GitHub REST API vs GHTorrent vs GitHub Archive: A Comparative Study

Thaís Mombach[1]**,Marco Tulio Valente**[1]

[1]Department of Computer Science – UFMG
Belo Horizonte – MG – Brazil

{thaismombach,mtov}@dcc.ufmg.br

Abstract. *GitHub is a popular platform for source code storage. This fact along with the API provided by the system contributes to make GitHub widely used on Software Engineering research. This paper presents and compares three systems that provide GitHub data: GitHub REST API, GitHub Archive and GHTorrent. For that, we rely on three queries where each system is used to collect the desired data. The goal of this paper is to provide a basic understanding about each system, their differences and in which case each one should be used. Our ultimate goal is to help researchers when choosing the best system to collect data required to the their research.*

1. Introduction

Nowadays, GitHub is the most popular platform for source code storage and sharing, with more than 87 million repositories and 30 million users.[1] GitHub is used by important Open Source Software (OSS) to store and share their code, and also to organize the contributions from developers around the world. This fact along with the friendly API that GitHub provides contribute to the platform being commonly used on Software Engineering research [Cosentino et al. 2017]. Particularly, GitHub provides an official REST API that allows access to information about projects through HTTP requests. Due to the large interest on GitHub for research, other platforms and datasets that collect data from the platform were create, such as GitHub Archive[2] and GHTorrent [Gousios and Spinellis 2012]. Essentially, these systems collect data from GitHub and make them available for further analysis.

GitHub REST API[3] is the official GitHub API that returns results in JSON format. This API provides data about repositories, users, issues, pull requests, among others. GitHub Archive[2] is a project created by Ilya Grigorik to collect GitHub data, so users can easily retrieve and use this data on their work. The system collects GitHub events and make them available in hourly archives that can be downloaded by HTTP client. Although GitHub Archive does not provide direct data about repositories and users, they can be obtained through Git events. The last project is GHTorrent [Gousios and Spinellis 2012] created by Georgios Gousios. This project provides an off-line mirror of GitHub data, which is retrieved by collecting events. The data is available in structured and unstructured formats. In the first case, the data is organized in tables on MySQL; in the latest one, the raw events data is stored in MongoDB.

[1] https://github.com/search

[2] https://www.gharchive.org/

[3] https://developer.github.com/v3/

The main purpose of this work is to compare these three projects in order to provide a basic understanding on how they work, when they should be used and what are the main differences among them. This paper is organized in six remaining sections. Section 2 includes a more detailed presentation of the three projects studied in this paper. Section 3 shows examples of queries that can be performed using these projects. In Section 4, we provide a comparative analysis of the three studied projects. Section 5 presents threat to validity. Section 6 presents related work, and Section 7 concludes the study.

2. Studied Systems

GitHub Rest API and GraphQL APIs provides access to data about hosted projects. However, this paper focus on the REST API, due to its popularity.

In order to retrieve data using the REST API, the requests should be made to *https://api.github.com/* specifying the desired data. For example, we can use *https://api.github.com/repo/owner/repo-name* to retrieve data about a repository called *owner/repo-name*. In fact, there is a large amount of data that can be retrieved using GitHub REST API, but the number of requests that can be performed is limited. This limit depends on the authentication provided by the user, including user and password, token, or key/secret. Currently, the rate limit for authenticated and unauthenticated requests is 5,000 and 60 requests per hour, respectively. However, under the REST API there is also a Search API with a different rate limit, 30 requests per minute for authenticated requests and 10 for unauthenticated.

GitHub Archive is a project created by Ilya Gregorik to store public events from GitHub, since February, 2011. Currently, the system extracts data from GitHub Events API, which provides more than 20 different types of events. The extracted data is available in hourly archives that can be downloaded using an HTTP client. Another option is using Google Big Query, where the data is stored in tables organized by year, month, and day. These tables have fields with basic information about the repository, the actor of the event, and a JSON object for the payload. Using Google Big Query, the data can be easily processed, since there is no need to download it; however, it is only possible to process 1 TB of data for free. In this paper, all examples are based on Google Big Query.

GHTorrent is a project created and maintained by Georgios Gousios. It provides an off-line mirror of GitHub data, which is available in two formats: structured data and raw data about events. The events are collected using GitHub Events API, and complemented with further requests to GitHub REST API to provide a structured representation. GHTorrent provides three ways to access its data: MySQL or MongoDB dumps, web service provided by GHTorrrent when accessing MySQL or MongoDB latest dumps, or Google Big Query when accessing from MySQL. In this paper, we focus on GHTorrent using MySQL. In this configuration, the tables represent projects, users, organizations, commits, pull requests, issues and others. Moreover, GHTorrent also provides geolocation data (city, state and country) of GitHub users, based on the location provided on their profile.

3. Comparative Queries

In this section, we provide examples of concrete queries using the studied systems.

3.1. Query #1: Top-1,000 Projects Number of Stars

GitHub REST API: The following URL returns a list of repositories sorted by number of stars in a descendant order.[4]

```
https://api.github.com/search/repositories?q=stars:1..*&sort=stars&order=desc
&page=1&per_page=100&access_token=xxxx
```

This URL has the following parts: (a) *https://api.github.com/* identifies requests to GitHub REST API; (b) *search* directs the request to the search API; (c) *repositories* defines that the search is for repositories; (d) *q=stars:1..** filters repositories according to the number of stars specified on a range; (e) *&sort=stars* defines by which field the result is sorted: stars, forks or update; (f) *&order=desc* specifies whether the result is in descendant or ascendant order; (g) *&page=1&per_page=100* specifies the desired page and the number of items per page (100 items, in this example).

GitHub Archive: Using this system, the following query returns a list of repositories sorted by number of stars in a descendant order.

```
1 SELECT repo.name, count(DISTINCT actor.id) AS num_stars
2 FROM (TABLE_QUERY([githubarchive:month], ’REGEXP_MATCH(table_id
    ,r”ˆ201\d0\d”)’))
3 WHERE type = ’WatchEvent’
4 GROUP BY repo.name
5 ORDER BY num_stars DESC
6 LIMIT 1000;
```

To understand this query, it is important to mention that GitHub Archive data is organized in tables by year, month, and by day; essentially, these tables store the events when they happened. There is no table for the current year, but there are tables for the current month and current day. In this way, the previous query relies on *month* tables (line 2) to search for *WatchEvents* (line 3), which are the events triggered when a user stars a repository. For each repository (line 4), the number of distinct authors of *WatchEvents* are counted to obtain the total number of stars (line 1). The final result is sorted in descendant order (line 5) and limited to 1,000 repositories.

GHTorrent: In order to obtain the top-1,000 projects by number of stars, *projects* and *watchers* tables are used to construct the following query.

[4] Stars are often used as proxy for the popularity of GitHub software [Borges et al. 2016b] and therefore frequently used when selecting systems for empirical studies in software engineering.

```
SELECT p.id, p.name, p.url, w.num_stars
FROM
(SELECT id, name, url FROM projects WHERE deleted = 0) p
INNER JOIN
(SELECT repo_id, COUNT(DISTINCT user_id) AS num_stars FROM
    watchers GROUP BY repo_id) w
ON p.id = w.repo_id
ORDER BY w.num_watchers DESC
LIMIT 1000;
```

As mentioned, this query uses two tables: *projects* (which stores information about GitHub repositories) and *watchers* (which store information about stars given by a user to a repository). On *watchers*, users and repositories are identified by an *id* (computed by GHTorrent) and each row represents a star. Therefore, to calculate the number of stars the repositories are grouped and the number of different users on the *watchers* table are counted. However, GHTorrent also stores information about deleted repositories; therefore the *projects* table is used to identify only active projects. The result of *watchers* and *projects* tables are combined, sorted in descendant order by number of stars and limited to the first 1,000 results.

3.2. Query #2: Number of Commits per Contributor of a Given Repository

GitHub REST API: This URL returns the number of commits per contributor of a repository called *owner/repo_name*.

```
http://api.github.com/repos/owner/repo_name/contributors?page=1&per_page=100
&access_token=xxxx
```

The previous URL has some parts that are different from the ones in Query #1, as follows: (1) *repos* directs the request to the repository endpoint of Github REST API; (2) *owner/repo_name* is the repository name, (3) *contributors* define that the desired data is about repository contributors. The answer is a list of contributors of *owner/repo_name* in descendant order by number of commits along with the total number of pages containing the answer.

GitHub Archive: The following query searches for *push events* triggered when a user pushes to *owner/repo* (line 3). For each contributor (line 4) the number of commits is computed adding the number of commits on each *PushEvent* (line 1).

```
1 SELECT actor.login, SUM(INTEGER(JSON_EXTRACT(payload, '$.size')
      )) AS num_commits
2 FROM (TABLE_QUERY([githubarchive:month], 'REGEXP_MATCH(table_id
      ,␣r"^201\d0\d")'))
3 WHERE type = 'PushEvent' AND repo.name = 'owner/repo'
4 GROUP BY actor.login;
```

GHTorrent: In this system, the *projects* table provides the *id* of *owner/repo*. This *id* is used to filter commits using *project_commits* table (line 2) and *commits* table (line 3). For each commit author (line 4) his/her number of commits is calculated (line 1). Finally, in order to obtain the contributor's name (instead of its *id*), the result is combined with the *user* table.

```
1 SELECT c.author_id, count(c.id) AS num_commits
2 FROM (SELECT * FROM project_commits WHERE project_id = (SELECT
     id FROM projects WHERE url = 'http://api.github.com/xxxx/
     xxxx') pc
3 INNER JOIN commits c ON pc.commit_id = c.id
4 GROUP BY c.author_id;
```

3.3. Query #3: Languages of Top-1,000 Projects

GitHub REST API: Essentially, for each repository returned by Query #1, the following URL returns its languages:

```
http://api.github.com/repos/owner/repository/languages&access_token=xxxx
```

This URL also refers to the repository API as in Query #2, but it differs on the *language* component, which requests the different languages used in a repository.

GitHub Archive: Using this system, it is not possible to answer this query because there is no event that lists all languages used in a repository.

GHTorrent: In the following query, the result from Query #1 (line 2) is combined with *project_language* table that contains the languages used on each project (line 3). Then, for each repository, the query returns its distinct languages (line 1).

```
1 SELECT DISTINCT ptop.url, plang.language
2 FROM (SELECT p.id, p.url FROM (SELECT id FROM projects WHERE
     deleted = 0) p INNER JOIN (SELECT repo_id, COUNT(DISTINCT
     user_id) AS num_stars FROM watchers GROUP BY repo_id) w ON p
     .id = w.repo_id ORDER BY w.num_stars DESC LIMIT 1000) ptop
3 INNER JOIN project_languages plang ON plang.project_id = ptop.
     id;
```

4. Analysis

The first aspect that must be taken into account is since when the data is required in a query. When using the official GitHub REST API, the information available covers all GitHub history, but this does not happen when using GitHub Archive and GHTorrent. GitHub Archive collects events since 02/12/2011. Therefore, even though it has a large amount of data, it does not contains the whole history of events from GitHub. The same happens to GHTorrent, which provides information in its MySQL database since 2012.

	GitHub REST API V3	GitHub Archive	GHTorrent
Data Since	-	02/12/2011	2012
Data From	-	GitHub REST API Events	GitHub REST API
Data Type	GitHub Data	GitHub Events	Structured GitHub Data
Data Update	Live	Hourly	Monthly

Table 1. Analysis of GitHub REST API V3, GitHub Archive, and GHTorrent.

In other words, GitHub Archive and GHTorrent aims to store part of the data available on GitHub and to make it easier for researchers to access this data without a rate limit. GitHub Archive stores data from events that occurs on GitHub organized in hourly files accordingly to the time they happened. GHTorrent provides structured data as MySQL dumps, which are monthly released. Therefore, an advantage of using GitHub Archive or GHTorrent is that it is possible to analyze how the data has evolved, and also to replicate a study with the same data. By contrast, as a disadvantage, the data is not always up to date, like on GitHub REST API. All these mentioned aspects are summarized on Table 1.

Particularly, in Query #1 it is more accurate to calculate the current top-1,000 repositories by number of stars using GitHub REST API, because the information is up to date and takes into account the whole GitHub history. By contrast, the result might not be the same for GHTorrent and GitHub Archive, since the data considered by such systems does not contain all history and therefore might not reflect the current status of GitHub. In addition, the query's complexity is lower for GitHub REST API than for GHTorrent and GitHub Achive. The main reason is that usually GHTorrent and GitHub Archive tables have to be combined to calculate the result.

For Query #2, depending on the selected repository the result might not be affected by the fact that GHTorrent and GitHub Archive do not provide up to date data from GitHub. The complexity of using GitHub Archive derives from the need to process all *month* tables available, if there is not time frame specified for the query. For GHTorrent, it is the need to retrieve the commits from all GitHub repositories. As a result, although the result might be almost the same for the three systems, using GHTorrent and GitHub Archive implies processing more data.

Regarding Query #3, we showed that it is not possible to use GitHub Archive to collect this information since there is no event that provides the required data (i.e., all programming languages used by a given repository). Using GitHub REST API, this query is expensive because for each project on top-1,000 list an additional query needs to be performed to retrieve all the languages used on such projects, on a total of 1,010 requests (10 to retrieve the top-1,000 list, assuming pages with the result of 100 project, plus 1,000 requests to obtain the languages of each project). Finally, when using GHTorrent, besides accessing the tables to calculate the top-1,000 projects, it is also necessary to access the table where the language for each project is stored.

5. Threats to Validity

In this paper, we compared three systems that can be used to retrieve data from GitHub. To the best of our knowledge, these are the principal systems to this purpose. However, we acknowledge that GitHub provides a new API, based on GraphQL technology, which can assume a key role in the future. Furthermore, we assessed the systems using only three queries. However, these queries were carefully selected based on our experience with the systems. Although limited in size, we claim these queries illustrate the pros and cons of using each system. Finally, our evaluation is qualitative by nature. In future work, we plan to conduct a quantitative evaluation of these systems, comparing for example runtime performance and, more importantly, precision and recall of the items returned by each studied system.

6. Related Studies

GitHub data has been widely used on Software Engineering Research; as a result, it is important to investigate the reliability of this data and the potential threats that might appear when mining GitHub. [Kalliamvakou et al. 2014] investigated this potential problems and proposed a set of recommendations to avoid the threats. [Cosentino et al. 2016] studied how researchers are mining GitHub, analyzing the empirical methods employed, used datasets and their limitations; regarding the datasets, the authors paper pointed out that GHTorrent is the most used dataset; however, GitHub API is also widely used.

A similar work to this paper presents a tutorial about how to analyze data from GitHub in order to avoid misleading results using GHTorrent as data source [Gousios and Spinellis 2017]. Another work investigates how GHTorrent dataset is used on researches and how much data is processed [Borges et al. 2016a]. However, to the best of our knowledge this is the first work that presents and compares the different sources of GitHub data.

7. Conclusion

The objective of this study was to provide a brief explanation and use scenarios for three solutions that can be used to mining GitHub data: GitHub REST API, GitHub Archive and GHTorrent. For each system was presented an explanation about their behavior, and available data. GitHub provides a REST API that provides through HTTP requests GitHub data as JSON objects. GitHub Archive collects GitHub events since 2011 and make them available in hourly files that can be downloaded for further analysis. The last one, GHTorrent, also access GitHub events to create dumps of raw and structured data, for MongoDB and MySQL databases, respectively. All this information are relevant when choosing a data source for a research, and it can impact on its result.

It was also presented three different comparative queries that were constructed using the presented systems. For each query it was discussed the positive and negative points of using this systems. The main goal of this paper was to provide an analysis of this different solutions that can be used to mining GitHub data system in order to help researches when choosing the one that best suits their research.

Based on our findings, we derive two practical recommendations to researchers:

- GitHub REST API is more appropriate when it is important to retrieve the most recent and up to date data (e.g., about number of stars).
- GHTorrent and GitHub Archive is more appropriate when it is important to retrieve historical data about projects (e.g., time series with number of stars per month).

As future work, we plan to work in four fronts: (a) include a comparison with GitHub GraphQL API; (b) extend our set of comparative queries; (c) evaluate the runtime performance of each system; (d) evaluate the precision and recall of GHTorrent and GitHub Archive, using the results provided by the official REST API as ground truth.

Acknowledgments

Our research is supported by CAPES, FAPEMIG, and CNPq.

References

Borges, H., Coelho, J., Carvalho, P., Fernandes, M., and Valente, M. T. (2016a). Como pesquisadores usam o dataset GHTorrent? In *5th Brazilian Workshop on Software Visualization, Evolution and Maintenance (VEM)*, pages 1–8.

Borges, H., Hora, A., and Valente, M. T. (2016b). Understanding the factors that impact the popularity of GitHub repositories. In *32nd IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 334–344.

Cosentino, V., Izquierdo, J. L. C., and Cabot, J. (2016). Findings from GitHub: methods, datasets and limitations. In *13th Working Conference on Mining Software Repositories (MSR)*, pages 137–141.

Cosentino, V., Izquierdo, J. L. C., and Cabot, J. (2017). A systematic mapping study of software development with GitHub. *IEEE Access*, 5:7173–7192.

Gousios, G. and Spinellis, D. (2012). GHTorrent: Github's data from a firehose. In *9th Working Conference on Mining Software Repositories (MSR)*, pages 12–21.

Gousios, G. and Spinellis, D. (2017). Mining software engineering data from GitHub. In *39th International Conference on Software Engineering Companion (ICSE)*, pages 501–502.

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., and Damian, D. (2014). The promises and perils of mining GitHub. In *11th Working Conference on Mining Software Repositories (MSR)*, pages 92–101.