CLONE DETECTION USING RABIN-KARP PARALLEL ALGORITHM

Rahadian Dustrial Dewandono[1], Fahmi Akbar Saputra[2], Siti Rochimah[3]
Department of Informatics, InstitutTeknologiSepuluhNopember
Surabaya, 60111
[1]dewandono11@mhs.if.its.ac.id, [2]fahmi11@mhs.if.its.ac.id, [3]siti@its-sby.edu

ABSTRACT

Code duplication is a common problem found in software development. It could generate several clones. The existence of clone is highly possible to increase the risk on software evolution. Method for detecting clone comprises textual, lexical, syntactic, and semantic approach.

In this paper, we evaluate by using several aspects based on the condition of each pair. We propose a novel method to detect clone by using Rabin-Karp parallel algorithm. The algorithm is more efficient than the traditional Rabin-Karp algorithm. In cases of evaluation, we built a detecting tool capable of processing source code in both lexical and syntactic manner.

We evaluate the performanceof the proposed method. To do so, we compare parallel Rabin-Karp to Traditional Rabin-Karp. The result shows parallel Rabin-Karp could gain better performance.

Keywords:Code Duplication, Clone Detection, Similarity Checking, Software Evolution, Rabin-Karp Algorithm.

1 INTRODUCTION

Code duplication is a common problem found in software development. Developers occasionally duplicate a block of codes to new line. This case could generate clone. Clone is a block of code, which is repeated many times on the source code. Previous research shows that approximately 7% up to 23% of source code of a software is duplicated [1].

The problem of clone could increase software development risk. It requires an extra effort to maintain clones [2]. If a clone is changed, the other clones might be not changed yet, since there is no link between each other. This issue could lead to a fatal failure on business process or the main algorithm. Furthermore, it is difficult to search and to correct error due to the presence of clones.

There are many researches on Software Engineering giving solutions to the problem of code duplication, including source code quality analysis, aspect mining, plagiarism detection, and copyright investigation. All of these researches need syntactic and semantic extraction of source code block on the software [3].

Previous researches on clone detection topic presents some methods to identify code clones on software, including textual, lexical, syntactic, and semantic analysis of code clones. In this paper, we first compare these four method of code clones detection technique and then provide a comparison based on three aspects. Based on this comparison, we found that the lexical and syntactic method is superior compared to the others.

In this research, we propose a new method to detect code clone. We combine the lexical and the syntactic method to detect code clones on source code of software. We build a tool, which is similar to PALEX [2] and LePALEX [3] to find code clones based on lexical and syntactic method combination. The output of this tool will be compared in parallel using the Rabin-Karp Parallel algorithm. Any code clone will be detected when there is same value in detection process.

2 MODEL, ANALISIS, DESIGN, AND IMPLEMENTATION

2.1 Terminology

Code Fragments (CF) refers to one or several lines of source code with comments. A CF can be identified by considering the name of archive and the order from starting line until final line. CF is recorded with the format as follows (CF, CF. [Archive-Name], CF. [Initial Line], CF. [End Line]). Clone is two or more CF-s having a similar function. The two CF-s belong to a clone if f(CF1) $=f$(CF2).

Clone analysis requires clone clustering based on type. The types of clone encompass clone type 1, type 2, type 3, and type 4. Clone type 1 is identical CF without any white space, layout, and comments. Clone type 2 is several CF, which are identical syntactically, except as an identifier, literal, type, white space, layout, and comment. Clone type 3 refers to CF, which is difficult to modify in certain syntactical pattern. Clone type 4 is several CF performing similar computation but written in different in a syntactic view.

2.2 Lexical Approach for Clone Detection

The procedure of lexical approach for clone detection comprise of converting source code into several lexical sequences and return original value of corresponding source code as a clone. This type of approach is good at evaluating minor revision found in source code [1].

This lexical method has been firstly introduced by Brenda Baker. Firstly, every source code is divided into some tokens by utilizing lexical analysis engine. Every taken entity is clustered into several token with parameter and token without parameter. Tokens with parameter is encoded in index based on their line number. Tokens without parameter belong to the lines summarized by utilizing hashing factor. Every prefix of sequence is represented by using suffix flow. If there are two suffixes that have general prefix, then there are more than one prefix included. Therefore, those two suffixes are categorized as a clone.

This method is able to identify clone for type 1, 2, and 3. In case of type 3, it could be accomplished by joining type 1 and type 2. If the lexical approach is not more than the determined threshold, then the source code is categorized as clone type 3. This process can be accomplished by using dynamic programming.

CCFinder is a tool providing a solution to detect clone. CCFinder utilizes an additional source in order to omit minor discrepancies, i.e., discrepancy within the bracket. This technique is a basic technique used by other techniques. As well as CP-Miner, it apply sequence of data mining to search for similar sequences from the tokens.

In lexical approach, syntactic is not a critical case. The clones is highly possible to overlap with other different unit of syntactic. However, this risk could be minimized by manipulating both pre-processing and post-processing phase. Hence, the clones corresponding to a syntactic block could be identified if block limitation is known [5].

2.3 Textual Approach for Clone Detection

Textual approach is a clone detection technique enabling the system processes source code directly without either any changing or any normalizing. Fingerprinting, a technique to hash several line codes that has exactly same numbers of line code, is popular to use in the context of textual approach. Code duplication will be inspected if the hashing values is same.

Ducasse uses dot plot to depict that there are some similar lines of codes. The x-axis and y-axis represent certain lines of codes in the process of comparison. The similarities are determined by the similarities hashing value among the compared lines. If hashing result is obtained with the same values, then a spot is shown. This spot identifies that there are lines of code, which are same with other lines of code. The diagonal line depicts that the source code contains a code duplication.

2.4 Semantic Approach for Clone Detection

This method utilizes static program analysis to obtain an accurate information. To implement the method, a dependency graph should be used. A graph node represents an expression or a statement, a graph content specifies a control and a dependency graph value. The methodology of clone detection is similar with that of isomorphic graph searching [6]. This method is an approximation approach due to whose NP-Hard complexity.

2.5 Syntactical Approach for Clone Detection

Syntactical approach for detecting clones is conducted by using tree and metrics approach. By using tree approach, codes have to be changed or be represented into parse tree or abstract syntax tree (AST). The code clones can be found by searching the same sub-tree. An improvement has been made to avoid sub-tree comparison complexity. This approach conducted by transforming sub-tree into sequence of nodes. By using this approach, clone detection speed will be equal to retrieved entities. Moreover, there exist another technique using characteristic vectors to estimate AST structure in Euclidean space. Further, clustering is conducted so that code clones can be found.

Clone detection method using the metrics approach is conducted by collecting some metrics CF to obtain vectors. Comparison between metric vectors is conducted in order to find code clones.

2.6 Comparison Method

We use systematical comparison to compare four methods for clone detection. The comparisons were performed using three aspects. The aspects of comparison consist of detection quality, clone size, and clone scope.

The selection of these three aspects are more based on the circumstances of the couple clone rather than based on an equivalent class. That is because the CF type 1 and type 2 have a basic similarity function that is reflexive, symmetric, and transitive, while CF type 3 do not have the basic similarity function that is transitive.

Detection Quality. Quality detection is defined as similarity level of methods to identify code clones. Measurements were performed using the values of recall and precision. Clones would be detected on its location and type. In the process of evaluation of the source code of some programs, the lexical approach can detect more clones than the other approach. Semantic approach to consider all clones detected as clones third type, whereas the lexical approach cannot report specific.

Clone Size. Clone size aspects represented as lines of code. With this measurement, we can investigate the number of lines of code that is proposed as a clone by a particular approach. From experiment, we find that the lexical approach could detect longer clones than resulted clones of another approach. The lexical approach also often detect a code snippet consisting more than one function.

Clone Scopes. Measurements on the clone scopes can investigate whether the code is more often copied in the same file or in different files. The four approaches in clone detection methods would be measured on its ratio between the copied code in the same file and in different files. From the experiments conducted, the lexical approach detects more clones that are copied in the same file rather than clones that are copied into a separate file. Meanwhile, the semantic approach could detects clones that are copied in the different files much more than clones that copied in the same file.

2.7 Comparison Matrices

From the analysis of the four approaches to clone detection using three aspects of the comparison, we define three classes of benchmarking results: low, medium, and high. Explanation of assessment criteria described in Table 1.

Detailed explanation of comparison results of the four-clones approaches presented in Table 2.

Table 2. Comparison Results

	Textual	Lexical	Syntactic	Semantic
Detection quality	High	High	Medium	Low
Clone size	Medium	Medium	High	High
Clone scope	Low	High	High	Medium

It is concluded that lexical is the best in cases of detection quality and clone scope. Syntactical approach is the best in cases of clone size and clone scope. Combing both lexical and syntactic approach is the best solution among the rest.

2.8 Rabin-Karp Algorithm

Rabin-Karp Algorithm is an algorithm used for matching two strings exactly [13].There at least two strings compared each other. The short string is a pattern, which is compared to other string. All of the patterns are compared to the full string. The algorithm has worst-time complexity O(n+m) in space O(m).

Rabin-Karp algorithm is frequently implemented in application for detecting plagiarism. Material source is given, Rabin–Karp uses such material to compare to a paper for instances of sentences. The algorithm ignoressome details (i.e., case, punctuation, etc.).

2.9 Parallel Processing

Parallel processing is a technique of information processing that emphasizes on concurrent manipulation of data that are owned by one or more processes so that problems can be solved [10]. This technique became popular due to the development of processor technology, more improvements directed toward parallel architecture than the speed of the processor [1].

Previous researches use parallel processing techniques to accelerate the processing time in an algorithm, such as the use of parallel processing techniques for particle swarm optimization algorithm (PSO) [2], genetic algorithms [3], and the fuzzy-based neural network [4].

3 RESULT

Research [7] has introduced clone detection method utilizing prefix tree algorithm. The output resulted of PALEX [2] is mapped on the weighted tree. Then, it is identified as the same suffix. Research [8] introducedLePALEX, which is able to convert the output of PALEX into three normal forms (NF1, NF2, and NF3). After that, clone detection using Rabin-Karb algorithm is performed.

Rabin-Karp algorithm lacks efficiency due to the complexity. The complexity lies in the range O(n+m) to O(nm); n denotes the length of text being processed and m denotes the length of pattern being compared. Moreover, this algorithm is performed in sequential manner.

In this research, we propose to apply Rabin-Karp Parallel algorithm in cases of clone detection. As presented in Table 3, Rabin-Karp parallel has better time response in cases of processing large source code. This algorithm is used to process outputs of LePALEX and to convert them into several parts. Those parts are processed using Rabin-Karp in parallel manner. Finally, those parts are compared each other to investigate if there are any clones.

Table 1. Measurement Assessment

Aspect	Low	Medium	High
Detection quality	Can only detect clones type 1	Can detect clones type 1 and type 2	Can detect clones type 1, type 2, and type 3.
Clone size	The error rate between the proposed lines and the code clones > 75%	The error rate between the proposed lines and code clones 25-75%	The error rate between the proposed lines and code clones < 25%
Clone scope	Can only detect clones in a single class.	Can detect clones across classes but cannot detect clones across files.	Can detect clones across files.

Table 3 A comparison between Traditional and Parallel Rabin-Karp

n	m	Traditional Rabin-Karp	Parallel Rabin-Karp
10000	10	0.026s	0.088s
	3000	0.026s	0.097s
3479998	10	3.553s	1.182s
	3000	3.888s	1.313s
	10000	3.936s	1.351s
6970438	10	7.114s	2.286s
	3000	7.757s	2.2520s
	10000	8.026s	2.598s
60258128	10	81.87s	24.74s
	3000	87.38s	27.25s
	10000	89.13s	27.75s

3.1 Source Code Partitioning

Source code is separated based in the identifier. In the proposed method, some identifiers play role as an initial marker of the source code partition. The identifier encompasses if, while, for, and other forms of identifier requiring a code block within. To elaborate more comprehensively, we illustrate each step of our proposed method by using source code examples.

```
for(inti = 0;i<5;i++)
{       i+=5;
        i--;
}
if(arr[0]<5)
{       arr[0]++;
        arr[1]++;
}
```

Figure 1. Ordinary Code Block

Figure 1 presents that are ‘for’ and ‘if’ as identifier. They have a role as initial marker of code separation. The result of code partition based on for and if as identifier is depicted Figure 2.

```
for(inti = 0;i<5;i++)
{
        i+=5;
        i--;
}
```

Figure 2. A “For” Code Block after Divided

```
if(arr[0]<5)
{
        arr[0]++;
        arr[1]++;
}
```

Figure 3. An “If” Code Block after Divided

Code blocks depicted in Figure 2 and Figure 3 are processed in a parallel manner.

3.2 Converting into Normal Form I (NF-1)

In order to implement Rabin-Karp algorithm in a parallel manner for the purpose of clone detection, each block of source code should be converted into Normal Form (NF) I, II, and III. NF I could detect the exact match of clones. NF II is used to inspect the syntactic match of clones. NF III checks for syntactically correct segments of clones.

Figure 4 and Figure 5 are the NF I of Figure 1. The code in the block is converted by eliminating white spaces.

```
for(inti=0;i<5;i++)
{
i+=5;
i--;
}
```

Figure 4. NF-1 “For” Code Block

```
if(arr[0]<5)
{
arr[0]++;
arr[1]++;
}
```

Figure 5. NF-1 "If" Code Block

3.3 Converting into Normal Form II (NF-2)

After converted into NF I, each block of source code is converted into NF II. The objective is to build hash function for text searching implemented on Rabin-Karp algorithm. The hash value is calculated for each block of code. Therefore, it could be performed in parallel manner. NF II is depicted in Figure 6 and 7.

```
$int,i,i,i:for($ $=0;$<5;$++)
{
$i:$+=5;
$i:$--;
}
```

Figure 6. NF-2 "For" Code Block

```
$arr[0]:if($<5)
{
$arr[0]:$++;
$arr[1]:$--;
}
```

Figure 7. NF-2 "If" Code Block

After being converted into Normal Form II, the hash function could be executed. The result of the hash function represents Normal Form II for each line of codes. Each line of code has unique pattern resulted from the corresponding hash function.

Parallel Rabin-Karp algorithm is applied in this step. Each block is compared to one another. For purposes of clone detection. To cut the cost of complexity, the proposed method only performs comparison between hash values of each block. In addition to that, the separation of data input improves responsiveness of the system.

3.4 Converting into Normal Form III (NF-3)

The conversion of code block into normal form III is performed for syntactical checking. This case could accommodate condition that is not solved by using exact string matching.

The key lies in the XML tags added into the block of code. After calculating hash value for each block, the values are inserted into a hash table. Clone detection is examined for each entry in the table. Finally, the result of NF-3 is used for decomposing clones into syntactical form with XML tags.

3.5 Evaluation

We build LePALEX-like tool to test validity and execution time of the proposed method. The tool can read the source code, breaking it into several blocks based on identifier of the source code, transform each block into the second normal form, and check the clones in parallel or traditional Rabin-Karp. Parallel processing uses many threads. The implemented is described in Figure 8.

After being processed in a parallel manner, performance is measured by recording time execution of each method. In this research, we conduct experiments ten times to compare Rabin-Karb to Parallel Rabin Karb. We utilize three categories of data set: small, medium, and large.

Table 4. Category of Data Set

Category	Kilo Line of Code (LoC)
Small	0-10000
Medium	10000-20000
Large	>20000

All of the experiments result that the execution time for every category of data set are various. Figure 9 describes the comparison between each category in terms of execution time.

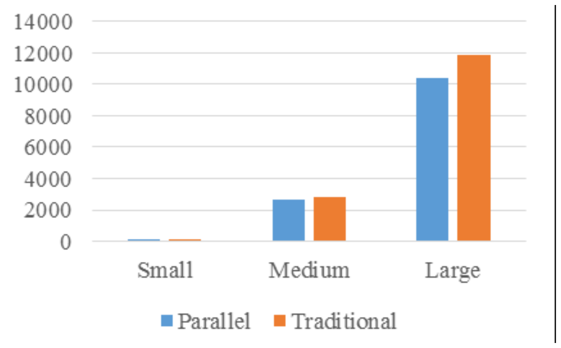

Figure8. Time Execution Comparison

From the result, we could conclude that Parallel Rabin Karb better than Rabin Karb does especially efficiency and execution time. The fewer line of codes, the fever deviation between the performances.

4 CONCLUSION

Clone Duplication is a serious problem found in field of software evolution. There are several methods to investigate clones: syntactical approach, semantic approach, lexical approach, and textual approach. The combination between syntactical and lexical approach is the best among such approaches.

In this paper, we develop a tool for purposes of clone detection. We conduct both syntactic and lexical approach utilizing Rabin-Karp algorithm. In order to increase performance, we perform the algorithm in a parallel manner. The evaluation of this research shows that parallel mechanism implemented on Rabin-Karp algorithm gain better performance compared to traditional one. The more lines of code to develop a program, the more significant the proposed method performs.

REFERENCE

[1] P. Brodanac, "Parallelized Rabin-Karp Method for Exact String Matching", Proceedings of the ITI 2011.

[2] J. Kim, "Optimal power system operation using parallel processing system and PSO algorithm", in *Electrical Power and Energy Systems*, 2011.

[3] P. A. E. Vidal, "A Multi-GPU Implementation of a Cellular Genetic Algorithm,"*IEEE,* 2010.

[4] C. C. T. C. Juang, "Speedup of Implementing Fuzzy Neural Networks With High-Dimensional Inputs Through Parallel Processing on Graphics Processing Units," *IEEE,* 2011.

[5] B. B., "On Finding Duplication and Near-Duplication in Large Software Systems,"*Proceedings of the 2nd Working Conference on Reverse Engineering,* pp. 86-95, 1995.

[6] C. L. d. E. M. J. Mayrand, "Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics," 1996.

[7] C. R. a. J. Cordy, "A Survey on Software Clone Detection Research," Queen's Technical Report, 2007.

[8] J. R. C. R. K. Chanchal K. Roy, "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach," 2009.

[9] L. Moonen, "Generating Robust Parsers Using Island Grammars," 2001.

[10] J. Krinke, "Identifying Similar Code with Program Dependence Graphs," 2001.

[11] K. Maeda, "Syntax Sensitive and Language Independent Detection of Code Clones,"*World Academy of Science, Engineering and Technology,* p. 36, 2009.

[12] K. Maeda, "An Extended Line-Based Approach to Detect Code Clones Using Syntactic and Lexical Information," in *Seventh International Conference on Information Technology*, 2010 .

[13] L. B. a. D. J. PredragBroanac, "Parallelized Rabin-Karp Method for Exact String Matching," in *Proceedings of the ITI 2011 33rd Int. Conf. on Information Technology Interfaces*, 2011.

[14] E. M. Rasmussen, "Introduction: Parallel Processing and Information Retrieval," *Information Processing & Management,* vol. 4, pp. 255-263, 1991.