Prototype of Web-based Application for Detecting Plagiarism

Meredita Susanti
Computer Science Departement
Gadjah Mada University
Yogyakarta
susantymeredita@yahoo.com

Khabib Mustofa
Computer Science Departement
Gadjah Mada University
Yogyakarta
khabib@ugm.ac.id

ABSTRACT
There exist applications and services which can detect document similarity, such as Turnitin [4] and MyDropBox[1]. Both compare a document in English to documents of their collections. TESSY (Test of Text Similarity) has also been developed in Gadjah Mada University [4]. This software checks documents similarities, but does not give information about which sentences are similar among the documents.

In this paper, we discuss a prototype of anti-plagiarism software that was developed as a web-based application built using Java. It has the ability to read documents in several formats, such as MS Word, MS Excel, MS Power Point, PDF, and plain text. The prototype functions by indexing documents, comparing documents and provides information about which sentences are similar in the documents together with the similarity types of sentences. In terms of speed performance, the software needs further improvement, but in general the result is promising as it can index and compare documents written in Indonesian. It also generates reports of the similar sentences position (by highlighting sentences), and similarity type of the sentences.

Keywords
plagiarism, document similarity, similarity type, morphological rules

1. INTRODUCTION
Plagiarism is a significant issue on most college and university campuses. In Indonesia, especially, in the beginning of 2010, the academician were shocked by the indication of plagiarism conducted by a professor [1]. This lead to the rising importance of implementing thoroughful plagiarism check for any kind of intelectual property, especially scientific documents. Plagiarism detection software is considered a powerful tool to fight against plagiarism.

This paper discusses a prototype of a web-based and open source anti-plagiarism software. Turnitin will be used as a reference application in developing the prototype. The prototype provides several features: comparison of documents in terms of sentence similarity, giving type of similarity (exact match, insertion, deletion, word change), display comparison result between two documents which contain similar sentences and generating reports of the comparison along with the highlighted sentences suspected to be plagiated form othe resources.

The paper is organized as follows. In section 2, we present some related work.s Section 3 explains the system design components used in the prototype. Section 4 describes the implementation of the design. Last but not least is section 5 which discuss the experimental result of the system performance, conclusion and the future works.

[1] http://www.mydropbox.com/

2. RELATED WORKS

2.1 Plagiarism Pattern
Comparing unit (chunking unit), overlap measure function, and plagiarism decision function crucially affect the performance of a Document Copy Detection system [2].

Let *So* is a part of the original document and *Sc* of the query document. The similarity – *Sim(So, Sc)* – can be calculated as follows [3]:

S_0 = {w1, w2, w3, ..., wn} , *Sc* = {w1, w2, w3, ..., wm}

Comm $(S_0, S_c) = S_0 \cap S_c$, *Diff*$(S_0, S_c) = S_0 - S_c$

Syn(w) = {synonym of w}

SynWord(S_0, S_c) $= \{w_i | w_i \in Diff(S_c, S_0) \cap Syn(w_i \in S_0)\}$

WordOverlap$(S_0, S_c) = \dfrac{|S_0|}{|Comm(S_0, S_c)| + \alpha . |SynWord(S_0, S_c)|}$,

where α is weight value

SizeOverlap $(S_0, S_c) = \sqrt{Diff(S_0, S_c) + Diff(S_c, S_0)}$

Sim $(S_0, S_c) = |S_0| . \left(\dfrac{1}{e^{WordOverlap(S_0, S_c) - 1} + SizeOverlap(S_0, S_c)} \right)$

Table 1 shows how to decide plagiarism patterns.

2.2 Stemming
Stemming is a core natural language processing technique for efficient and effective Information Retrieval. It is used to transform word variants to their common root word by applying morphological rules. For Stemming Bahasa (Indonesian Language), the prototype will use algorithm form Asian, Williams, and Tahaghogi [2]

Before considering how the scheme works, we consider the basic groupings of affixes used as a basis for the approach, and how these definitions are combined to form a framework to implement the rules. The scheme groups affixes into categories:

1. Inflection suffixes – the set of suffixes that do not alter the root word. The inflections are further divided into:

   (a). Particles (P) – including "-lah" and "-kah"

(b). Possessive pronouns (PP) – including “-ku", “-mu", and “-nya"

Table 1. Plagiarism Pattern Decision Parameter

Plagiarism Pattern	Decision Parameter	
Copy Exactly	WordOverlap(S_0, S_C) =1	SizeOverlap(S_0,S_C) = 0
Word Insertion	SizeOverlap (S_0, S_C) ≠ 0	Diff(S_0, S_C) > 1
Word removal	SizeOverlap (S_0, S_C) ≠ 0	Diff(S_0, S_C) > 1
Changing word	1<WordOverlap(S_0, S_C)<∞	SizeOverlap (S_0, S_C) = 0
Changing Structure	WordOverlap(S_0, S_C) = 1	SizeOverlap (S_0, S_C) = 0

Particle and possessive pronoun inflections can appear together and, if they do, possessive pronouns appear before particles. A word can have at most one particle and one possessive pronoun, and these may be applied directly to root words or to words that have a derivation suffix. For example, “makan" (to eat) may be appended with derivation suffix “-an" to give “makanan" (food). This can be suffixed with “-nya" to give “makanannya" (a possessive form of “food")

2. Derivation suffixes – the set of suffixes that are directly applied to root words. There can be only one derivation suffix per word. For example, the word “lapor" (to report) can be suffixed by the derivation suffix ”-kan" to become “laporkan" (go to report). In turn, this can be suffixed with, for example, an inflection suffix “-lah" to become “laporkanlah" (please go to report)
3. Derivation prefixes : the set of prefixes that are applied either directly to root words, or to words that have up to two other derivation prefixes. For example, the derivation prefixes “mem-" and “-per-" may be prepended to “indahkannya" to give “memperindahkannya" (the act of beautifying). The classification of affixes as inflections and derivations leads to an order of use:

[DP+[DP+[DP+]]] root-word [[+DS][+PP][+P]]

3. SYSTEM COMPONENT DESIGN

We employed a use case diagram as shown in figure 1 to describe system functionality. There are 3 main scenarios, `UploadDocument`, `CheckDocument`, and `DownloadPdfReport`. This paper will only describe the `CheckDocument` scenario. In this scenarion, the cases as mentioned in the table 1 to be covered are: copy exactly, word removal, word insertion and structure change and limited changing word .

The `CheckDocument` scenario will check the query document (document to be compared) with existing documents (original documents) in the database. After indexing the query document, the application gets the identity of document, such as; author, created date, and file extension. File extension is used to determine which API will extract document, then it compares words, sentences, or paragraphs of the original document with query document. Sentences in the query document that are similar to the original document will be highlighted with a different color.

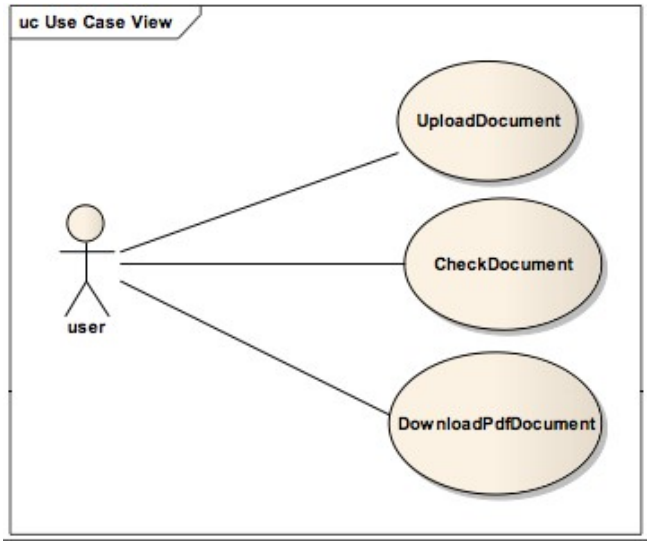

Figure 1 Use Case Diagram

The detail process of the `CheckDocument` scenario is as follows:

1. get content and metadata of the documents : query document (QD) and original/reference document (RD)
2. parse content into sentences
3. compare sentenceQD with sentenceRD. If both are equal, it results in degreeOfCopy to be “copy exactly” and then do step 5, otherwise do step 4
4. parse sentence into words and trim the words (eliminating connecting words, such as: “di”, “ke”, “dari”, “untuk”). If trimmed words of RD contains all trimmed words of QD, this can be classified to be “copy exactly” case, otherwise do stemming prosess (removing prefix, suffix or infix to find bare words by consulting to a dictionary)
   a. if the set of stemmed words of RD contains all stemmed words of QD this means a word removal case. Conversely, it is classified as word insertion case.
   b. If the set of stemmed words of RD is equal with set of stemmed words of QD, this can be considered a changing structure
   c. if it is necessary to check the word changing, the process is more complicated as one more step is needed. The process includes checking each root/bare word resulting from the stemming process of the QD to find their synonym as recorded in the dictionary. If a word of QD is not part of the set of stemmed words of RD but the synonym of the word of QD is part of the set of stemmed word of RD then the sentence of QD is classified to be a case of word change pattern.

   If none of the above case is matched, the sentence of QD is not suspected as plagiated from any sources, the spe continues to step 7.

5. record index sentenceQD, index sentenceRD, filenameRD, degreeOfCopy
6. add HTML tags to mark the suspected sentences and give proper colouring for each tags in the document.
7. Repeat step 3 – 6 until all sentences are compared.
8. generate report of PDF format from the coloured document
9. present report into web and provide link to download PDF report

4. SYSTEM ARCHITECTURE

The architecture of the system developed is shown on Figure 2.

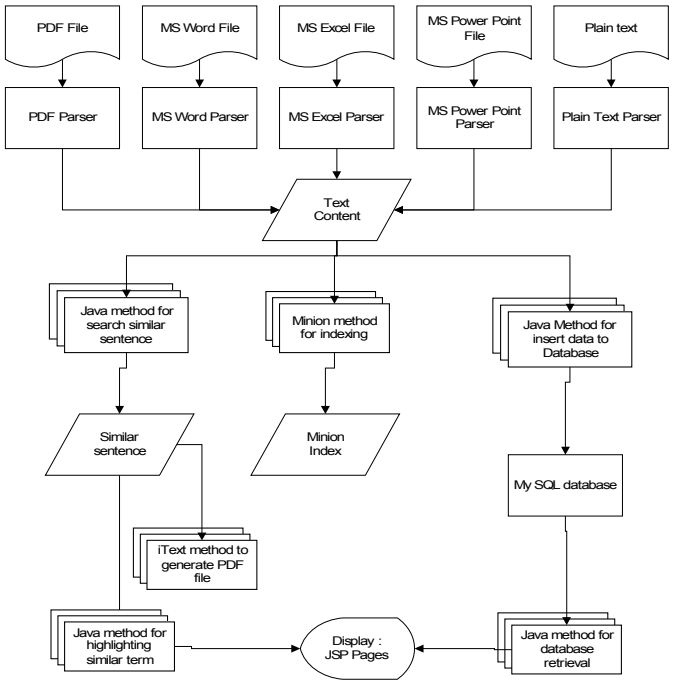

Figure 2 System Architecture of the prototype

Document content will be read using the appropriate API based on the file extension. The document content is then converted into a `String`. Then another API – Minion – indexes this `String`. The application do comparison on file content as described above using the `CheckDocument` scenario, the comparing unit is word. We get similar sentence position and the similarity pattern. The result of the application is a report that lists position, similar sentence(s), and the similarity type. The report also shows content of document that was highlighted by colors. Different highlight colors show the similarity patterns. Besides being displayed in the Web browser, the report can be downloaded as well.

5. Class Design

As mentioned above, the main focus of the discussion is only on the process of checking the document, including stemming and similarity pattern finding process. The overview of the class developed for checking the document is shown in Figure 3.

5.1.1 Stemming

Stemming process in this application is using stemming algorithm from Jelita, Williams, and Tahaghoghi as described in section 2.2 Class `Stemmer` is the class which performs the stemming process. Followings are the methods in Class `Stemmer` and their functions:

a. `getRootWord(String word)` : Do stemming and transform word variants to their common root.

b. `checkRootWord(String word)` : Check if the common root of the word exists in the dictionary. If the word exists, it means the word is in a common root form.

c. `getSuffix(String word)` : get the suffix of a word.

d. `getPrefix(String word)` : get the prefix of a word.

e. `removeInflectionSuffix(String word, String suffix)` : remove inflection suffix from a word, such as "-lah", "-kah", "-pun", "-nya", "-ku", "-mu".

f. `removeDerivationSuffix(String word, String suffix)` : remove derivation suffix from a word, such as "-kan", "-an", "-i".

g. `removePrefix(String word, String prefix)` : remove prefix from a word.

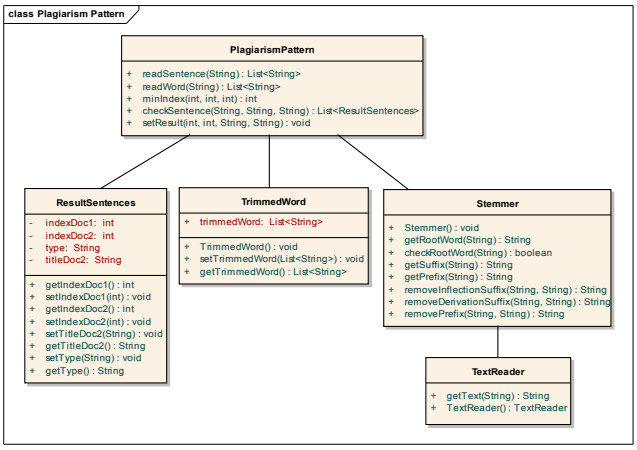

Figure 3 Class Diagram for CheckDocument

Unlike Andriani algorithm – as described in Jelita's paper [2]. – this algorithm does not record removed affixes.

In doing stemming, first the `getRootWord(String)` method is called. This method checks if the word is in the dictionary. If the word does not exist in the dictionary, it checks whether the word is a plural form. If the word is in plural form, the singular form is the common root. If the word is not in plural form, `getSuffix(String)` method is called to get the suffix. Once we have the suffix, it will be removed from the word. After removing all suffixes, we check the prefix using `getPrefix(String)`. We remove all prefixes until we get the common root. The order of this process is shown in Figure 4.

5.1.2 Find Plagiarism Pattern

Class `PlagiarismPattern` is used to compare sentences in the original document and query document, and get the position of a similar sentence and it's pattern. These are the methods in `PlagiarismPattern` class:

a. `readSentence(String content)` : breaks file content into sentences. Sentences are stored in `ArrayList`.

b. `readWord(String sentence)` : breaks sentence into words. Words are stored ini `ArrayList`.

c. `minIndex(int index1, int index2, int index3)` : return the position of nearest punctuation.

d. `checkSentence(String firstReader, String secondReader, String secondFilename)`: determines the similarity pattern of a sentence.

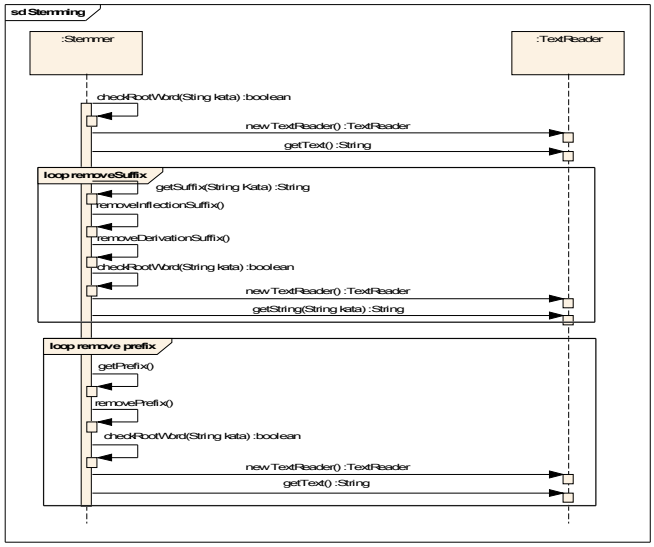

Figure 4 . Sequence Diagram for Stemming

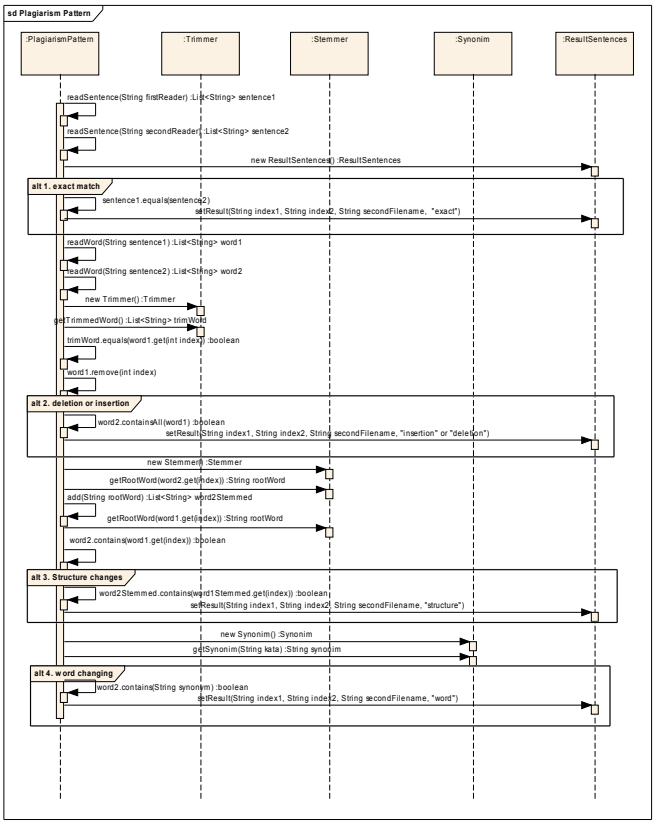

Figure 5 Sequence Diagram To Find Plagiarism Pattern

In order to get information about position of a similar sentence and it's similarity pattern, the document content is broken into sentences. Then we compare each sentences in the first document with the second document. Next we record the position of the similar sentence and it's similarity pattern in an `ArrayList`. Those similar sentences will be highlighted in the report. The order of this process is shown in Figure 5

6. IMPLEMENTATION

6.1 Implementation ToFind Similar Sentence

Suppose the first document is *documentA* and it will be compared to the second document called *document1*. We do the comparison using `PlagiarismPattern.checkSentence(content of documentA, content of document1, document1)` method.

Comparison processes are run MxNx3 times to compare the first document with the second document (where M is total words in *documentA*, and N is total words in second *document1*). Beside word comparison, we also do 1 up to 6 comparisons to get the common root of a word, depending on the number of affixes in the word.

Method `checkSentence` as an implementation of Kang and Han [3] does the following steps in order to get information about the position of similar sentence between two documents. Suppose we compare *documentA* and *document1*.

1. Content of *documentA* and *document1* are broken down into sentences. Every sentence is stored in `ArrayList` of String.
2. Compare each sentence in *documentA* with sentences in *document1*, if a sentence is exactly the same, the index of the sentence in *documentA* and *document1* are stored in `ResultSentences` object. Besides storing index of sentence, we also store similarity pattern in this object.
3. If the sentences is not "same exactly", the sentence in document A and document 1 are broken down into words and are stored in `ArrayList` of String.
4. Trimming words in document A
5. Compare words in document A with words in document 1. If `ArrayList` of words in document 1 contains all words in `ArrayList` of words in document A, index of these sentences are stored in `ResultSentences`. If the number of words in document 1 > the number of words in document A before trimming, then the similarity pattern is word insertion; otherwise it is word deletion
6. If `ArrayList` of word in document 1 does not match any words in `ArrayList` of document A, perform stemming the words in both documents. Save the stemming result in `ArrayList stemWord`
7. Compare `stemWord` document 1 with `stemWord` document A. If there a is similar word between them, the type of similarity is "structure change", then we save

the index of sentences and its similarity type in ResultSentence

We use `readSentence(String)` method to break down content of document into sentences. End of a sentences is a ".", "!", or "?". We get the index of this metacharacter using `indexOf(metacharacter)` method. If a metacharacter is not found, this method will return -1.

Since we do not know which is the first metacharacter, all metacharacter indexes are compared using `minIndex(int, int, int)` method. String from index 0 until reaching the smallest metacharacter index (not "-1") considered as a sentence. Then this sentence is stored in an `ArrayList` named `sentence`. We'll continue reading document to the end of file.

Trimming is a process for removing words that do not have significant meaning. For example, "di", "ke", "dari", "pada". Class Trimmer has a list of word that can be trimmed. This class has a method, `getTrimmedWord(String)`, returning a Boolean value. If the word does not exist in the trimmed word list, it will return a value of true. This method is called inside `checkSentences` method. Each word is checked through `getTrimmedWord()`, if the return value is true, then we remove that word from `ArrayList`.

Stemming is a process removing affixes from word variant to get the common root form. Here are the steps for stemming:

1. Check the word in the dictionary. If it does not exist, find character "-".
2. If character "-" is found, compare the word before and after that character. If both words are the same, one of them is a common root.
3. Otherwise, check word length. If the word lengths are the same, it was a common root. For example, "bolak-balik".
4. If the words and their lengths are different, do stemming for both. If stemming results are different, the original word is sent as a return value.
5. When there is no character "-" in a word, continue to get the affixes.
6. When we find affixes, removes them until we get the common root.
7. If we do not find the affixes, return the original form.

Inside Class `PlagiarismPattern`, if a word does not exist in the dictionary and return value of `getRootWord()` method does not change the word form, then this word will be added into the dictionary. We add a word into the dictionary using `addKata()` method. This method is `Kamus()` class.

Here is the source code for adding a word.

```
public class Kamus {
    FileReader rd = new FileReader("kamus.txt");
    BufferedReader reader = new
BufferedReader(rd);
    StringBuilder sb = new StringBuilder();
    String currLine = null;
public Kamus() throws IOException{
        while ((currLine = reader.readLine()) !=
null) {
          sb = sb.append(currLine.concat("\r\n"));
        }
    }
    public void addKata(String kata){
        try{
            FileWriter ryt=new
FileWriter("kamus.txt");
            BufferedWriter out=new
BufferedWriter(ryt);
            out.write(sb.toString());
            out.write(new
Date().toString().concat("\r\n"));
            out.write(kata.concat("\r\n"));
            out.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

6.2 Testing

This prototype was able to compare two documents and get information about the index of similar sentences and their similarity pattern. Unfortunately, it takes a long time to run this application. Table 2 and table 3 show comparisons of the number of sentences, words, and execution time if we are comparing two documents in two different computers.

Table 2. Execution Time Comparison – Computer 1

First Document		Second Document		Time
Total Sentences	**Total Words**	**Total Sentences**	**Total Words**	
27	227	27	227	16"
54	454	54	454	47"
540	4540	540	4540	59'41"

Table 3. Execution Time Comparison – Computer 2

First Document		Second Document		Time
Total Sentences	**Total Words**	**Total Sentences**	**Total Words**	
27	227	27	227	53"
54	454	54	454	2'30"
540	4540	540	4540	167'47"

The specification of Computer 1 is Processor Intel Core 2 duo 7300 2GHz, RAM 2 GB, and Windows XP OS. Specification for computer 2 is, Processor Intel Dual Core T2080 1,73 GHz, RAM 1GHz, and Windows Vista Home Premium OS.

7. Example of Result

To clarify and better description of the prototype performance, the following Figure 6 and Figure 7 show some examples of the prototype output.

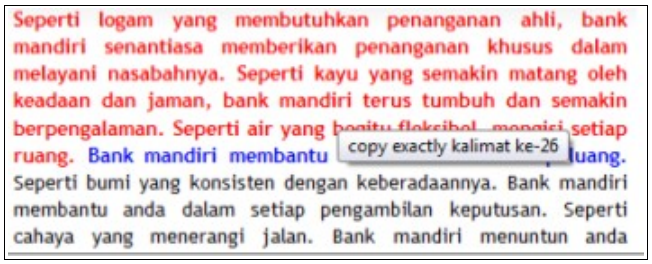

Figure 6 Example of highlighted sentence and its pattern of plagiarism

Kalimat ke	Judul dokumen mirip	Kalimat ke	Type
1	file2.doc	1	exact
3	file2.doc	10	exact
4	file2.doc	11	insertion
5	file2.doc	12	exact
6	file2.doc	11	deletion
8	file2.doc	13	insertion
7	file2.doc	14	exact
8	file2.doc	15	exact

Figure 7 Report generated by prototype showing the index of the sentence and the type of match

8. Conclusion and Future Works

A prototype of a system for checking plagiarism on a document has been successfully developed and tested, even though the performance still need further improvement. The prototype has been capable of detecting several plagiarism patterns: exact match, word removal, word insertion, structure change and limited word change (synonym). From the testing we conclude that the number of sentences and words in a document determines execution time. There are other factors that also impact execution time, including similarity type and hardware of computer. If the similarity pattern is “copy exactly”, it consumes less time than other patterns. It is because when a sentence is recognize as “copy exactly”, application does not necessarily stem each word in that sentence. Similarity pattern “word change” consumes most time. It is because the application must execute the stemming of words and find the synonym for each word in the sentence.

This prototype only compares and is tested using two documents. However, nowadays, huge amounts of documents are open for access on the internet and thus susceptible to plagiarism. Further research on using documents available on Internet is necessary.

Execution time is the biggest issue in this prototype and further optimization is much needed. To reduce the process time, from software point of view, the prototype can be revised in terms of stemming process by not repeating words that have been stemmed before. On the other hand, from hardware point of view, the implementation may involve a paralel processing using more than one computer (cluster, grid) to speed up the process.

9. REFERENCES

[1] Hireka Eric,"*Prof. Banyu Perwita: Plagiat ini Bukan yang Pertama!*" ,Jakarta, 2010 [online] access date 14 Feb 2010 http://edukasi.kompasiana.com/2010/02/07/prof-banyu-perwita-plagiat-ini-bukan-yang-pertama/

[2]: Jelita, A. , Williams, H. , Tahaghoghi, S., "Stemming Indonesian" ,*Proceedings of the 28th Australasian conference on Computer Science*, 2005 [online] access date 10 Feb 2010 http://crpit.com/confpapers/CRPITV38Asian.pdf

[3]Kang, N., Han, S., Alexander, G.,"PPChecker: Plagiarism Pattern Checker in Document Copy Detection " ,LNCS 4188,Springer Berlin, Heidelberg, 2006 [online] access date 14 Feb 2010 http://nlp.cic.ipn.mx/Publications/2006/TSD-2006-Plagiarism.pdf

[4] Yandi M.R. and Muh. Syaifullah,"*Satpam Digital di Bulaksumur*" ,Yogyakarta, 2008 [online] access date 10 Feb 2010 http://www.www.korantempo.com/id/arsip/2008/11/03/TI/mbm.20081103.TI128613.id.html