ISBN 978-981-11-3671-9
Proceedings of 2017 the 7th International Workshop on Computer Science and Engineering (WCSE 2017)
Beijing, 25-27 June, 2017, pp. 144-149
doi: 10.18178/wcse.2017.06.025

Kraken: A Continuous Incremental Data Acquisition System for GitHub and Git Repositories

Lingbin Zeng [+], Gang Yin, Tao Wang, Yue Yu, Qiang Fan, Zhi-Xing Li, Jie Yu, and H. M. Wang

National Laboratory for Parallel and Distributed Processing, National University of Defense Technology, 410073, Changsha, China

Abstract. With the quick development of open source software, quantity of software is produced in the open source community (OSC) [1]. Lots of researches are launched to study the internal regular patterns of OSC [2], [3]. GitHub is one of the most famous open source community which owns thousands software projects. As a result, there are massive and abundant data of software development activities in GitHub. With the purpose to offer an accuracy and efficient dataset of GitHub, this paper proposes *Kraken* which is a continuous incremental data acquisition system for GitHub. *Kraken* contains three main modules which are independent with each other. *Kraken* gets the data of GitHub from two ways: git repositories and rest API. The final result shows that *Kraken* could extract the commits information of git repositories and get pull requests(PRs) and issues through rest API. The commits information contains the detail development history of software and the feedbacks and wisdom of software engineers are showed through PRs and issues.

Keywords: GitHub, open source software, data extraction, rest API

1. Introduction

With the development of open source system, there are quantities of software produced by software engineers from all over the world. As a result, there are lots of software for developers and users to choose. However, the quality of software is uneven in open source community. Searching the reasons why some of the open source software could be such successful and analysing the development activities are hot topics in the software engineering fields.

Analysing the software development activities need abundant and massive dataset. It will be convenient and useful with the detailed and comprehensive data. To achieve the purpose, this paper proposes an open source data acquisition system named *Kraken*. *Kraken* is focused on the data in GitHub which is the most famous and successful open source community all over the world. The traditional way to get data of GitHub is through GHTorrent[4]. However, data provided by GHTorrent is not timely and fully. If you want study some specific software projects and want to get the newest data, *Kraken* may be a good choice.

There are many kinds of data in the GitHub, such as commits, PRs, issues and so on. But we could divide the data into two types according the acquisition ways: git repository data and rest API data. Git repository includes history of all development activities, for example, the added codes, the minus codes, commits message and many other detail things in every git commits. Besides, GitHub mainly designed two ways to collect the wisdom of contributors, PRs and issues. PRs collect the commits and Issues collect the bugs as well as suggestions. What's more, GitHub has well designed rest API architecture. As a result, people could get the PRs, issues and other history information of the open source projects through rest API.

Kraken is designed as a professional but easy to reuse system to acquire the comprehensive data of GitHub. It is a robust, incremental, scalable, efficient data acquisition system. With the help of *Kraken*,

[+] Corresponding author.
E-mail address: zenglingbin12@163.com.

thousands of git repositories could be cloned and updated at regular time. Besides, PRs and issues of git repositories could be acquired in an incremental pattern.

The rest of this paper is organized as below: Section 2 is to introduce the related work. Section 3 describes the architecture and working principle of *Kraken*. Section 4 is to analyse the results and some studies with the acquired data. Section 5 is a conclusion of this paper and some future work will be mentioned.

2. Related Work

Lots of studies work in solving problems of acquiring data of open source community.

G Gousios et al proposed GHTorrent to create a scalable off line mirror of GitHub's event streams and persistent data, and offer it to the research community. GHTorrent supports massive data of GitHub through rest API [4]. However, data are acquired only through API.It can not extract the useful information through software repositories and not a light weight system.

D Patrick et al introduce *Mothur* aims to be a comprehensive software package that allows users to use a single piece of software to analyse community sequence data [5]. It could provide flexible and powerful software package for acquiring and analysing sequencing data.

Zhi-Xing Li et al creates a system named Octopus, a data acquisition system for open source software [1]. Octopus is a well-designed system which could collect two kinds of open source software (OSS) data sources: software production communities and software consumption communities. However, Octopus could not acquire the data of GitHub.

Most of these works do not work on GitHub. GHTorrent works on GitHub, but the data acquired by it is not entirely. GHTorrent only through the rest API to get the feedback of users and does not get the detail data from the software repositories. *Kraken* could get the data not only through API but also through software repository.

3. Architecture of *Kraken*

3.1. Overview

The architecture of *Kraken* is described in Figure 1. As we can see. *Kraken* is made up by three main modules: *Confirming Goal Repository*, *Software Repository Spider* and *Rest API spider*.

The three main modules contain many submodules. *Confirming Goal Repository* creates the goal repository and deliver to the other two modules. The main function of *Software Repository Spider* module aims to cloned remote repository and extract the data contained in repositories. *Rest API Spider* module aims to spider the contribution through extensive rest API supported by GitHub. The two modules are paralleled with each other and run in an efficient way.

3.2. Confirming Goal Repository

The function of this module is to get the aim projects. There are two ways to get the aim projects list. One is to catching the list of popular projects which have quantities development history and contributions. The other is to spider the projects input by people who use the *Kraken*.

As it is shown in Figure 1. First of all, *Spidering Repo* uses the extensive rest API to acquire the basic information of projects in GitHub[6]. The basic information includes star number, watch number, fork number. Star number is a number that shows the number of people who interested in the projects. Users will receive notifications of the project if he or she watches the project. Fork number is a digit showing the number of people who clone the repository of the project. With the support of these data, *Selecting Repo* will rank the projects with star number, watch number and fork number. Then, *Repo Generator* could acquire the list of ranked projects. Finally, if we want to spider the specific projects in some condition, we could input the name of projects into the *Repo Generator*.

3.3. Software Repository Spider

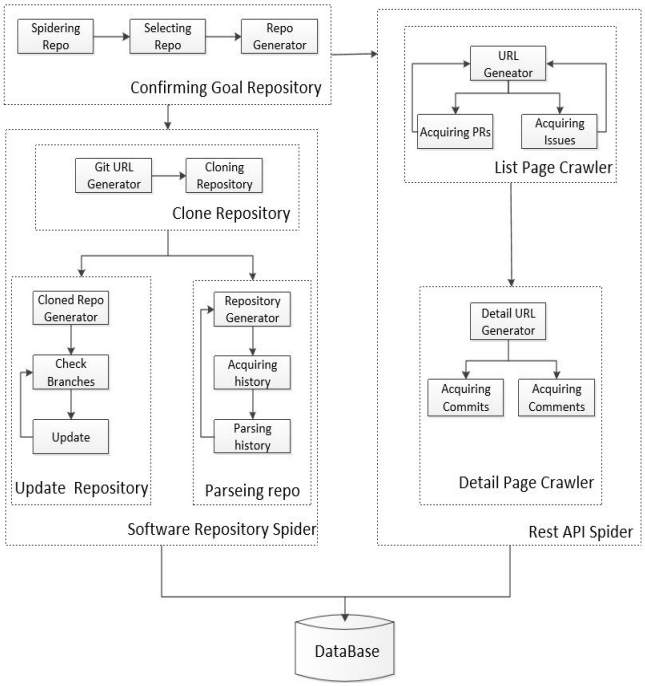

Fig. 1: The architecture of *Kraken.*

Software Repository Spider is the module that could acquire and analyse the software repository. First of all, *Kraken* will generate the git address of aim projects. Then, the system will clone the remote software repositories(RSR) and store them in local storage. Besides, repositories will be updated on regular time in order to keep the same with RSR. What's more, the system will acquire the logs of commits and parse the logs.

Clone Repository: The first step of *Clone Repository* is generating the git address of projects. GitHub uses and obey git protocol. As a result, *Kraken* could utilize the name of projects to structure the git address of projects. With the git address, *Kraken* could use *git clone*, one of the git commands, to clone the RSR [7].

Update Repository: *Kraken* already have cloned the RSR into the local storage. However, the RSR are continuous updating all the time. *Update Repository* is designed to make local software repository (LSR) and remote software repository stay the same. Software repository has many branches in remote, however, there is only one branch in LSR when the software repository cloned at the first time. *Check Branches* deals this problem. First, *Check Branches* uses *git branch -va* to show the branches in the remote software repository. Second, the git command *git checkout* is used to generate the branches. As a result, Branches could stay the same between LSR and RSR.

Update is the module that could update all the branches after *Check Branches* module make the branches stay the same between LSR and RSR. *Update* makes use of two git command to realize functions. First of all, *git pull* that is a command to keep current branch stay the same with remote branch. Besides, *git checkout* command is used to check into another branch that is not updated [8]. The loop will not stop until all the branches are updated.

Parsing Repository: Software repository contains many information like *commits logs*[9]. The function of *Parsing Repository* is getting and analyzing the information contained in software repository.

First of all, *Repository Generator* will select the list of all the LSR and put these duties of LSR into *redis* database. Then, *Repository Generator* will distribute works to *Acquiring history* in order to make all works parallel.

Acquiring history could incremental access the logs of commits. First of all, *Acquiring history* uses *git log -p* to catch the logs of commits and store it in the text document. There are many fields in a commit like *commit sha, Author, Date, Message, file name, minus codes* and *added codes*. The *commit sha* of latest commit will be stored in database. When LSR are updated, *Acquiring history* will fetch the *commit sha* stored in database and acquire the logs of commits which are later than that *commit sha*. As a result, an

incremental access to the logs of commits are realized. The commit information is stored in text document and has a complex construction.

Table 1: The Database Structure of Commits

Fields	meaning
user_name	user name of projects
repo_name	repository name of project
Sha	hash values of commits
add_count	count of added lines
minus_count	count of deleted lines
add_line	the specific added codes
minus_line	the specific deleted codes
old_path	old path of changed files
new_path	new path of change files

Parsing history is designed to parse and extract the key information of commits[10]. *Parsing history* designs a set of regular expressions to extract the information contained in commits logs. The information of every commit will be stored in database. As shown in table 1, *user_name* and *repo_name* is the name of software repository. Besides, *sha* is the previous seven characters of *commit sha* and *add_count*,*minus_count* is to count added lines and deleted lines respectively in that commit. What's more, *add_line*, *minus_line* are two fields that store the specific codes that added or deleted. The file path before changing is stored in *old_path* and *new_path* stores the file path after changing. The branch which contains the commit is stored in *branch* field.

3.4. Rest API Spider

GitHub supports well-designed rest API. The *Rest API Spide*r utilizes the API supported by GitHub to acquire the development information of projects. *Rest API Spider* is made up by *Detail Page Crawler* and *List Page Crawler*. The job of *List Page Crawler* is to catch the basic information of PRs and issues. The duty of *Detail Page Crawler* is to acquire the detail information of PRs, issues and so on.

List Page Crawler： *List Page Crawler* is designed to acquire list of PRs and issues through rest API. GitHub gives 60 times API visits per hour for every IP address while gives every registered user an *access token* which allow user visits GitHub through API 5000 times per hour. To improve the visits times, lots of users need to be registered.

The function of *URL Generator* is to generate the API URL of every project, such as, *https://api.github.com/repos/rails/rails/issues*.GitHub will return data in JSON after visiting the API URL. However, PRs and issues will return together as response when the URL of issues are visited. PRs and issues will be separated on the basis of the feature that PR own a special field named *pull_request*.

GitHub supports a mechanism to visit PRs and issues which are updated after specific time. The newest updated time of every project will be stored in the database and *List Page Crawler* will use it when the project needs to be visited again. According to it, *List Page Crawler* could catch all of the PRs and issues which are updated recently. Consequently, it will improve the efficiency and avoid to acquire information which have been got already[11].

Detail Page Crawler： The job of *Detail Page Crawler* is to acquire the detail information of PRs and issues liking comments and commits. *Detail URL Generator* will listen news from *List Page Crawler* all the time and create the URL based on it.

The information that comes from *List Page Crawler* include name of projects, number of issues, numbers of comments and commits. However, GitHub answers no more than 30 message per page. To deal with it, *Detail Page Crawler* will set page number according to the count of comments and commits of every issues or PR. The response from GitHub will be analyzed and stored in database.

4. Results and Discussion

In this section, the results and analysis will be presented and we will analyse the characteristics of *Kraken.*

4.1. Results and Analysis

The software repository we spider are according to the value of *star*. The *star* values of a project is an important index to show the amount of people who really interested in the project. Consequently, *Kraken* spider the top 40000 software repositories ranked in *star*. The 40000 software repositories take up 698 GB. Besides, *Kraken* updates all of the forty thousand software repositories once a week.

Table 2: Contributors and Commits

	maximum	minimum	Average
Commits number	566836	1	671
Contributor number	15002	1	24

What's more, we have done some analysis to the 400000 software repositories and find that there are 4 billion lines of codes in *master* branch while 63 billion lines of codes in all branches. As shown in table 2. The maximum commits per projects are 566836 while the minimum is 1. Besides, there are 15002 contributors in the maximum project while 1 contributor in some other projects.

The top 1140 projects are selected due to their amounts of PRs. *Kraken* spider PRs, issues, commits, comments and other development information of these projects. As shown in table 3, there are 1219183 PRs,951918 issues,3789607 comments of PRs and 3903661 comments of issues acquired until now. Besides, 3534000 commits of these 1140 are fetched. *Kraken* monitors all the projects and catches the newest information of projects when they are updated. All of these data are stored in MySQL database. *Kraken* acquiring these data friendly, we do use *access token* way to fetch data instead of IP rolling which will do huge pressure to GitHub.

Table 3: Already Acquire Data

name	Number
projects	1140
PRs	1219183
issues	951918
comments of PRs	3789607
comments of issues	3903661
commits of PRs	3534000

Kraken could give powerful supports for people who study GitHub or software engineering with the gigantic data and convenience functions.

4.2. System Operation Performance

Kraken is a scalable, robust and efficient system. We will explain these features in detail.

Scalable: *Kraken* will acquire all of the information that are updated recent time and never catch the information that already have been got. Obviously, the system proposed is incremental. Besides, we could add the number of *access token* to increase the frequency of accessing to GitHub. What's more, the system could acquire many other information of projects with changing some parameters.

Robust: There are many special conditions that will affect the operation of system, for example, network, electricity. To deal with this problem, *Kraken* designs sets of mechanisms. The system could restart to work based on works of last time and set a threshold time. If single visit beyond the threshold time, the system will pass the work and put it to the end of the message queue.

Efficient: *Kraken* is a distributed system which could wok in multiprocessing way. *Kraken* could spider 800000 messages per hour. Besides, the speed could enhance quickly as long as we get more *access token*. What's more, *Kraken* could parse about 60 million commits per hour in local endpoint. It saves time and could parse commits without Internet.

To sum up, *Kraken* is a good choice for people who study GitHub or major in big data analysis of software engineering.

5. Conclusion and Future Work

In this paper, we proposed a GitHub data acquisition named *Kraken*. We acquire 40000 software repositories and get software development information of 1140 projects through rest API. Besides, we do some basic analysis of these data. *Kraken* is a robust, incremental, scalable and efficient system for acquiring data of GitHub. However, there are some limitation could be enhanced. The information contained in software repositories is abundant. We could extract more useful and interesting information from it, so increasing the function of parsing software repositories will be a meaningful work in the future.

Besides, all programs of *Kraken* will put in *Trustie*, a famous open source community in China [12]. What's more, we intend to share our data in order to help people who need this data do empirical software engineering studies.

6. Acknowledgements

Thank you for people who helped me to finish my study, especially to Yue Yu and ZhiXing Li. This research is supported by the National Science Foundation of China with Grant No. 61502512, No.61432020, No. 61472430 and No.61532004.

7. References

[1] ZhiXing Li, Gang Yin, Tao Wang, YiAng Gan, Yun Zhan, Yang Zhang. Octopus: a data acquisition system for open source software communities [C]. In Proceedings of the 2016 International Conference on Software Engineering and Application.

[2] Yu Y, Yin G, Wang H, et al. Exploring the patterns of social behavior in github[C]//Proceedings of the 1st international workshop on crowd-based software development methods and technologies. ACM, 2014: 31-36.

[3] Dabbish L, Stuart C, Tsay J, et al. Social coding in GitHub: transparency and collaboration in an open software repository[C]//Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work. ACM, 2012: 1277-1286.

[4] Gousios G, Spinellis D. GHTorrent: GitHub's data from a firehose[C]//Proceedings of the 9th IEEE Working Conference on Mining Software Repositories. IEEE Press, 2012: 12-21.

[5] Schloss P D, Westcott S L, Ryabin T, et al. Introducing mothur: open-source, platform-independent, community-supported software for describing and comparing microbial communities[J]. Applied and environmental microbiology, 2009, 75(23): 7537-7541.

[6] Russell M A. Mining the Social Web: Data Mining Facebook, Twitter, LinkedIn, Google+, GitHub, and More[M]. " O'Reilly Media, Inc.", 2013.

[7] Blischak J D, Davenport E R, Wilson G. A quick introduction to version control with Git and GitHub[J]. PLoS Comput Biol, 2016, 12(1): e1004668.

[8] Loeliger J, McCullough M. Version Control with Git: Powerful tools and techniques for collaborative software development[M]. " O'Reilly Media, Inc.", 2012.

[9] Bird C, Rigby P C, Barr E T, et al. The promises and perils of mining git[C]//Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on. IEEE, 2009: 1-10.

[10] Nguyen N H, Song Z, Bates S T, et al. FUNGuild: an open annotation tool for parsing fungal community datasets by ecological guild[J]. Fungal Ecology, 2016, 20: 241-248.

[11] Zeinalipour-Yazti D, Dikaiakos M. Design and implementation of a distributed crawler and filtering processor[C]//International Workshop on Next Generation Information Technologies and Systems. Springer Berlin Heidelberg, 2002: 58-74.

[12] Wang H, Yin G, Li X, et al. TRUSTIE: A software development platform for crowdsourcing[M]//Crowdsourcing. Springer Berlin Heidelberg, 2015: 165-190.