Finding File Clones in FreeBSD Ports Collection

Yusuke Sasaki* Tetsuo Yamamoto† Yasuhiro Hayase* Katsuro Inoue*
*Graduate School of Information Science and Technology, Osaka University
1-5 Yamadaoka, Suita, Osaka, 565-0871, Japan
Email: {ysk-ssk,y-hayase,inoue}@ist.osaka-u.ac.jp
†College of Information Science and Engineering, Ritsumeikan University
1-1-1 Nojihigashi, Kusatsu, Shiga, 525-8577, Japan
Email: tetsuo@cs.ritsumei.ac.jp

***Abstract*—In Open Source System (OSS) development, software components are often imported and reused; for this reason we might expect that files are copied in multiple projects (file clones). In this paper, we propose a file clone detection tool called `FCFinder` and show the analysis performed with it on the FreeBSD Ports Collection, a large OSS project collection. We found many file clones among similar or related projects, which are systematically introduced from base projects.**

***Keywords*-File clone; hashing; power law; open source system.**

I. Introduction

These days, researches on code clones are very active, and various code clone analysis applications have been explored including bug finding, refactoring, code evolution trace, and plagiarism detection[1].

In the past, our laboratory performed an experiment on large-scale code clone analysis using the distributed code clone detection system D-CCFinder[2]. One of the targets of the experiment was the FreeBSD Ports Collection, which is the set of all the applications of FreeBSD and contains about 10 GB source code.

In this experiment, we found many cases in which files of a project are simply copied into another project without any (or just slight) modifications in comments or headers. We call these copied files *file clones*. The characteristics of the file clones found in a large collection of source code have not been explored yet.

File clones can be detected using tools such as CCFinder[3]. However, even with a fast and distributed environment, code clone detection is still a very resource intensive process requiring precise string matching.

In this paper, we propose an efficient file clone detection method using comparison of the tokenized files' hash values, and show its implementation as a tool named `FCFinder` (File Clone Finder). Using `FCFinder`, we have analyzed the FreeBSD's Ports Collection and found several interesting details on the nature of the file clones in that collection. Our findings are summarized in such a way that the distributions of the file size and file clone set size almost follow the so-called "*power law*", with several anomalous points which are made by systematical file duplication among projects.

The contributions of this paper are as follows.

1) A hash-based efficient file clone detection method is proposed and implemented as a tool (`FCFinder`).
2) A large collection of Open Source Systems (OSS) is analyzed and visualized from a view point of file clone.

In Section II, we will show the related works. In Section III, the approach of `FCFinder` will be described. Section IV will discuss the application result of `FCFinder` to the FreeBSD Ports Collection. Section V will conclude our discussion with future works.

II. Background

There are various code clone detection methods proposed as research trials or as practical implemented tools[3], [4] but their scalability is mostly limited by the amount of available memory. Livieri *et al.* have overcome this limitation by partitioning the source code into small pieces and detecting the code clones between each pair of pieces[2], [5].

Mayrand *et al.* proposed a method for code clone detection at function level using 21 metrics[6]. Successively, they extended their approach to object oriented programs to efficiently find similar classes[7].

To the best of our knowledge, there is no research aiming to find file clones in an efficient and scalable way; also, the distribution of file clones in a large source code collection is unknown.

III. `FCFinder`: File Clone Detection Tool

`FCFinder` receives in input a collection of source files and reports the file clones in it. Figure 1 shows the overall `FCFinder`'s process.

1) *Indexing*: for each `.h` and `.c` file in the input collection, `FCFinder` extract and store several information (e.g., file sizes and project names).
2) *Tokenization*: each file is parsed into a token sequence; comments, redundant white spaces, line breaks, and carriage returns are removed. During this step, files shorter than 15 tokens are removed from the input list.

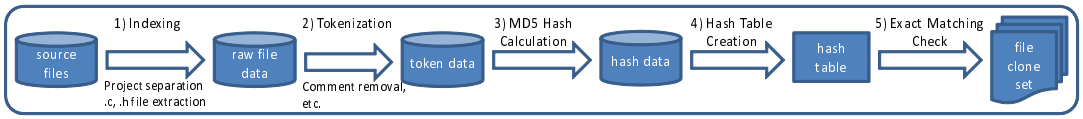


Figure 1. Overview of `FCFinder`

Table I
MINING TARGET CHARACTERISTICS

Number of files (.c, .h)	1,355,098
Number of projects	7,209
Total size	11.2 GBytes

3) *MD5 Hash Calculation*: an MD5 hash value[8] is computed for each tokenized file.
4) *Hash Table Creation*: a hash table is created with the computed hash values as keys and the corresponding file paths as values; files with the same MD5 value are grouped together.
5) *Exact Matching Check*: for each key in the table, the token sequences of the corresponding files are compared using a simple exact matching; matching files are reported as a *file clone set*; the files in a file clone set have exactly same token sequences.

IV. EXPERIMENT

In this section, we describe the experiment performed applying `FCFinder` to the FreeBSD Ports Collection (hereinafter referred as *target*), a source code collection used in one of our previous experiments[2]. These files were downloaded on Sep. 6th in 2007.The target's characteristics are shown in Table I. The target is divided into 45 categories (database, multimedia, etc.) and each category consists of multiple projects (PHP5, emacs, etc.); there are 7,209 projects in the target. As described in III, we used only `.c` and `.h` files in our experiment.

The experiment was performed on a 64 bits FreeBSD workstation equipped with 2.5GHz CPU and 16 gigabytes of main memory. During the experiment, the maximum memory usage was four gigabytes.

A. Preliminary Analysis of the Target

We have analyzed the target in order to expose various characteristics. Figure 2 shows the size distribution of the files in the target; the *x*-axis displays the file size in 10 kilobytes intervals; the *y*-axis displays the number of files whose size falls into each interval; both axes use a logarithmic scale.

Figure 2 shows that the number of files decreases with the increasing of the file size. The relationship is almost linear and it is a case of so-called *power law*[9], [10] characteristic of file size distribution. We did a linear regression analysis

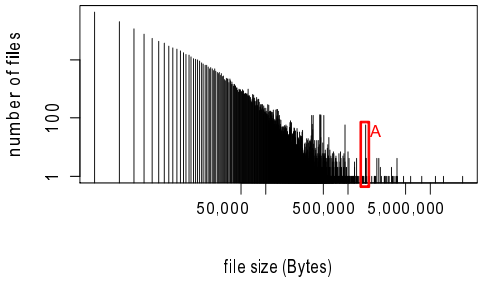


Figure 2. File size distribution of the target

for figure 2, and its coefficient of determination adjusted for the degrees of freedom R^{*2} becomes about 0.9024.

Area A presents some interesting findings. The area is an anomaly and does not fit the general power law profile of the file size distribution. We performed a detailed analysis of this area finding 59 file clones of some PHP5's files related to the time zone management; these files have a size of about 1.6 megabytes, and their clones increased the frequency of the corresponding size interval.

Figure 3 shows the distribution of the number of `.c` and `.h` files in each project; the *x*-axis represents the number of files inside a project; the *y*-axis shows the number of projects; both axes use a logarithmic scale.

Area B highlights 59 projects that have 1,316 files and 61 projects that have 1,070 files which are outliers with respect to the power law distribution. We found that these projects have many file clones of files in the PHP4 and PHP5 projects.

Similarly, area C highlights 14 projects, each one having 6,506 files of cross compilers for RTEMS (Real-Time Executive for Multiprocessor Systems). A part of the files in these projects are customized depending on their target architectures.

B. File Clone Analysis of the Target

We have extracted the file clone information for the target using `FCFinder`.

The number of elements in a file clone set is called "*population*" of the file clone set.

Figure 4 shows the distribution of the population for each file clone set in the target; the *x*-axis displays the population

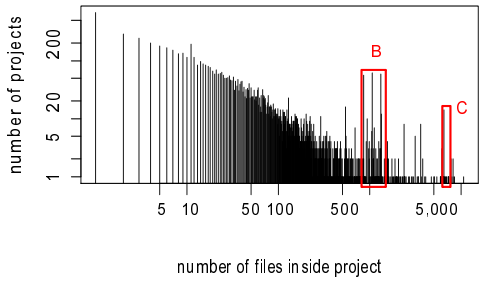


Figure 3. Distribution of the number of files in project

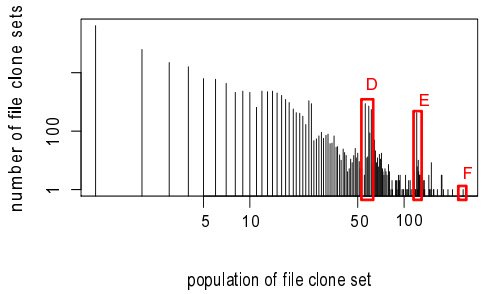


Figure 4. Distribution of population of file clone set

of file clone set; the *y*-axis displays the number of the file clone sets; both axes use a logarithmic scale.

In the target we found 915,409 file clones, with a total size of 7,6 Gigabytes; about 68% of the files in the target are file clones. The reason why the target files have large numbers of file clones is that large numbers of projects in the target use many common files among them.

The general tendency is that small population sets are more frequent than the large population sets. And the coefficient of determination R^{*2} for figure 4 with a linear regression analysis becomes about 0.8508.There are, however, several exceptions.

Area D highlights the file clone sets containing the files corresponding to area A in Figure 2.

Area E marks the file clone sets of 419 different files belonging to the PHP4 project or PHP5 project; each of these files has 120 copies. Area F corresponds to PHP project's files duplicated 240 times.

Figure 5 presents the distribution of the file clones in each project. The *x*-axis displays the number of file clones in a project; intra-project file clones are ignored. The *y*-axis displays the number of projects with such amount of file clones. The average number of file clones per project is 126.

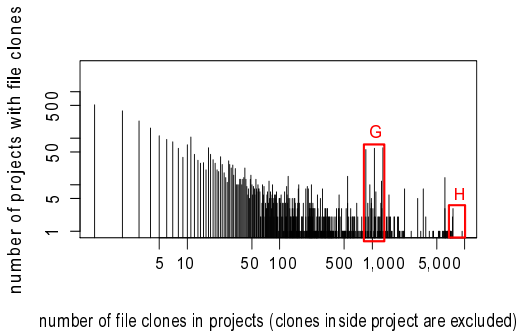


Figure 5. Distribution of file clones in project

This graph shows that the number of projects with a specified amount of file clones decreases as the number of file clones increases. The coefficient of determination R^{*2} for figure 4 with a linear regression analysis becomes about 0.8263.

Area G marks PHP related projects corresponding to area B in Figure 3 and area D in Figure 4.

Figure 6 shows some of the relations between the projects corresponding to area H. Each node represents a project and each edge represents the number of file clones between two projects.

Figure 6 (a) shows the case of GNU Compiler Collection projects: gcc41, gcc41-withgcjawt (java), gcc42, and gfortran. Three of those share 7,764 file clones.

Figure 6 (b) shows the case of OpenOffice.org where three related projects share between 7,440 and 8,101 files.

Figure 6 (c) shows that Thunderbird, Firefox, Firefox-devel, Seamonkey (an integrated network tool), and XULRunner (application development kit) share about 6,000 files; these are different projects sharing the same code base.

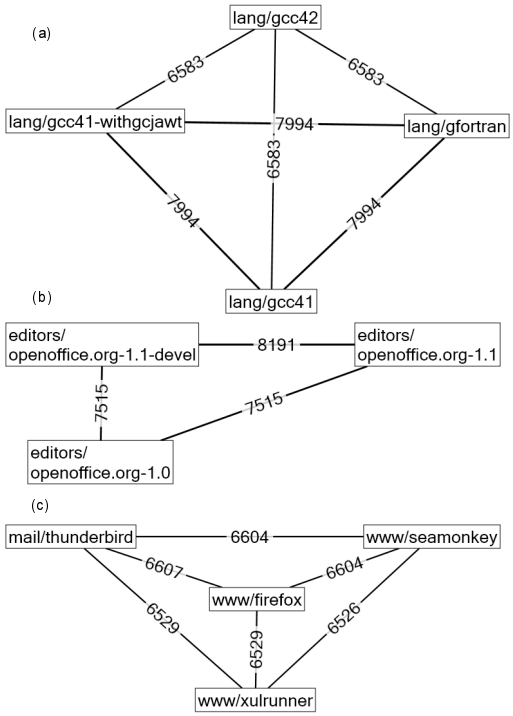
C. Evaluation of `FCFinder` output

We have randomly chosen 50 file clone sets from the analysis result of the target, and performed a manual investigation. We have found that those file clones correspond to files that are syntactically identical after they have been tokenized.

We have also randomly chosen 83 file clone pairs from the CCFinder result for the same target. We verified that all of those file pairs have been detected by `FCFinder`. Both precision and recall of `FCFinder` are 100% with respect to the results of CCFinder; however, we need to extend the number of samples to strengthen this evaluation.

D. Analysis Time

Table II shows the running time for the different stages of our experiment. In our previous experiment using 80 PC cluster machines for the same target analysis, it took about 51 hours to complete[2]. Assuming a linear relationship between the number of computers used and the running time,



- Nodes show the projects.
- Edges between projects show the number of file clones between two projects.

Figure 6. Project relations with file clones

Table II
ANALYSIS TIME

Step	Time
Indexing	9.42h
Tokenization	
MD5 Hash Calculation	1.13h
Hash Table Creation	
Exact Matching Check	6.61 h
Total	17.16 h

the same analysis performed with a single computer will require about 40 days. In the experiment with `FCFinder`, it took only 17.16 hours with a single workstation, as shown in Table II. This is due to the difference of analysis objectives such that D-CCFinder computes all the code clones including code fragments, while `FCFinder` finds only file clones. This reduced the computation complexity, and allowed a light-weight implementation using existing hashing and database tools. Further improvement of the `FCFinder`'s performance will require intensive usage of the cache of the database with careful tuning.

V. CONCLUSION

We have implemented an efficient file clone detector called `FCFinder`, and applied it to the FreeBSD Ports Collection. The experiment allowed us to find several interesting characteristics of the file sizes and file clones distributions. Our findings have been summarized in such a way that the distributions of the file size, file clone set population, and project's file clones are shown to follow the power law; anomalies in the distribution have been proven to be caused by systematic duplication of file among projects.

File clone analysis is very important in order to know the relationships between different software projects. Through file clones, we can investigate the evolution of projects and migration of legacy files.

As a future work, we plan to improve the file tokenization process to allow for small differences in the source code's syntax, such as differences in identifiers and constants naming.

ACKNOWLEDGMENT

We are grateful to Dr. Simone Livieri and Dr. Makoto Ichii for their useful comments.This work has been partially supported by Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research (A) (No.21240002), and also by MEXT Stage Project, the Development of Next Generation IT Infrastructure.

REFERENCES

[1] *3rd International Workshop on Software Clones*, Kaiserslautern, Germany, 2009. [Online]. Available: http://www.informatik.uni-bremen.de/st/IWSC/

[2] S. Livieri, Y. Higo, M. Matushita, and K. Inoue, "Very-large scale code clone analysis and visualization of open source programs using distributed CCFinder: D-CCFinder," in *29th IEEE International Conference on Software Engineering*, Minneapolis, MN, 2007, pp. 106–115.

[3] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A multilinguistic token-based code clone detection system for large scale source code," *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 654–670, 2002.

[4] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, "Comparison and evaluation of clone detection tools," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 577–591, 2007.

[5] S. Livieri, Y. Higo, M. Matsushita, and K. Inoue, "Analysis of the linux kernel evolution using code clone coverage," in *4th International Workshop on Mining Software Repositories*, Minneapolis, MN, 2007, pp. 22.1–4.

[6] J. Mayrand, C. Leblanc, and E. M. Merlo, "Experiment on the automatic detection of function clones in a software system using metrics," in *12th IEEE International Conference on Software Maintenance*, Monterey, CA, 1996, pp. 244–253.

[7] E. Merlo, G. Antoniol, M. D. Penta, and V. F. Rollo, "Linear complexity object-oriented similarity for clone detection and software evolution analyses," in *20th IEEE International Conference on Software Maintenance*, Illinois, CA, 2004, pp. 412–416.

[8] R. Rivest, "The MD5 Message-Digest Algorithm," RFC 1321 (Informational), Internet Engineering Task Force, Apr. 1992. [Online]. Available: http://www.ietf.org/rfc/rfc1321.txt

[9] M. Mitzenmacher, "A brief history of generative models for power law and lognormal distributions," *Internet Mathematics*, vol. 1, no. 2, pp. 226–251, 2003.

[10] G. Concas, M. Marchesi, S. Pinna, and N. Serra, "Power-laws in a large object-oriented software system," *IEEE Transactions on Software Engineering*, vol. 33, pp. 687–708, 2007.