

Big Data final project report- GitHub pull request analisis

Maria del Carmen Sacristan Benjet & Jin Yang

<https://github.com/beishanyangjin/cs585FinalProj>

Abstract

This project examines the trends for repositories and users found in pull requests throughout 2 days, 2019-03-11 and 2019-05-20. The pull request data from these days comes from a kaggle dataset. The project makes use of Docker, Hadoop and Spark to manage the complexities of big data processing.

Introduction

GitHub is an essential tool for code developers, as such it counts on approximately 83 million users, 32 million monthly users and 200 million repositories. Given that GitHub manages the development process, its data can help us understand patterns and habits of developers. The GitHub Pull Requests database holds a massive amount of data points about pull request details.

Database description

The kaggle database “ghtorrent” contains all the publicly available pull requests during 5 days (non-holiday) in 2019. For each pull request there are the following data: actor_login, actor_id, component_id, comment, repo, language, author_login, author_id, pr_id, c_id (commit id), commit_date. The database is 92.53 GB in size. Because just downloading the 92.53 proved a major issue memory and time wise, we decided to use only two days of the 5 (2019-03-11 and 2019-05-20) for a total of 37.15 GB.

Problem statement

For this project we will analyze if there are any common trends that pull requests have. We viewed the data from 3 perspectives: individual pull requests, individual contributors, and repositories. We wanted to answer the following questions: Is there a time period throughout the day where more pull requests are made? How many pull requests do users make and how is this distributed? What are the trends in regards to pull request

comment length? Which are the largest repositories in terms of pull request as well as number of individual contributors? What are the most used languages?

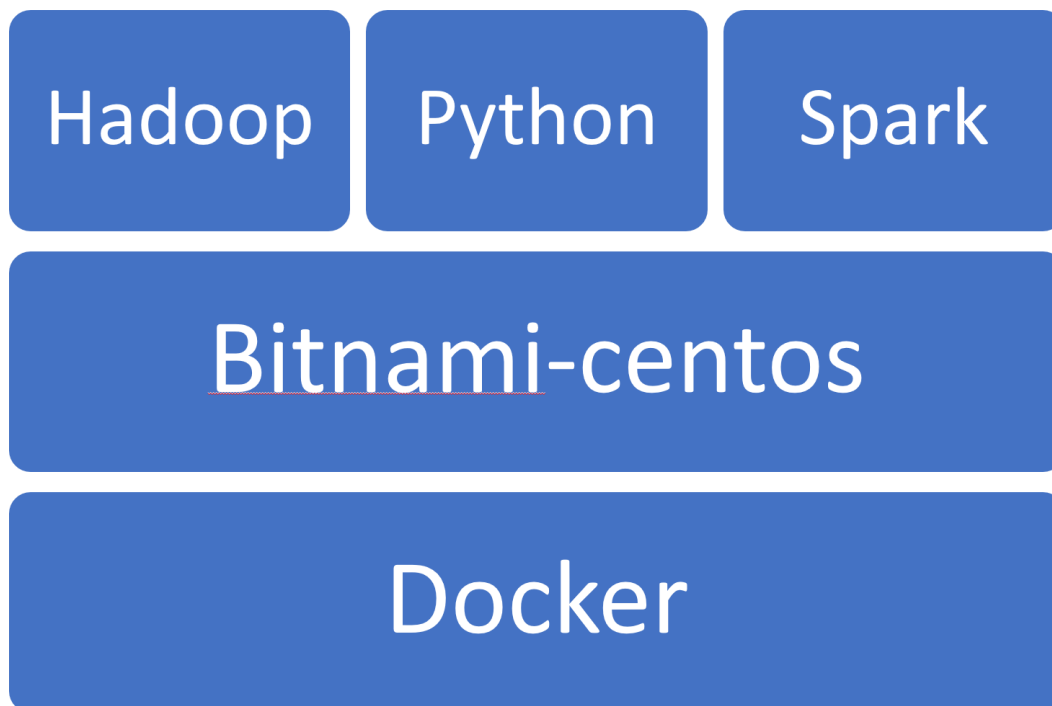
Technologies

The technologies used were docker, hadoop and spark. Docker was key to the project as it allowed for a scalable isolated environment to run the code. We chose to use bitnami images to build the docker container because it provided a framework that had what we needed, was simple and up to date. Hadoop was necessary to store and manage the large dataset. We chose to use spark because it is very efficient when handling large volumes of data and reduces the amount of code necessary for map reduce functions.

Method: Pipeline and execution

We built our project inside a docker container. We used Bitnami images to build our hadoop and spark container. Inside the container, hadoop is run to create the hdfs file system where the raw data is stored. The spark code is then pulled from the hdfs file system to read the data. After the data processing is done, the output is stored back into the hdfs.

Figure 1: Project architecture pipeline

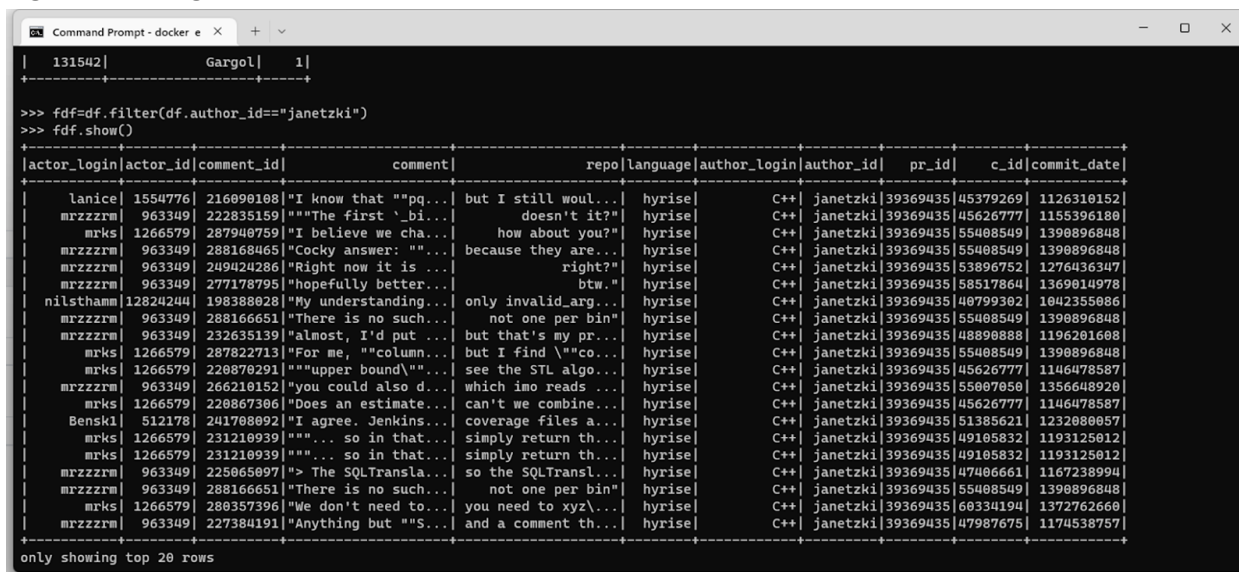


Challenges and Obstacles

Working with such a large database was a challenge concerning both memory and time. It was an issue just to clear up enough space on the computer. From there uploading the data to the hdfs took a matter of hours. Thanks to the efficiency of spark and thoughtful design, reading, parsing and analyzing the data only took a more reasonable 30 min.

Another struggle was the format of the data. The comment field in the data included strings that could have any characters. This meant that when spark read the csv into a dataframe, it messed up the columns in the data. We figured out through debugging that when a string contained quotation marks, spark interpreted this as a separate string. Meaning that the remainder of the string would be assigned to the new column and all the subsequent fields would be shifted one column over. Figure 2, illustrates this issue.

Figure 2: Sting format issue



```
Command Prompt - docker e X + v
| 131542 | Gargol | 1 |
|-----|-----|-----|
>>> fdf=df.filter(df.author_id=="janetzki")
>>> fdf.show()
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|actor_login|actor_id|comment_id|comment|repo|language|author_login|author_id|pr_id|c_id|commit_date|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|lanice|1554776|216090108|"I know that "pq... but I still woul...|hyrise|C++|janetzki|39369435|45379269|1126310152|
|mrzzrrm|963349|222835159|""The first 'bi... doesn't it?"|hyrise|C++|janetzki|39369435|45626777|1155396180|
|mrks|1266579|287940759|"I believe we cha... how about you?"|hyrise|C++|janetzki|39369435|55408549|1390896848|
|mrzzrrm|963349|288168465|"Cocky answer: "..." because they are...|hyrise|C++|janetzki|39369435|55408549|1390896848|
|mrzzrrm|963349|249424286|"Right now it is ... right?"|hyrise|C++|janetzki|39369435|53896752|1276436347|
|mrzzrrm|963349|277178795|"hopefully better... btw."|hyrise|C++|janetzki|39369435|58517864|1369014978|
|nilsthamm|12824244|198388028|"My understanding... only invalid_arg...|hyrise|C++|janetzki|39369435|40799302|1042355086|
|mrzzrrm|963349|288166651|"There is no such... not one per bin"|hyrise|C++|janetzki|39369435|55408549|1390896848|
|mrzzrrm|963349|232635139|"almost, I'd put ... but that's my pr...|hyrise|C++|janetzki|39369435|48890888|1196201608|
|mrks|1266579|287822713|"For me, "column... but I find \"co...|hyrise|C++|janetzki|39369435|55408549|1390896848|
|mrks|1266579|220870291|""upper bound\""... see the STL algo...|hyrise|C++|janetzki|39369435|45626777|1146478587|
|mrzzrrm|963349|266210152|"you could also d... which imo reads ...|hyrise|C++|janetzki|39369435|55007050|1356648920|
|mrks|1266579|220867306|"Does an estimate... can't we combine...|hyrise|C++|janetzki|39369435|45626777|1146478587|
|Benski|512178|241708092|"I agree. Jenkins... coverage files a...|hyrise|C++|janetzki|39369435|51385621|1232080057|
|mrks|1266579|231210939|""... so in that... simply return th...|hyrise|C++|janetzki|39369435|49105832|1193125012|
|mrks|1266579|231210939|""... so in that... simply return th...|hyrise|C++|janetzki|39369435|49105832|1193125012|
|mrzzrrm|963349|225065097|"> The SQLTransla... so the SQLTransl...|hyrise|C++|janetzki|39369435|47406661|1167238994|
|mrzzrrm|963349|288166651|"There is no such... not one per bin"|hyrise|C++|janetzki|39369435|55408549|1390896848|
|mrks|1266579|280357396|"We don't need to... you need to xyz\...|hyrise|C++|janetzki|39369435|60334194|1372762660|
|mrzzrrm|963349|227384191|"Anything but "S... and a comment th...|hyrise|C++|janetzki|39369435|47987675|1174538757|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 20 rows
```

We solved this by adding an instruction to the file read that escapes the quotation characters in the string, As shown by figure 3.

Figure 3: String format fix

```
spark.read.csv(r"hdfs://master:9000/ghtorrent-2019-03-11.csv")
↓
spark.read.csv(r"hdfs://master:9000/ghtorrent-2019-03-11.csv", escape = '"')
```

In order to confirm that this fix had worked we queried the author_ids that we knew to have issues. Before the fix we noted that “janetzki” appeared often in the author_id column despite the column supposedly only containing a unique number id, clearly data that was from a different column (Figure 2 illustrates this). After the fix when we queried for the value “janetzki” in the author_id column and got no rows in return (as seen in Figure 4), meaning the fix worked.

Figure 4: Data in now in the correct column

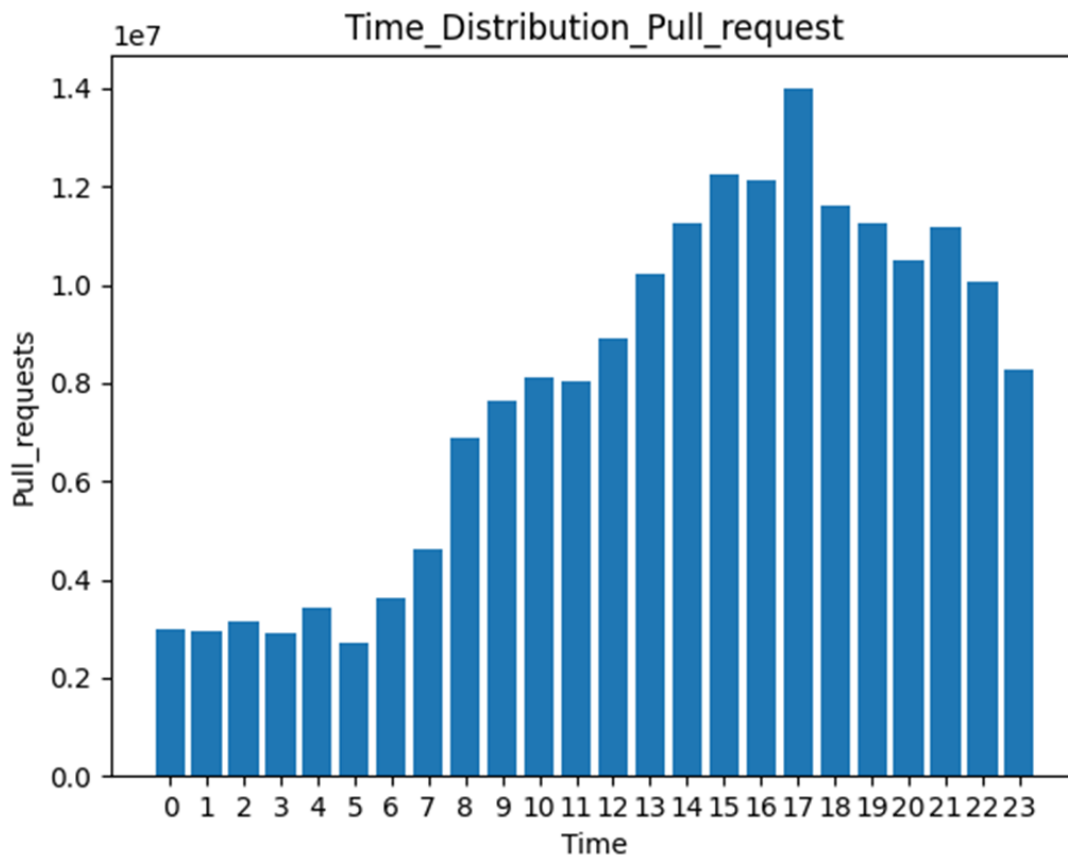
```
SyntaxError: EOF while scanning string literal
>>> df = spark.read.csv(r"file:///opt/share/ghorrent-2019-03-11.csv", escape = '').toDF('actor_login','actor_
guage','author_login','author_id','pr_id','c_id','commit_date')
>>> fdf=df.filter(df.author_id=="janetzki")
>>> fdf.show()
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|actor_login|actor_id|comment_id|comment|repo|language|author_login|author_id|pr_id|c_id|commit_date|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

Results

Analysis of pull request time frequency in UTC

We selected the column that contained the UTF date and time stamp and parsed the data to extract the hour of day. We then grouped the time of day into 24 buckets to make a histogram of the distribution of Pull Request throughout the day (Figure x). We found that there was a trend for the time of day Pull requests were made peaking at 5pm UTC time.

Figure 5: Time distribution of Pull Requests



Analysis of actor trends

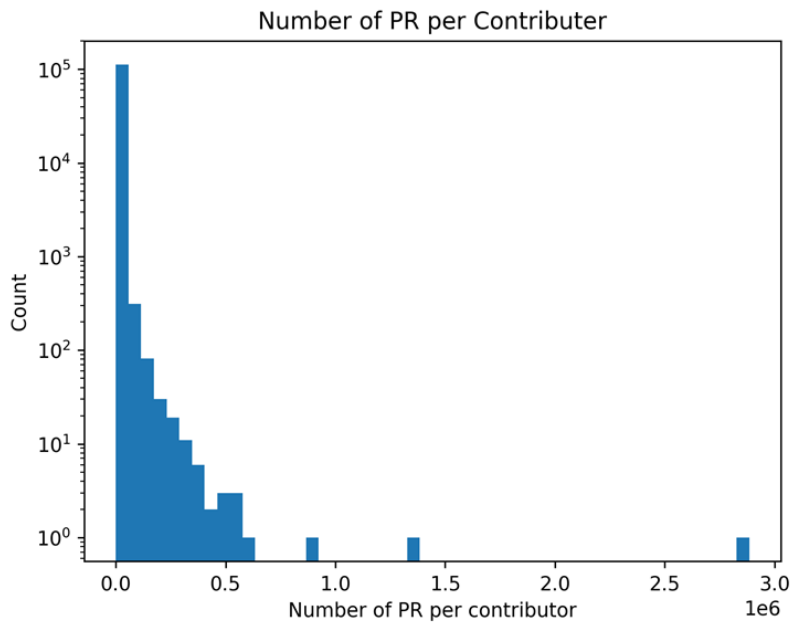
We also analyzed trends when it came to individual actors. We perform a group by operation on actor_id and actor_login. With this operation we were able to rank the most active actors. Through this analysis we found that there were bots that were creating massive amounts of pull requests per day. The most active actor called houndci-bot made 2884918 pull requests. This bot had an average length of comment of 66 characters, which is below average. Figure 6 shows the top 5 most active users and the 5 least active users with only 2 Pull requests. Overall it seems that the least active users had a longer average length of comment than the most active bots. Nonetheless, bots don't necessarily always have a low average length of comment. The third most active actor "codacy-bot" with 913176 had an average length of 201 characters which is well beyond the actor average of 100 characters.

Figure 6: Activity of individual users data frame

	actor_id	actor_login	count	length_of_comment
0	11568779	houndci-bot	2884918	66.316176
1	1435621	MartinHjelmare	1368852	81.186590
2	16780017	codacy-bot	913176	201.223267
3	718011	jreback	585524	68.693642
4	1816524	seanlip	571536	116.972222
...
112888	26609735	sragia	2	254.000000
112889	49831207	ChronicPwnage	2	254.000000
112890	3730222	obicke	2	254.000000
112891	456496	skagedal	2	254.000000
112892	2126894	xipmix	2	254.000000
[112893 rows x 4 columns]				

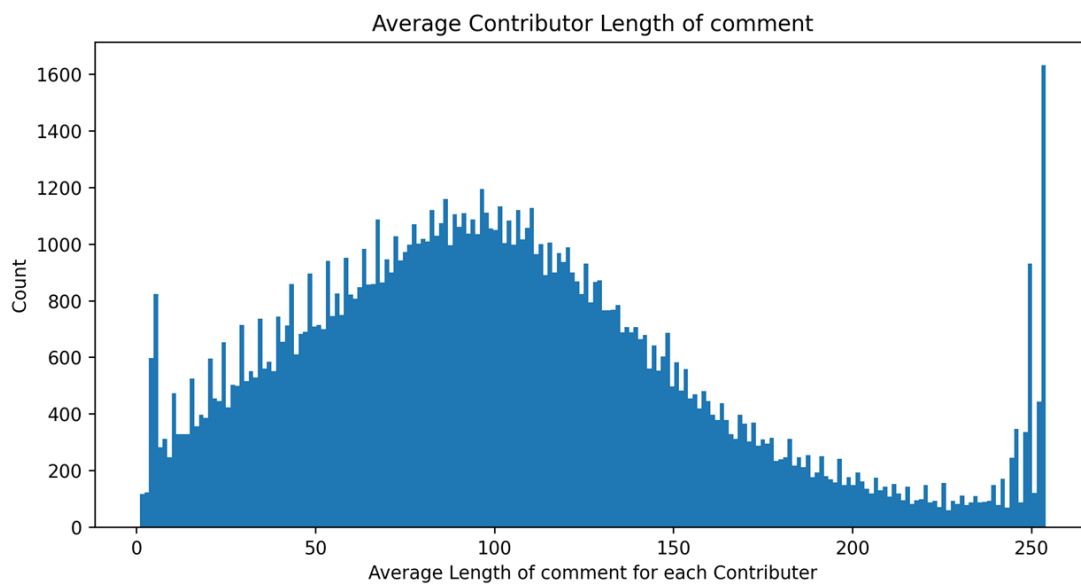
The distribution of the number of Pull Requests per actor, shown in Figure 7, shows that actors that have an unreasonably large amount of Pull Requests are a small minority. The histogram uses logarithmic axes in order to be able to visualize these outliers.

Figure 7: Number of Pull Requests per contributor



From this data that is grouped by actor, we also considered it interesting to see the distribution on average comment length per actor. The median average comment length is at 100 characters. We found it interesting that comment length trends towards 100 characters but also have peaks at close to 0 and over 250 characters. The distribution is shown in Figure 8.

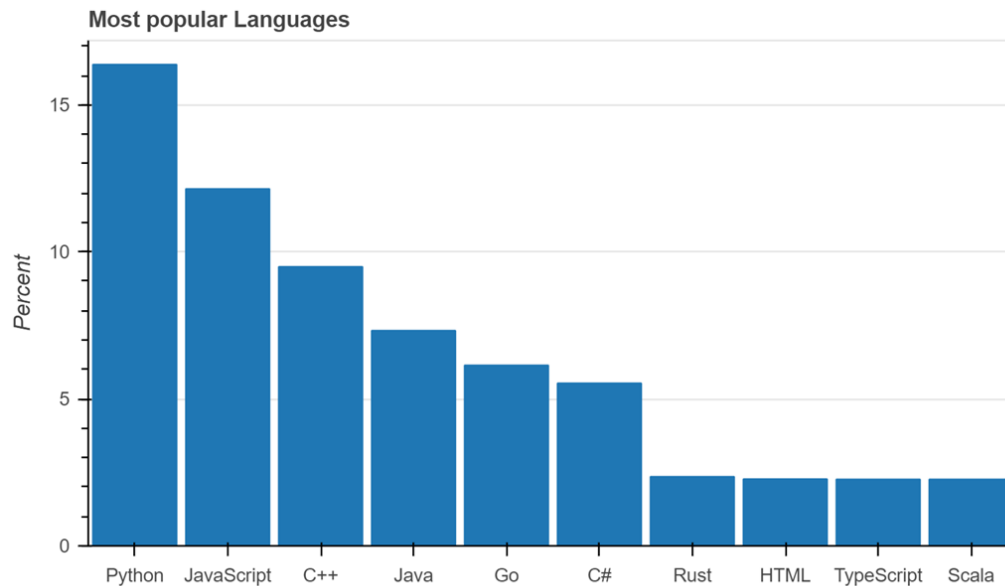
Figure 8 :Average Contributor Length of Comment



Analysis of Pull Requests most commonly used language

Furthermore, we wanted to know what the most common languages the Pull Requests used. Figure 9 shows 10 most popular and their prevalence.

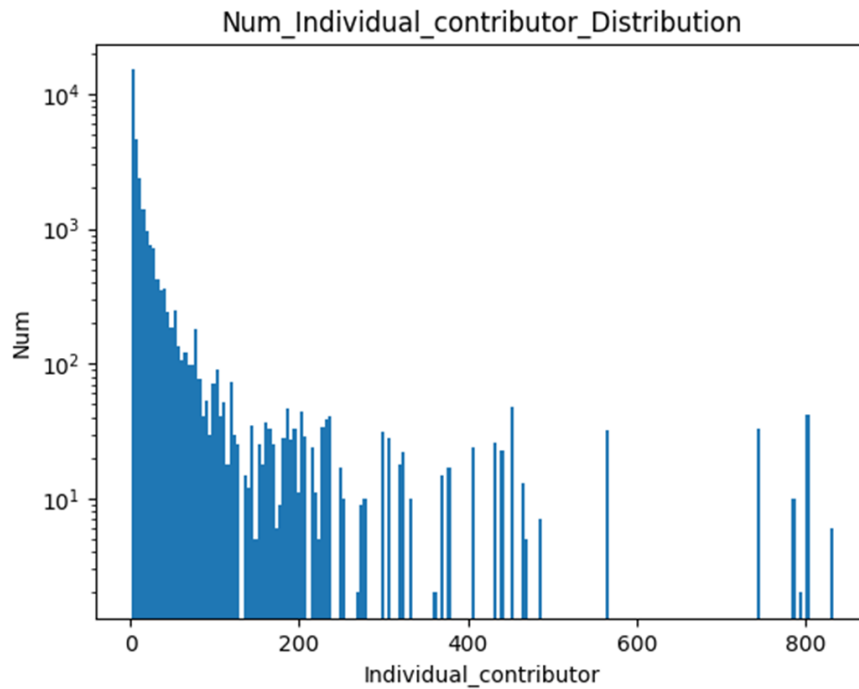
Figure 9: Most Used Languages



Analysis of Repository trends

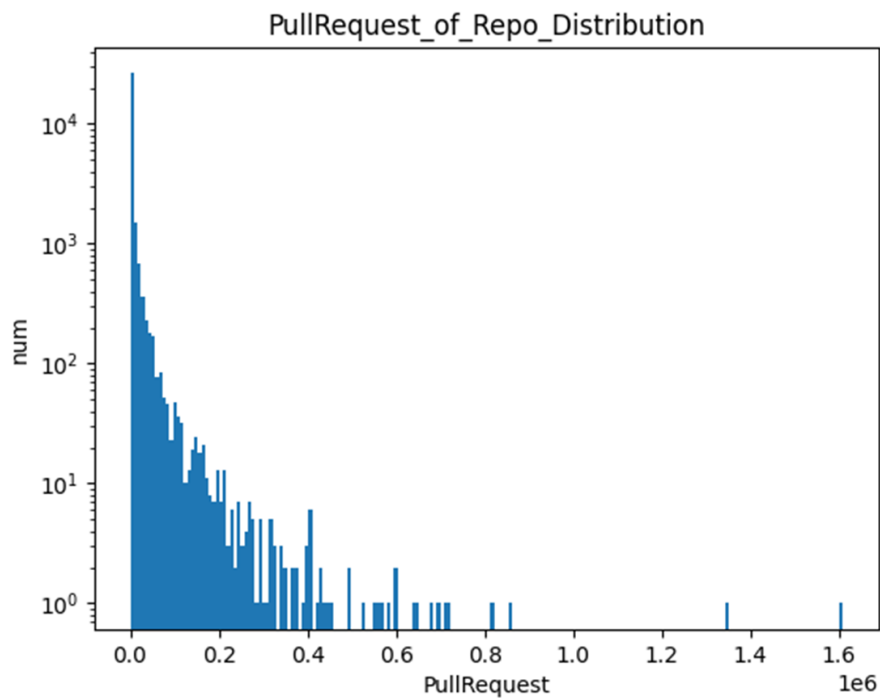
We also found the distribution of the number of Pull Requests and number of individual contributors to individual repositories. Figure 10 shows that the vast majority of repositories had only one or 2 distinct contributors. There were a minority of repos that did however have distinct contributors in the hundreds over the 2 days studied.

Figure 10: Distribution of the number of individual contributors per repository



The distribution of the number of Pull Requests per repo showed a similar trend. As shown in Figure 11.

Figure 11: Distribution of the number pull requests per repository



Conclusion

Through the project we uncovered some interesting trends about github pull requests. On the user side of things we found that despite people around the world having different time zones, there is a peak of activity around 5 pm UTC time. Users also write comments of an average length of 100 characters. Our analysis also uncovered the presence of bots that make automated pull requests. Python was calculated to be the language used in 16% of pull requests, this makes it the most used language. On the repository aspect, we found a large range in regards to pull requests and individual contributors, however the vast majority of repositories have only a couple of each, with only a few outliers in the hundreds. This project taught us about the difficulties of memory and time that arise with big data. Spark proved very impressive in terms of efficiency. The entire data processing in spark took a fraction of the time it took to just download the data. All while requiring a few lines of code to achieve. Docker and hadoop also proved essential to store and manage big data, without it the scale of the project would not have been manageable.

References:

GitHub repository for the project:

<https://github.com/beishanyangjin/cs585FinalProj>

The kaggle data was retrieved from:

<https://www.kaggle.com/datasets/stephangerland/ghtorrent-pull-requests?select=ghtorrent-2019-05-20.csv>

Bitnami resource:

<https://bitnami.com/>