Octopus: a data acquisition system for open source communities

Zhi-Xing Li, Gang Yin, Tao Wang, Yi-Ang Gan, Yun Zhan, Yang Zhang
National Laboratory for Parallel and Distributed Processing,
National University of Defense Technology, 410073, Changsha, China
E-mail: starleelzx@163.com, cloud_zhan@163.com

With the rapid development of open source software (OSS), a lot of reusable software resources have been produced in open source communities. This leads to spotty quality and high dispersion of OSS resources so that lots of research works have been conducted for effective location of reliable software resources. To provide convenient data source for such studies, this paper introduces *Octopus*, a data acquisition system for resource of open source software. Octopus is a robust, scalable and efficient system which consists of three main modules that have been well decoupled and can be updated at runtime with flexible configurations. The experiment results show that, *Octopus* can successfully collect two kinds of OSS data sources: software production communities and software consumption communities. The former produce structured software artifacts such as project profiles while the latter contain rich user feedback such as posts and blogs.

Keywords: Open Source Software; Data Acquisition; Web Crawler.

1. Introduction

With the continuous development of open source software, massive reusable software resources are available in open source software community [1]. Such great many of open source software projects provide rich choices for developers, but it also result in spotty quality and high dispersion [2]. Searching suitable software of high quality for reuse is a tough job with so many candidate resource available. To help user locate reliable software resources effectively and conveniently, more and more researches have been conducted [1][2][8].

Abundant and convenient dataset contribute a lot to such research. With this goal in mind, this paper proposes Octopus a data acquisition system for resource of open source software. The common way to build data acquisition system is to take advantage of web spider and crawl target data [8][9]. Commercial search engines are usually built on the base of general web spider, but for studies on a specific topic, focused spider is a better choice [10]. The main functionality of

Octopus to crawl data from Internet is achieved by the re-development of Webmagic (webmagic.io) an open source focused web spider.

There are two kinds of communities in open source software domain: software production communities and software consumption communities [1]. Structured software artifacts such as source code, documents and issues are produced in production communities like Openhub (www.openhub.net), SourceForge (sourceforge.net) and Oschina (www.oschina.net) while user feedback such as posts and blogs come from consumption communities like StackOverflow (stackoverflow.com) and CSDN (www.csdn.net). For scalable purpose, we design a uniform interface to process them equally.

Octopus is designed as a staged process and is configured with external file. It is a robust , scalable, efficient acquisition system for open source communities. For now, Octopus has covered 10 software consumption communities (SCC) and 6 software production communities (SPC).

The rest of paper is organized like this: Section 2 reviews briefly related work and Section 3 explains the architecture of Octopus. Section 4 discusses the result and characteristic of Octopus. Section 5 concludes this paper with future work.

2. Related Work

A lot of research work has been conducted on the problem of data acquisition for open source communities.

Gousios G et al. [3] proposed GHTorrent a scalable off line mirror of GitHub's event streams and persistent data. GHTorrent retrieves raw event contents, store them in MongoDB collections and replay events on top of an initial state. It has provided much service to the research focused on the data of GitHub.

FLOSSmole [4] is originally built by Howison et al. to collect data of open source software projects hosted on SourceForge. The collected data contains numbers of downloads, programming language and other metadata of a project. For now, FLOSSmole has included much more open source software repositories.

S Bajracharya present Sourcerer [5] to gather Java projects from SourceForge and Apache. It employs the structural information like reference in source code, the dependences between libraries and so on to achieve large scale source code indexing and searching.

BlackDuck [6] a massive open source knowledge base is built by Black Duck. It inspects thousands of open source sites and collects millions of open source projects. Upon this they also build a code search engine Koders and an open source software community OpenHub.

Most of these works focus on the software production communities, and some of them even restrict themselves to specific topic. Little work has been paid on the user feedback in consumption communities. This limit the application scope of their dataset. Different from them, Octopus try to include both kinds of communities as many as possible and provide abundant and convenient dataset for crowd wisdom mining in open source communities [1].

3. Architecture of Octopus

3.1. *Overview*

The architecture of Octopus is shown in Figure 1. The main process of Octopus is divided into three modules: *list page crawler*, *URL extractor*, and *detail page crawler*. These three parts are decoupled with database and synchronized with the help of *monitor* so that they can run in parallel. The underlying programming model of crawler is the re-development of WebMagic an open source web spider.

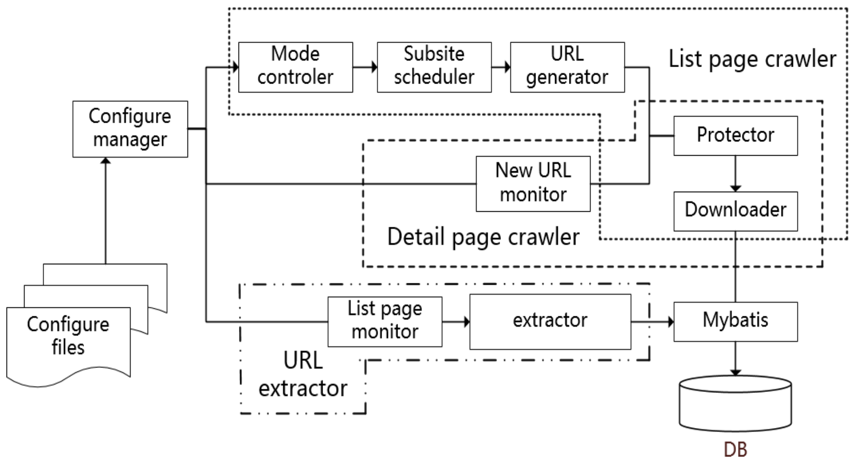
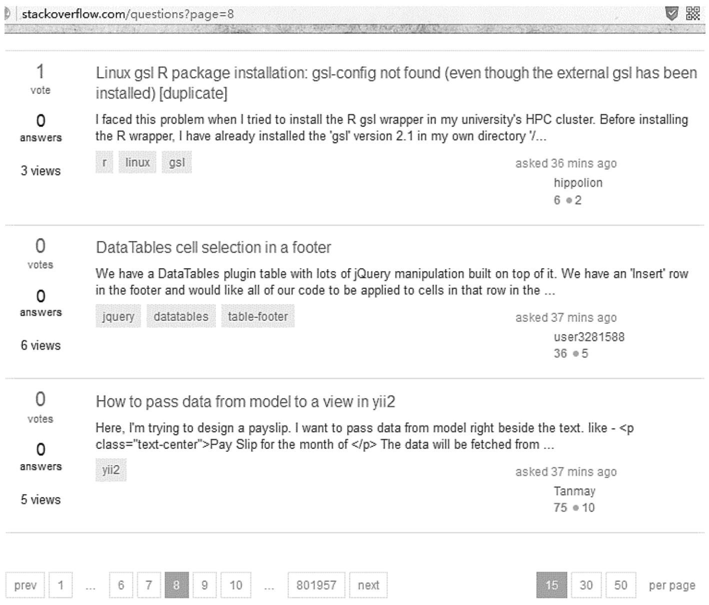


Fig. 1. The architecture of Octopus

The job of *Configure manager* is to load various configure files and influence several behaviors of Octopus. The rule to generate URL of list page, the rate to crawl some specific website, the rule to extract URL of detail page from list page, etc. are all configurable with external xml file. To make Octopus more friendly and tenacious, protector is designed for rate control and alternation of related parameter which will be taken into consideration unless it is absolutely necessary [11]. MyBatis (www.mybatis.org) a first class persistence framework is used to do the data persistence. In the following sections, the implementation of Octopus will be introduced in detail.

3.2. *List page crawler*

As discussed, Octopus is efficient because of its precise location of target resource. Unlike traditional focused web crawler filter valuable information from crawled data[9], Octopus only crawl needed resource. Like figure 2 shows, most sites relevant to open source software show their content in list pages.



stackoverflow.com/questions?page=8

1 vote | 0 answers | 3 views
Linux gsl R package installation: gsl-config not found (even though the external gsl has been installed) [duplicate]
I faced this problem when I tried to install the R gsl wrapper in my university's HPC cluster. Before installing the R wrapper, I have already installed the 'gsl' version 2.1 in my own directory '/...
r linux gsl
asked 36 mins ago hippolion 6 ● 2

0 votes | 0 answers | 6 views
DataTables cell selection in a footer
We have a DataTables plugin table with lots of jQuery manipulation built on top of it. We have an 'Insert' row in the footer and would like all of our code to be applied to cells in that row in the ...
jquery datatables table-footer
asked 37 mins ago user3281588 36 ● 5

0 votes | 0 answers | 5 views
How to pass data from model to a view in yii2
Here, I'm trying to design a payslip. I want to pass data from model right beside the text. like - <p class="text-center">Pay Slip for the month of </p> The data will be fetched from ...
yii2
asked 37 mins ago Tanmay 75 ● 10

prev 1 ... 6 7 8 9 10 ... 801957 next   15 30 50 per page

Fig. 2. List page of StackOverflow

The first step of Octopus is to crawl these list pages. The URLs of list page in a site have the same regular expression: *prefix_url:page_number:postfix_url*. The *prefix_url* and *postfix_url* is invariable for a specific site and *page_number* is an incremental number in the corresponding range. We call this type of regular expression as 3I *(invariable-incremental-invariable)* regular expression. In StackOverflow, these parameters is "stackoverflow.com/questions?page=", a number between (1,801956) for now, and a null string respectively. With this regularization, after start index and end index of *page_number* have been determined, it is convenient to construct all the available URLs of list page automatically by assembling the three parts which is read from external xml file specifying the rule of URLs of list page in a specific site.

To stay updated on new data produced anytime, we give Octopus the ability to be in different mode: mirror mode and increment mode. In the former mode Octopus crawl the history data in one site and in the later mode Octopus crawl the new data produced since last crawl time. For one site, Octopus start to collect data in mirror mode, and change into increment mode after all the history data has been collected. When it is in increment mode, Octopus visit the website periodically according to the update rate which is decided manually with a conservative estimate under actual situation. The general flow chart of *list page crawler* is illustrated in figure 3. They both call the common modules to assemble URL and download and store pages.

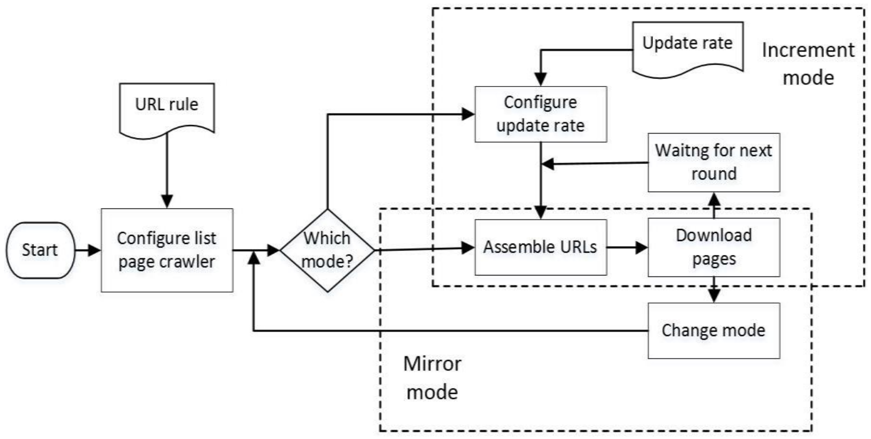


Fig. 3. The flow chart of list page crawler

For forum websites, there is no 3I regular expression of the URL for several subsections. To treat forum websites equally with other sites like StackOverflow, we model a website as a two-depth tree structure where the root node indicates the main site and leaf nodes indicate subsites. For those websites that have 3I regular expression of the URL, like StackOverflow and OpenHub, there is only one leaf node, and for those websites that don’t have, the number of leaf nodes is the number of their subsites. List page crawler will process each subsite in a round-robin schedule.

3.3. *Detail page crawler*

Like figure 4 shows, *URL extractor* listens to the result of *list page crawler* with the help of *monitor*. Once new data has come, *URL extractor* will process the html content of list page and extract the URLs contained in the page. To extract specific URL from a web page, lots of method has been proposed. We apply Jsoup [12] to do this job which is a powerful and easy-to-use HTML parsing tool. With Jsoup, CSSPath can be used to select any specific item in html file. The format of the grammar of CSSPath is like this : "*#postlist>h1>a*". The character "#" before a string indicates the string is an id of some item in a html and character ">" indicates the item before it is the parent of the item behind it. The CSSPath expression of an item in html can be many and varied, but the one that starts with some item specified with id and has a short length is a better one.

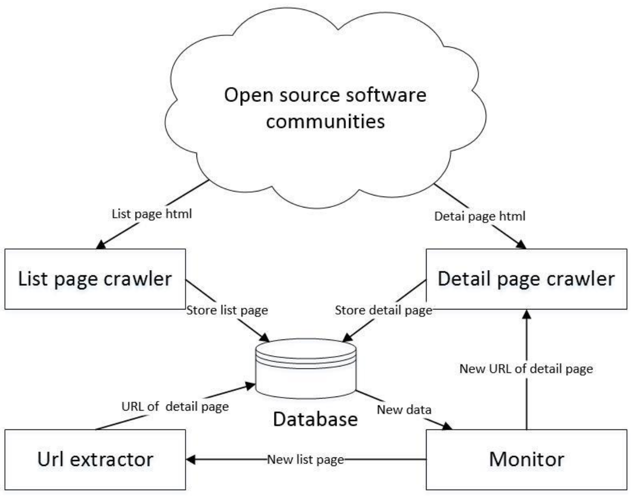


Fig. 4. Interaction between modules

The final step of Octopus is to crawl the detail pages which is our real goal. Like *URL extractor* listens to *list page crawler*, *detail page crawler* listens to *URL extractor*. Because the URLs of detail page have been extracted to local database, there is no tough job for detail page crawler. What it need to do is just download the detail page according to corresponding URL and persist it.

3.4. *Protector*

Some websites have made a set of criteria to resist web crawler[11]. Sometimes these criteria are so rigorous that crawler can download a very little data against them. There are many ways to make a web crawler smarter to overcome these restricts. Websites usually recognize a web crawler with its User Agent. In HTTP, the User Agent is often used for content negotiation, where the server end response with suitable content[12]. Web crawler need to identify itself in User Agent, such as Baidu's User Agent is "Mozilla/5.0 (compatible; Baiduspider/2.0; +http://www.baidu.com/search/spider.html)". For this reason, many designers build a User Agent pool which containing plenty of available User Agents. When their crawler issues a request it select a User Agent randomly from the pool to act as a famous web crawler or web browser. Another way to decrease the risk of being forbidden is to switch the IP of the server in which the crawler runs in. Websites also analyzes the IP traffic over an interval to decide whether a specific IP is requesting too much and restrict it if so. Dynamic IP switch can be achieved by IP proxy or dial-up connection. But we don't adopt these rude ways to improve the performance of Octopus. Reducing the crawling speed is the most friendly way.

In order to prevent disturbing the performance of target communities, we set Octopus at a friendly state to collect data. First, the crawling speed of Octopus is at a very low level. We set a range of crawl period in advance to constraint it. This range is a conservative estimate which can satisfy our need and don't bother the target communities at the same time. Every time, after Octopus finish downloading a page, it choose a value between that period range and sleep corresponding time to wait for next round. To better keeping Octopus running at an continuous state, we let it work intermittently. After Octopus working for a short interval, it will go into hibernation for a relatively long time. Both work interval and hibernation interval are random value generated in a specific range.

4. Results and discussion

In this Section, we present the result of the proposed system and the characteristic analysis of Octopus.

4.1. *Results*

Until now, Octopus has covered 10 software consumption communities including StackOverflow, CSDN, 51CTO, etc. and 6 software production communities including SourceForge, OpenHub, Apache, etc.. The data we have collected is shown in table 1.

For software production communities, data scale means the number of open source software project that one website have hosted, for software consumption communities data scale is the number of posts or blogs that one website has produced. In some communities, like CSDN, there are several individual sub-communities. The data scale is the total number of all the sub-communities.

Table 1. Collecting result of Octopus

Website	Type	Data scale	Website	Type	Data scale
StackOverflow	SCC	3,351,477	CodeProject	SCC	5,296
CSDN	SCC	1,063,453	LinuxTone	SCC	5,552
Cnblogs	SCC	170,086	OpehHub	SPC	660,992
51CTO	SCC	29,429	SourceForge	SPC	465,650
Dewen	SCC	12,311	Oschina	SPC	40,192
PHPChina	SCC	47,505	Freecode	SPC	40,716
ITEye blog	SCC	4,582	Gna	SPC	1,455
Slashdot	SCC	2,172	Apache	SPC	248

As we stated, Octopus collects data from open source communities friendly. It follows the robots exclusion protocol. Robots exclusion protocol, also known as simply robots.txt, is a standard used by websites to communicate with web crawlers to specify which resources of the website should not be processed or scanned. When Octopus crawls, it leaves out the disallowed resources and don’t violate their privacy. Because of friendly strategy, Octopus works in good condition and hasn’t been forbidden by any website.

4.2. *Characteristic analysis*

Octopus is a scalable, robust and efficient data acquisition system. In the following, we will explain these characteristic.

(1) Scalable: Octopus is configured with external xml file. The rules to generate URL of list page, extract URL of detail page form list page, and control of the work rate are all defined in configure file. This provides a plugin-based structure for adding new communities. The only effort to include another community is just to write a new configure file and specify all these rules.

(2) Robust: Open source communities grow at amazing speed and the structure and layout of their web page are often changing. It is difficult to identify whether an error is created in the process of extraction or other valuable information is included in changed pages. To handle this problem, Octopus is designed as a staged process including list page crawling, URL extract, and detail page crawling. These three modules are decoupled with database, and their outputs have all been stored into database. Under this design, we can redo any extraction when error has been found or extract extra valuable

information from pages stored in database without network access and crawl them again which is the most time-consuming operation.

(3) Efficient: Unlike traditional focused web crawler filter valuable information from crawled data, Octopus only crawl needed resource. Detail page is downloaded with the URL extracted from list page. There is no irrelevant data for Octopus to collect, so it saves much time on identify and filter.

All these characteristic of Octopus make it a perfect choice for data acquisition of open source communities.

5. Conclusion and future work

In this paper, we proposed Octopus a data acquisition system for the resource of open source software. With Octopus, we have collect 6 software production communities and 10 software consumption communities. The actual running state shows that Octopus is a robust, scalable, efficient system. However, there are some limitations need to be improved. As we have said, there are more and more open source communities, so it is a meaningful work in the future to discover unknown websites automatically that are relevant to open source software.

What's more, to help more empirical software engineering studies and satisfy their need on data acquisition of open source communities, we plan to offer both public dataset and online query interface.

Acknowledgments

Lots of thanks should be given to OSSEAN team members, especially to Xunhui Zhang and Li Li for their valuable help. This research is supported by the National High Technology Research and Development Program of China (Grant No.2012AA011201) and National Science Foundation of China (Grant No. 61432020 and 61472430).

References

1. Yin G, Wang T, Wang H, et al. OSSEAN: Mining Crowd Wisdom in Open Source Communities[C]//Service-Oriented System Engineering (SOSE), 2015 IEEE Symposium on. IEEE, 2015: 367-371.
2. Fan Q, Wang H, Yin G, et al. Ranking open source software based on crowd wisdom[C]//Software Engineering and Service Science (ICSESS), 2015 6th IEEE International Conference on. IEEE, 2015: 966-972

3. Gousios G, Spinellis D. GHTorrent: GitHub's data from a firehose[C]//Mining software repositories (msr), 2012 9th ieee working conference on. IEEE, 2012: 12-21.
4. Howison J, Conklin M, Crowston K. FLOSSmole: A collaborative repository for FLOSS research data and analyses[J]. International Journal of Information Technology and Web Engineering (IJITWE), 2006, 1(3): 17-26.
5. Linstead E, Bajracharya S, Ngo T, et al. Sourcerer: mining and searching internet-scale software repositories[J]. Data Mining and Knowledge Discovery, 2009, 18(2): 300-336.
6. Bagley C E, Lane D. Black Duck Software. Case Study[J]. Harvard Business School Publishing, 2006.
7. Vasilescu B, Filkov V, Serebrenik A. StackOverflow and GitHub: Associations between software development and crowdsourced knowledge[C]//Social Computing (SocialCom), 2013 International Conference on. IEEE, 2013: 188-195.
8. Heydon A, Najork M. Mercator: A scalable, extensible web crawler[J]. World Wide Web, 1999, 2(4): 219-229.
9. Thelwall M. A web crawler design for data mining[J]. Journal of Information Science, 2001, 27(5): 319-325.
10. S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific Web resource discovery. Computer Networks, 31(11-16):1623−1640, 1999.
11. Fox V, Camp A A, Ibel M, et al. System and method for enabling website owners to manage crawl rate in a website indexing system: U.S. Patent 7,599,920[P]. 2009-10-6.
12. Hedley J. jsoup: Java html parser[J]. 2010.
13. Fielding R, Gettys J, Mogul J, et al. Hypertext transfer protocol--HTTP/1.1[R]. 1999.