Proxy Herd with asyncio Report

Abstract

In this project, we implemented asynchronous programming for Python with asyncio to host an application server herd such that multiple application servers can communicate directly to each other with certain command. The communication is consisted of three type: IAMAT, AT, and WHATSAT, each representing unique query or response. In this paper, we examine the pros and cons of this implementation, the effectiveness and performance of asyncio-based programs, its relevance of python version, and comparison with Java-based approach and asyncio of Node.js.

1 Introduction

This project is aimed to examine a new architecture called an "application server herd" for a new Wikipedia-style service. This architecture let multiple application servers communicate directly to each other as well as via the core database and caches, which is designed for rapidly-evolving data whereas the database server will still be used for more-stable data.

To realize this service, we are assigned to use Python's asyncio asynchronous networking library, since asyncio's event-driven nature should allow an update to be processed and forwarded rapidly to other servers in the herd. So, a simple and parallelizable proxy for the Google Places API is the main implementation as our prototype project.

Specifically, we host five servers that some two of them can communicate with TCP connection with IP address and DNS names. A client should be able to send its location to the server by sending a message using this format:

IAMAT [client ID] [latitude & longitude] [POSIX time]

The server should also respond to clients with a message using this format:

AT [server ID] [time diff] [client] [loc] [time]

Also, clients can query for information about places near other clients' locations, with a query using this format:

WHATSAT [client] [radius] [upper bound]

The server responds with a AT message in the same format as before, giving the most recent location reported by the client, followed by a JSON-format message in the format as Google Place API for a Nearby Search request.

Also, a flooding algorithm is implemented such that each server would propagate the location updates to the servers that it can reach until no new server receives this update.

2 Comparison with Java

2.1 Type checking

The main difference of type checking between python and java is that python use dynamic type checking at the run time, while java mainly uses static type checking at the compile time. In java, basically all variable is defined with specific type, but python doesn't specify the type of the variable while declaring it, such that the variable in python can be flexible changed into different type.

The main advantage of python's type checking is that people don't have to worry about type when coding, so that people can use duck-typing and avoid unnecessary type check with flexible type coercion. Also, the code appears to be more concise. Comparatively, Java's type checking at the compile time can avoid ambiguity and result in less error when running the program, so it is easier to be maintained in the long term.

Since this project is a prototype to examine whether asyncio is a suitable framework for the application, python is a better choice for the swift development of the program. However, if the company aims to maintain the application in the long term with sophistication, the advantages of type-checking of Java makes Java a more robust and reliable programming language for this application.

2.2 Memory Management

Python and Java both use garbage collector to allocate and free memory, such that we don't need to manually free the allocated memory, since the collector itself would free the memory when the memory is running low by deleting the objects that are not referenced or can be reached in the program.

However, they have some differences. Python uses the reference counts such that each object has a count that depends on the number of times that it is referenced by another object. Then, the object can be deallocated when its count reaches zero. Comparatively, Java uses concurrent mark sweep(CMS) method such that the memory manager sweeps through objects from the root of the object tree and marks all the unreferenced objects/ Then, the marked objects would be deleted in the next round of sweep. Although Java's approach is slower with two steps, Python's approach is less reliable when circular dependencies exist.

Since there's no circular dependencies in our application, Python's approach of memory management appears to be reliable and more efficient.

2.3 Multithreading

While Java has Java Virtual Machine(JVM) that supports multithreading on multiple CPU cores, Python fundamentally doesn't support multithreading, because it implemented global interpreter lock(GIL) so that python has asynchronous reference counting. Also, since python is written in C, further implementing python program into multithreading would result in concurrency problems.

Since our application doesn't have heavy computation that can be optimized by parallelism, the single-thread program in python doesn't affect the performance a lot compared with implementation in Java.

2.4 Conclusion

Python is a good programming language for our application, since our application would not have tough memory management issue under python, and can work with good performance with single-threading. Also, python is easier when implementing with many library and dynamic typing. However, if the application is going to be extended with massive functions and dependencies, Java would be a better choice with its static type checking, reliable garbage collector, and ability of multithreading.

3 Comparison of asyncio with Node.js

3.1 Overview

Asyncio and Node.js are very similar in many aspects. First, both Python and JavaScript doesn't support multithreading, and asyncio and node.js respectively provides a solution for its programming language. However, node.js is designed to be asynchronous, so node.js basically has better efficiency. Second, both python and JavaScript are dynamically typed, while Node.js has slightly more static typing compared with asyncio. Also, they share the await keyword can handle asynchronous operations without concurrency issue.

However, asyncio is better at processing I/O with many built-in functionalities and performance optimization. For example, asyncio.start_server() can directly start a TCP connection between two servers.

3.2 Conclusion

Since our application has massive I/O and TCP connections among the servers, we can say that asyncio is a more suitable choice for our application.

4 Analysis of Asyncio

4.1 Asyncio for server herds

It is relatively very easy to implement server herds with asyncio. First, there are some built-in functions to establish TCP connections (asyncio.start_server()), maintain the connection

(server.serve_forever()), and send and receive message between servers(by reader and writer). Also, the keywords 'await' and 'async with' deal with the concurrency issues for the server herds in an easy way. Thus, since the server herds mainly need operations based on TCP connections and faces concurrency situations with flooding algorithm of propagation, we can conclude that asyncio is very easy and suitable to implement server herds.

4.2 Pros and Cons

The main advantage of asyncio is that it deal with concurrency issue for python and can deal with multiple requests. Also, the built-in functions and keywords also make the implementation of inter-server communication very easy.

The disadvantages of asyncio involve that it cannot support multithreading compared with other solutions in other programming languages. Also, asyncio cannot deal with a specific situation in our serve herds that when WHATSAT message is processed before IAMAT request, it cannot be processed and would be regarded as an invalid message without the ability of waiting for latter message, due to the attribute of synchronous operations on reader and writer.

4.3 Importance of Python 3.9 or later

In python 3.9, there are many changes concerning asyncio has been made, but mostly are minor changes. For example, due to security concerns, the reuse_address parameter of asyncio.loop.create_datagram_endpoint() is no longer support. Also, a new coroutine shutdown_defaut_executor() is added that it schedules a shutdown for the default executor that waits on the ThreadPoolExecutor to finish closing. Further, asyncio.run has also been updated to use the new coroutine.

Since our implementation relies on TCP connection and doesn't involve the changes of asyncio in python 3.9, we can safely use older versions of python with minor modification.

5 Conclusion

Python is a good programming language for this application if it will not be extended to large-scale system, given that type-checking of python is convenient, and the downsides of memory management and multithreading aspects are not very important in the implementations. Also, Asyncio is very suitable for implementing the server herds, given its efficiency and convenience of setting TCP communications for servers.

6 References

Project documentation:
https://web.cs.ucla.edu/classes/winter21/cs131/hw/pr.html

Python 3.9 updates:
https://docs.python.org/3/whatsnew/3.9.html

Asyncio documentation:
https://docs.python.org/3/library/asyncio.html