CS131 Project Report

Weizhen Wang

Conclusion:

For IO bounded tasks—as in our project—asyncio is a great candidate. However, Python asyncio is not a Panacea: Python's dynamic type checking can lead to complexities when the project is written in multiple languages, and the Global Interpreter Lock of Python interpreter makes asyncio an unsuitable framework for CPU bounded tasks.

1. Suitability of Python:

In this server herd project, the bulk of the tasks are reading user's message, save client ID with its coordinate, and reuse that coordinate for http request to Google API. We also have inter-server communication implemented with flooding algorithm. Essentially, we have a lot of network I/O and file I/O while running the servers, and this fact make asyncio—asynchronous, single-threaded, and cooperative—the perfect framework candidate. With asyncio, we will no longer be blocked by I/O operations, and multiple I/O operations can be overlapped to save time. While we can also enjoy concurrent I/O using multithreading—the Java way—in asyncio, we can achieve the same, properly-behaved functionality without the OS overhead introduced by thread management. Also, we won't have blocked threads waiting for I/Os while consuming all the system resources. Therefore, from a performance point of view, Python asyncio is a more light-weighted solution to achieve the same functionality as Java multithreading without all the headaches of its error-prone nature

Moreover, asyncio is a suitable framework because of Python's simplicity. The syntax of Python is well-formed and natural to follow, and Python's dynamic typing makes creating variable straightforward without type concerns. Moreover, since Python use a garbage collector with reference count, object-oriented programming is easy—without the complications of deallocation. In this task, since we don't need to create complicated data structures that can incur cyclic reference, the reference count technique of garbage collecting employed by Python is efficient and simple.

To conclude this section, Python asyncio is a suitable framework because of its great I/O performance, linguistic simplicity and efficient memory management.

2. Pros of Python:

Following from the previous section, one pro of Python asyncio is its suitability for I/O bounded tasks. This is because Python asyncio employs the concept of "event loop"—like Node.js—to allow multiple non-blocking I/Os, while the asyncio enjoys the more concise syntax and flexible data types than Javascript, the language of Node.js. Since asyncio uses single-threaded event loop, it takes less system resources than multi-threaded approach, and it scales well while the number of I/O tasks increases.

Another pro of Python is its simplicity: without variable type declaration as in other statically typed language and memory deallocation in language without garbage collections, Python is easy for programmer without much exposure to object-oriented programming or asynchronous/concurrent programming.

botulisms tastes five quite schizophrenic tickets, and umpteen orifices grew up. Five extremely progressive televisions kisses one chrysanthemum. Tokyo ran away. Subways incinerated the obese wart hogs, because two quixotic trailers easily tastes the speedy tickets, and one elephant auctioned off sheep, although five obese elephants kisses the slightly

3. Cons of Python:

The merits of Python can turn into its drawbacks. The dynamic typing of Python can create unexpected errors when the result of computation in Python code is supplied into another part of the Program that can use a different language. Moreover, the reference count method of Python garbage collector can fail when more complicated data structure is introduced. Global Interpreter Lock of the Python interpreter also limited Python program to one native thread, making multi-threading in Python an obsolete method. Therefore, for computations that's CPU bound, Python asyncio will be intolerably slow. One can get around GIL with parallel programming—creating a lot of process and use inter-process communications such as piping—but it leads to more memory usage and complications. Therefore, with all its merits, Python asyncio is suitable only for a specific subset of computation tasks.