Report for Python Project

Shannon Hoang
UCLA - CS131 Winter 2019 Eggert

Abstract

The aim of the project was to examine how suitable the "application server herd" architecture implemented using asyncio Python servers (specifically the stream API) was for a Wikimedia-style service. This Wikimedia-style service unlike Wikipedia, would have article updates much more often, require access via various protocols (besides HTTP), and allow clients to be more mobile. For communication among servers a simple flooding algorithm was used in which the servers had a bidirectional communication relationship. In this report I'll also be comparing the asyncio implementation to two alternatives: Node.js and Java.

1. Introduction

The type of architecture that will be the main focus of our investigation is called "application server herd," in which multiple servers will communicate and propogate information to each other via information flooding.

1.1. Implementation

In general, the asyncio asynchronous library was used for replacing the Wikimedia platform for our server application. This is because asyncio's event driven nature allows us to quickly update servers when new information is given. More specifically, to implement the applications server herd I used streams -- high-level async-await-ready primitives. Streams are a coroutine based API that the transmitting and receiving of data without having to use callbacks and low-level protocalls and transports.

Note that unlike in the get_event_loop, API, we never manage or touch the loop directly. Instead the event_loop is managed by asyncio.

When a client connects to our servers, they may send two different types of messages: IAMAT and WHATSAT. An IAMAT message is intended to be used to update the server on a client's location information. A WHATSAT message on the other hand is intended to be used to inquire about locations in the vicinity of a specific client. The last type of message that may be sent is an AT message. These are constructed with the intention of only being used between servers to propagate information about client location.

1.2. COMMANDS – IAMAT

Using the IAMAT command, clients can inform and update servers on their current location. IAMAT messages have the format: 'IAMAT' (client ID) (ISO 6709 latitude + longitude coordinates) (POSIX time for message send).
After processing the IAMAT command, the server will send back a message of the format: "AT" (serverID) (difference between message sent and received) (copy of client data). Please note the time difference may be negative if there is enough clock skew in the client's direction).

1.3. COMMANDS – WHATSAT

The WHATSAT command allows clients to query for information about places near other client's location.
The format of a WHATAT command is as follows:
'WHATSAT' (clientid) (radius in km) (information bound). The Google Places API was used to retrieve information on places near the specific client. As a result, note that the maximum radius must be 50km and the information bound must be at most 20 items as this is what the Google API allows.
After successfully receiving the WHATAT request from the client, the server should respond with an AT message of the same format as previously mentioned, and then the JSON-format message received from the Google API (with the results array limited to the given information bound).
If the server does not have information on the requested client, the server should print the corresponding error message (? (message sent by client) and throw an error as specified by the spec.

1.3. COMMANDS – AT

AT messages were used for server-server communication during the propagation of a client's location information.

The format is the same as mentioned in a server's response to an IAMAT message in the "Commands-IAMAT" section.

1.4. Error Handling

Bad commands were handled by printing a "?" a space, and then a copy of the bad command. Incorrect IAMAT, and WHATSAT requests were handled individually in the corresponding message function. AT messages did not require error handling as AT messages were only ever sent via our servers – which we would guarantee would format the AT messages correctly. Errors that can be made in IAMAT messages are: invalid clientname (not a string of non-whitespace characters), latitude or longitude out of range or not convertible to floats, and an incorrect number of parameters in the message itself. Errors that can be made in WHATSAT message are: again, an invalid client name, a radius greater than 50km, and an information bound greater than 20 items. Also for both types of messages, none of the fields must contain any whitespace.

2.0 Python Versus Java

2.1 Type Checking

Python utilizes dynamic type-checking while Java uses Static type checking. More specifically, Python type-checking is "duck type checking," meaning that if the functionality of an ambiguous object is the same as a defined type, the object is assumed to be of the defined type. However, Java because it uses dynamic type checking, an objects methods are determined by its type. The Java compiler determines the types of objects during runtime and restricts the objects behaviors accordingly.
As a result of Python's more relaxed method of type-checking, certain errors can pass by silently – which can yield hard to find bugs -- unlike in Java. However, the advantage of Python's style of type checking allows greater propensity for interfacing and encapsulation.

2.1 Memory Management

Memory management in Python is done via a heap that is internally managed by Python and reference counting. There are object-specific allocators that manage memory for specific types. For instances, integers are managed differently from strings.

In Python there is Garbage collection and this is done by reference counting. Every object has a specific reference count. This reference count can increase for a few different reasons: if you assign it to another variable, if you pass it as an argument, if you put it in a list. If the reference count falls to 0, the object's memory is freed so that that space can be used.

Meanwhile Java combines several memory management methods – which is mostly motivated by the desire to make garbage collection – the freeing of unused memory – as efficiently as possible. Java has stack and heap memory. The stack memory is used for holding pointers to objects on the heap and primitive types values. There is a certain scope limitation to the data on the stack as well – only information that is within the current scope is used. The heap memory is used for storing objects which can be accessed via references that are stored on the stack.

For garbage collection, Java uses a "mark and sweep" process. First, Java will look through all the variaables on the stack and mark the objects that still need to exist. After, all the unused objects have their memory freed.

However, the heap actually has multiple sections to further optimize garbage collection. The Eden(1) space is used for the newest objects. The garbage collector will run on the Eden space and mark alive objects. The garbage collector will run on the Eden space most often because newer objects are less likely to stay alive. The S0(2) space is the first place where objects that survive get moved. The next time the garbage collector runs on the Eden space it will move all surviving objects to the S1(3) space, also all objects in the S0(2) space are promoted to S1(3) space and so on. Essentially, the longer an object survives, the further it'll progress in the sections of the heap.

This method for garbage collection is logical as newer objects tend to be more likely to die, whereas older surviving objects tend to be used frequently and throughout the program.

Python's memory management has issues however. Because Python uses reference counting it runs into problems in terms of objects that are self-referencing or objects that reference each other. This can cause the python garbage collector to be stuck in a sort of infinite loop when garbage collecting and never delete the cyclically referencing objects, resulting in a memory leak.

Moreover, Java's garbage collection is faster. Python's use of reference counting forces the Python to have to update and check every object's reference count quite often, as reference counts must be updated every time an object is used in Python. However, because Java has lazy garbage collection, garbage collection is only done when it must be done and is done in bulk. Therefore in Java, more memory may be more occupied at a time then necessary and cause stalls in memory management as a result of mass garbage collection. Python with its more consistent garbage collection would only take the memory it needs (for the most part).

2.2 Multithreading

Multithreading is actually a major weakness of Python as Python cannot run in parallel completely on its own. This is because of the Global Interpreter Lock that restricts access to Python objects to a single thread. As a result, differing threads in Python cannot share memory, taking away many of the benefits of a multi-threaded program. As a result, Python is disadvantageous for our purposes in that we cannot implement our servers using multi-threading.

However, Python does support asynchronous programming which allows which is a means of parallel programming by allowing another process to work while another process stalls. Asynchronous programming as a result can speed the efficiency of programs without true parallel programming.

Java on the other hand, fully supports parallelism, multi-threading and the use of multiple cores. Java has an entire class devoted to Threads that come with the necessary methods for multithreading.

However, this difference in multi-threading may not actually matter that much as the potential race-conditions that exist in multi-threading may pose problems in our server herd implementation. As a result, we can see that even though Python is not that advantageous for multi-threading, this point is not that relevant to our examination.

3.0 Asyncio vs. Node.js

In terms of concurrency, Asyncio and Node.js are actually in the same in that they run singly-threaded. In addition the tools in asyncio and node.js can be seen as very similar as well. Node.js callbacks are almost identical asyncio-coroutines that can pause and resume execution. However an advantage asyncio has over Node.js is the coroutine – one program can be paused and allow another to take over if it is advantageous. This can allow for a more efficient program run.

However, in terms of performance and scalability, Node.js wins. Node.js was constructed on the extremely efficient and reliable V8 engine for Chrome and was made with the intent of creating websites that operate in real-time that have massive potential for scalability.

However in terms of accessibility, I would argue that the library for asyncio and Python is much more accessible compared to Node.js. Python as a language is more readable and intuitive than JavaScript and this can be an advantage when exporting programs.

4.0 Conclusion

Based on our examination and successful implementation we can see that asyncio was the best choice for our proxy server herd implementation. The concurrency provided by using asyncio streams allows us to take requests from clients and propogate information among the servers easily and efficiently.

While asyncio in Python was suitable for our prototype, our research shows that potentially Node.js may be a better alternative for a later iteration of our server herd as it possesses better scalability.

One challenge that should be noted was the prevention of infinite loops among clients when propogating location updates. This was solved by checking the timestamp on a message and comparing it to previously received data. If the timestamp was older than previously received data, the message was ignored.

References

Python Memory Management
https://docs.python.org/2/c-api/memory.html

Java Memory Management
https://dzone.com/articles/java-memory-management

Node.js
https://nodejs.org/en/docs/guides/

Node.js vs Python Asyncio
https://medium.com/@interfacer/intro-to-async-concurrency-in-python-and-node-js-69315b1e3e36

Learn Concurrency (Java)

https://docs.oracle.com/javase/tutorial/essential/concurrency/