Volume 4, Issue 9, September 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Code Clone Detection using Decentralized Architecture and Parallel Processing-Latest Short Review

Mr. Ritesh V. Patil
Research Scholar, Dept. of Comp.
Bharath University,
Chennai, India

Dr. S. D. Joshi
Professor, Dept. of Computer Engg.
Bharati Vidyapeeth COE,
Pune, India

Dr. V. Khanna
Dean, Dept. of Comp.
Bharath University
Chennai, India

Abstract--- Code clones are those kinds of codes which gives same output when we apply the same input i.e. functionality of codes is same. Basically it is a reuse approach of existing code in software development life cycle. While code cloning is a convenient way for developers to reuse existing code it could potentially lead to negative impacts, such as code size unnecessarily increased and may lead to unused, dead code. There are various clone detection techniques based on different comparison parameters. Discovered clone detection tools and techniques do not sufficiently satisfy with regards to speed and accuracy. Thus implementation of a Hybrid approach can prove useful for eliminating the flaws in individual techniques. CFG and weighted graph are used for finding all four types of clone. The technique can achieve optimization in the reference of time and space by implementing of a decentralized architecture. Decentralized architecture can improve flexibility of the tool. Parallel programming is used to reduce the detection time.

Keywords: - Code clones, Clone Detection, Decentralized Architecture, Hybrid Approach, Parallel Programming.

I. INTRODUCTION

Clone is keyword which is used for duplication. Sequences of duplicate code are known as *code clones*. The Automated process of finding duplicate code is known as *clone detection*. In software development reuse of code by copying and pasting source code is a common practice. So-called software clones are the results of this reuse approach."Accidental clones" are not the result of direct copy and paste activities but by using the same set of APIs (Application Programming Interfaces) to implement similar protocols. Code clone helps the developers from probable mistakes, to save time and effort in devising the logic, to help in decoupling of classes or components and more important it reduces development cost. But Duplicate code is generally considered as undesirable for number of reasons. Introduction of bad design and lack of good inheritance structure or abstraction may be caused due to code clones. Because of duplicated code the additional time and attention is needed to understand the existing cloned implementation. If a cloned code segment is found to be contained a bug, all of it similar counterparts should be examined for correcting the bug in question as there is no guarantee that this bug has been already removed from other similar parts at the time of reusing or during maintenance. The higher growth rate of the system size is introduced due to duplication of code.

Code Clone Types:-

Basically there are two kinds of similarities between two code fragments. Two code fragments can be similar either textually i.e. depending on the similarity of their program text or functionally i.e. they can be similar in their functionalities without being similar textually.

TEXTUAL SIMILARITY:

Considering the textual similarity we distinguish the clones as following types.

Type I: -Identical code fragments except for variations in whitespace (may be also variations in layout) and comments.

Type II: - Code fragment that is the same as the original except for some possible variations about the corresponding names of user-defined identifiers.

Type III: -Copied fragments with further modifications such as addition or removal of statement as well as addition to variations in identifiers, literals, types, layout and comments.

FUNCTIONAL SIMILARITY:

If the functionalities of the two code fragments are identical or similar i.e., we call the clones as semantic clones and referred as Type IV clones when their pre and post conditions are similar.

Type IV: -Type IV clones are the results of semantic similarity between two or more code fragments.

II. CODE CLONE DETECTION TECHNIQUE

Text-based Approach:-In this approach, the target source program is considered as sequence of lines/strings. Two code fragments are compared with each other to find sequences of same text/strings. Once two or more code fragments are

found to be similar in their maximum possible extent (e.g., w.r.t. maximum no. of lines) are returned as clone pair or clone class by the detection technique.

Token-based Approach: -Sequence of tokens is formed by lexing/parsing/transforming the entire source code in the token-based detection technique. This sequence is then scanned for finding duplicated subsequences of tokens and finally, the original code portions representing the duplicated subsequences returned as clones. Compared to text-based approaches, robustness of a token-based approach is usually more against code changes such as formatting and spacing.

Tree-based Approach:-In the tree-based approach a program is parsed to a parse tree or an abstract syntax tree (AST) using a parser of the language of interest. Matching technique is used for searching similar sub-trees and the corresponding source code of the similar sub-trees is returned as clones pairs or clone classes.

Program Dependency Graph (PDG): -In this approach, we achieve high abstraction of the source code by going one step further than other approaches by considering the semantic information of the source code. The control flow and data flow information of a program is available in PDG and hence carries semantic information. Isomorphic sub-graph matching algorithm is applied once a set of PDGs are obtained from a subject program, for finding similar sub-graphs which are returned as clones.

Metric-based Approach:-Metrics-based approaches gather different metrics for code fragments and instead of comparing code directly these matrices are compared. There are number of clone detection techniques which use different software metrics for detection of similar code. First, a set of software metrics called fingerprinting functions are calculated for one or more syntactic units such as a class, a function, or a method or even statement and then the comparison of the matrices values is done to find the code clones over syntactic units.

Hashing Technique: - An optimization technique is used by the tool using a hash function for string that reduces the computation complexity by a factor, which is a constant determined by the number of characters in a line. Again, meaningful clone resolution is difficult to achieve in a language-independent manner because it is hard to guarantee that detected clones represents cohesive unit in the language being analysed. The process of resolution itself also depends on the language in question.

III. LITERATURE SEARCH

One of the leading state of the art token-based techniques is CCFinder [2] of Kamiya et al. First, each line of source files is divided into tokens by a lexer and the tokens of all source files are then concatenated into a single token sequence. The token sequence is then transformed, i.e., tokens are added, removed, or changed. It searches clones in NT kernel first. It is used widely because of its good availability. It has client-server architecture. Due to this multi-core processing is not possible. But the problem with the CCFinder is it requires lot of time to detect the clones. CCFinder is only capable of finding TYPE I and TYPE II clones.

AST-based clone technique is that of Baxter et al.'s CloneDR. [3].A compiler generator is used to generate an annotated parse tree (AST) and compares its sub-trees by characterization metrics based on a hash function through tree matching [2]. Source codes of similar sub-trees are then returned as clones. Clone DR identifies not only exact but near-missed clones. It is mainly designed to remove redundant code during clone management. Parallel computing implemented efficiently. Bug fixing may prove costly.

The Duplicate Code Detector (Solid SDD) is a software tool for finding and analyzing duplicate code (i.e., code clones). It can be used to analyse large projects and detect code that has been cloned during development, for example by copy-paste-modify operations. Finding such duplicate fragments is of great importance for: Facilitating development and maintenance; Reducing the application memory footprint. The tool is fast and scalable, configurable and easy to integrate.

Clone Digger is tool used for detection of code clones in Python and Java program. It is platform independent tool. The tool is constructed using Abstract Syntax Tree and Suffix Tree techniques for clone. In this tool first of all construction of Abstract Syntax Tree (AST) is carried out. The concept of anti-unification is the key of the clone digger. It supports clone parameterization and more complex clones. It does not detect gapped clones as well as the clones with inserted and/or deleted statements.

Iclone [2] is an incremental tool which is written in Java and therefore it works on most platforms. It is based on Suffix Tree implementation for clone detection. This tool can be used at personal and academic level. It detects only Type-1 and near missed clones. CYCLONE [13] is multi-way evolution inspection tool. It provides five different views to detect clones.

VisCad [1] is a flexible code clone support for NiCad clone detection tool. NiCad detects exact, renamed and near-missed clones. It has good accuracy and gives an interactive output in HTML in addition to its XML text output. But NiCad fails in the case of large code. VisCad visualizes clone detection from NiCad. It quickly locates and inspects the cloning regions from higher level of abstraction to source code level. It can be plugged to detection tool giving textual output references.

IV. INTEGRATED TECHNIQUES

Integrated techniques are nothing but *Hybridizing Approach* towards the clone detection. Every technique has its own merits and demerits. So Hybridization is done to remove the one's demerit by some another feature of other technique. Various tools are based on the hybridizing approach only.

Type-I and Type-II clones can easily find out by individual technique. But Type-III and Type-IV clone detection is really difficult with the help of available techniques. But Hybridizing approach is really useful to find out such difficult

clones. Accuracy of detection is increased with the help of hybrid approach.

Combination of tree based and text based approach saves time and maintenance cost. It is very simple to implement. This approach can find renamed clones; but memory requirement remains as it is. Another hybrid approach is based on tree and matrices. It can find exact clones; but it is really difficult to implement. Token based approach is hybridized with dynamic pattern matching technique. Tokens are compared with each other at run time. Comparison time is saved. It has a demerit that is system will slow down if tokens are in huge amount.

V. DECENTRALIZED ARCHITECTURE

Decentralized system is a software system in which components are located on networked computers. The components interact with each other to achieve a common goal. In the case of clone detection the software tool needs to process a huge length codes. In most of the clone detection tools two tier architecture is implemented.

Decentralized architecture leads accuracy and simple to design but it fails in detection of huge length codes. Client-server architecture uses single core i.e. there is a limited memory available. This mainly restricts the scalability due to lack of resources. The use of distributed architecture facilitates the use of multi-core systems and thereby increases the speed and optimization for the clone detection. Decentralized architecture provides flexibility to our tool. The architecture is simple and easy to use. It is independent of available storage. It has fixed dataflow.

VI. PARALLEL PROCESSING

Parallel processing is simply processing the multiple processes or programs simultaneously. In clone detection, time required to detect the clone is an essential parameter. The tool is said to be efficient, if time required to detect clones is less. Parallel processing provides this feature. Due to the simultaneous execution of processes or programs, same amount of time is required for searching clones in different files. At a time, it will check for the clones in different files. Parallel processing must be supported by distributed architecture.

Parallel processing provides high throughput due to the less searching time. Parallel processing and distributed architecture will work together to achieve the flexibility and the speed. The tool based on CFG and weighted graph finds the all four types of clone. To increase the efficiency of that tool we can use the parallel processing. To optimize the time and space we use parallel processing and distributed architecture.

VII. INCREMENTAL APPROACH

Clone management tools rely on accurate cloning information to indicate cloning relationships in the IDE while developers maintain code. To remain useful, cloning information must be updated continuously as the software system under development evolves. For this, detection algorithms need to be able to very rapidly adapt results to changing code, even for very large code bases. In most of the clone detection tools the processing occurs in batch mode i.e. it reads the entire software system and detects all clones in a single step. If used for clone management, batch algorithms either have to be executed on-demand, or clone management has to use pre-computed results that are updated on a regular basis.

In software evolving stage their occur modifications into the code this may introduce clones into the code at many places. The redetection of whole code is a costly computation. It also consumes time. In incremental approach the only modified part of the code is examined for detection. This approach is also called real time approach. All the efforts needed for update only is based on size of the change and not on the entire code base.

VIII. CONCLUSION

This paper is mainly carried out for providing the flexibility and speed up the tool. Efficiency of the tool can be increased with the help of parallel processing and distributed architecture. We obtained a global overview of code clones among these programs, and the use of the same source code in multiple projects was visible as easy recognizable artifacts. The objective of the paper is to find out the clone from large database system in very little time. The paper is concerned about architecture and design of the efficient system.

ACKNOWLEDGEMNT

We are very much thankful to Dr. S. D. Joshi for encouraging us for the research study of software clone detection. We are also thankful to Dr. V. Khanna, Bharat University, Chennai for giving resources for research studies.

REFERENCE

[1] M. Asaduzzaman, C. K. Roy, and K. A. Schneider. VisCad: Flexible code clone analysis support for NiCad. In IWSC, pages 77–78, 2011.

[2] T. Kamiya, S. Kusumoto, K. Inoue. CCFinder: a multilinguistic token-based code clone detection system for large-scale source code. *IEEE Trans. Software. Eng.*, 28(7):654–670, 2002.

[3] I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone Detection Using Abstract Syntax Trees", Proc. IEEE Int'l Conf. on Software Maintenance (ICSM) '98, pp. 368-377, Bethesda, Maryland, Nov. 1998.

[4] B. Baker. On finding duplication and near-duplication in large software systems. In WCRE, pages 86 –95, 1995.

[5] H. Basit and S. Jarzabek. Detecting higher-level similarity patterns in programs. SIGSOFT Softw. Eng. Notes, 30:156–165, 2005.

[6] H. Basit, D. Rajapakse, and S. Jarzabek. An empirical study on limits of clone unification using generics. In SEKE, pages 109–114, 2005.
[7] S. Bazrafshan and R. Koschke. An empirical study of clone removals. In ICSM, pages 50–59, 2013.
[8] S. Ducasse, M. Rieger, and S. Demeyer. A language independent approach for detecting duplicated code. In ICSM, pages 109 –118, 1999.
[9] M. Fowler, K. Beck, J.Brant, W. Opdyke, and D. Roberts. Refactoring: Improving the Design of Existing Code. Addison Wesley, 1999
[10] Y. Fukushima, R. Kula, S. Kawaguchi, K. Fushida, M. Nagura, and HIida. Code clone graph metrics for detecting diffused code clones. In APSEC, pages 373 –380, 2009.
[11] S. Giesecke. Generic modelling of code clones. In DRSS, pages 1–23, 2007.
[12] N. Göde and R. Koschke. Studying clone evolution using incremental clone detection. J. of Soft.:Evol. and Proc., 25(2):165–192, 2013.
[13] J. Harder and N. Göde. Efficiently handling clone data: RCF and cyclone. In IWSC, pages 81–82. ACM, 2011.
[14] J. Harder and N. Göde. Cloned code: stable code. Journal of Soft.:Evol. and Proc., 25(10):1063–1088, 2013.
[15] Y. Higo, U. Yasushi, M. Nishino, and S. Kusumoto. Incremental code clone detection: A PDG-based approach. In WCRE, pages 3 –12, 2011.
[16] 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 ICSE, pages 106–115, 2007.
[17] E. Juergens, F. Deissenboeck, and B. Hummel. CloneDetective – a workbench for clone detection research. In ICSE, pages 603–606, 2009.
[18] N. Pham, H. Nguyen, T. Nguyen, J. Al-Kofahi, and T. Nguyen. Complete and accurate clone detection in graph-based models. In ICSE, pages 276–286, 2009.
[19] Index-Based Code Clone Detection:Incremental, Distributed, Scalable by Benjamin Hummel ElmarJuergens Lars Heinemann R. Koschke, "Survey of research on software clones," in Duplication, Redundancy, and Similarity in Software, 2007.
[20] J. Dean and S. Ghemawat, "MapReduce: a flexible data processing tool," Commun. ACM, vol. 53, no. 1, pp. 72–77, 2010.
[21] S. Livieri, Y. Higo, M. Matsushita, and K. Inoue, "Very-large scale code clone analysis and visualization of open source programs using distributed CCFinder: D-CCFinder," in ICSE'07, 2007.
[22] R. Tairas and J. Gray. Increasing clone maintenance support by unifying clone detection and refactoring activities. Info. & Soft. Tech.,54(12):1297–1307, 2012.
[23] M. Uddin, C. K. Roy, K. A. Schneider, and A. Hindle. On the effectiveness of simhash for detecting near-miss clones in large scale software systems. In WCRE, pages 13 –22, 2011.
[24] M. F. Zibran and C. K. Roy. Towards flexible code clone detection, management, and refactoring in IDE. In IWSC, pages 75–76, 2011.
[25] J. Krinke, "Identifying similar code with program dependence graphs," in WCRE'01, 2001.
[26] S. Chen. Cheetah: a high performance, custom data warehouse on top of MapReduce. Proceedings of the VLDB, 3(1-2):1459–1468, 2010.

