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

Originally, code clone detection technique 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 debug process, code clone detection prevent us from overlooking some of code fragments simultaneously.

Originally, code clone detection technique was developed for investigating duplicated code in a single software system, or catching plagiarism between two or three software systems. But, applying code clone detection technique to a large set of software systems should be able to identify useful duplication in it.

However, there is a big problem in applying code clone detection technique to a large set of software systems; existing detection technique is for a few software systems; the scalability of it is not enough to handle a large set. We need to handle several hundred software systems (or several billion 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 [2], and have conducted case studies on a large set of software systems [1]. But, we believe that using grid environment can achieve more ease of use, more portability, and more scalability.

2 Code Clone Detection in Grid Environment

In grid environment, there are two kinds of nodes, master and slave.

Master node: master node divides the target software systems into small sets. Then, it assigns each small set to slave nodes until no small set remains. Also, master node receives code clone detection results from each slave node, and merge them as a single result.

Slave node: slave node receives a small set from the master node. Then, it detects code clones in it by using existing code clone detection technique. The detection result is returned to the master node. After returning the result, the slave node received another small set.

In order to run this system fast, we need a method to merge slave node's detection results efficiently.

3 Assumed Applications

At present, we are developing a system running on 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 the same kinds of functions in different projects. If we could identify duplicated functions between different project's source code developed by a department, the duplicated functions may be able to be useful libraries for the department. 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 source code licensing violations

Code clone detection from a large set of software systems is used for catching source code licensing violations. For example, if we write source code based on GPL'ed software, the source code must be licensed with 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 source code licensing.

4 Conclusion

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

Acknowledgements

This work has been 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 has been performed under Grant-in-Aid for Scientific Research (A)(17200001) supported by Japan Society for the Promotion of Science.

References

[1] 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.

[2] 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.