Async vs Sync for Non-Computation Heavy Applications

Drake UNIVERSITY

Jaehyeok Choi, Jacob Danner, Riley Rongere, Blythe Kelly, Gonzalo Valdenebro, Dr. Eric Manley

Abstract

In recent years, the number of web applications that implement the asynchronous way of handling requests are increasing. Existing tools such as Spring Boot or Django are starting to provide options to implement the logic in asynchronous ways.

With the benefit of wasting less time on I/O compared to traditional synchronous applications, much of the time, asynchronous applications are considered more performant than the synchronous applications

Considering these benefits that asynchronous programming provides, it would be logical for new projects or applications to be architected in an asynchronous manner. But what about the existing applications that are designed to be synchronous? Sometimes converting a project from synchronous to asynchronous can be time and resource consuming.

If it is possible for an application to have minimal tasks, where it takes minimal time to complete, therefore the blocking caused by synchronous programming is almost non-existent in real life, wouldn't it make more sense to keep the synchronous way of programming if performance difference is minimal?

Consider the BankAPI application. It is a simple application, with core functionality of retrieving data from the database and returning JSON value. It was originally built as an synchronous application, but thanks to aiomysql and FastAPI, we converted it to N asynchronous application.

Because this application fits the category of being non-computation heavy, and both synchronous and asynchronous code exists, the load test that simulates intensive web requests was timed to compare the performance differences.

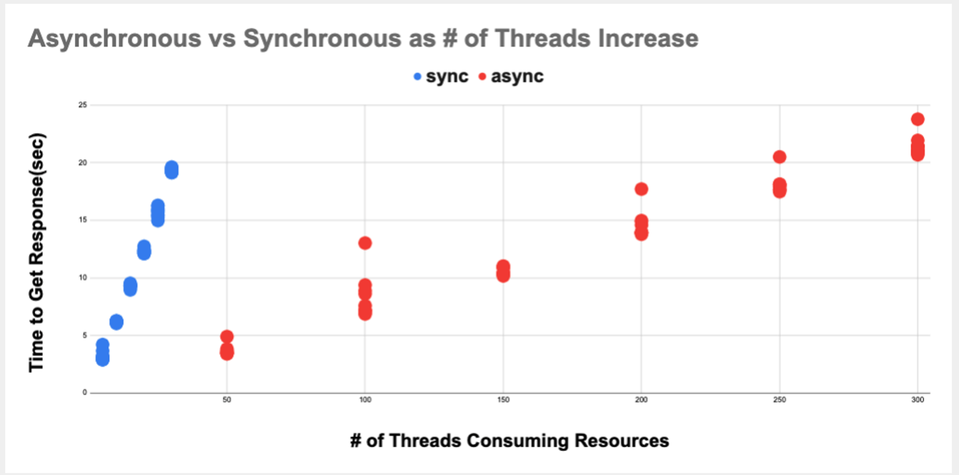
With the data gathered from this experiment, it can be determined that migrating existing synchronous applications to asynchronous applications may often be worth the time.

Future

- Move the test env from local to cloud
- Create a test to check if they operate differently when scaling
- Create a test to see how they behave differently in serverless env

Acknowledgement

- https://aiomysql.readthedocs.io/en/stable/
- https://fastapi.tiangolo.com/
- https://stackoverflow.com/questions/15085348/what-is-the-use-of-join-in-threading



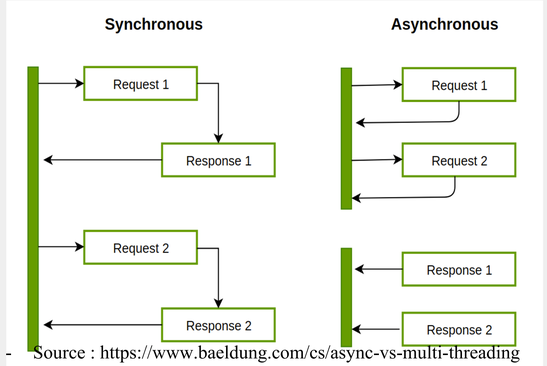
Test Setup

- The 3 endpoints were specified where each endpoint returns 1000 rows from database
- Using the threads, each thread called a method that send request to the 5 endpoints synchronously
- For each iteration, the number of threads would increase, and the time it took to finish the tasks were measured
- For asynchronous applications, the test was done with 100, 200, 300, 400, and 500 threads.
- For synchronous applications, the test was done with 10, 20, and 30 threads due to the server blowing up after.
- The server was started locally, in order to provide the same environment but the architecture.
- The spec of the computer that ran the server locally is following - 2020 Macbook Pro 13 inch, base model

```python
def ping():
    requests.get("http://127.0.0.1:8000/audit/data_dict")
    requests.get("http://127.0.0.1:8000/audit/CALL8786")
    requests.get("http://127.0.0.1:8000/audit/RCFA5311")
    # requests.get("http://127.0.0.1:8000/audit/RCFA8274")
    # requests.get("http://127.0.0.1:8000/audit/RCFAA224")

def main():
    data = []
    for num_of_threads in range(0,31,5):
        print(num_of_threads)
        for _ in range(10):
            threads = []
            for i in range(num_of_threads):
                thread = threading.Thread(target=ping)
                threads.append(thread)
            start = time.perf_counter()
            for thread in threads:
                thread.start()
            for thread in threads:
                thread.join()
            end = time.perf_counter()
            # print(f"{end - start:0.4f} seconds")
            data.append([num_of_threads,f"{end - start:0.4f}"])
```

Resouce used : https://docs.python.org/3/library/threading.html



- Source : https://www.baeldung.com/cs/async-vs-multi-threading

Variables

- Async vs Asynchronous architecture, please refer to the diagram above from Baeldung.
- Management of database pooling -> aiomysql vs mysql-connector
- # of database base pool available at the given time of test
- Request library provided
- Async request vs sync request
- How to implement the loading of the server: threading, futures, or other library
- Usage of the platform or the language that will be used to send request to the server

For this specific testing, the db pools were maxed at 32 connections, and generic python requests library and threads tools were used

Result

- Async architecture was about 10x more performant
- Sync application was more prone to internal errors with db connections or due to sheer amount of request
- Managing the number of pools in database connection made a performance difference, however, even when sync application maxed the pool size, and async application was not, async application still outperformed sync application

Conclusion

- Async performed 10x better than sync application in this specific test
- Neither the async nor the sync were deployed in any form of server
- Neither of the application used the tools to scale as the strain on server increased
- Spec of the database could have effected as sync application needed more connections to database at the given time, but the database spec was constant.
- If it truly needs to be async vs sync applications, the sync application for the clarification or even to better "represent" sync application should use WSGI instead of ASGI
- This test could be used to show that how async applications deals with loads better than the sync applications, however, it would be a stretch to use the numbers from the testing since, although minimalized, the other factors mentioned could have affected the overall performance.

Test Code with both application

API CODE