Final Project Report
Team: CloudyWithAChanceOfClusterFailure
Renuka Nannapaneni, Mihir Pandya, Aidan San, James Wang

Introduction

The goal of our project was to analyze the trends of different programming languages across the different aspects of a programmers life, spanning from their open source contributions, to their questions, to their jobs. We wanted to see if there was a relationship between the three datasets that represented these facets of a programmer's life. We built a website which shows overall popularity of languages, trends of language use over time, as well as language popularity in certain timezones.

Datasets

We used three datasets to perform our analysis: GitHub, StackOverflow, and Dice. Both the GitHub and StackOverflow data came from Google BigQuery, while the Dice dataset came from Kaggle. The overall sizes ended up being ~150 million rows for GitHub, ~2 million rows for StackOverflow, and ~22,000 rows for Dice.

We chose to use BigQuery because it allowed us to interact with the datasets using SQL queries and choose what columns we wanted from the dataset. It also had a side-effect of allowing us to download the entire dataset onto our laptops before transferring them to the cluster (The entire GitHub dataset is close to 1TB!). The other sources we found for GitHub and StackOverflow were too large to download onto any of our laptops.

All of the data is licensed under the Creative Commons License, and is free to use. The appropriate links leading to the datasets were put on a README file and added to HDFS along with the data itself.

Frameworks

Of course, the most important Framework we used was Spark. Both the StackOverflow and Github datasets were too large to use any other framework or language. Spark also gave us a very easy way to not only extract values we needed, combine them with data from other datasets, and perform analysis, but also quickly test different combinations of data values for efficiency. We used the Pyspark interactive shell for exploratory analysis and testing.

Here are some code examples of our usage of Spark:

```
#this method takes the post data, parses it, and returns the
#desired data in the form (uid, (date, correct_words))
def get_post_pairs(entry):
    keywords = ['javascript', 'sql', 'java', 'c#', 'php', 'python', 'c++', 'js', 'ruby', 'objective-c'] # + c
    if (entry[0] == "id"):
        return ("#", "#")
    date = entry[1]
    date_vals = (date.split(" ")[0]).split("-")
    date = date_vals[1] + "/" + date_vals[2] + "/" + date_vals[0]
    tags = entry[2].split("|")
    correct_words = []
    for tag in tags:
        for word in keywords:
            if word in tag.lower():
                if (word == 'js'):
                    correct_words.append('javascript')
                else:
                    correct_words.append(word)
            if tag.lower().strip() == 'c':
                correct_words.append('c')
    if len(correct_words) == 0:
        return ("#", "#")
    uid = entry[3]
    return (uid, (date, correct_words))
```

This is an example of how the initial data (namely, the StackOverflow posts dataset) is cleaned and simplified.

```
#structure of joined_data is (location, (date, tags))
def query_2(sc, joined_data, q1_results):
    #creates data pairs for all the tags and months in the year
    data = joined_data.flatMap(lambda x: [((tag, x[1][0].split("/")[0]), 1) for tag in x[1][1]]).reduceByKey(lambda x, y: x + y)

    #restructures the data to match the format (tag, (month, amount of instances))
    data = data.map(lambda x: query_2_restructure(x))

    #groups keys to match all the months to the same key
    data = data.groupByKey().mapValues(list)

    #join the data to the yearly count data
    data = data.join(sc.parallelize(q1_results))
    return data

def query_2_restructure(entry):
    month = entry[0][1]
    return (entry[0][0],(month, entry[1]))
```

The image above is an example of some Spark processing techniques used when processing the StackOverflow dataset.

```
#joining the commits table with the repositories table on the repository name
joined_df = commits.join(repos, 'repo', 'inner')
...
#pulling the languages we are tracking from the language tag
fix_date = fix_date.map(lambda x: check_lang(x, langs))
...
#counting the key ((date, language), count)
map_count = map_count.reduceByKey(lambda x, y: x + y)
```

The image above is an example of some Spark processing techniques used to combine the commit and repository data in the GitHub dataset.

The Dice Jobs dataset was significantly smaller than the other two datasets (only 22,000 rows). If we were to use Spark to analyze the data it would probably take longer to queue than it would have taken to run the program on just a single core. An important lesson we learned in CS398 was to use the correct tool for the job. Since the dataset was so small we decided to use Jupyter and Pandas instead of using Spark to both conserve the resources of the cluster in addition to saving us time.

A key goal of our project was to make our analysis easy to access and use. We decided that a public website would be the easiest way to achieve this. We used a JavaScript library called ChartJS. Because of the many languages we were visualizing and the very large number of points (all days in the year), the graph showing every language was very cluttered. Another feature of ChartJS was the ability to click on labels to hide that language so that the person using the website can choose only the relevant languages that he or she wants to look at.

Features

From the GitHub dataset we kept information on both repositories and commits. For the commits, we kept track of author timestamps (UTC), author timezone offsets, the date of the commit, and finally the name of the repository containing the commit. For each repository, we kept track of the repository name and the languages that it contained. All the Github data that was necessary was complete, so no values had to be omitted.

From the StackOverflow dataset, we kept information on both posts and users. From each post, we kept track of timestamps (UTC), tags, and the id of the user who created the post. For users, we simply kept track of the user's id and the location they set on their profile. There was no timezone offset for the timestamps, so we had to use the location to find the timezone. StackOverflow was missing some location data, and as that was necessary for the analysis, those data were omitted.

From the Dice dataset, we kept state information and then changed it into timezone data. We also kept track of date data, so that we could produce the time based line graphs.

Analysis

Our first hypothesis was that the most popular languages for developers would be reflected in Github and StackOverflow activity. We saw that the top three languages, which included JavaScript, Java, and SQL/Python, were consistently represented from the visualizations (E.g. Fig. 1, Fig. 2, Fig. 3).

Our second hypothesis was that the frequency of activity we saw for individual languages on GitHub and StackOverflow would be represented in job postings in the

Dice dataset. To explore this theory, we looked at our visualizations of daily data, day of the week data, and monthly data. Although the rise and fall of the languages over the year in the Dice, did not completely match those in the developer dataset, these graphs largely supported our hypothesis, showing that the the most popular languages on the developer sites like JavaScript, Java, and Python were also very apparent in the Dice jobs.

One of the disparities between the developer datasets and the Dice jobs dataset was the popularity of C. C is mostly a specialized language in the industry so it would make sense that there aren't as many jobs for it. Lots of tools are written in C, and since it's more complex, it makes sense that it would be more prevalent on Github and StackOverflow.

Another disparity is the popularity of SQL, which is very popular on Dice but not as popular on the developer sites. GitHub separates SQL into PLSQL and SQL, but we decided to focus solely on SQL as StackOverflow and Dice did not have PLSQL. SQL is also mainly run using libraries that are written other languages like Python or Java. These factors could have contributed to the disparity we see in the data. Disregarding SQL and C, the hypothesis is largely supported by our findings.

From the time-based monthly data, we were able to see a trend; after visualizing the Dice dataset, we were able to see the recruitment cycles in March, September, and December, and were able to see the end of year drop off on developer websites. Using the timezone visualization, we were also able to see that PST and EST were the timezones with the most activity on both Dice and StackOverflow. (E.g. Fig. 5)

A notable outlier was a beginning of the year spike in Github Ruby activity. Upon further research, we found that there was a new version of Homebrew being released. In the days leading up to the release, the creators were "autocorrecting" their code and making a large amount of menial changes. This was the spike that we say in the daily data visualization (E.g. Fig. 4)

Issues

The first issue that we encountered was getting access to our initial source of GitHub data, GHTorrent. Each month's data was at least 50GB, so we needed access to their MySQL database. However, in order to gain access, we needed to submit a pull request to their repository containing our ssh public key. While they said on their website that they would accept these pull requests on a weekly basis, they didn't merge any pull requests for several weeks after we submitted ours, so we switched to BigQuery.

Overall one of our biggest goals was consistency. Instead of having 3 different analyses of each dataset we wanted to have one unified analysis. This caused us some problems, because all three datasets did not provide us with the same pieces of

information. We did not have location information from GitHub, only offset information, so we instead had to sacrifice some specificity in out analysis of location by transforming our Dice and StackOverflow location information which was originally in the form of user inputted city and state fields into timezone information.

Performance

Github: The entire application ran for 112 minutes and 46 seconds. The largest job, processing for the daily data, ran for 1459 seconds.

StackOverflow: The entire application ran for 3 minutes and 28 seconds. The largest job, processing for the yearly data, ran for 141 seconds.

Dice: Since, the dataset was so small, the application ran in a trivial amount of time.

Conclusion

Looking forward, we should find a better way to handle time offset data, and extend our analysis to frameworks and other languages. If we had more resources, we would also be able to purchase the remainder of the Dice dataset and increase the amount of languages we look at to track language popularity over a longer period of time.

We began by asking a question relevant to developers everywhere: What languages will help me the most in the real world. We analyzed the data using Spark and other technologies and finally made a publicly facing application, making it easy for developers to access and use.

Mentioned Visualizations:
https://www.tinyurl.com/plangvis

Fig. 1:

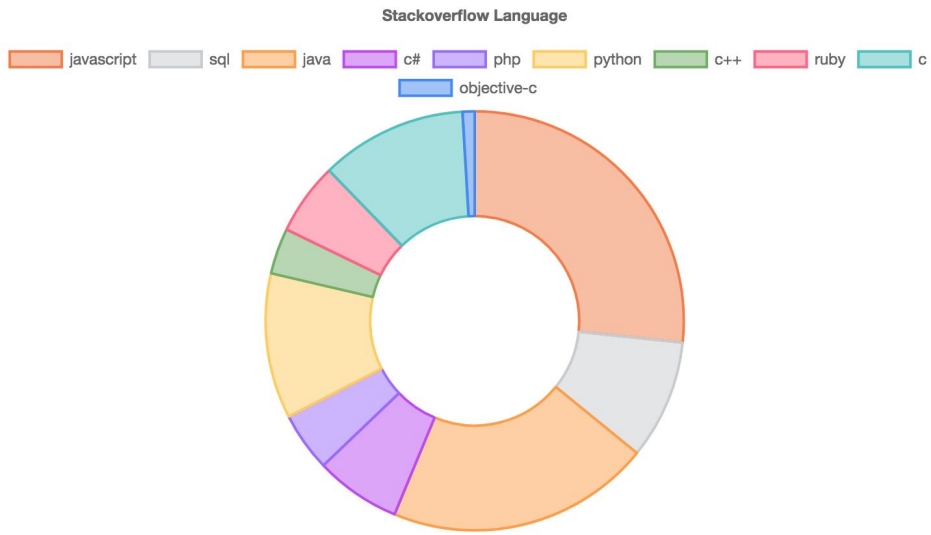


Fig. 2:

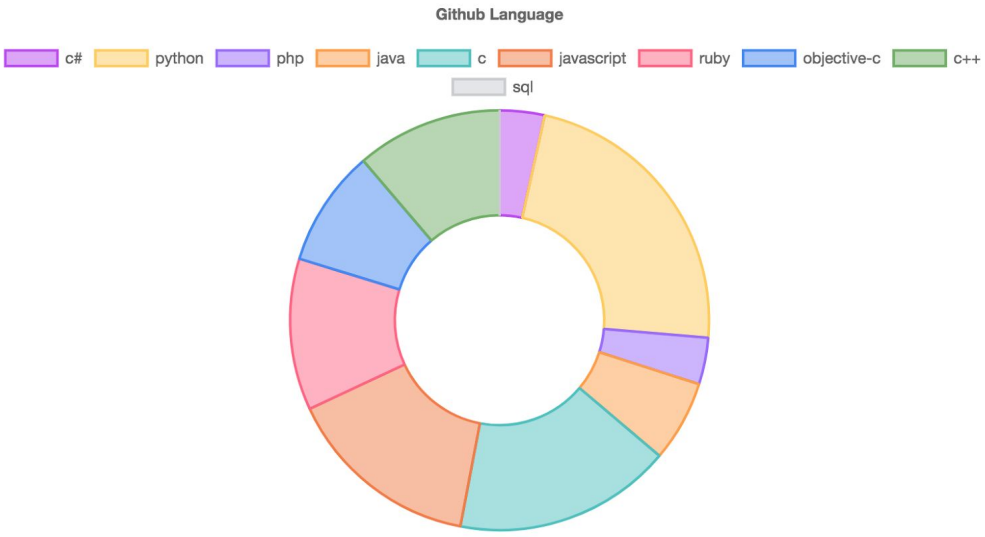


Fig. 3:

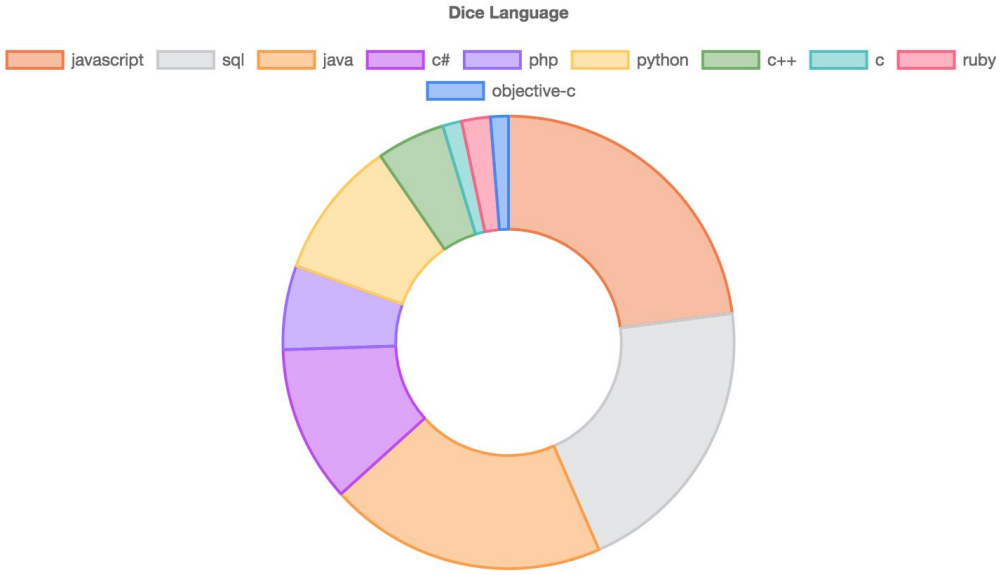

Dice Language
javascript
sql
java
c#
php
python
c++
c
ruby
objective-c

Fig. 4:

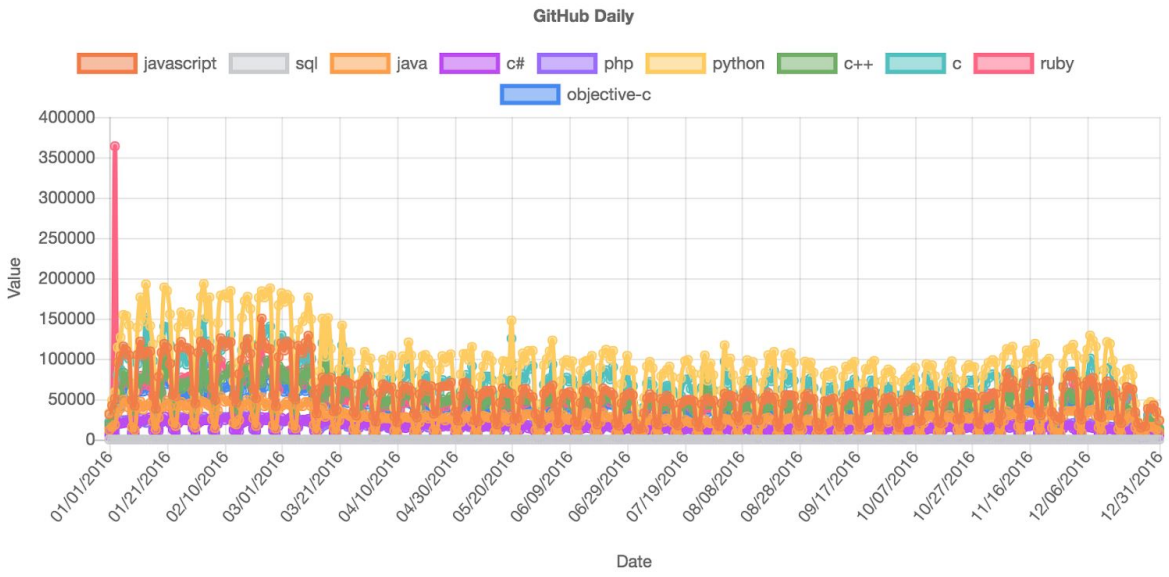
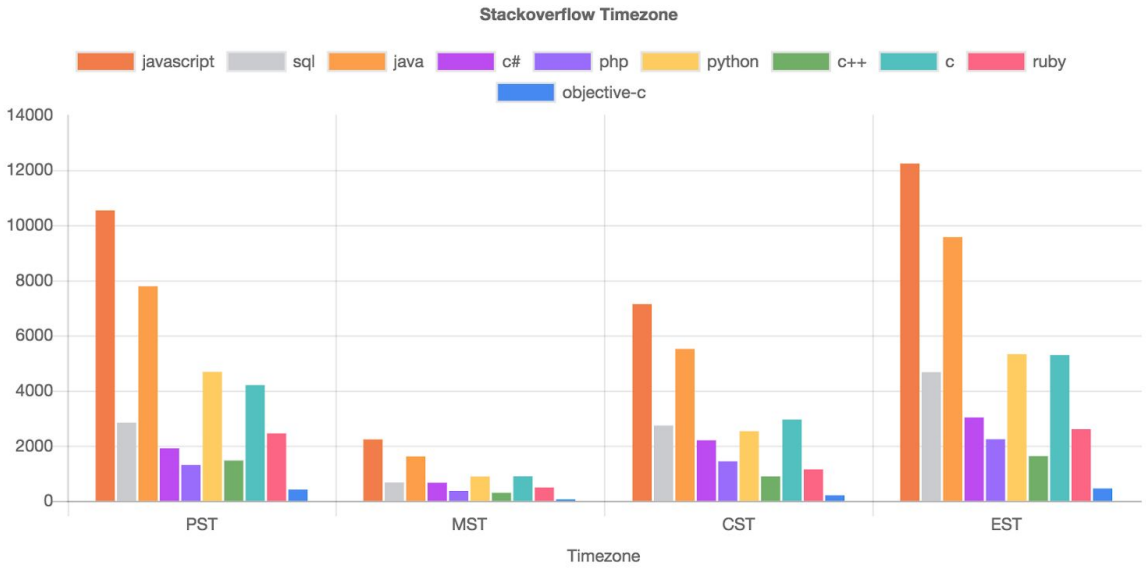

GitHub Daily
javascript
sql
java
c#
php
python
c++
c
ruby
objective-c
400000
350000
300000
250000
200000
150000
100000
50000
0
Value
01/01/2016
01/21/2016
02/10/2016
03/01/2016
03/21/2016
04/10/2016
04/30/2016
05/20/2016
06/09/2016
06/29/2016
07/19/2016
08/08/2016
08/28/2016
09/17/2016
10/07/2016
10/27/2016
11/16/2016
12/06/2016
12/31/2016
Date

Fig. 5:


Stackoverflow Timezone
javascript
sql
java
c#
php
python
c++
c
ruby
objective-c
14000
12000
10000
8000
6000
4000
2000
0
PST
MST
CST
EST
Timezone