Analysis of Open Source Downloaders

Akshay Saxena, Shantanu Bhoyar, Ashish Pawar, Aditya Shah
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
{asaxena5, sbhoyar, akpawar, ahshah4}@ncsu.edu

***Abstract*—The Internet offers a broad spectrum of resources for different purposes. Being able to easily and securely access these resources is vital to the success of many projects. Downloading utilities provide many options for such accesses. But choosing the right tool for the job can be a difficult task. This report[1] compares three open-source command-line download utilities, namely, cURL, Aria2 and Wget. We use three metrics static code analysis, applicability of the software and community-health, to determine which tool to use in specific contexts. We then analyze the results to rank the tools in terms of each of those metrics and conclude with our findings.**

Keywords: Aria2, Coverity, cURL, GitHub, git-of-theseus, gitrob, SonarQube, StackExchange, Wget.

I. Overview

cURL[2], Wget[3] and Aria2[4] are command line utilities to download files and exchange data from/to a server. These tools support multiple protocols (like FTP, HTTP, HTTPS), provide various options (eg recursive download, support for proxies, UI) and can be ported to many platforms. For any open source software, it is vital to understand the community support behind it. An active and supportive community is more likely to produce better quality software with fewer bugs and support for newer features.

II. Approach

We developed homegrown scripts in addition to accessing open source tools to determine which of the three OSS was the best for our use case as well as to identify the healthiest development community. We can broadly categorize our approaches into the following categories:

A. Applicability

To compare the tools based on their overall applicability, we analyzed their performance as well as support to different platforms, protocols and features.

1) Performance: As the three applications are primarily designed to be downloading utilities, the bit-rate offered by the utility is a major indicator of their relative performance. To enable this comparison and eventual analysis, we implemented a Node.js script to poll a file located on a remote server every 5 seconds (non-overlapping) by executing the command line operations for each utility simultaneously. The results were then parsed to extract the bit-rate based on the operation's complete turnaround time. These results were stored in CSV files for each utility. Note, as the unit of bit-rate differed across each platform, we converted the results to bytes per second for uniformity. We wrote a Python script that consumes Matplotlib plotting library to plot the data from the CSV files to produce a graph comparing the tools w.r.t their downloading speed.

2) Platforms, protocols, features: We studied individual support for transport-layer protocols and accessibility on different Operating Systems using reference manuals. We compared various options and modes offered by each utility. Additionally, we considered their User Interface support and ease of use.

B. Static Analysis

For any software project, static analysis is critical as it does a thorough analysis of the source code and helps in detecting bugs and improving performance. It also helps in identifying potential vulnerabilities. To perform static analysis we used two open source tools namely Coverity[5] and SonarQube[6].

Coverity provided a descriptive analysis of vulnerabilities as well as flow of code from source to the exact location of the issue to enable root cause analysis. To begin with, a central analysis was performed followed by an automated process that checked for the entire source code. After successful configuration of the project, the "build & analyze" cycle was initiated for generating graphs like analysis metrics, outstanding defects per issue type etc.

SonarQube helped us uncover code smells, test coverage, and code duplication, in addition to code quality related issues. SonarQube helped in performing drill down operations on issues.

We also used an open source tool called Gitrob[7] to detect files that contained unwanted sensitive information.

C. Community Health

We considered community health metrics to get a sense of how open and responsive the community and maintainers of a utility are. These metrics would help potential users figure out which utility is more likely to be well-maintained in the foreseeable future, which one has a better track-record of adding new features and fixing bugs, and which one has

[1] https://github.com/ashish9995/Downloaders-Analysis
[2] https://curl.haxx.se/
[3] https://www.gnu.org/software/wget/
[4] https://aria2.github.io/
[5] https://scan.coverity.com/
[6] https://www.sonarqube.org
[7] https://github.com/michenriksen/gitrob

the best resolution time for queries on popular forums. The sources of our assertions are:

1) git-of-theseus: We used this open-source tool to obtain meta data for the Github repositories of the three utilities. It allowed us to gauge the maintainability of the utilities by calculating relative author contributions, the amount of code added over last 5 years and the amount of code that has survived in the repository over 5 years (survival rate).

We believe an OSS with more distributed and active development community will more likely have on-going or future developments. The amount of code added is an indicator of how active these authors have been recently (last 5 years as assumed a reasonable time frame). The survival rate gives an indication of the amount of scalable code written in those years, thereby implying forethought in design as lesser the changes, lesser would be the chance of buggy software.

2) GitHub API: We implemented a Nodejs script to fetch data using GitHub's REST APIs for calculating the time duration between an issue being logged to the repository, till the time the fix was merged or the issue was closed (whichever came earlier). This turn-around is a fair indicator of the contributors' responsiveness. Similarly, data was retrieved to measure the time taken to merge pull requests, enabling us to determine how frequently new features are likely to show up and the health of developer community. Both results were then charted using the D3.js JavaScript library.

3) Stackexchange API: The ease and speed of getting a query resolved is critical to new users adopting a software. StackExchange, being a popular resource for query resolution for software users facilitated in quantifying this. We queried its API to measure the time taken by user communities to respond to queries posted on StackExchange. This helped us understand the promptness of user community.

III. Results and Inference

A. Applicability

1) Performance: The results of the performance script shown in Fig.1 indicate that Wget has relatively higher bit-rate, followed by Aria2 and then cURL.

2) Platforms, protocols, features: An exhaustive list of supported protocols can be found at [4]. cURL supports widest range of transport-layer protocols. It makes use of its cross platform library *libcURL* to support variety of Unix based systems, OS/400, TPF. Wget supports recursive download option *-r* which is not available in cURL and Aria2. Only Aria2 supports parallel & segmented downloads by using multiple threads. It provides an option *-j* to specify number of concurrent connections. This allows Aria2 to be used for *BitTorrent* applications.

B. Static Analysis

1) Coverity: It generated graphs featuring defect density for each utility. Defect density is the number of defects (or flaws) found per thousand line of codes. It was found that cURL had the least defect density (almost negligible) followed by Aria2 (0.4) and then by Wget (0.54) as shown in Fig.2, 4 and 3. Coverity also generated a graph comparing outstanding defects over fixed defects over a fixed period of time as shown in Fig.5. It displays higher fixed defects vs outstanding defects in cURL followed by Wget and then by Aria2.

2) SonarQube & Gitrob: It was found that cURL contained more code smells overall. However, these results were formed on the test suite used for internal testing and may not be reflective of the production code quality. Aria2 contained the highest number of code duplication. The output from Gitrob showed us that none of the utilities contain any sensitive information and the tool only highlights mock data for tests.

C. Community Health

1) git-of-theseus: This tool gave three important results:

The first one was author contributions. We found that almost 90% of the current Aria2 code is contributed by a single author [5]. In contrast, Wget has a healthy contribution of six authors, with each contributing at least around ten percent of the current code [5]. cURL has 40% contributions by a single author 40% amongst three more, with other minor contributor [5].

The second result is lines of code added over a period of 5 years (2013 - 2018). In this period, Aria2 development has been nearly static with barely 35,000 lines(16% of current code) being added [5]. Wget has added around 20,000 lines(30% of current code) [5]. cURL has had a more active development with around 180000 lines(54% of current code) being added since 2013 [5].

The third result gives the average number of lines that have survived in a 5 year time frame, as shown in [5]. For Aria2, around 60% of code did not survive more than 5 years on average. Wget had 65% code removed in 5 year span. cURL on the other hand stands at 58%.

2) issue-turn-around-time: It takes around 1309.32 hours to fix an issue in Aria2 (Table I). The results show 1486.09 hours and 639.02 hours for Wget and cURL respectively (Table I)

3) pull-request-turn-around-time: It takes around 75.96 hours for a Pull Request on Github to be merged for Aria2 (Table I). In the case of Wget and cURL, this is more than 2000 hours and 467.34 hours respectively. The Wget anomaly is due to a PR being open from 2015 (Table I).

4) StackExchange response time: The average response time for answering an Aria2 query on StackExchange is roughly 71 minutes. For Wget, this time is 37 minutes while for cURL it is 43 minutes. All this data can be seen in Table III.

IV. Conclusion

cURL has had the least number of issues and existing defects were addressed most promptly but suffers from more code smells. On the applicability aspect, Wget had the highest bit-rate, while cURL outnumbers others in terms of protocols and portability. cURL turned out to be the winner when it came to community engagement, future prospect and development activity. Further, the appendix provides results on other metrics to help readers gain more insight about the analysis undertaken.

REFERENCES

[1] cURL source code Git repository
`https://github.com/cURL/cURL`
[2] Wget source code Git repository (mirror)
`https://github.com/mirror/Wget`
[3] Aria2 source code Git repository
`https://github.com/Aria2/Aria2`
[4] Utility comparison sheet
`https://goo.gl/2NFAvY`
[5] git-of-theseus analysis
`https://goo.gl/x7AMau`

V. APPENDIX

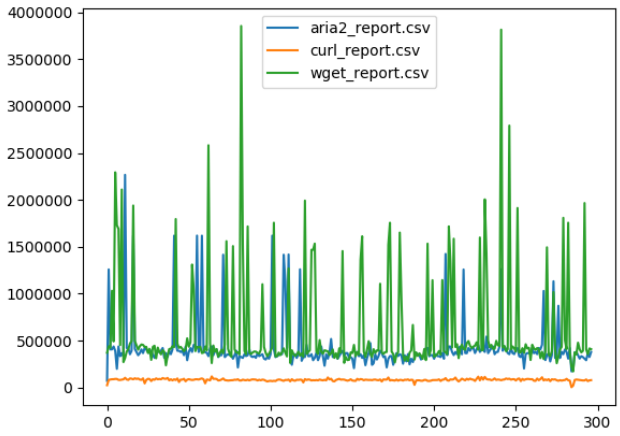

Fig. 1. Performance Analysis in Bitrate

Analysis Metrics per Components

Component Name	Pattern	Ignore	Line of Code	Defect density
libcurl	.*/lib/.*	No	81,488	0.00
tests	.*/tests/.*	Yes	0	N/A
curl tool	.*/src/.*	No	21,832	0.00
Other	.*	No	72,740	0.00

Fig. 2. Defect density per component for cURL

Analysis Metrics per Components

Component Name	Pattern	Ignore	Line of Code	Defect density
custom gcc	/usr/local/gcc.*	Yes	0	N/A
test	.*/test/.*	Yes	0	N/A
src	.*/src/.*	No	79,711	0.39
Other	.*	No	136,876	0.01

Fig. 3. Defect density per component for aria2

Analysis Metrics per Components

Component Name	Pattern	Ignore	Line of Code	Defect density
Gnulib	.*/lib/.*	Yes	34,823	N/A
Gnulib2	/lib/.*	Yes	0	N/A
System includes	/usr/include/.*	Yes	24,225	N/A
Other	.*	No	50,465	0.54

Fig. 4. Defect density per component for Wget

TABLE I
PULL REQUEST AND ISSUE TURNAROUND TIMES

Utility	Pull request(hours)	Issue(hours)
aria2	75.96	1309.32
curl	467.34	639.02
wget	2000	1486.09

TABLE II
SONARQUBE STATIC ANALYSIS RESULTS

Utility	curl	wget	aria2
Total	87	32	89
Bugs	19	12	41
Vulnerabilities	9	1	5
Code smells	53	19	43
Security hotspots	6	0	0

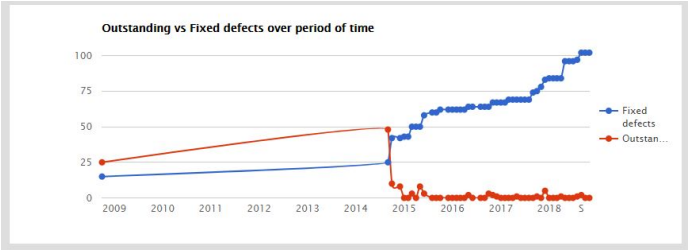

Fig. 5. Outstanding vs fixed defects for cURL

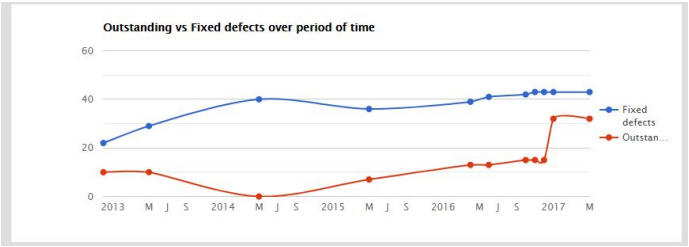

Fig. 6. Outstanding vs fixed defects for Aria2

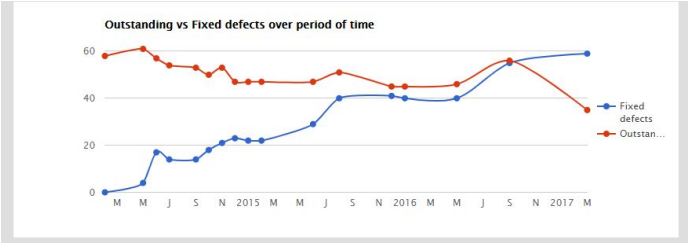

Fig. 7. Outstanding vs fixed defects for Wget

TABLE III
AVG RESPONSE TIME ON STACK EXCHANGE FORUMS

Tag Name	Average (minutes)
aria2	71
curl	43
wget	38