Python Project Report
CS 131

Abstract

This paper analyzes the pros and cons of Python's asyncio library to build an application server herd prototype as an alternative implementation for Wikimedia to improve the speed of client server communications. The paper analyzes asyncio on its own, compares the choice of Python over Java, and examines the similarities and differences of asyncio and Node.js.

1 Introduction

Wikipedia is currently implemented using the Wikimedia server platform which is based on Linux, Apache, MariaDB, PHP, and JavaScript. This combination is very efficient for use by Wikipedia which consists mostly of large and less frequent requests. However, for a service that has more frequent requests, has various protocols other than HTTP, and mobile clients, this setup will not be very efficient.

This paper looks into using an application server herd approach to handle such requests. The setup consists of multiple servers that each cache data individually and share it among themselves instead of updating one common database. When one server receives a request, it updates its local cache accordingly and propagates the new data to its neighboring servers through a technique called flooding. This approach allows all servers to be rapidly updated with the new data without having to access the core database.

To implement this approach, this paper uses the asyncio Python module. This module is used to write concurrent code in Python that runs on a single thread. The paper will discuss the benefits and costs of using this library.

2 Evaluating asyncio

asyncio provides the user with numerous tools to develop asynchronous applications. Through abstracting event loops and coroutines to a couple methods that can be called by the user, the Python library makes writing asynchronous code very straightforward. Coupled with the simplicity and readability of Python syntax, building concurrent applications in Python with asyncio is very simple.

The reliability of asyncio is also very strong due to the implementation of the event loop. The event loop keeps track of processes that need to be executed and decides based on the requirements of each one which one to execute next. Assuming the developers that built the implementation of the event loop did a good job, asyncio should be very reliable in handling the large number of requests received by each server.

3 Python vs. Java

Java has a lot of features missing in Python, so it is important to consider the language as an alternative and discuss why Python with asyncio was the better choice.

3.1 Type Checking

One major difference between Python and Java is that Python is dynamically type checked while Java is statically type checked. Since Python is an interpreted language, types of variables and statements are only checked during runtime. However, Java verifies the types of its variables and statements during compile time.

The main issue here is that Java will catch more bugs at compile time and will be more reliable in terms of types. This issue arises when parsing through cached data when they are stored as strings but are treated in other parts of the code as floats. This can be avoided through careful coding practice and documentation and is not a significant detriment to picking Python over Java.

3.2 Memory Management

In both languages, memory management is hidden from the implementation. Each language employs its own version of cleaning up unused data. Python keeps track of reference counts to every object. The reference count is the number of pointers that point to the object. When the reference count is 0, meaning that no pointers are pointing to the object, the language frees up the memory used by the object. This fails when there are circular references because if A points to B and B points to A and nothing else points to A or B, both objects have reference counts of 1 but are not accessible by the program. To solve this, Python implements a garbage collector that will find these memory leaks every once and a while and free up the space. Java, on the other hand, utilizes a MARK and SWEEP garbage collector that doesn't allow for any temporary memory leaks.

Anyhow, in either case, memory is abstracted from the developer and neither language has any significant benefit in memory management over the other, so this does not play a role in deciding which language to implement the approach in.

3.3 Multithreading

Python handles multithreading inefficiently through the Global Interpreter Lock which protects against race conditions. Java, on the other hand, has the Java Memory Model which is known for its efficient multithreading capabilities. Due to this, Java is a much better candidate for multithreading.

However, an application server herd does not simply rely on multithreading. Since there is a large amount of communication to the servers and between servers, there are a lot of network requests being sent which are usually considerably slower than the computation that happens on each server. This means the benefit and speedup gained from being able to execute other tasks because of the event loop is far more than the benefit of efficient multithreading within each server. For this reason, using asyncio and being able to asynchronously execute other tasks while waiting for network requests makes the Python + asyncio a much more efficient implementation for our application than a multithreaded Java approach.

4 asyncio vs Node.js

Node.js and async.io both tackle the problem of developing asynchronous programs in JavaScript and Python respectively. Both are used when developing web servers that handle large numbers of requests and in general are very similar to each other except the language they are built for.

However, there is one key difference between Node.js and asyncio. When using asyncio in Python, asynchronous methods must be explicitly defined with the keyword 'async' before the 'def' keyword. In JavaScript with Node.js, on the other hand, all functions are treated as asynchronous by default. This means the developer gets more flexibility with asyncio but Node.js is most likely more optimized in terms of performance.

Either way, both Python and JavaScript are popular languages with widespread use and have huge community support and documentation. It is difficult to determine which of the two approaches would be more beneficial for this application without further research and testing.

5 Prototype Design

The design of the application server herd prototype was fairly straightforward. Every server operates on its own with its own local cache and the existence or failure of other neighboring servers doesn't affect its performance.

When a request is received by a server, it parses the request, by determining what type of request it is and then extracting the relative arguments. If the type of request isn't recognized by the server, it responds to the client with a '?' followed by the request received.

If the request is of the type 'IAMAT', the server updates its local cache with the new information about the client if the client's sent timestamp is after the timestamp the server has of the clients most recent request. The server then responds to the client with an 'AT' command. Next, it sends the 'AT' command to all its neighboring servers that it can successfully make a connection with as part of the flooding implementation.

If the command is of type 'WHATSAT', the server determines if it has the information about that clientid stored locally, and if it does responds with an 'AT' command and sends the data from a Google Places API request.

Finally, the server can also handle commands of type 'AT'. This command is used to implement flooding between servers. The server processes the data received in the 'AT' command and determines if it has already received that data or has more updated data by comparing the timestamp of the data in the 'AT' command the timestamp of the data it has cached. If the server decides to update its local cache, it then propagates the 'AT' command to all its neighbors.

A logger based on Python's logging module logs opened and closed connections between servers and clients and between servers.

6 Conclusion

The implementation of an application server herd must be done by an asynchronous framework. asyncio is definitely one of the right choices for building this application. The handling of asynchronous network requests by the event loop provides efficient and robust code to handle a large number of requests. The simplicity and readability of Python makes this approach an even more favorable. However, this report does not dive into enough analysis to determine if asyncio is the best approach for this application but rather concludes that it is one of the top choices.