UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Concurrent benchmark system for web-frameworks on Python

Author:
Andriy PANKIV

Supervisor:
Oles DOBOSEVYCH

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2019

Declaration of Authorship

I, Andriy PANKIV, declare that this thesis titled, "Concurrent benchmark system for web-frameworks on Python" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Concurrent benchmark system for web-frameworks on Python

by Andriy PANKIV

Abstract

The frameworks have come a long way, and each new developer is faced not only with learning the language but also with the choice of the first framework for himself. The current choice is the result of incredible innovations in a relatively short period. As recently as 2004, Google released Gmail, which is considered the first product to be an all-in-browser; Today, such products are called one-page applications. Have you ever tried to create a front-end web interface using only HTML, CSS, and JavaScript? Well, nowadays it's not so hard. If the requirements are not too complicated, a small project can be completed relatively quickly. As for medium and large projects, to cope with the complexity of user requirements, you will need at least one framework.

Here you can find links to GitHub repositories for every part of my project:

- TestModuleExecutor
- FlaskTestModule
- DjangoTestModule
- PyramidTestModule

Acknowledgements

I would first like to thank my diploma supervisor Oles Dobosevych of the Faculty of Applied Sciences at Ukrainian Catholic University. The door to the Oles office was always open whenever I ran into a trouble spot or had a question about my research or writing.

Also, I am thankful to the Ukrainian Catholic University for the possibility to join very comfortable and maintainable atmosphere, which give me the appropriate impulse to improve myself every day.

Contents

List of Tables

List of Abbreviations

JSON	**J**ava **S**cript **O**bject **N**otation
HTTP	**H**yper **T**ext **T**ransfer **P**rotocol
PaaS	**P**latform **a**s **a** **S**ervice
Iaas	**I**nfrastructure **a**s **a** **S**ervice
CaaS	**C**ontainers **a**s **a** **S**ervice
SQL	**S**tructured **Q**uery **L**anguage
IT	**I**nformation **T**echnology
AWS	**A**mazon **W**eb **S**ervices
IoT	**I**nternet **o**f **T**hings
ORM	**O**bject **R**elational **M**apping
DB	**D**ata **B**ase
CRUD	**C**reate **R**ead **U**pdate **D**elete
CSS	**C**ascading **S**tyle **S**heets

Chapter 1

Introduction

In 2019, whatever your want to do, business or non-profit project, government or your personal idea, you need to communicate with others. You can yell about your thought on the street or attract customers with flyers or brochures. All this methods can be quite effective, but percentage of the people who would see your work, can be quite low comparing with world scales. So, how to attract those attention more effectively? How to yell all over the world? Answer is simple – build a website, where you can talk to your customers in any point on earth, in any language you need. But how to build effective and fast website? You can use popular website builder such as WordPress or Wix, but such solution may be slow and not as customizable as you want it to be. As a result, you need to develop your own web project to satisfy all your and your customer needs. In modern world, there is no need to reinvent the wheel, you or your employees need to use modern instruments to make work done. Those instruments is called frameworks. Nowadays, it is hard to choose from such wide variety of the ready-to-use solutions. With this diploma work I will try to make this decision easier for you, not among all possible web-frameworks, but, at least I will help to choose between popular Python projects, such as Flask, Django and Pyramid.

Chapter 2

Related works

2.1 TechEmpower first steps and last results

The problem of choosing a web framework to create its new web server has existed since the growing popularity of the World Wide Web and, accordingly, the widespread popularity of web browsers. With such rapid growth, programmers from all over the world have demonstrated their options for designing and maintaining sites of varying complexity and purpose. There are over 500 frames in more than 30 languages, each with its disadvantages, pros, and its unique capabilities. It can take a lot of time to make a choice, with such a wide assortment. For this, TechEmpower programmers have developed their first benchmark since March 2013 and published the results of the testing of 24 web-based frameworks. After that, for five years, they have released 16 more results of their achievements, the last of which took place in October 2018. For such an extended period, the team from TechEmpowers has developed 1774 tests for 431 frameworks in 26 languages.

2.2 How they build their tests

Even after the first tests, TechEmpower realized that among the many factors that need to be taken into account when choosing a web-based framework, it's easier to evaluate performance objectively. Application performance can be directly reflected in the final project cost, and for a young company hosting costs can be a pain point. Weak performance can also lead to premature scaling, degradation of user experience and related issues.

The first tests were aimed at providing a "starting point" for various frameworks. Under the "starting point" they mean the point from which the performance of any real program can only worsen. They want to know the upper bound of the framework on the unit of web equipment.

But they also want to implement some of the framework components, such as JSON serialization and database connetion. Although, each test is limited to measuring the number of requests per second that can be processed by one server, they perform a selection of components provided by modern frameworks, so they consider it a reasonable starting point.

For test-related data, they deliberately built tests to avoid any caching layer provided by the framework. They want this test to require repeated requests to external services (for example, MySQL or MongoDB) to execute the data mapping code implemented by the framework.

2.3 How their machines are configured

For all tests, they used two machines configured in the following roles:

- Application server. This machine is responsible for hosting web applications only.
- Client and database server. This machine is responsible for client side, it generate HTTP traffic to the application server using WeigHTTP, as well as for hosting the database server. In all their tests, the database server (MySQL or MongoDB) used very little processor time; processor resource was not exhausted. In the database tests, the network was used to provide the application server sets of results and to return HTTP responses in the opposite direction. However, even with the fastest frameworks, the use of networks was lower in database tests than in the usual JSON tests, so it is unlikely to be anxious.

After completing Round 15, they took the challenge to rewrite all ~ 460 test implementations from their home configuration to an enormous array of Docker containers. It took some time, but the Great Docking Certification gave significant advantages.

Due to the duplication, reproducibility, and consistency of their measurements have become much better than in previous rounds. Coupled with their continuous testing, they now see significantly less variation between each startup.

Indeed, what they do with this project is perfect for Docker. Or the Docker is ideal for this.

Chapter 3

What are Docker and Docker-Compose

3.1 Docker

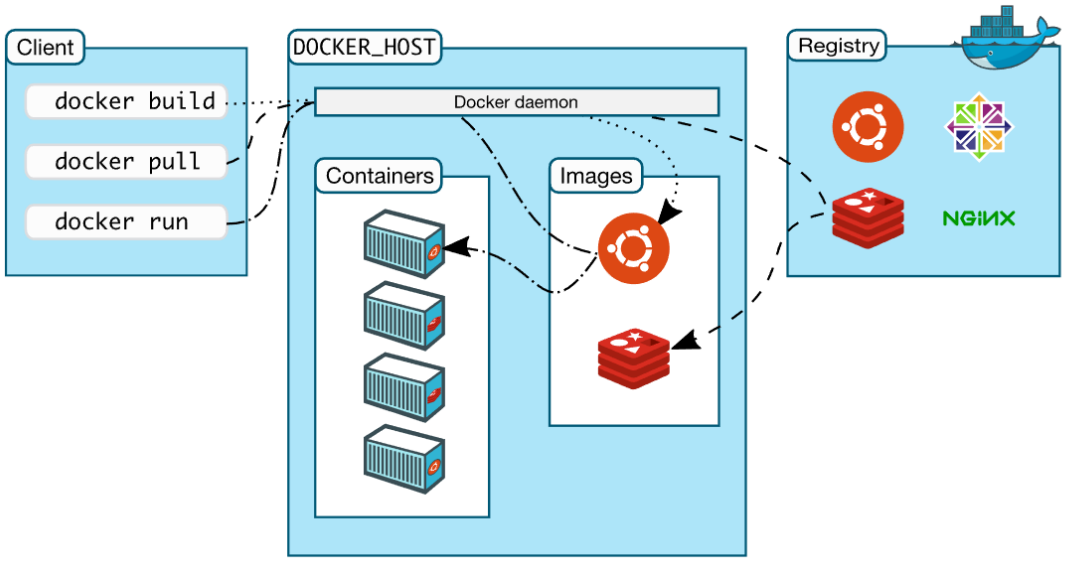
Along the years, developers have been looking for the best solutions of their app deployment. They discovered that PaaS models are too high-level for black box environment style. Furthermore, IaaS proposals with their appropriate container are not enough because they provide short-sightedness which is useful exclusively in this structure. In looking for the best solution, companies are offering a CaaS solution to provide agility for developers and cross-platform portability of their applications.

The Docker platform provides an integrated set of features for CaaS infrastructure models. Thanks to this solution, the IT team can provide and manage both infrastructures resources and content of the basic programs, and developers can create and deploy their programs in a self-service way.

In 2013, Docker entered the landscape with app containers to build, ship and run apps anywhere.("Modern App Architecture for the Enterprise") Docker was able to take any application and its dependencies and put them into a slight container. Similar to how shipping containers are today, software containers are simply a standard unit of software that looks the same on the outside regardless of what code and dependencies are included on the inside. ("Modern App Architecture for the Enterprise") This simplifies application transportation across environments without the need to modify anything. The Docker journey begins here. ("Modern App Architecture for the Enterprise")

3.1.1 How it works

Nowadays, IT-part of every company defines its success, whatever you are selling. Software is how you attract customers, reach new users, understand their behavior and their needs. To do this well, today's software is custom-made for every problem. Small parts of the code that are designed for a different task are called microservices. The main goal of such project structure is to gain system with independent parts, which can be easily modified or changed in the future and reduce the delay between writing your application and running it on production.



In the diagram above, you can see a basic example of how Docker works. The developer chose a list of services he needs, daemon pull them from the registry and put them in containers. In that way, there is no need to install libraries or follow long installation process of services because all this work was done for you, so you work with ready-to-use black-boxes.

3.2 Docker-Compose

Now you know what Docker is, but in real life projects, it can be hard and tedious to define and run multi-container Docker application and establish correct communication among them. So, here begins the journey of a Docker-Compose. For better understanding take a look at Docker-Compose configuration file below.

```
version: '3'
services:
  web:
    build: .
    command: python /code/app/app.py
    ports:
     - "8082:8082"
    volumes:
      - .:/code
    depends_on:
      - db
    hostname: myappserver
  db:
    hostname: mysqlserver
    image: mysql
    environment:
      MYSQL_ROOT_PASSWORD: p@ssw0rd123
      MYSQL_DATABASE: wordpress
```

This file will create two Docker containers 'web' and 'db'.
'web':

- will start with command 'python code/app/app.py'
- map port 8082 to 8082 for World Wide Web
- map folder on your machine with docker container folder

- make it depend on ‘db’ docker container, so if MySQL server will not start, it aborts execution of web server
- set hostname to Docker container so that you can connect to it by name, not just port on localhost

’db’:

- set hostname
- pull image from the registry(watch image above)
- set database password and name

If there where no such suitable instrument, you should set all those parameters every time you need to rebuild your project Dockers. With Docker-Compose you need to write it only once.

Chapter 4

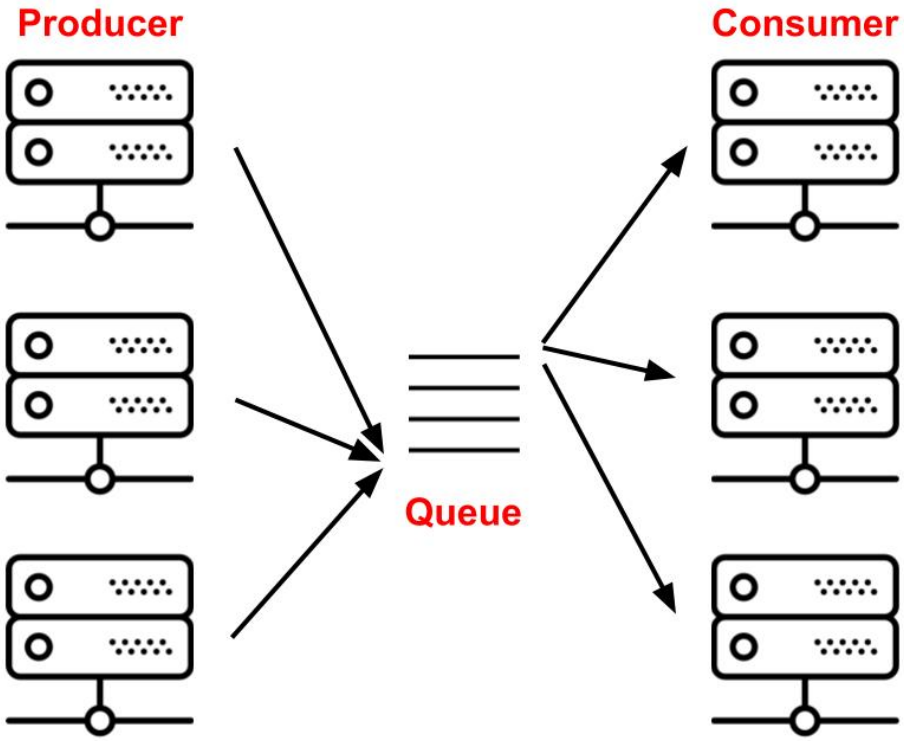
What are Celery and Redis

4.1 Celery

We have an excellent system to build, deploy and improve stability of our test. Also, we have a comfortable instrument to group and manage them together, but how to simulate real-world user activity? If I will write a script with "for loop" which will make an HTTP request to the server it may work, but is it close enough to real stress-test of every test module? I don't think so, because, in real life, the user will never wait for other users to finish their staff, they want everything now, so I need something which can work asynchronous and independent at the same moment.

Here the work of Celery begins. It is a reliable, flexible and straightforward system to process a set of tasks from message broker(will be defined in next sections), these tasks are executed concurrently with one or more workers using multiprocessing.

The problem of running concurrent task can be easily shown by "Producer and Consumer" model. Producers add jobs to the queue. Consumers than check for the new tasks, pick the first one and start processing it.



In the Celery context, Producers often the web nodes, Query as message broker and Consumers as workers. The idea of a broker is a simple queue, but how to implement a modern queue with IT? One of the straightforward solutions would be a text file. On that way, we can hold an order of tasks, but the problem is that text files were not developed to work through the network are maintain concurrent access. So text files are not reasonable solutions for our problem. How about SQL databases? They are capable of running in a network and can support concurrent access, but now we are stumbled at speed limits. We may use NoSQL databases, they are fast, but they can not prove the reliability we need. So, we need something fast, reliable and concurrence tool such as Redis.

4.2 Redis

Redis (stands for Remote Dictionary Server) is a fast repository of key-value data in open source memory for use as a database, cache, message broker or queue. ("AWS Amazon") Redis provides a fraction of a millisecond response time and allows real-time applications to perform millions of requests per second. Such applications are in demand in the field of games, advertising technologies, financial services, health care, and IoT. Redis is widely used for caching, session management, game development, creating leader boards, real-time analytics, working with geospatial data, taxi service support, chat and messaging services, multimedia streaming and applications with sending messages using the Publisher and Subscriber model.

All data in Redis is stored in memory, not on disks or solid-state drives, as in other databases. Redis, like other in-memory data storages, does not need disk access, this excludes search delays. Due to this, the number of operations performed increases many times and the response time is reduced. The result is extremely high performance. On average, read or write operations take less than a millisecond; the speed of operation reaches millions of operations per second. Redis features include support for a variety of data structures, high availability, working with geospatial data, creating Lua scripts, conducting transactions, permanently storing data on disk, and supporting clusters. All this makes it easy to create real-time applications for the entire Internet.

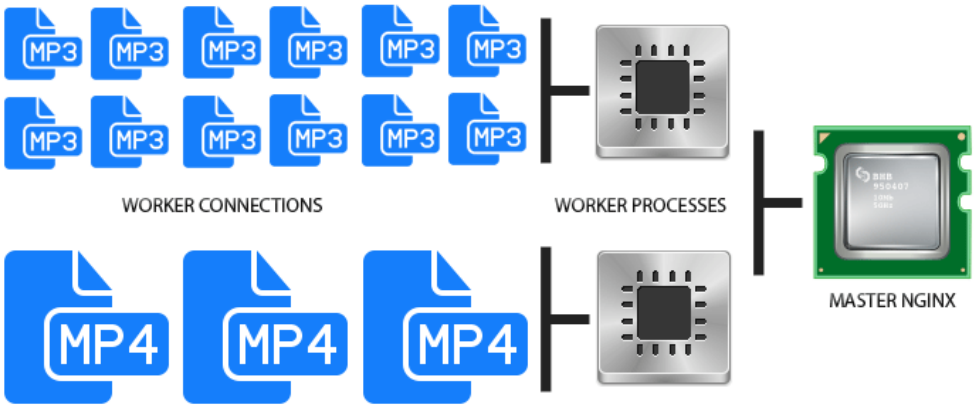
Unlike simplified storage based on key-value pairs that support a limited set of data structures, Redis supports a vast variety of data structures to meet the needs of diverse applications. Redis data types include:

- strings - text or binary data up to 512 MB in size
- arrays - a collection of rows, ordered in order of addition
- sets - an unordered collection of lines with the ability to intersect, combine and compare with other types of sets
- sorted sets - sets ordered by value
- hash tables - data structures for storing lists of fields and values
- bit arrays - a data type that allows you to perform bit-level operations
- HyperLogLog structures - probabilistic data structures used to estimate the number of unique elements in a data set

Chapter 5

Nginx

Before digging into how Nginx works, let's see for what web server stands for. When someone makes a request to open a web page, the browser contacts the server of that website. The server then looking for the requested data for the page and returns it to the web client. This is just the simplest example of a web scenario.



The above example is also considered as one stream. Web nodes create a separate thread for each request, but not Nginx. It works with concurrency, event-driven architecture. This means that a single workflow controls similar threads, and each workflow contains smaller blocks, called workloads. This whole block is responsible for processing the request flows. Work connections deliver requests to the work process, which also sends it to the main process. Finally, the main process provides the result of these queries.

This may look like a simple architecture, but Nginx can handle up to 1024 similar requests. Thanks to this, Nginx can efficiently process thousands of requests. This is also the reason Nginx is excellent for downloaded websites such as online stores, search engines, and cloud storage.

So, what is Nginx? Nginx is a web server that also acts as a proxy email server, reverse proxy server, and load balancer. The software structure is asynchronous and event-driven, which allows you to process many requests at the same time. Nginx also scales well, which means that its service grows with customer traffic.

Chapter 6

Project Structure

6.1 Microservice architecture

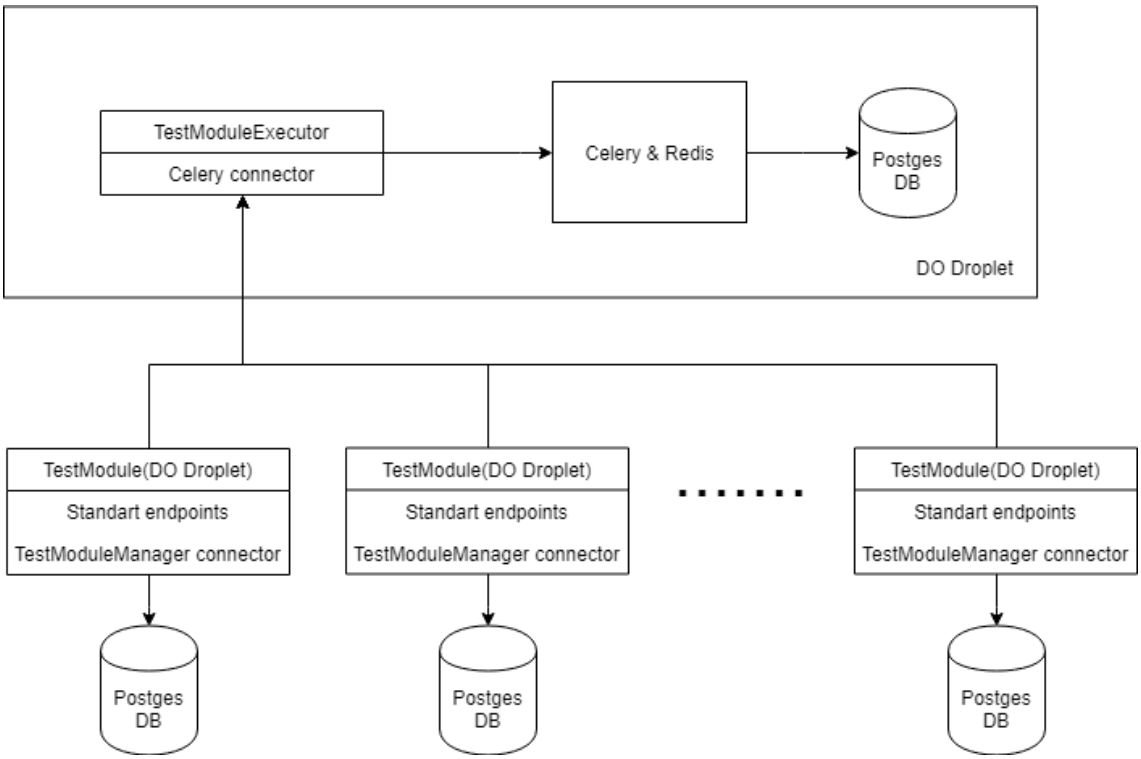
For this project, I choose microservice architecture, because of its list of benefits, such as:

- Writing and maintaining small services is always easier than large ones. The smaller the code, the easier it is to fit it in the head;
- Modularity. If you want to make an update - roll out the service. If you see by the metrics that something is wrong - roll it back. At some point, you noticed that no one else goes to the service - turn it off. Instead of refactoring, it is easier to throw away the service and write it from scratch.
- Tracking the dependencies between the services is easy. If all services depend on each other, probably something in the architecture of your system is wrong. If the number of connections is not much higher than the number of nodes, then you are in the right way. It is also easier to find a place and reason for critical failure.
- Reliability. If one module raises an error, rest parts of the system, which don't depend on it, will perform successfully, but with a monolithic architecture, if one part is dead – the whole system is dead.

6.2 Project parts

My project contains from three parts:

- Stack of TestModules. Fully independent instances of different web-frameworks. Due to the Docker, all of them are executed in the same environment, so all benchmarks are honest. Moreover, every TestModule is configured to work with Nginx, so it can handle asynchronous HTTP requests.
- Celery & Redis. Combination of two reliable systems to simulate client activity. Celery is responsible for managing and executing tasks, while Redis is responsible for stacking messages for Celery workers and storing its results.
- TestModuleExecutor – heart of the project, it is responsible for registering new web-framework nodes, managing and collecting results from Redis.



6.3 Workflow

Every TestModule is executed successively. So, project flow is next:

- TestModuleExecutor fill Redis message stack.
- Celery workers asynchronously take messages from Redis stack.
- Web-module receive a HTTP request, process it and return result to the source.
- Celery worker store result to the Redis and take another message from stack, if there are some.
- TestModuleExecutor catch time of every bunch of tests and return results as simple JSON response to your browser.

Chapter 7

Results

My benchmarks was based on three main tasks of back-end part of almost every modern web-project, no matter if it is simple blog, mobile application or giant projects like “YouTube”, “Instagram” or “Facebook”. This tasks are:

- Database connections
- JSON serialization
- JSON deserialization

7.1 CRUD Testflow

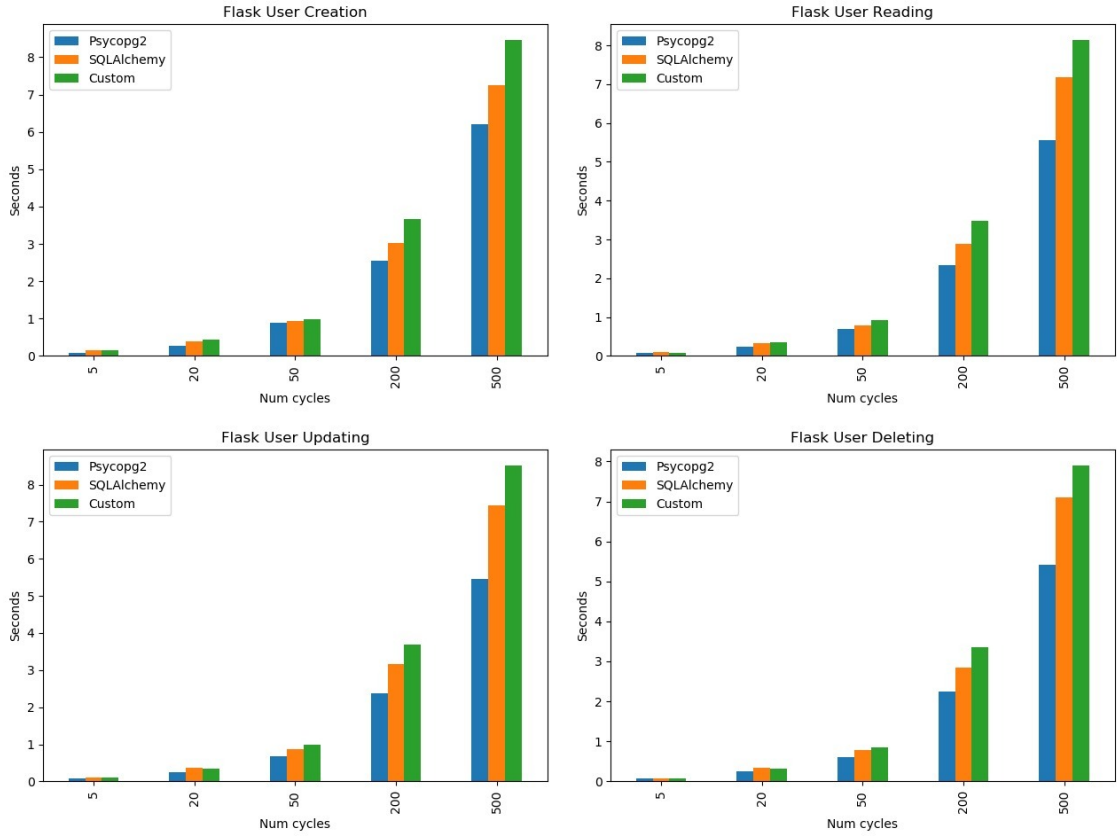
What is **CRUD**? You should know from abbreviation section, that **CRUD** is **C**reate **R**ead **U**pdate **D**elete, but what does that mean? The term **CRUD** is also used in conjunction with the user interface. For example, in an address book program, a base object in contact with contact data. As a minimum, the application must provide the next feature:

- **C**reating Records
- **R**eading records
- **U**pdating records
- **D**eleting existing records

Without these basic operations, the program can not be considered suitable for use. As I mentioned above, every of this group of tests was executed one after another and TestModuleExecutor catch time of every group. Database connection can be configured in different ways, but I choose three most popular ways – pure DB connectors, SQLALchemy ORM and custom DB connection of every framework. So, it was tested in next way:

- Each database of every TestModule have simple table “User” with name and surname fields
- TestModuleExecutor fill Redis stack with proper messages to create, read, update and delete users, count how much time every framework needs to do this and return results as HTTP response.
- Benchmark was repeated with all three DB connections and with 5, 20, 50, 100, 200 and 500 users.

7.1.1 Flask CRUD results

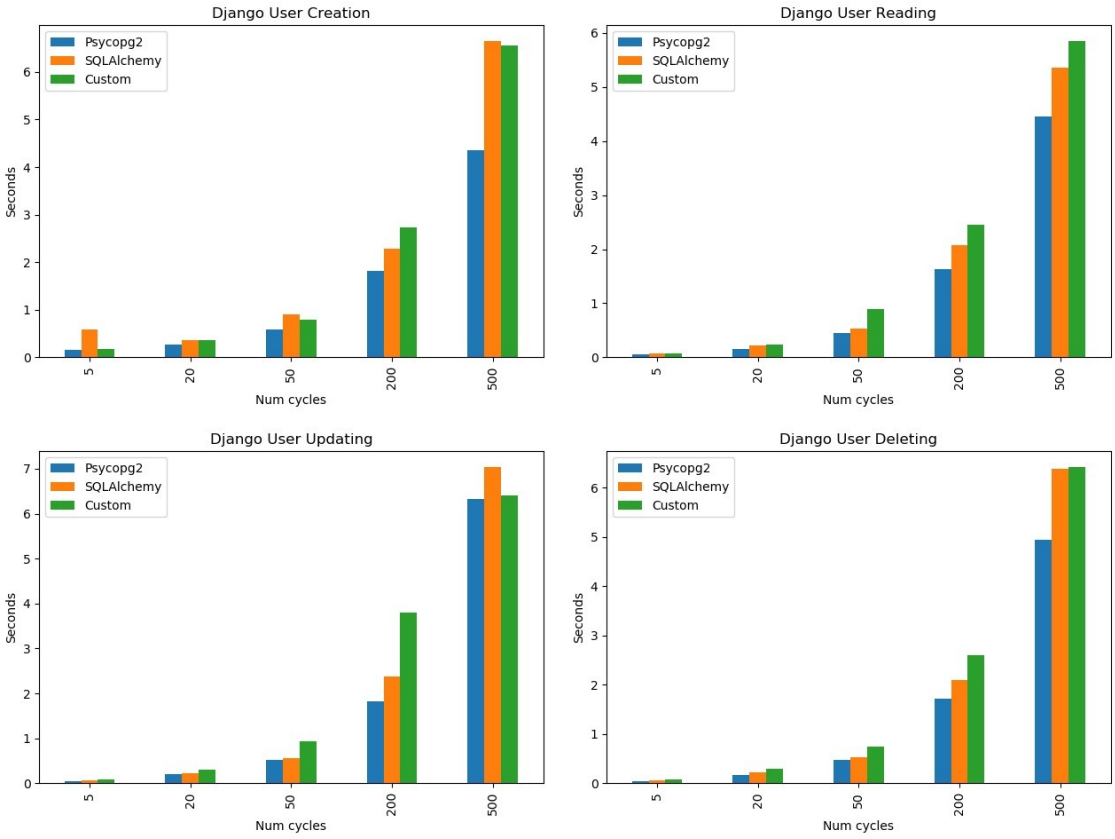


Framework		Django				
№		5	20	50	200	500
User Creation	Psycopg2	0.0797	0.2783	0.8939	2.5379	6.2167
	SQLAlchemy	0.1521	0.3913	0.9321	3.0314	7.2550
	Custom	0.1454	0.4367	0.9831	3.6727	8.4593
User Reading	Psycopg2	0.0678	0.2414	0.6981	2.3283	5.5679
	SQLAlchemy	0.0947	0.3258	0.7854	2.8802	7.1743
	Custom	0.0803	0.3522	0.9319	3.4739	8.1361
User Updating	Psycopg2	0.0781	0.2422	0.6750	2.3633	5.4689
	SQLAlchemy	0.0928	0.3611	0.8733	3.1625	7.4355
	Custom	0.1034	0.3373	0.9915	3.6847	8.5134
User Deleting	Psycopg2	0.0756	0.2426	0.6081	2.2454	5.4196
	SQLAlchemy	0.0831	0.3325	0.7903	2.8355	7.0950
	Custom	0.0805	0.3127	0.8580	3.3514	7.8948

TABLE 7.1: Flask CRUD results

As it was expected, pure connectors is the fastest way to work with DB. Second in speed is extremely popular among Python Developers library - SQLALchemy and last is custom DB connection. It was very predictable that custom Flask DB connection is the slowest one, because it is a framework over SQLALchemy ORM. So, it would be extremely weird if it was better than its “father”.

7.1.2 Django CRUD results

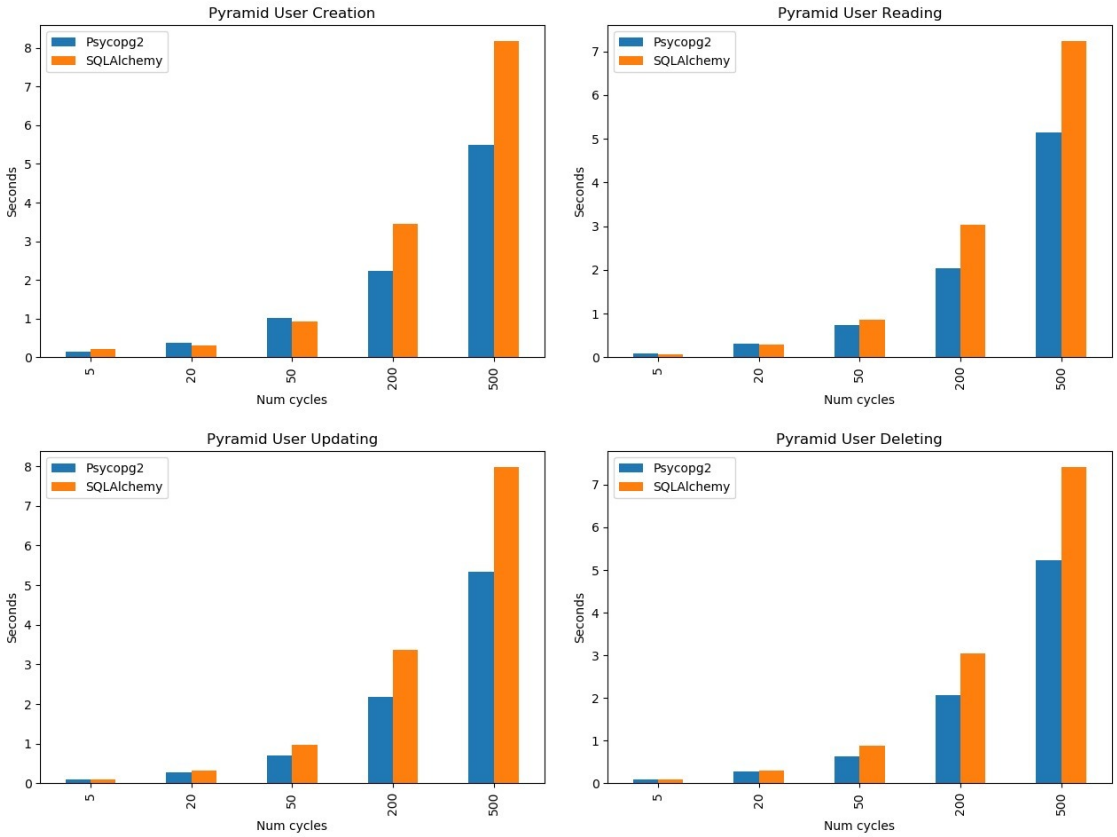


Framework		Django				
№		5	20	50	200	500
User Creation	Psycopg2	0.5764	0.3625	0.9059	2.2865	6.6484
	SQLAlchemy	0.1595	0.2666	0.5820	1.8108	4.3645
	Custom	0.1759	0.3620	0.7822	2.7360	6.5580
User Reading	Psycopg2	0.0720	0.2254	0.5238	2.0757	5.3576
	SQLAlchemy	0.0561	0.1520	0.4443	1.6275	4.4521
	Custom	0.0663	0.2410	0.8893	2.4468	5.8491
User Updating	Psycopg2	0.0653	0.2176	0.5544	2.3731	7.0341
	SQLAlchemy	0.0560	0.2112	0.5119	1.8259	6.3338
	Custom	0.0785	0.2936	0.9441	3.7895	6.4006
User Deleting	Psycopg2	0.0627	0.2285	0.5233	2.0982	6.3888
	SQLAlchemy	0.0462	0.1747	0.4697	1.7086	4.9413
	Custom	0.0740	0.2991	0.7433	2.6081	6.4164

TABLE 7.2: Django CRUD results

Among DB connector, Django results was quite similar to the Flask results, only difference is with its custom connectors, because they are quite good written as whole framework in general. To my mind, this is one of the main reasons why "Instagram" developers choose this web-framework as it main back-end technology.

7.1.3 Pyramid CRUD results



Framework		Pyramid				
№		5	20	50	200	500
User Creation	Psycopg2	0.1395	0.3642	1.0159	2.2403	5.4904
	SQLAlchemy	0.2033	0.3042	0.9170	3.4414	8.1742
User Reading	Psycopg2	0.0821	0.3018	0.7299	2.0425	5.1531
	SQLAlchemy	0.0777	0.2826	0.8620	3.0315	7.2312
User Updating	Psycopg2	0.0940	0.2870	0.6932	2.1818	5.3345
	SQLAlchemy	0.0976	0.3301	0.9613	3.3645	7.9776
User Deleting	Psycopg2	0.0841	0.2822	0.6395	2.0705	5.2240
	SQLAlchemy	0.0806	0.2937	0.8720	3.0396	7.4118

TABLE 7.3: Pyramid CRUD results

As you may notice there is no bar for custom connection on the Pyramid chart. Why I exclude them? Answer is pretty simple – there is no such functionality in Pyramid, if you will visit official Pyramid documentation about DB connection, you will find explanation how to add SQLAlchemy to your project.

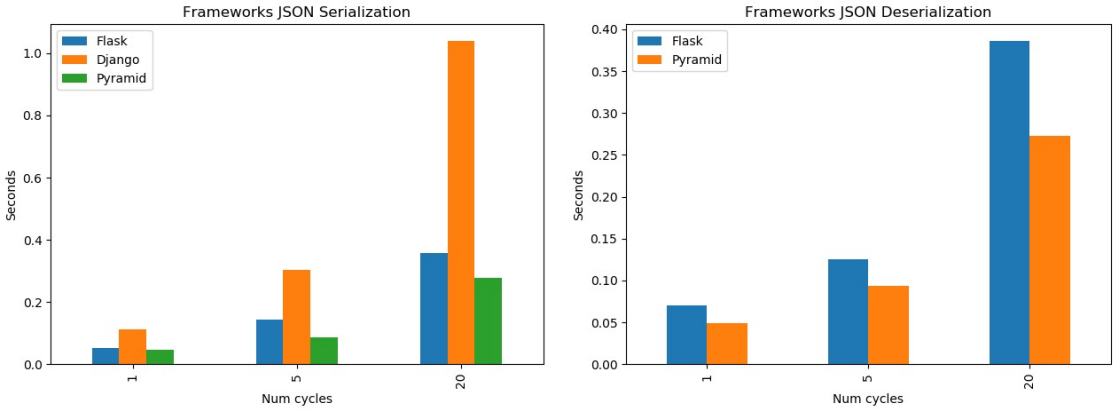
7.2 JSON Testflow

You should know what is **JSON** from abbreviation section, that **JSON** is **J**ava **S**cript **O**bject, but what does that mean? This is a text format for data exchange between computers. JSON is based on text, can be read by a person. The format allows you to describe objects and other data structures. This format is mainly used for the transmission of structured information over a network (due to the processes called

serialization and deserialization).
JSON serialization and deserialization was tested in quite similar way as DB connection. The same algorithm, but instead of user information it was working with JSON as you might have already guessed. It was processed in this way:

- Every framework and TestModuleExecutor has it own copy of JSON files
- TestModuleExecutor fill Redis stack with proper messages to serialize or deserialize JSON
- Benchmark was repeated with every JSON files, there are wide variety of big and nested JSONs of every type you or your team will be using in your project.

7.2.1 Frameworks JSON results



Framework	Flask			Django			Pyramid		
№	1	5	20	1	5	20	1	5	20
JSON Serialization	0,051	0,142	0,356	0,111	0,304	1,040	0,105	0,142	0,356
JSON Deserialization	0,069	0,125	0,386	-	-	-	0,069	0,125	0,386

TABLE 7.4: JSON Serialization results

You may asking, why there is no metrics for JSON deserialization in Django. It was quite surprising for me, that so popular and huge framework dose not support such basic functionality, but there is build-in python library which will do that for you, so this will never make struggle while working with Django frameworks.

Chapter 8

Conclusion

You may decide that Django is the best choice for your project, as it is the fastest among others and you can use built-in python JSON deserializer if there is a real need, but it depends on the size and possible popularity of your web-module. If you are as ambitious as Mark Zuckerberg and you want to create web-service with millions of users per second – your choice should stop on the Django. If you're going to use microservice architecture where every module is responsible for small tasks or a simple website, which may be not as popular as a child of Mark, you should decide between two popular lightweight frameworks - Flask and Pyramid. To my mind, you should choose Flask as it is more popular than the Pyramid so you can find more solutions to your problems over the internet. Moreover, much more popular web-based libraries are ready to combine with Flask.

Bibliography

Amazon.com, Inc. “AWS Amazon”. In: p. 1.
Docker, Inc. “Modern App Architecture for the Enterprise”. In: p. 3.