

Topics in Software Engineering: Final project report

Team Name: Appify

Project Title: Web Application development frameworks

Group members:

Vibhor Gaur (UNI: vg2376)

Prateek Gupta (UNI: pg2512)

A. Introduction/Overview

In this project, we have developed a web app using different python frameworks. The aim is to compare these web frameworks by testing the app developed using different dimensions that would be common to most web applications. The application has been described in detail in section B. Web application development is a very common phenomenon these days ranging from people developing apps for fun to the burst of start-ups giving web app based solutions for daily life problems to major companies providing/shifting to the app based market solutions. Most of the mobile apps today are also hybrid apps which basically are web apps wrapped in wrappers to provide an environment to be able to execute in corresponding to the native OS mobile platform. Hence, through this project we would help a web developer assess the scope of using one out of the two major frameworks used for web app development. The tools chosen for comparison were selected based on popularity among most popular python web frameworks for development. We will be comparing the following two frameworks:

- 1) Flask
- 2) Django

We tried to choose dimensions for comparison such that they generalize well to most of the current web applications and those that will be developed in future. The major dimensions on which we will try to compare them are as follows:

- 1) Network access (load testing)
- 2) Caching
- 3) Performance for interaction with UI (render_template vs HTTP_response)
- 4) File handling (Upload file)
- 5) Security (sessions/cookies/user authentication, Cross scripting/HTML injection)
- 6) Architectural differences/ Other APIs
- 7) Developer experience
- 8) Major Strength/limitations

B. App developed for comparison (Pollarion)

We developed a web app that could generalize to future web apps that developers would develop in the sense that it involves front end (loading HTML templates), back end (data base access), sessions/cookies, caching, file handling (upload/download)), redirection to new page. Comparing flask and django on such an app would give future developers a good insight as to which of these two popular frameworks should they use or when should they shift from one platform to another (like when the number of users increase) depending on the specifics of their app most of which are covered in the app developed by us in this project [4], [5], [6], [7], [8], [11], [12].

An app for generating polls where users ask questions and answer questions posted by other users, users can vote for answers and/or write their own answers).Users can comment or reply on any of the posts. They can also like/unlike posts and comments and vote/unvote an answer which they previously voted. Specifically, the features of the app are:

1. Sign up
2. login
3. Upload profile picture
4. Create new post
5. Like post
6. Unlike post
7. Add answer
8. Vote answer
9. Unvote answer
10. Add comment
11. Like comment
12. Unlike comment
13. Reply to comment (all features available with comment are available with reply)
14. Ban user by admin

The user home page will contain all questions posted by different users, to view this question and/or do activity on these posts he can just click the post to open it.

To ensure consistency among the two apps so that measurement across the two apps are comparable primarily along the lines of the framework

For both the apps, we used the same UI/front end code (HTML templates,), features/functionalities for user interaction, backend (database, database schema, DB queries, code for application logic, code for communication with the Database). The common environment used in both the apps is:

1. Database: Postgresql
2. Execution Environment: 64 bit i3 processor, 2.3GHz, 4 GB RAM
3. HTML: Jinja and twitter templates

C. Experiments and Results

1. Network access (Load testing)

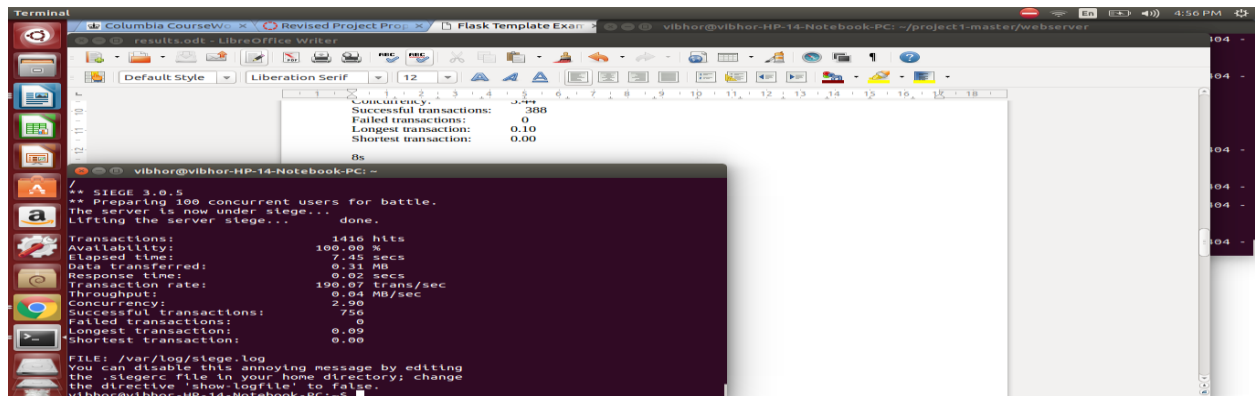
We used the siege software to perform load tests on our web application using the following command:

```
siege -c <num_users> <time_to_load <num_users>> <http end point where app is running>
```

The results of the experiments are outputted in a format as follows:

```
Transactions:      2742 hits
Availability:      96.52 %
Elapsed time:      19.42 secs
Data transferred:  0.67 MB
Response time:     2.48 secs
Transaction rate:  141.19 trans/sec
Throughput:        0.03 MB/sec
Concurrency:       349.91
Successful transactions: 1806
Failed transactions: 99
Longest transaction: 16.77
Shortest transaction: 0.00
```

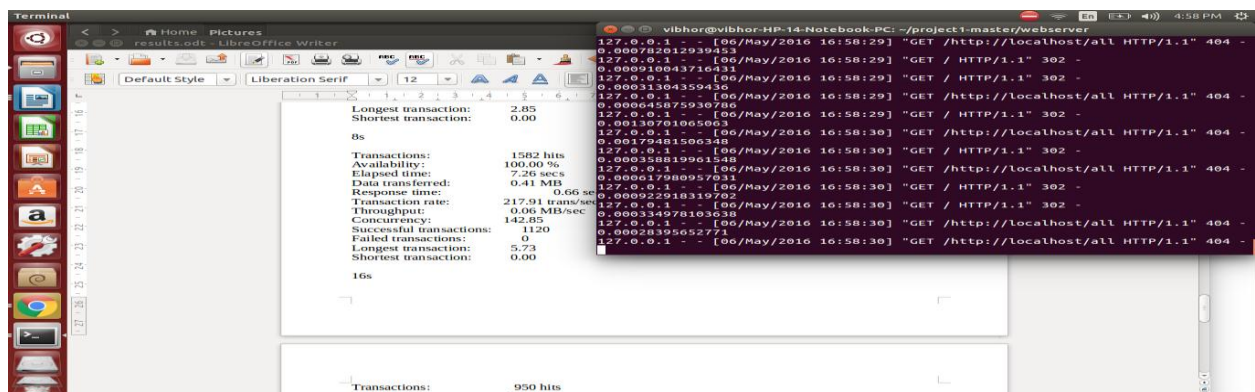
Sample screenshot for load testing



```
Terminal
Columbia CourseW... Revised Project Pro... Flask Template Exer... vibhor@vibhor-HP-14-Notebook-PC: ~/project1-master/webserver
Default Style Liberation Serif 12
Concurren...
Successful transactions: 388
Failed transactions: 0
Longest transaction: 0.10
Shortest transaction: 0.00
8s
vibhor@vibhor-HP-14-Notebook-PC: ~
** SIEGE 3.0.5
** Preparing 100 concurrent users for battle.
The server is now under siege...
Lifting the server siege... done.

Transactions:      1416 hits
Availability:      100.00 %
Elapsed time:      7.45 secs
Data transferred:  0.31 MB
Response time:     0.02 secs
Transaction rate:  190.07 trans/sec
Throughput:        0.04 MB/sec
Concurrency:       2.90
Successful transactions: 756
Failed transactions: 0
Longest transaction: 0.09
Shortest transaction: 0.00

FILE: /var/log/siege.log
You can disable this annoying message by editing
the .siegerc file in your home directory; change
the directive 'show logfile' to false.
vibhor@vibhor-HP-14-Notebook-PC: ~
```



```
Terminal
Home Pictures
results.txt - LibreOffice Writer
Default Style Liberation Serif 12
vibhor@vibhor-HP-14-Notebook-PC: ~/project1-master/webserver
127.0.0.1 - - [06/May/2016 16:58:29] "GET /http://localhost/all HTTP/1.1" 404 -
0.000782012939453
127.0.0.1 - - [06/May/2016 16:58:29] "GET / HTTP/1.1" 302 -
0.000910043716431
127.0.0.1 - - [06/May/2016 16:58:29] "GET / HTTP/1.1" 302 -
0.000313043594356
127.0.0.1 - - [06/May/2016 16:58:29] "GET /http://localhost/all HTTP/1.1" 404 -
0.00045675930786
127.0.0.1 - - [06/May/2016 16:58:29] "GET / HTTP/1.1" 302 -
0.00138701005863
127.0.0.1 - - [06/May/2016 16:58:30] "GET /http://localhost/all HTTP/1.1" 404 -
0.00179481506348
127.0.0.1 - - [06/May/2016 16:58:30] "GET / HTTP/1.1" 302 -
0.000358819961548
127.0.0.1 - - [06/May/2016 16:58:30] "GET /http://localhost/all HTTP/1.1" 404 -
0.000617980957031
127.0.0.1 - - [06/May/2016 16:58:30] "GET / HTTP/1.1" 302 -
0.000922910319702
127.0.0.1 - - [06/May/2016 16:58:30] "GET / HTTP/1.1" 302 -
0.000334978103038
127.0.0.1 - - [06/May/2016 16:58:30] "GET /http://localhost/all HTTP/1.1" 404 -
0.00028394082771
127.0.0.1 - - [06/May/2016 16:58:30] "GET /http://localhost/all HTTP/1.1" 404 -

Longest transaction: 2.85
Shortest transaction: 0.00
8s
Transactions:      1582 hits
Availability:      100.00 %
Elapsed time:      7.26 secs
Data transferred:  0.41 MB
Response time:     0.66 secs
Transaction rate:  217.91 trans/sec
Throughput:        0.06 MB/sec
Concurrency:       142.85
Successful transactions: 1120
Failed transactions: 0
Longest transaction: 5.73
Shortest transaction: 0.00
16s
Transactions:      950 hits
Availability:      100.00 %
Elapsed time:      7.26 secs
Data transferred:  0.41 MB
Response time:     0.66 secs
Transaction rate:  130.85 trans/sec
Throughput:        0.06 MB/sec
Concurrency:       142.85
Successful transactions: 1120
Failed transactions: 0
Longest transaction: 5.73
Shortest transaction: 0.00
```

The following experiments were setup for both the apps:

1. 3 users
2. 10 users
3. 100 users
4. 1000 users

For each experiment above the time to load users was varied as

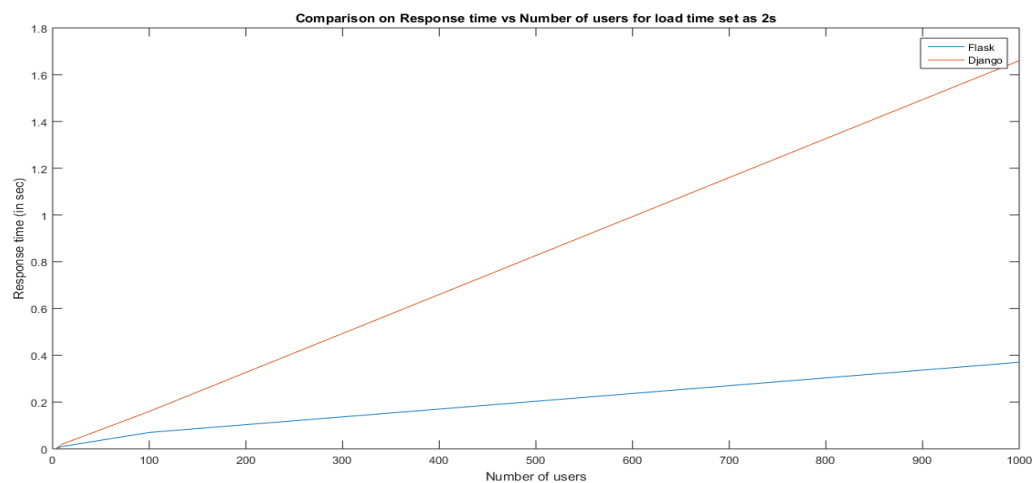
1. 1s
2. 2s
3. 4s
4. 8s
5. 16s

We evaluate the performance of load testing on the following criterion setting maximum load time to load the users as 2s.

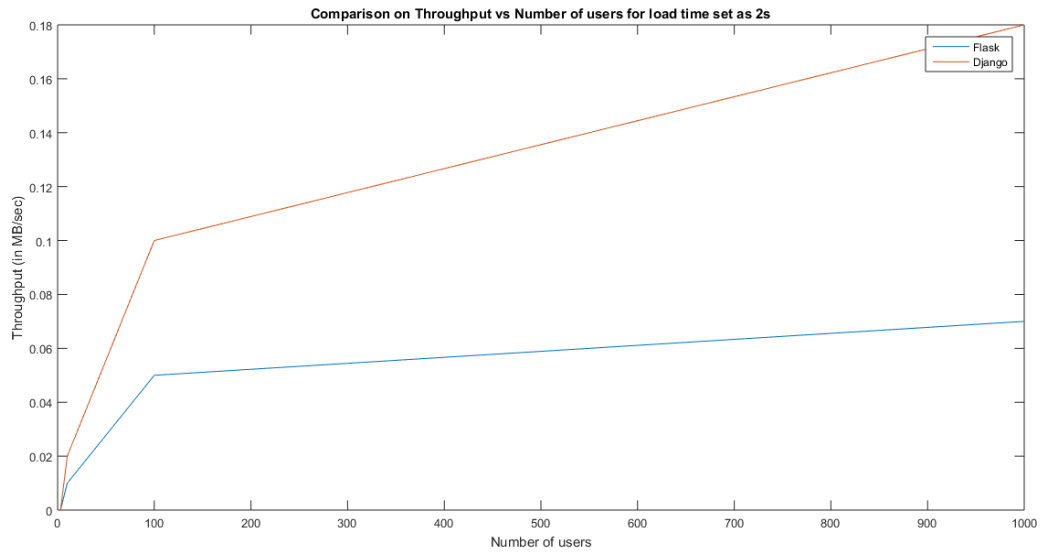
- a. Average Response time
- b. Throughput
- c. Concurrency
- d. Longest transaction time (reflection of worst case waiting time for user)

Results are as follows:

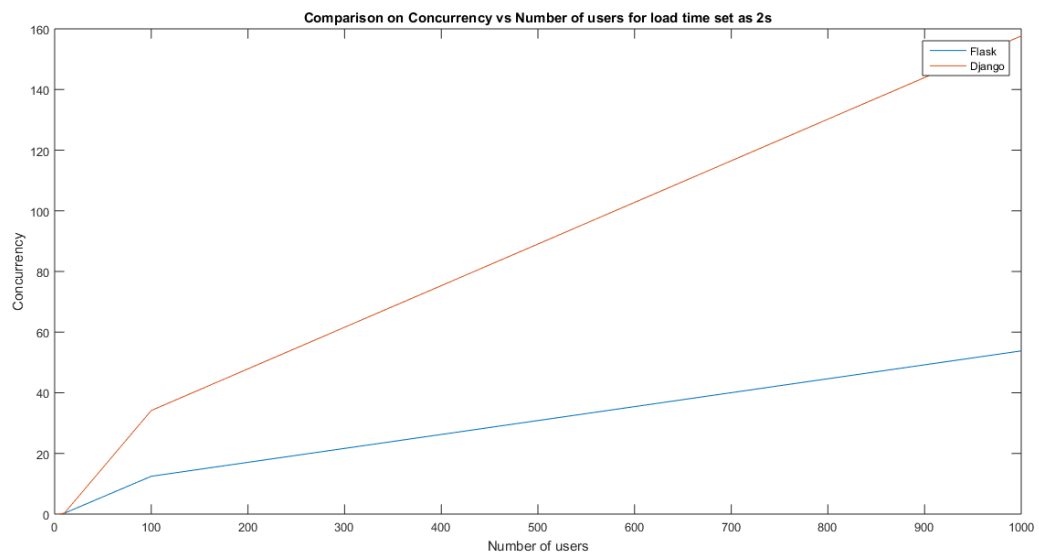
a. Average Response time



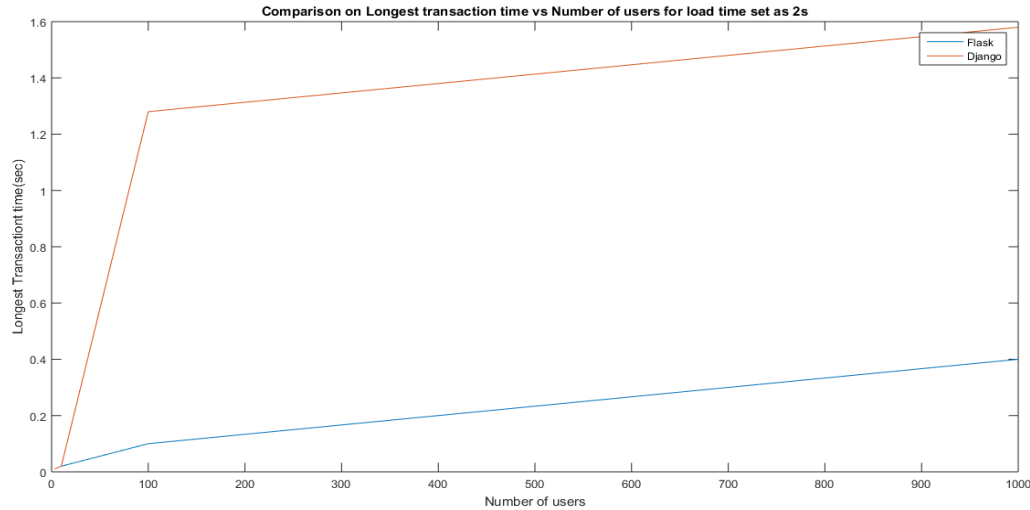
b. Throughput



c. Concurrency



d. Longest Transaction Time



Analysis: We observe that flask takes on an average less response time and performs significantly better as the number of users increase. Likewise it does better on Concurrency of handling users and the longest transaction time for a given transaction.

Django performs better than flask which is kind of intuitive as throughput increases as response time increases and decreases with decrease in response time.

2) Caching

We used the caching mechanism provided by the respective frameworks in our app as follows:

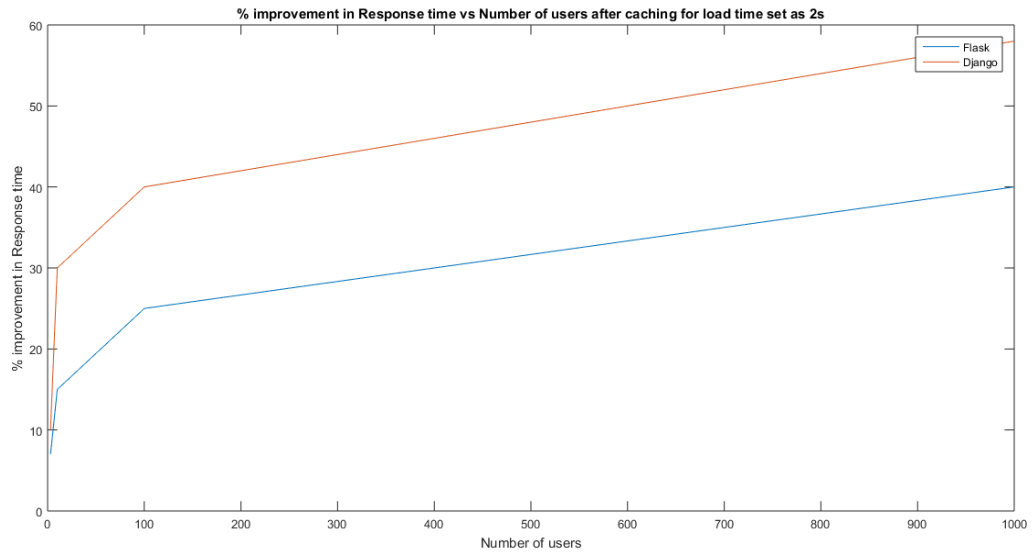
Flask: Import the flask cache extension, build a cache object by specifying cache, timeout for caching, attach the cache object to the app. In order to cache some specific functions/HTMLs frequently used we use the @cache decorator [11].

Django: Here we use memcache. In settings.py file create a Cache by specifying the ip and port of the server locations where django will store the cache [10].

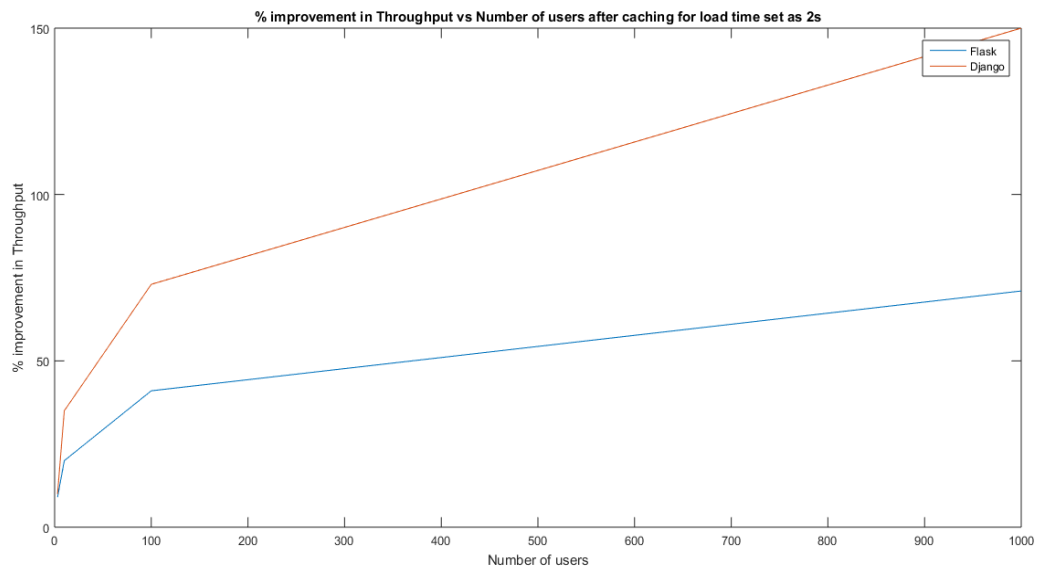
Now, for all the load testing experiments above we repeat these experiments with caching enabled in each of the flask and django apps. We then calculate the percentage improvement in performance. Now, instead of plotting the raw values, we plot the percentage improvement with the number of users. This would enable us to compare the caching feature in Django and flask to determine which framework's caching mechanism is more efficient. (Note: this would also include the caching overhead involved)

Results are as follows:

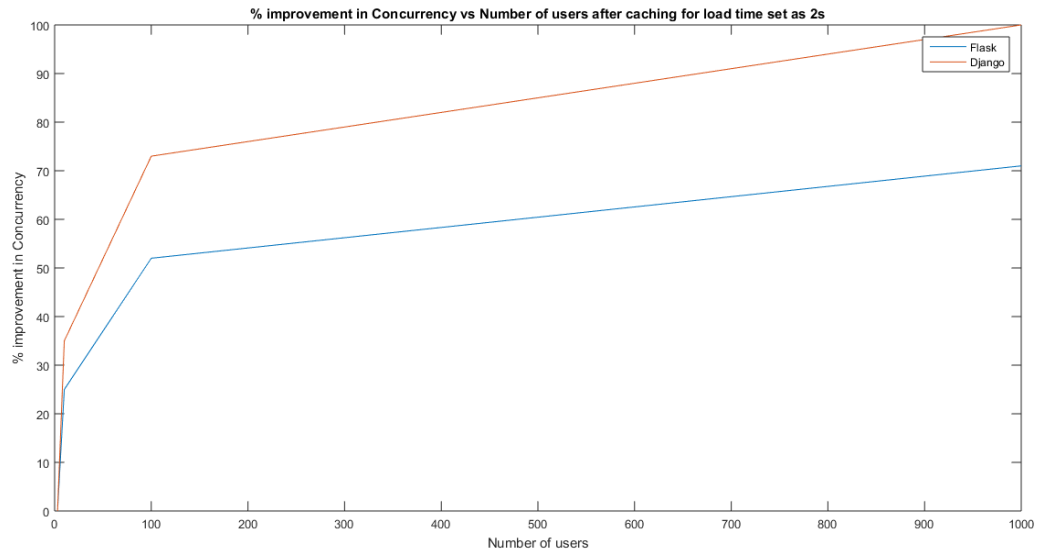
1. Response time



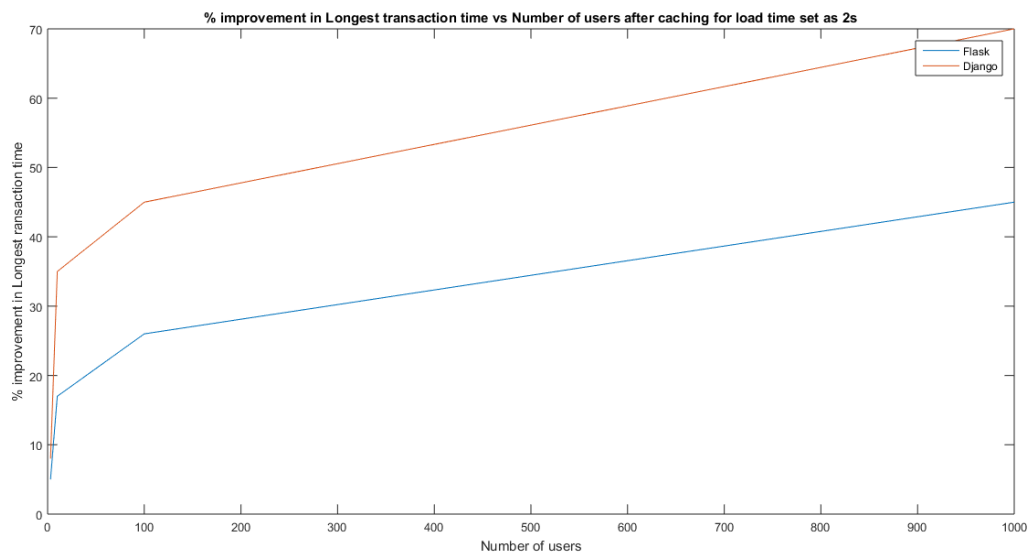
2. Throughput



c. Concurrency



d. Longest Transaction Time



Analysis: Django's caching mechanism is much better than that of Flask as the percentage speed-up/improvement for all metrics for Django is much better than that of Flask when caching is enabled.

Note: Flask still performs better overall but the percent improvement/speed-up in case of Django is better which indicates Django's caching mechanism is better than that of Flask. Improvement in performance was compared by repeating all the above experiments and then comparing the improvement.

3) Performance for interaction with UI (render_template vs HTTP_response)

User experience is a critical part of an application. A slow and unresponsive web page hampers user experience and may eventually lead to the user not using the app. In order to compare the two frameworks on rendering HTML pages we did a profiling test in order to compare the time it takes for each framework to load pages and perform different actions like creating post, writing comment, clicking like button etc.

Mainly render_template is used by Flask and Http_response is used in Django to render content on the HTML page. Following are the major functionalities of the app which were benchmarked:

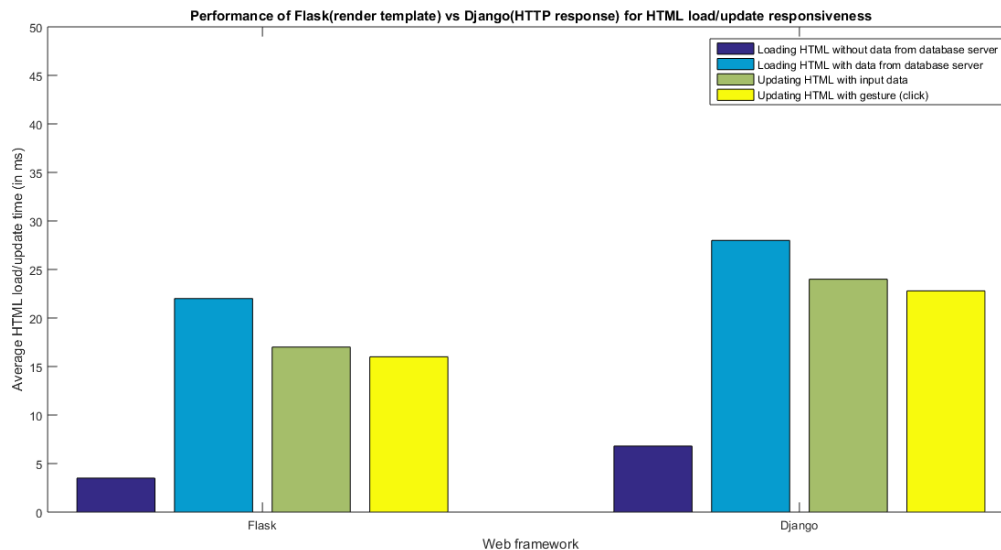
- 1) Loading HTML without data from database server
 - Loading Login page
 - Loading Signup page
- 2) Loading HTML with data from database server
 - Loading all posts
 - Loading all details of a post (question,author,date,answers,votes,comments,likes,replies)
- 3) Updating HTML with input data
 - Writing posts
 - Writing answers
 - Writing comment/reply
- 4) Updating HTML with gesture, click
 - Vote an answer
 - Like a comment/post

The experiments were carried out by interacting with the application in real time and using python's inbuilt time() function to compute the time taken by Flask's render_template and Django's function. This function is called before and after the render_template. This helps us to get the loading/execution time for the 4 major classes described above. The results of the profiling test performed on the two frameworks are presented in a tabular manner follows:

Functionality	Flask (in ms) render_template	Django(in ms)
1.Loading HTML without data from database server		
a. Login	3.4	6.3

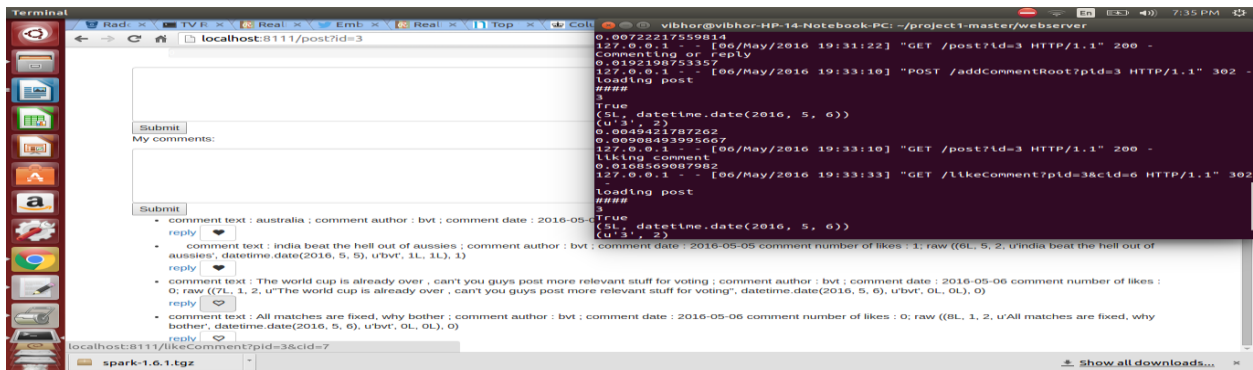
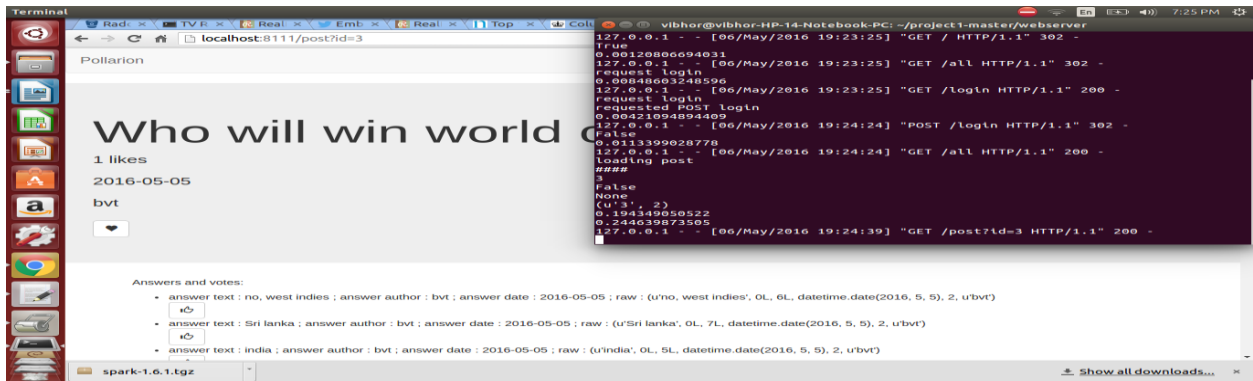
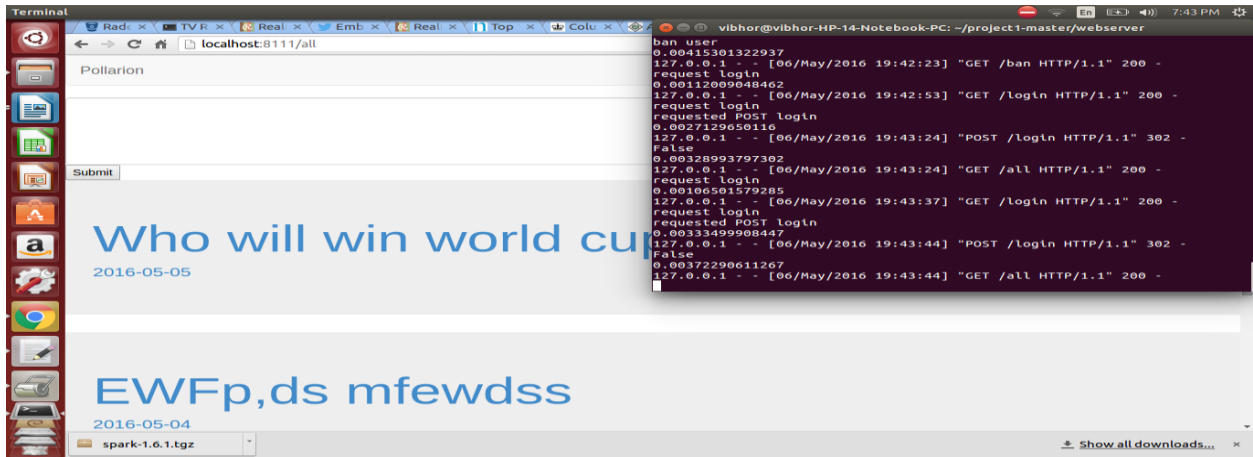
b. Signup	3.5	7.4
2. Loading HTML with data from database server		
a. Loading all posts	23.8	30.3
b. Loading all details of a clicked post	20.4	27.4
3.Updating HTML with input data		
a. Writing posts	18.4	24.5
b. Writing answers	14.4	22.1
c. Writing comment/reply	19.2	28.5
4. Updating HTML with gesture, click		
a. Vote answer	17.4	23.6
b. Like comment/post	15.3	22.3

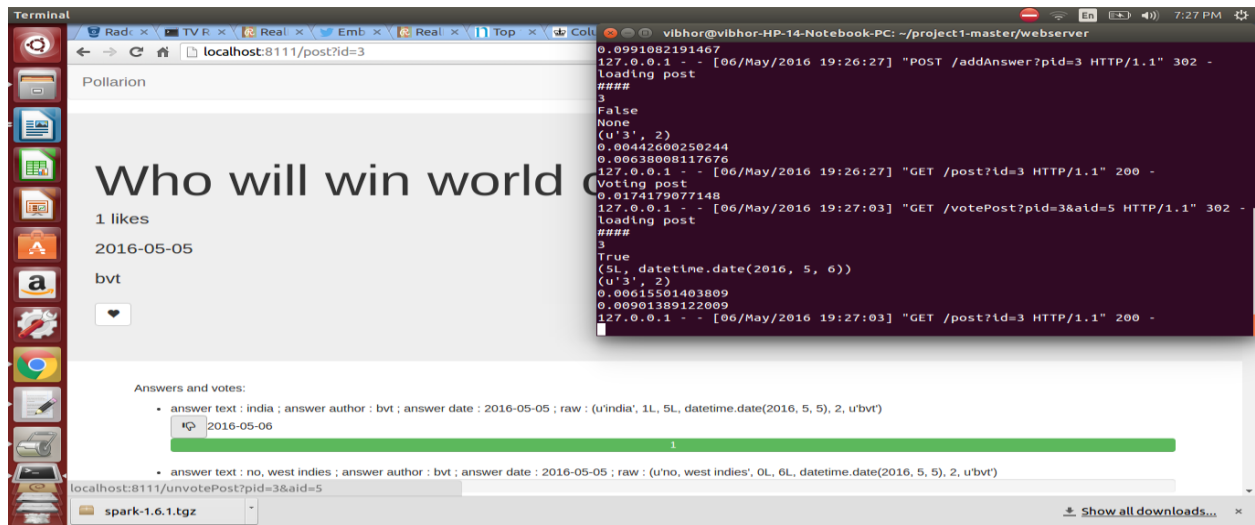
Plot for the performance of Flask vs Django (for each of the 4 categories above time is measured by averaging the time of all the functions involved in that category)



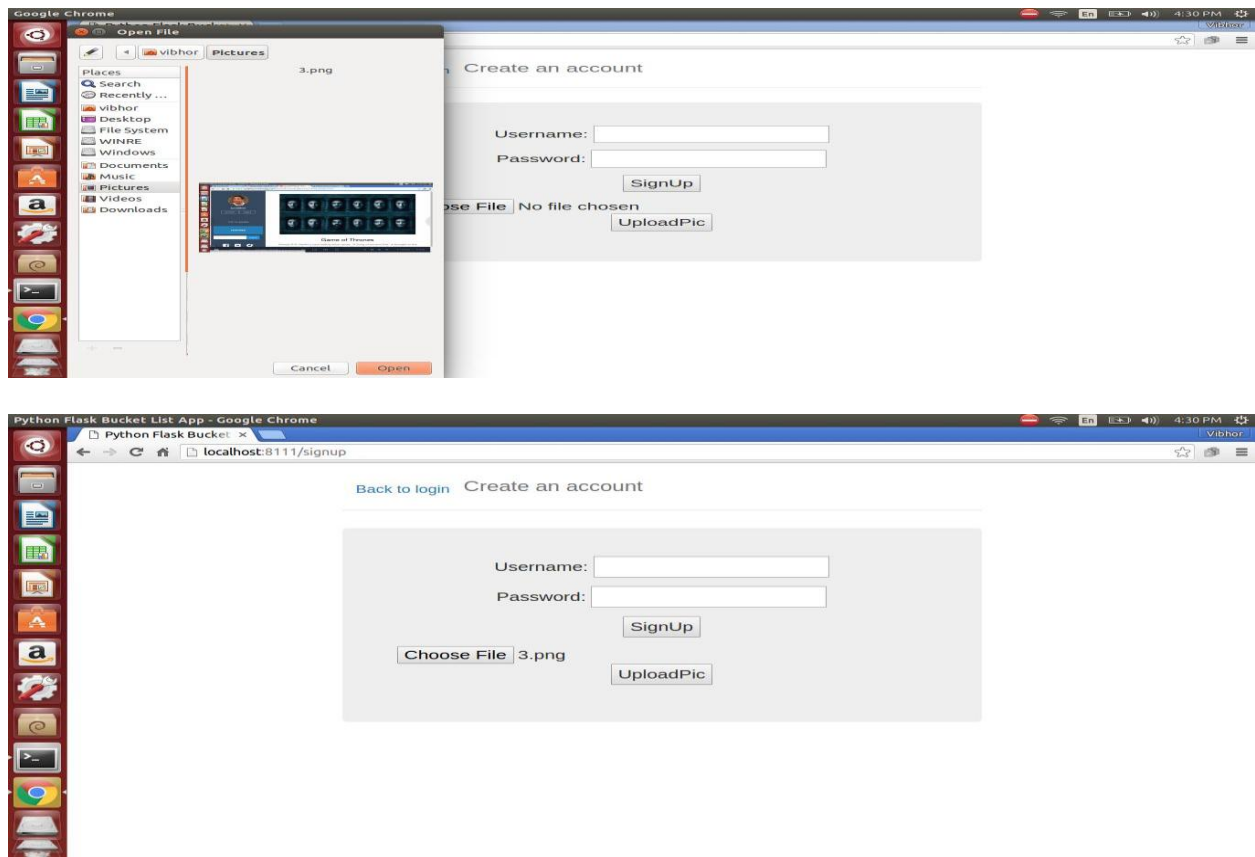
Analysis: Flask does better than Django in terms of response time for loading/updating HTML on all 4 types of HTML interaction involved while using the respective APIs.

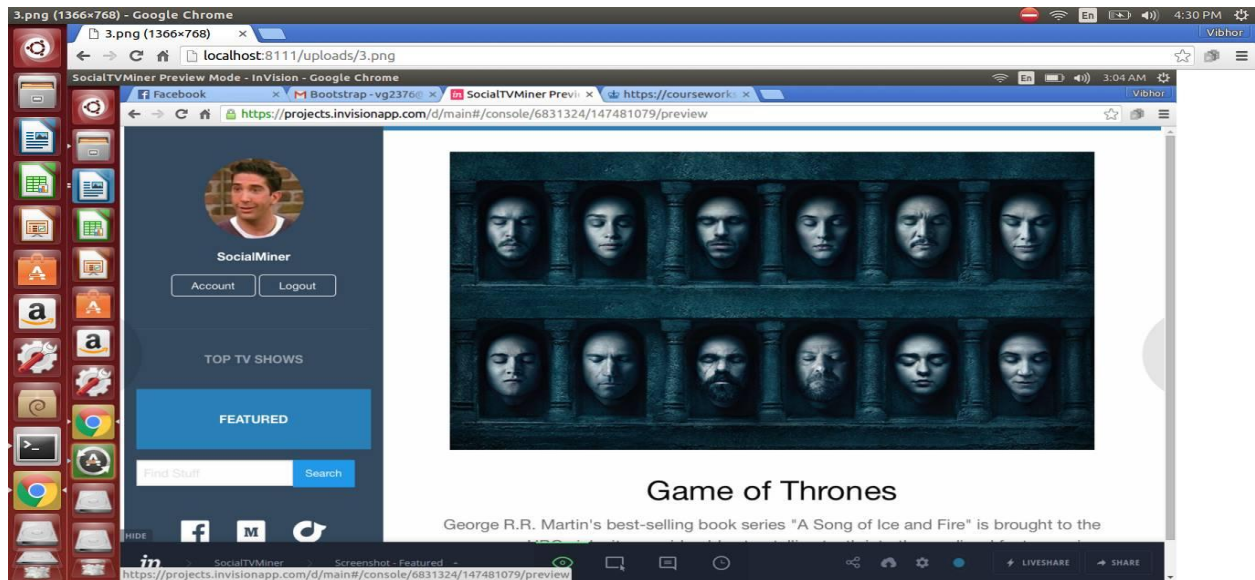
Sample screenshots for the tests carried out above are as follows:





4) File handling (Upload profile picture)





Flask:

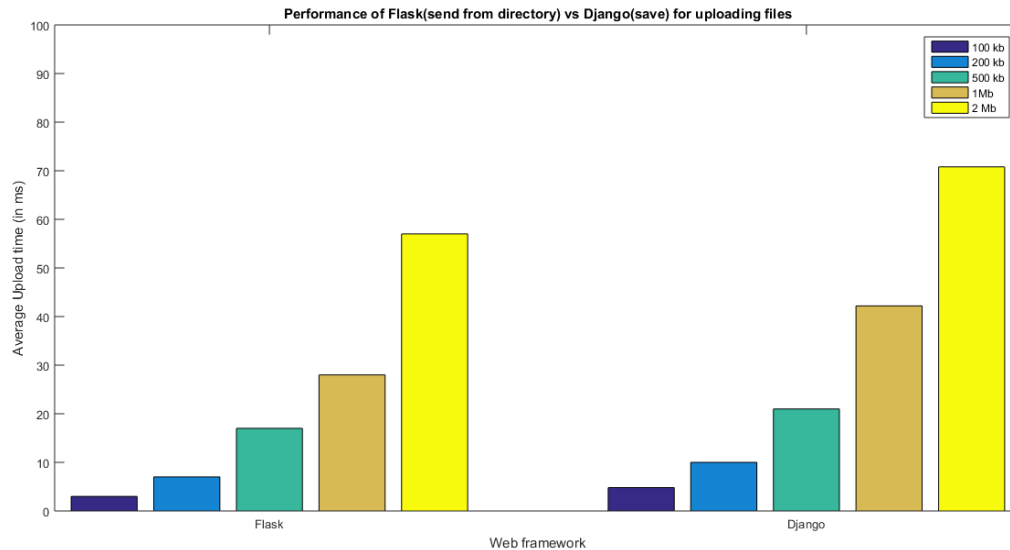
`send_from_directory()` : uploads file to the specified directory in `UPLOAD_FOLDER` attribute of app object.[2]

Django

`save()`: The directory to which the file is to be uploaded is specified in `mysite/polls/model.py` file [1]. The base directory is specified in the `settings.py` file

We experimented by uploading the files of varying sizes and computing the time taken (in ms) by each of these two functions using the `time()` function as in previous experiments to upload a file (Called before and after the function call and then time difference is taken).

We also experimented by uploading files of sizes of 100Kb, 200 Kb, 500 Kb, 1Mb, 2Mb. Below are the results of the upload performance of the two frameworks



Analysis: Performance of flask is better in terms of uploading files and the difference increases as the file size increases.

5) Security (Cross scripting, sessions/cookies/user authentication)

The following features were used for ensuring security in the app

Flask:

1) Cross scripting

Jinja2 templates are auto configured to escape text and check against the input given by the user in order to prevent a malicious script (Java Script or CSS) from getting injected through user input which could access session's cookies to gain access to information like user's password, delete his account and many other malicious activities. Even in that case the developer must be careful in.[15]

- a) Handling data from files which are uploaded.
- b) Use quotes in the template wherever attribute values are used

2) CSRF (Cross site request forgery)/User authentication

Basically cross scripting as described above to gain access to user's authentication credentials. To protect against this flask provides the feature of generating a csrf token appending it to the session. This string can then be verified against a field in the post request which is hidden.[13]

3) Sessions/cookies

Sessions can be created by importing the sessions package from flask and create a session password. Now, one can use as many attributes as he wants for the session such as 'username', 'password'. For eg. `session['username']=request.form('username')` which sets the user for the current session with the value inserted in the username field of the HTML form. These are visible across all other files and can be used in HTML to get the user details and other information. For cookies one can simply use the `@cookie` decorator in the function and use a response variable to which one can add as many values using the `response.set_cookie()` function everytime.

Django

1) Cross scripting

Using the same Jinja2 templates as in flask helped us to get the feature of autoescape text and validate user input.

Django also uses CSRF nonce in each POST request thus eliminating replay attacks completely. `CsrfViewMiddleware` [14] along with HTTPS ensures that the subdomain and port are set to the same URL as that of HTTP refer header on the origin. Unlike flask, the developer need not take care about security handling due to uploaded files and missing quotation around attributes. With these features, the developer practically doesn't need to worry much about cross scripting and injection attacks apart from making sure that they don't manually set a decorator to `csrf_exempt` (unless they are very sure).

2) Sessions/cookies

Like flask we can create `request.session` objects and add attributes to these for the current session. Like flask cookies can be set using a response object and the `set_cookie` method .

6) Architectural differences/ Brief comparison of other APIs

Django

Django is very easy to setup and start working with. Starting a new Django project creates a set of default files which the developer can use. These include the `manage.py` file which is used to run the server, `settings.py` file which contains the settings and is used to configure the system and `urls.py` file which contains the mappings between the url and the corresponding function which is executed. All the views/functions in Django are written in a single file called `views.py`, placed in the project or the application directory. The view function takes a web request and returns a web response. Django supports all the different template types and also provides its own templates if we make use of the automatic admin functionality. It also allows the user to make use of his own templates by adding the templates to the template directory created in the application. Django allows you to create models. Each model maps to a single database table. The models are stored in a single file `models.py` and can be accessed using python and without

writing raw SQL queries. Django makes use of migrations to propagate the changes we make to the models into the database schema. Migrations are automatic but it is upto developer to decide when to make migrations and when to run them.

Flask

Flask is a micro framework which makes it very easy to handle critical things. It comes with a lot of tools and extensions which makes it very easy to implement anything from log-ins to logging. It is different from django in the respect that it relies only partially on configuration files, which makes things easier to handle. Flask applications are mostly developed in their own virtual environment which contains all the dependencies for the application. Flask makes use of the route decorator to bind a function to a URL instead of creating a separate file like in Django for mapping URLs and functions. It is also different from Django with respect to it's feedback system. It provides a flashing system which makes it possible to record a message at the end of a request and access it on the next request. Similar to Django, Flask provides the option to create database using models. Models makes use of classes to create a database architecture and makes it easier to setup the database programatically, eliminating the need for manual effort in writing database queries and executing them.

7) Developer experience

The developer experience has been summarized in the table below.

8) Overall strengths and weaknesses

These have been summarized in the table below.

D. Summary of Results

Dimension for comparison	Flask	Django
Network access	Better on response time, concurrency, largest transaction time compared to Django	Better on throughput as compared to Flask
Caching	Achieves lesser speedup as compared to Django	Achieves more speedup as compared to Flask
Performance for interaction with UI	API: render_template Achieves better HTML load/rendering time as compared to Django	API: HTTP_response Takes more HTML load/rendering time as compared to Flask
File handling (Uploading files)	Upload time faster than Django for experiments in range 100 kb to 2Mb files	Upload time slower than Flask for experiments in range 100 kb to 2Mb files

Security (Cross scripting, sessions/cookies/user authentication)	Auto secure features provided by Jinja2 template, CSRF tokens to prevent session/cookie information being stolen. Developer needs to take care of file handling data and adding quotes in attribute fields	Along with security features in Flask (Jinja autosecure, CSRF) CsrfViewMiddleware in Django provides enhanced security features and developer doesn't have to worry about uploading files, quotation around attributes
Developer experience	Easier to learn than Django and start hands on development	Little tough to get started as compared to flask
Strength	Faster than Django for most common operations like network access, file handling, loading HTML.	Uses MVC architecture, easier to augment code, add features, ideal for managing dynamic and complex architectures with changing design requirements, has better caching mechanism than Flask
Weakness	A light python based framework which is not ideal for complex architectures with dynamic and changing design specifications	Slower than Flask for common operations like network access, file handling, loading HTML

E. Challenges

- 1) Developing and designing the initial application which could generalize to many web applications in the future.
- 2) Using the Jinja2 templates in Django was a very big challenge since Django does not support the new syntaxes for the Jinja2 templates and finding those deprecated syntaxes was a bit tough.
- 3) Django makes use of regular expressions to map the URLs to views. Writing these regular expressions can be very difficult especially for longer URLs.
- 4) Performing the experiments had a major challenge since we need to keep the UI for both the applications the same so that we can compare the frameworks independent of the UI.
- 5) Designing the experiments and keeping the experimental settings accurate such that the results are as accurate as possible.

F. Conclusion

- 1) Flask is a lightweight architecture which allows the developer to pick his own tools for development. Django is comparatively heavy architecture as compared to flask and makes use of many default settings and modules.
- 2) Flask outperforms django when we compared it based on important web app features like network access, Interaction with UI and File handling.
- 3) Security and caching are better in Django as compared to Flask.

G. Learnings from the project

The crux of the learning from this project which forms the basis of our suggestion to future web app developers is the following:

Use Django to build the initial application when the specifications and design requirements are dynamically changing as it provides a better framework through it's MVC architecture to handle these changes along with a stronger security framework. When the designs and code base get somewhat steady in the sense the high level design of the app doesn't change much and the developers want to scale to a larger number of users for their app it is better to switch to flask as it provides enhanced performance (mainly as it is light weight) on many important user experience dimensions as Network access, UI interaction, file handling.

F. Software (Deliverable)

The source code for the pollarion application developed in Flask and Django along with the respective README files can be downloaded from the following URLs. All experimentation and results have been documented in this report itself.

Download URL for Flask Application: github.com/vibhor1510/COMS-6156-Flask-App

Download URL for Django Application: github.com/prateekgupta89/COMSE6156_Django

G. Division of work

Vibhor Gaur: Designed and developed the initial application in flask, performed all the experiments for the apps made in flask and Django and wrote bulk of the report

Prateek Gupta: Developed the application in Django and worked on the report.

References

[1] Need a minimal Django file upload example. (n.d.). Retrieved May 10, 2016, from <http://stackoverflow.com/questions/5871730/need-a-minimal-django-file-upload-example>

- [2] Uploading Files¶. (n.d.). Retrieved May 10, 2016, from <http://flask.pocoo.org/docs/0.10/patterns/fileuploads/>
- [3] Joe Dog Software. (n.d.). Retrieved May 10, 2016, from <https://www.joedog.org/siege-home/>
- [4] Installation¶. (n.d.). Retrieved May 10, 2016, from <http://flask.pocoo.org/docs/0.10/installation/>
- [5] (n.d.). Retrieved May 10, 2016, from <https://www.digitalocean.com/community/tutorials/how-to-deploy-a-flask-application-on-anubuntu-vps>
- [6] Google's Python Class. (n.d.). Retrieved May 10, 2016, from <https://developers.google.com/edu/python/>
- [7] (n.d.). Retrieved May 10, 2016, from <http://flask.pocoo.org/docs/0.10/.latex/Flask.pdf>
- [8] Documentation. (n.d.). Retrieved May 10, 2016, from <https://docs.djangoproject.com/en/dev/intro/>
- [9] Documentation. (n.d.). Retrieved May 10, 2016, from <https://docs.djangoproject.com/en/1.9/topics/cache/>
- [10] Flask-Cache¶. (n.d.). Retrieved May 10, 2016, from <https://pythonhosted.org/Flask-Cache/>
- [11] Documentation. (n.d.). Retrieved May 10, 2016, from <https://docs.djangoproject.com/en/1.7/ref/templates/api/>
- [12] (n.d.). Retrieved May 10, 2016, from Grinberg, Miguel. Flask Web Development: Developing Web Applications with Python. " O'Reilly Media, Inc.", 2014. Holovaty, Adrian, and Jacob Kaplan-Moss. The definitive guide to Django: Web development done right. Apress, 2009.
- [13] Flask Snippets. (n.d.). Retrieved May 10, 2016, from <http://flask.pocoo.org/snippets/3/>
- [14] Documentation. (n.d.). Retrieved May 10, 2016, from <https://docs.djangoproject.com/es/1.9/topics/security/>
- [15] Security Considerations¶. (n.d.). Retrieved May 10, 2016, from <http://flask.pocoo.org/docs/0.10/security/>