CS 131 Project Report

Sinan Cem Cetin

Abstract

The aim of the project was to judge the effectiveness of Python's asyncio library when implementing a parallelizable proxy for the Google Places API. This is used to figure out if the event-driven networking of the library would be a good fit for an application server herd architecture to replace part of or all of the Wikimedia platform for this application. I was able to examine the pros and cons of using such a method by testing the program I made.

Introduction

The Wikimedia platform used by websites such as Wikipedia uses a LAMP platform that uses multiple, redundant web servers behind a load-balancing virtual router for the sake of reliability and performance. While this platform works well for Wikipedia, this might not be the case for a Wikimedia-style service designed for news where the updates to articles happen way more often, access will be required using various protocols and not just HTTP and clients will be far more mobile. In such a service, this platform would lead to bottlenecking and would make adding new servers, such as cell phones harder.

My aim in this particular project was to determine if an application server herd style architecture would be a better fit for such a service using Python's event-driven networking library, asyncio. While asyncio's event-driven nature would make updates processed and forwarded rapidly to other servers in the herd, my job was to assess how well it worked in process.

Problems That Had To Be Overcome

Writing in asyncio, which is a library I had no prior experience with was very difficult at first. There were very few resources to help with my understanding of the library outside the asyncio documentation itself as asyncio is a relatively new library to Python. Another rather confusing aspect of the writing a program using this library is that the sample code given in the documentation was outdated compared to the version of Python I was writing for, which was version 3.6.3. This particular lack of knowledge on how to apply the functions within the library made it particularly difficult to figure out how to implement the application server herd. Another particularly difficult part of the code was sending the GET request to the Google Places API. At first there was the issue of the GET request having to be very precise with any imperfection in the request leading to a 404 or 400 error being returned from the API. Then after the API returns a response, there was the matter of processing the JSON that Google was sending back. This required not only using another library but also removing random strings that were inserted into the JSON. While this process could have been easily simplified with the use of the Python aiohttp library, this was not allowed in the spec for full credit. This slowed down the development of my program significantly.

Comparison to Java based Approach

To figure out if asyncio is a suitable library to use in practice, I had to first look at the areas concering Python's type checking, memory management and multithreading. Asyncio is a great abstraction to many low level implementations that would be harder to grasp and write otherwise. As a result, development of an application server herd is far easier and more robust than without it. Since Python is dynamically typed, as a developer, I did not need to worry about the types that objects had to be and that made writing the code much easier. Java, which is statically typed wouldn't be as readable but would be clearer. Since Pyhton's interpreter uses duck typing to figure out the type of an object based on the context of its usage, it is hard for someone who doesn't know how an application works to grasp the the details of the application.

Python's memory management system creates references to the objects created and then uses reference counting and non-moving mark-and-sweep garbage collection as opposed to the usual standard mark-and-sweep garbage collector. This ensures that within the application server herd, the objects created are deleted when they are no longer referenced and there is no need to wait for the garbage collector to free up memory. Since Python is on a heap based memory model, the program becomes easier to develop as the programmer does not need to worry about allocating objects on the head like in Java.

Asyncio uses a single-threaded event driven asynchronous model to perform networking operations because of Python's single-threaded implementation. Doing the same in Java would have to use a multi-threaded approach which would rely more on the system the server is running on and the number of available processors. A similar application in Java could be faster or slower depending on the machine running the application while the event loop in asyncio would have similar performance on different machines and would work better when using the horizontal scaling of a server herd.

Recommendation and Node.js

Using the aiohttp library in conjunction with the asyncio library makes writing a program using asyncio much easier and significantly speeds up the communication with the Google Places API and also eliminates the issue of the JSON response from the API containing random strings within. I recommend using the asyncio library with aiohttp as I didn't have to filter the JSON response from Google to find added random strings and the speed of the response is far faster as the aiohttp get function doesn't have to be awaited and the GET request doesn't have to be written on its own.

All in all though, asyncio was a very powerful and simple library in Python, which is a very useful programming language. I would recommend the use of Python and asyncio due to its robustness and its abstraction of more complicated lower level procedures. However, it might be worth checking out Node.js as a possible alternative to creating an application server herd. Since Node.js is built on top of the asynchronous model too, so it would be useful to see which implementation would work better. Node.js also runs atop of Google's V8 JavaScript Engine and this makes it easy to integrate it with existing web applications that need a server. Since asyncio is built on Python it would be more difficult to integrate it into web based servers.