbacks implemented with semaphores, unlike asyncio which appends coroutines to the event loop. This requires Node.js to have to nest its callbacks together which causes problems for us. In hopes of simplifying this callback chain the Promises object was designed and if the procedure is successful we see that the Promise’s *then()* function and will handle whether the Promise is resolved or rejected. Node is useful for creating good server-side web applications, but it lacks the ability to be really good at processor intensive task with I/O, which is why our choice of using asyncio is better than the alternative of Node.js. Lastly, using asyncio is not only better performance wise for this particular task, but it is also written in a more synchronous “style” and gives us good readability through the use of keywords that help us write and maintain the server code.

## 6. Conclusion

From our investigation, we can see that although there are a number of things that Python and asyncio can do to improve, it is still very good to use for the task that we have at hand. There are no major performance pitfalls, and compared to alternative implementations this is one of our best choices. To conclude, our study recommends using Python and asyncio in the implementation of this Wikimedia-style server herd application.

## References

[1] "Asyncio - Asynchronous I/O ¶." *Asyncio - Asynchronous I/O - Python 3.7.3rc1 Documentation*, www.docs.python.org/3/library/asyncio.html.

[2] "Memory Management Reference." *4. Memory Management in Various Languages - Memory Management Reference 4.0 Documentation*, www.memorymanagement.org/mmref/lang.html.

[3] Saba, Sahand. "Understanding Asynchronous IO With Python's Asyncio And Node.js." *Math Code by Sahand Saba Full Atom*, 10 Oct. 2014, www.sahandsaba.com/understanding-asyncio-node-js-python-3-4.html

[4] Radcliffe, Tom RadcliffeTom. "Python vs. Java: Duck Typing, Parsing on Whitespace and Other Cool Differences." *ActiveState*, 3 Feb. 2019, www.activestate.com/blog/python-vs-java-duck-typing-parsing-whitespace-and-other-cool-differences/.

[5] "Global Interpreter Lock." *Wikipedia*, Wikimedia Foundation, 20 July 2018, www.en.wikipedia.org/wiki/Global_interpreter_lock.