Toward Efficient Code Clone Detection on Grid Environment

Yuki Manabe† Yoshiki Higo† Katsuro Inoue†
†Graduate School of Information Science and Technology, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan
{y-manabe, higo, inoue}@ist.osaka-u.ac.jp

Abstract

Code clone detection technique originally was developed for investigating duplicated code in a single software system or between two or three ones. If it can be applied to a large amount set of software systems, we should identify useful duplicate in it. This paper describes how we are going to scale up code clone detection technique for handling many software systems.

1 Introduction and Motivation

Recently, code clone detection technique attracts much attention. A code clone is a code fragment having code fragments identical or similar to it in the source code. It is widely accepted that automated code clone detection can help software development and maintenance. For example, in the debugging process, code clone detection prevents us from overlooking some of code fragments simultaneously.

In offshore developments, different teams tend to create similar libraries independently because of the lack of inter-team communication. For efficient offshore development, developers in different teams should share these libraries. Duplicated functions (Code clones) between existing software systems will be required by software systems under development or ones developed in the future. In other words, Code clones between existing software systems should be useful libraries for future developments.

However, there is a big problem in applying code clone detection technique to a large set of software systems; the scalability of code clone detection is not enough. Since, code clone detection technique was originally developed for investigating duplicated code in a single software system, or catching plagiarism between two or three software systems. We need to handle hundreds of software systems (billions lines of code) all together.

In order to satisfy this requirement, we are trying to applying existing code clone detection technique in grid environment. We have already implemented D-CCFinder, which is a code clone detection system in a distributed environment [4], and have conducted case studies on a large set of software systems [3]. But, we believe that grid-based CCFinder can achieve further usability, portability, and scalability.

2 Code Clone Detection on Grid Environment

This section describes our prototype of a code clone detection system on grid environment. Our system comprises a single master node and many slave nodes working on OurGrid [1], which is one of the popular grid middleware systems. Fig 1 shows the relation between the master node and slave nodes.

The following describes the code clone detection process of our system. The numbers in the process boxes of Fig 1 correspond to the following items.

1. The master node receives target source files, and divides them into groups that include a specified number of the source files. Fig 2 illustrates the relation between source files, software and groups.
2. The master node sends a pair of the groups to an idle slave node.
3. A slave node detects code clones from the assigned pair using CCFinder [2].
4. The slave node returns the detection result to the master node.
5. After detecting code clones all the pair of the groups, the master node merges the detection results as a single detection result.

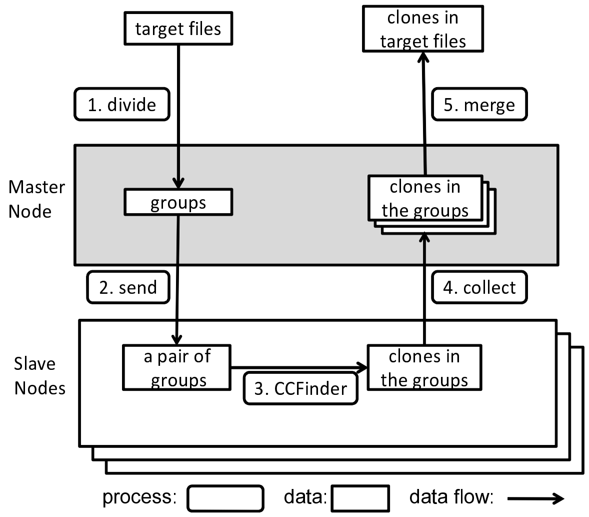

Figure 1. Relation between the master node and slave nodes

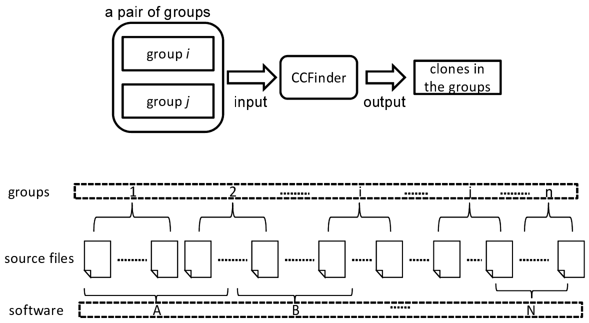

Figure 2. Relation between source files, software and groups

In order to detect code clones in a large file set quickly, we need a method to merge slave node's detection results efficiently.

3 Applications

At present, we are developing a system running on the grid environment, and so have no case study. The remainder of this section describes some assumed applications of our developing tool.

3.1 Creating useful libraries

It may be possible that a single software development department in different projects or software develop departments in a same project develop the same kinds of functions. If we could identify duplicated functions, the duplicated functions may be able to be useful libraries. Creating useful libraries can prevent the department from writing the same kinds of code in the future, and can reduce the development cost.

3.2 Catching licensing violations of source code

Code clone detection from a large set of software systems is able to catch license violations of source code. For example, if we write source code based on GPL'ed software, the source code must be licensed under GPL. If many code clones are detected between GPL'ed source code and non-GPL'ed source code, non-GPL'ed source code may violate the GPL license.

4 Conclusion

In this paper, we describe code clone detection on grid environment for handling a large set of software systems and also discussed its applications.

Acknowledgements

This work is being conducted as a part of EASE Project, Comprehensive Development of e-Society Foundation Software Program, and Grant-in-Aid for Exploratory Research(186500006), both supported by Ministry of Education, Culture, Sports, Science and Technology of Japan. Also it is being performed under Grant-in-Aid for Scientific Research (A)(17200001) supported by Japan Society for the Promotion of Science.

References

[1] Ourgrid. http://www.ourgrid.org/.

[2] T. Kamiya, S. Kusumoto, and K. Inoue. Ccfinder: A multi-linguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, 28(7):654–670, Jul 2002.

[3] S. Livieri, Y. Higo, M. Matsushita, and K. Inoue. Analysis of the linux kernel evolution using code clone coverage. In *Proc. of the 4th Workshop on Mining Software Repositories*, pages 22.1–22.4, May 2007.

[4] S. Livieri, Y. Higo, M. Matsushita, and K. Inoue. Very-large scale code clone analysis and visualization of open source program using distributed ccfinder: D-ccfinder. In *Proc. of the 29th International Conference on Software Engineering*, pages 106–115, May 2007.