JADE based Virtual Checker to Avoid Plagiarism in MOOC's

Gaurav Shah
Shah & Anchor Kutchhi
Engineering College
W.T.Patil Marg, Next to Dukes
Co, Chembur, Mumbai-88,
India

ABSTRACT
The MOOC's (Massive open online classes) have ushered in a new era of learning overcoming the boundaries of time and geography to provide high quality education to masses who cannot afford university education. The major drawback however is that although they have best tried to capture classroom environment but facets such as teacher-student interaction and the usefulness of assignments as a source for enhanced understanding of subject matter have not been captured. Also it has led to plagiarism, which not only is a serious hazard but is also staining the otherwise excellent image the MOOC's could have had. In this paper we propose a JADE based multi-agent system(MAS) as an effective step to solve the problem of plagiarism. The system has considered 2 types of plagiarism namely ***local*** and ***internet-based*** plagiarism. This will be used to filter out the students caught in plagiarism and students who come out clean will then have their solutions checked by using LEXICAL ANALYSIS to determine the styling and subjected to tests as well as code-checker to check for logical and syntactical correctness.

General Terms
Pattern matching, Lexical analysis, local, internet-based, plagiarism, JADE, agents

Keywords
Virtual Checker, MOOC's

1. INTRODUCTION
The MOOC's try to capture the class room environment. It consists of lectures, in-class quizzes followed by an assignment based on the subject matter that was taught during the week. The basic flaw that these systems have is that the student cannot interact directly with the professor and hence has no source to solve his doubt. Another facet that is equally important is the assignment checking part. The assignment is aimed at improving the student's knowledge on subject matter but since MOOC's lack a fool-proof plagiarism checking system, student take advantage of this drawback and try to get the right solution. In this process they fail to realize that it is ultimately affecting their understanding of the subject. The plagiarism is divided into two categories: ***Internet-Based plagiarism*** and ***Local Plagiarism.***

INTERNET-BASED PLAGIARISM :- This is the category of plagiarism in which the person involved obtains the solution from the internet by using certain problem statements or certain keywords in the problem statement in the search query on an internet-search engine to arrive at an exact solution to the problem and present it as their own.

LOCAL PLAGIARISM :- This is the category of plagiarism in which the person involved copies the solution obtained by another student involved in the course and presents the solution as his own.

In order to obtain the "Proof of concept" a survey was conducted in which 20 students of a college classroom were given a problem and were told that the solutions are available on the internet and were asked to present solution in 4 days time with the condition that the sooner they answer the higher grade they get. Only 3 students submitted original solutions while 12 of them were involved in Internet-based plagiarism and the rest in local plagiarism.

The proposed system is a JADE based multi-agent system (MAS) that is used to check plagiarism.. To illustrate the entire system, though-out the paper an example of a dummy Programming course offered by the MOOC is taken. Initially the students will submit their solutions to the MOOC server. The solutions submitted will then be checked for both the types of plagiarism.

A .Internet-based plagiarism
In order to deal with the Internet-Based Plagiarism, the proposed system assumes subscription to google's web search service. The solution submitted by the student would be divided into smaller parts according to the different parts required to be implemented by the student for the assignment. Each part of the solution would then be converted into a sequence of Strings which will be used as a search query . The google search index uses each String in the search query as a key and the resulting URL's or Web-Pages as a value stored against that key. As we have an access to all of the search corpus, we can use that to locate the source from where the code was copied if at all an exact and complete match exists otherwise we move on to the next solution. This procedure is followed for each and every solution that is submitted.

B. Local Plagiarism
In order to deal with the Local Plagiarism, the system makes use of the Boyer-Moore algorithm. Boyer-Moore algorithm is a text-matching algorithm best-suited for the application because the application is going to have large sized pattern strings and the Boyer-Moore algorithm goes faster with the increase in the length of the pattern string. Also the pre-processing time and the matching time are significantly lower as compared to the other text matching algorithm which would result in a significant amount of time saved considering the large number of times the string matching needs to be carried out in order to determine whether the local plagiarism exists among a set of solutions. In this case the pattern string is going to be the token sequence associated with sequence of

string that was being used as a search query in the Internet Based plagiarism and the text string is going to be the token sequence corresponding to the entire solution of the other student. Since the solution of student under consideration(e.g student A) has been divided into parts, they can be distributed over the network to other JADE agents in the cluster and can be simultaneously matched against the same other student's solution(e.g all will match their part against solution of student B) which also will be passed over the network by the co-ordinator agent. If a match is found then the agent must return a 1 and if not the agent must return a 0. The co-ordinator will then take a sum of all the results and then divide it by the total no of parts to obtain a value in between 0 and 1. If that value is greater than the plagiarism threshold set (typically around the 0.6 mark) then both student A and student B are involved in plagiarism else they are not. This procedure will be repeated for each student and their solution will be matched in a similar fashion against the solution of every other student who has submitted the solution.

2. JADE FRAMEWORK

Agent-oriented applications are an exciting blend of artificial intelligence and distributed system techniques which model the various structural components as agents. Each agent is autonomous and has the ability to communicate with other agents to achieve personal and communal goals. Such applications employ an architectural model which enables each agent to communicate with every other agent in the application. They are generic in nature and can be easily extended to model all the components in a given domain and thus can help in capturing the workflow of that domain by simply allowing the developer to concentrate on business logic.

JADE (Java Agent DEvelopment Framework) simplifies the implementation of multi-agent systems through a middle-ware that claims to comply with the FIPA specifications and through a set of tools that supports the debugging and deployment phase. The agent platform is truly independent as it can be distributed across machines which don't even need to run the same OS. JADE is a software platform written in Java that provides basic middleware-layer functionalities which are independent of the specific application and which simplify the realization of distributed applications that exploit the software agent abstraction. Each Agent being FIPA compliant will exhibit qualities such as proactiveness, responsiveness, autonomy, co-operation, mobility, adaptability and rationality. JADE-based systems are generally loosely coupled and message passing among agents is asynchronous. Every agent has a thread dedicated to it which controls its life-cycle and is used to perform autonomous tasks by the agent.

2.1 JADE Architecture

A JADE platform consists of a runtime environment (also called containers) that can be distributed over the network and provides all the services needed for hosting and executing agents. A special container, called the *main container* must be the first container to start and all other normal containers register with it as soon as they start and must therefore know the main container's host address and port. A diagram showing the typical architecture of the JADE platform is shown in Figure 2. Starting another main container elsewhere in the network constitutes a different platform to which new normal containers can possibly register. The main container manages the container table (CT), which is the registry of object references and transport addresses of all container nodes in the platform; manages the global agent descriptor table GADT), which is the registry of all agents present in the platform, including their current status and location; and hosts the AMS (Agent Management Service) and DF (Directory Facilitator), the two special agents that provide the agent management service and the default yellow page service of the platform respectively. The AMS will be used in our system.

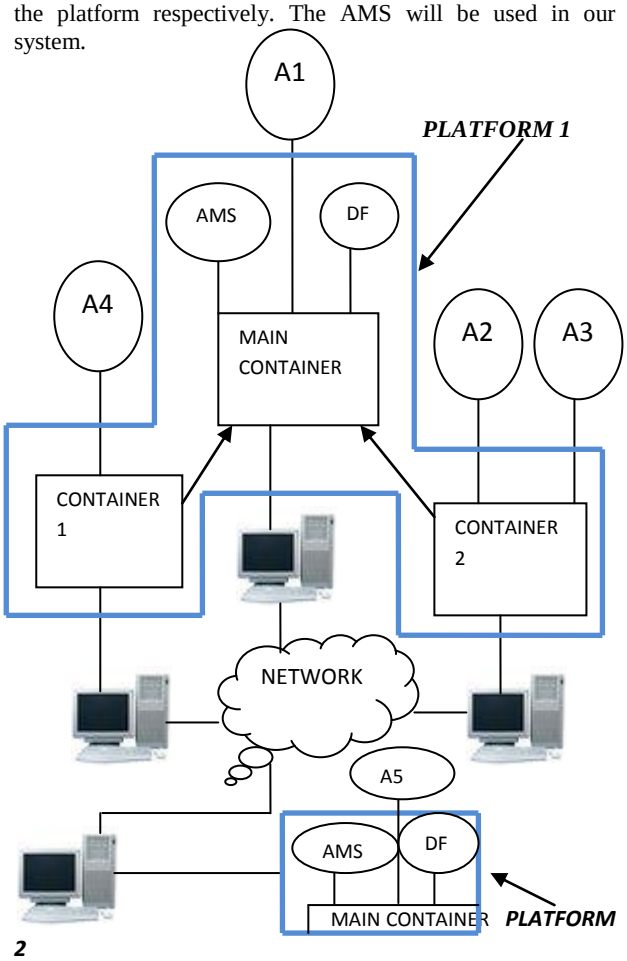

Fig 1: JADE architecture – Relationship between containers and platform

Every other container manages their LADT (Local Agent Descriptor Table) and cache of GADT (Global Agent Descriptor Table). Whenever the agent wants to communicate with the other agent the LADT is searched first. If a match is not found then the main container's GADT is searched and is cached for future usage. Agents in JADE are identified by a globally unique name called an Agent Identifier (AID) consisting basically of the agent‟s local name and its addresses (usually inherited from the platform). Each agent can communicate transparently regardless of their actual location: same container (e.g. A2 & A3 in Figure 1), different containers in the same platform (A1 & A2) or different platforms (A4 & A5) if they know other‟s agent identifier.

2.2 Message Transport Service

JADE consists of a Message Transport Service (MTS) that manages all message communications within platform as well as between platforms. To promote interoperability between different non-JADE platforms, all standard Message Transport Protocols (MTPs) defined by FIPA are implemented by this service. Each MTP comprises of standard encoding of the message envelope and definition of transport protocol. HTTP-MTP is started by default when main container is initialized. The Normal containers have no default MTP. This creates a server socket on the main

container host and listens for incoming connections over HTTP at the URL specified. Whenever an incoming connection is established and a valid message is received over that connection, the MTP routes the message to its final destination which, in general, is one of the agents located within the distributed platform

2.3 Agent Communication

The communication paradigm adopted in JADE is the asynchronous message passing. Each agent has a message queue where the JADE runtime environment posts messages sent by other agents similar to a mailbox; whenever a message is posted in the message queue, the receiving agent is alerted. It's upto the programmer to decide if and when the agent actually picks up the message from the queue for further processing. This procedure is illustrated in Figure 2.

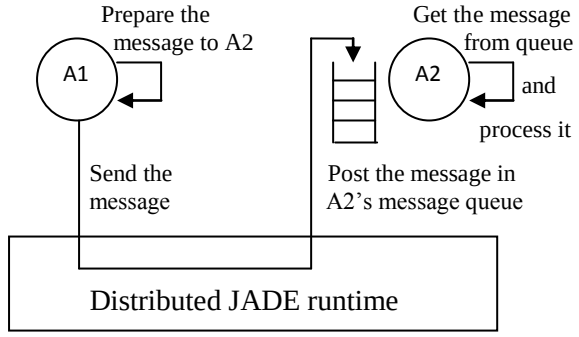

Fig 2: JADE asynchronous message passing paradigm

JADE uses FIPA compliant ACL message structure specifications and has fields such as the sender, list of receivers, communicative act (REQUEST, INFORM, PROPOSE etc), content, content language and ontology.

3. DESIGN OF PROPOSED SYSTEM

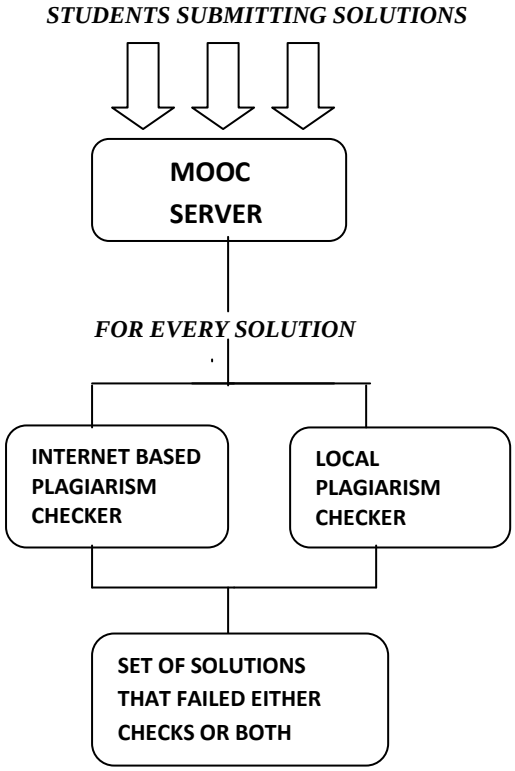

Fig 3: Overall system design

The Design of the proposed system is pretty straightforward as can be seen in figure 3**.** The students upload their submissions to the MOOC server. The internet based and local plagiarism checker then pick each solution from the submitted set and check it for respective types of plagiarism. Those solutions that pass these tests are forwarded to the next stage of processing and those solution that don't are outputted as the result of the plagiarism checker system. The internet based checker is discussed in section 3.1 and local plagiarism checker in section 3.2. This is followed by a discussion of Boyer Moore algorithm which is used in both checkers.

3.1 Internet-based plagiarism checker

The internet based plagiarism checker is implemented as shown in the figure:

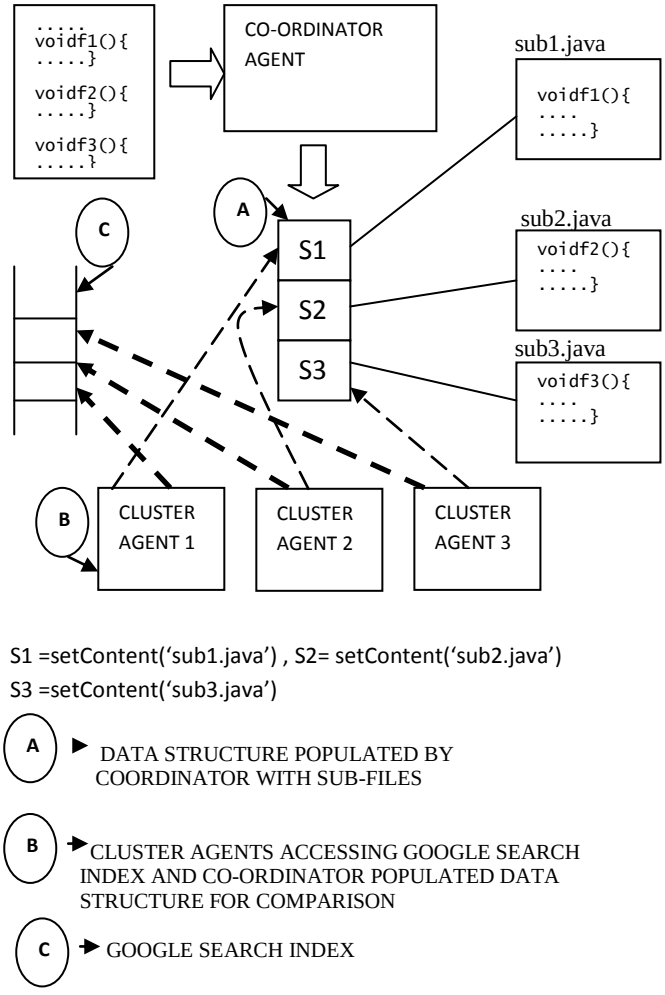

Fig 4: Internet based plagiarism checker

The co-ordinator agent splits up the solution file according to the number of functions to be implemented such that each function is 1 file whose content is stored in the form of a string in the data structure shown in the figure. Each cluster agent then accesses this data structure once it receives a message from the co-ordinator. The cluster agent then also queries the google search index to obtain the string in which the pattern string i.e. the string form of the function is to be searched using the Boyer-Moore Algorithm.

3.1.1 The Google Search Index Generation

Google runs on a distributed network of thousands of low-cost computers and can therefore carry out fast parallel processing. Google has three distinct parts:

A. GoogleBot

Googlebot is Google's web crawling robot, which finds and retrieves pages on the web and hands them off to the Google indexer. In reality Googlebot doesn't traverse the web at all.

Googlebot consists of many computers requesting and fetching pages much more quickly than you can with your web browser. In fact, Googlebot can request thousands of different pages simultaneously. Googlebot finds pages in two ways: through an add form and through finding links by crawling the web.

B. Search Indexer

Googlebot gives the indexer the full text of the pages it finds. These pages are stored in Google's index database. This index is sorted alphabetically by search term, with each index entry storing a list of documents in which the term appears and the location within the text where it occurs.
To improve search performance, Google ignores (doesn't index) common words called *stopwords(*such as *the, is, on, or, of, how, why,* as well as certain single digits and single letters.

C. Query Processor

The query processor has several parts, including the user interface (search box), the "engine" that evaluates queries and matches them to relevant documents, and the results formatter.

PageRank is Google's system for ranking web pages. A page with a higher PageRank is deemed more important and is more likely to be listed above a page with a lower PageRank.

Google considers over a hundred factors in computing a PageRank and determining which documents are most relevant to a query, including the popularity of the page, the position and size of the search terms within the page, and the proximity of the search terms to one another on the page.

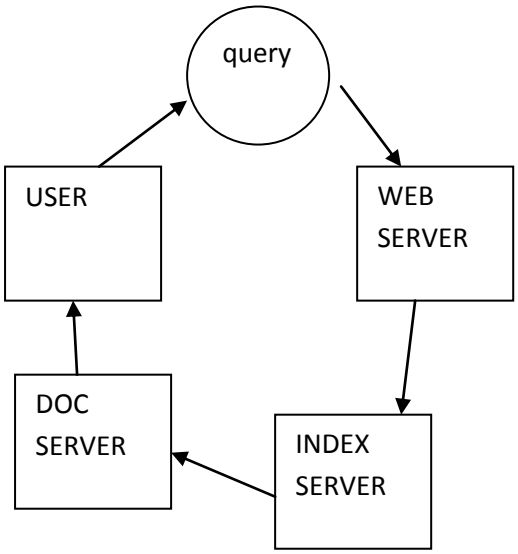

Fig 5:The Google Web Search process

1 – The Web Server sends query to index server.Content inside the index server is identical to index at the nack of the book.It tells which pages contain the words that match a term.

2 – The query travels to the doc server that actually retreives the stored document. Snippets are generated to describe each search result.

3 – The search result is returned to user in fraction of second

3.1.2 The m4 macro preprocessor for file split

m4 is a general purpose macro processor designed by Brian Kernighan and Dennis Ritchie. m4 is an extension of an earlier macro processor **m3**, written by Ritchie for the AP-3 minicomputer.

m4 has many features like a free-form syntax, rather than line based syntax, a high degree of macro expansion (arguments get expanded during scan and again during interpolation), text replacement, parameter substitution, file inclusion, string manipulation, conditional evaluation, arithmetic expressions, system interface etc.

Unlike most earlier macro processors, m4 does not target any particular computer or human language.

This m4 will be used by the co-ordinator agent to split the code files according to the number of functions so there will be as many files as there are functions. A practical example of file splitting is as follows.

Suppose we have the following Java class:

```
// File Foo.java

class Foo{
  // Some properties of the Foo class

  public int aProperty1;
  public int aProperty2;
  public int aProperty3;

  // Some methods of the Foo class
  public void apple()
  {
    // Method body of apple method goes here
  }

public void banana()
  {
    // Method body of banana method goes here
  }

public void carrot()
  {
    // Method body of carrot method goes here
  }
}
```

If we make the following changes to the Foo class:

```
// File class-Foo.java
m4_changecom()
m4_changequote(,)
class Foo
{
  // Some properties of the Foo class
```

```
  public int aProperty1;
  public int aProperty2;
  public int aProperty3;

  // Some methods of the Foo class
  m4_include(class-Foo-method-apple.java);
  m4_include(class-Foo-method-banana.java);
  m4_include(class-Foo-method-carrot.java);
}
```

The first two lines are needed to change the behaviour of m4 and can be ignored by users of this system. Note that the purpose of the semicolons after the include directive is so that indentation works correctly inside Emacs. If you don't use Emacs then you can avoid those semicolons. Using this system allows us to have methods of the Foo class in separate files, like so:

```
// File class-Foo-method-apple.java
public void apple()
{
  // Method body of apple method goes here
}

// File class-Foo-method-banana.java
public void banana()
{
  // Method body of banana method goes here
}

// File class-Foo-method-carrot.java
public void carrot()
{
  // Method body of carrot method goes here
}
```

To get the building of the preprocessor to work, you will need to add the following lines to your Makefile.

```
%.java: class-%.java class-%-method-*.java
      m4 -P class-$*.java >$*.java

.PRECIOUS: %.java
```

The above example illustrates the use of m4 macro. Since we are using it for a programming assignment. The "m4_include" statements can be put in the assignment for all the functions that need to be completed in the assignment before its passed on to the student so that when the completed programs returns we can straight away split them into files. The call to the preprocessor will be initialized from the setup method of the coordinator JADE agent

```
import jade.core.Agent;
public class CoordinatorAgent extends Agent {

protected void setup() {
 // call to the preprocessor to split the files
 // after splitting convert the content of each file into a string and store them in a data structure for future    access
   }
}
```

3.1.3 The action() method and message passing among JADE agents.

A behaviour represents a task that an agent can carry out and is implemented as an object of a class that extends ***jade.core.behaviours.Behaviour***. The
Agent class, which must be extended by agent programmers, exposes two methods: ***addBehaviour(Behaviour)*** and ***removeBehaviour(Behaviour)***, which allow to
manage the ready tasks queue of a specific agent. Notice that behaviours and sub-behaviours can be added whenever is needed, and not only within Agent***.setup()*** method. A scheduler, implemented by the base Agent class and hidden to the programmer, carries out a round-robin non-preemptive scheduling policy among all behaviours available in the ready queue, executing a Behaviour-derived class until it will release control (this happens when ***action()*** method returns). If the task relinquishing the control has not yet completed, it will berescheduled the next round. A behaviour can also block, waiting for a message to arrive. In detail, the agent scheduler executes ***action()*** method of each behaviour present in the ready behaviours queue; when ***action()*** returns, the method ***done()*** is called to check if the behaviour has completed its task. If so, the behaviour object is removed from the queue.

The coordinator agents ***action()*** method will be used to convert the content of each of the file obtained by splitting the Solution code file, into a string and passing that string to the cluster agents which will use it for matching against the search index.

```
public class CoordinatorBehaviour extends Behaviour {

public void action() {
// access the data structure of subfiles created in the
   setup() method of agent.


//distribute Strings from the datastructure to the cluster agents
}

public boolean done() {
return true;
}
}
…..
import jade.core.Agent;
public class CoordinatorAgent extends Agent {
    protected void setup() {
 …..
addBehaviour (new CoordinatorBehaviour)
  }
}
```

3.1.4 Inter Agent Communication

Messages exchanged by JADE agents have a format specified by the ACL language defined by the FIPA international

standard for agent interoperability. This format comprises a number of fields and in particular:

• The ***sender*** of the message
• The list of ***receivers***
• The communicative intention (also called "***performative***") indicating what the sender intends to achieve by sending the message.
• The ***content*** i.e. the actual information
• The content ***language*** i.e. the syntax used for expression
• The ***ontology*** i.e. the vocabulary of the symbols used in the content and their meaning

A message in JADE is implemented as an object of the ***jade.lang.acl.ACLMessage*** class that provides ***get*** and ***set*** methods for handling all fields of a message.

3.1.4.1 Sending Messages

So to send a message to one agent in a cluster the co-ordinator will communicate as follows:

```
{…
// setting the performative for the message
ACLMessage msg =new ACLMessage(ACLMessage.INFORM);
msg.addReceiver(newAID("nickname", AID.ISLOCALNAME));

msg.setLanguage("English");
msg.setOntology("Assignment-Ontology");
msg.setContent("string from the datastructure ");
send(msg);
}
```

3.1.4.2 Receiving Messages

As mentioned above the JADE runtime automatically posts messages in the receiver's private message queue as soon as they arrive. An agent can pick up messages from its message queue by means of the ***receive()*** method. This method returns the first message in the message queue (removing it) or ***null*** if the message queue is empty and immediately returns.

```
{
 ACLMessage msg = receive();
 if (msg != null) {
// Process the message
// compare the content of message obtained with each of the google search indices to find a match.
 }
}
```

Since we have already subscribed to the Google services, so we will be provided with the search index. We need to just search the content of message sent by the co-ordinator and search the string returned in the content obtained by querying the google search index. If a match is found the student is involved in plagiarism else we move on to the next function of the assignment or to the next student's solution and the same procedure would be repeated.

3.2. LOCAL PLAGIARISM CHECKER

The local plagiarism checker is implemented as shown in the Fig 6.

As can be seen in the figure the initial part upto the splitting of the solution into individual files containing 1 function is same as that in the internet- based plagiarism checker. After splitting, each file is given to the Lexer which converts the File content into a sequence of tokens.

If 2 expressions are written in the same language and in the same manner then the token sequence generated by the lexer will also be the same. Since local plagiarism involves direct copying of the content of the solution of another person so the above mentioned concept is used.

Solution of all other students stored in the database is then converted into token sequences and then the token sequence of a function of the earlier student is searched in the token sequence corresponding to the complete solution submitted by every other student. If a match is found then both the students are involved in the act of plagiarism.

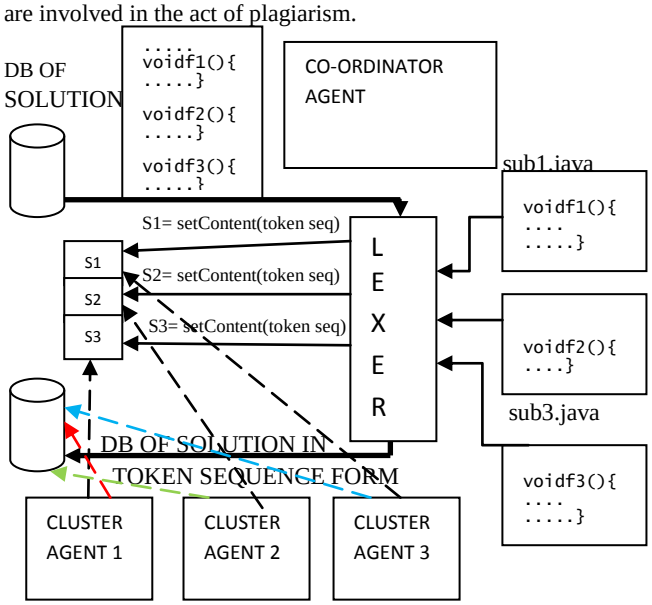

Fig 6: Local Plagiarism Checker

3.2.1 Token-Sequence output of Lexer

A lexical analyzer is a program that transforms a stream of characters into a stream of 'atomic chunks of meaning', as shown in the figure below:

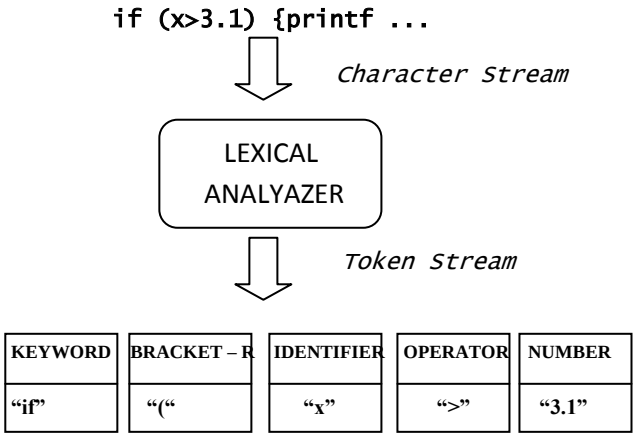

Fig 7: Lexical analyzer concept

As a result of the lexical analysis process the list of 'atomic chunks of meaning', so called 'tokens', prepare the interpretation on some higher level. Each token consists at least of a type identifier. Additionally, some parameters about the matching 'lexeme' might be stored in the token.

So using lexer (lexical analyzer) we obtain the token sequence output for 1 of the files (i.e. any 1 function) from the solution of student A and for the entire solution of student B. This sequence of Tokens is then converted into a String. The string corresponding to student A's function is the pattern string and that corresponding to the complete solution of student B is the text string in which the pattern string will be searched using the Boyer-Moore Algorithm.

4. THE BOYER-MOORE ALGORITHM

The **Boyer–Moore string search algorithm** is one of the most efficient algorithm which over the years has proven to be the benchmark for String searching algorithms. The algorithm preprocesses the pattern string being searched for , but not the text string in which the search is to be carried out. It is thus well-suited for applications in which the text does not persist across multiple searches. It uses information gathered during the preprocess step to skip sections of the text, resulting in a lower constant factor than many other string algorithms. In general, the algorithm runs faster as the length of pattern increases.

Some of the key features are:

- performs the comparisons from right to left;
- preprocessing phase in $O(m+\sigma)$ time and space complexity;
- searching phase in $O(mn)$ time complexity;
- 3n text character comparisons in the worst case when searching for a non periodic pattern;
- $O(n / m)$ best performance.

The algorithm scans the characters of the pattern from right to left beginning with the rightmost one. In case of a mismatch (or a complete match of the whole pattern) it uses two precomputed functions to shift the window to the right. These two shift functions are called the ***good-suffix shift*** (also called matching shift and the ***bad-character shift*** (also called the occurrence shift).

Assume that a mismatch occurs between the character $x[i]=a$ of the pattern and the character $y[i+j]=b$ of the text during an attempt at position j. Then, $x[i+1 .. m-1]=y[i+j+1 .. j+m-1]=u$ and $x[i] != y[i+j]$. The good-suffix shift consists in aligning the segment $y[i+j+1 .. j+m-1]=x[i+1 .. m-1]$ with its rightmost occurrence in x that is preceded by a character different from $x[i]$ (*see Fig 8*)**.**

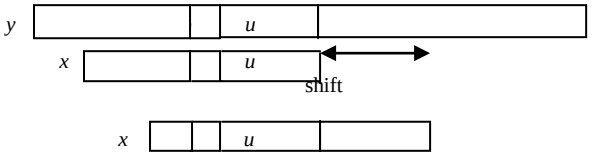

Fig 8: The good-suffix shift, *u* re-occurs preceded by a character *c* different from *a*

If there exists no such segment, the shift consists in aligning the longest suffix v of $y[i+j+1 .. j+m-1]$ with a matching prefix of x (*see Fig 9*).

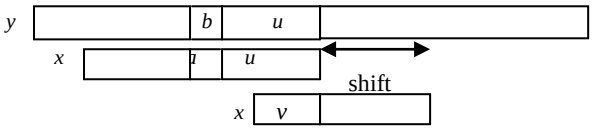

Fig 9: The good-suffix shift, only a suffix of *u* re-occurs in *x*.

The bad-character shift consists in aligning the text character $y[i+j]$ with its rightmost occurrence in $x[0 .. m-2]$. (*see Fig 10*)

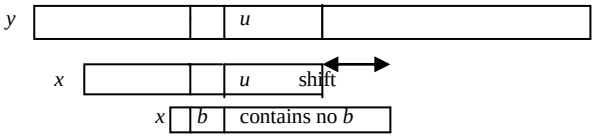

Fig 10: The bad-character shift, *a* occurs in *x*

If $y[i+j]$ does not occur in the pattern x, no occurrence of x in y can include $y[i+j]$, and the left end of the window is aligned with the character immediately after $y[i+j]$, namely $y[i+j+1]$ (*see Fig 11*)

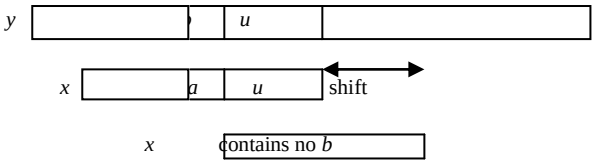

Fig 11: The bad-character shift, *b* does not occur in *x*

Note that the bad-character shift can be negative, thus for shifting the window, the Boyer-Moore algorithm applies the maximum between the the good-suffix shift and bad-character shift. More formally the two shift functions are defined as follows. The good-suffix shift function is stored in a table *bmGs* of size $m+1$.

So the token sequence for 1 function of solution A is searched in the token sequence of the entire program submitted by another student B. However there can be an element of chance involved where 2 students who might be thinking in the same way may come up with identical solution. To completely eliminate that factor we determine a threshold value of 0.6 for the degree of plagiarism. In order to compute the degree of plagiarism what is done is that each cluster agent assigns a 1 when token sequence corresponding to student A's function is

found in token sequence corresponding to the entire solution of any other student B and 0 when it is not found. This result is returned to the co-ordinator agent which then sums it up and divides it by the total number of functions to be implemented.

So for example if 2 student student A and Student B's solution are being compared and there are *N* functions which need to be implemented then if

X => token sequence for a function of student A's solution
Y=> token sequence for the entire solution of student B
∩=> Boyer-Moore applied, where 1st operand is searched in 2nd operand
DOP => Degree of Plagiarism

$$DOP = \left(\left(X \cap Y \right)_{func1} + \ldots + \left(X \cap Y \right)_{funcN} \right) / N$$

This will result in a DOP value between 0 and 1 which is evaluated according to the

$DOP > threshold$ => case of plagiarism
$DOP < threshold$ => not a case of plagiarism

Thus this will detect the required level of plagiarism taken into consideration the occasional chance that a part of code might be similar but certainly the 2 codes cannot be similar entirely.

The entire Boyer Moore algorithm will be implemented in the message processing section of the cluster agent as shown in section (3.1.4.2)

4. CONCLUSION

The proposed system is filter for plagiarism which provides an efficient check for both types of plagiarism. The file splitting mechanism is appropriate as there is no data loss involved and extreme care is taken to avoid the same. The system aims at helping the MOOC's to achieve its goal of eradicating limitations to achieve quality education and also help the students to explore the subject matter by eradicating plagiarism techniques, which will benefit the student in the long run.

5. FUTURE WORK

The proposed plagiarism system design is expected to be quite effective for the MOOC's in particular and the academic domain in general. However plagiarism is rampant in other domains like business enterprises etc. So we aim to develop a generic system that, when configured can map to any required domain and act as a plagiarism checker for that field. The internet based plagiarism checker currently uses exact text matching which also needs to be handled to check plagiarism on part of those students who copy from the internet but make slightest of modifications and call it their code. Also if the file splitting mechanism, which is currently handling 1 file at a time, is modified then it can be competent enough to handle multiple files which will increase the efficiency of the system. Thus the system will be much more robust, scalable, and will help in modeling any domain effectively for plagiarism checking.

6. REFERENCES

[1] Multi-Agent Systems in a Computational Environment of Education: A Chatterbot Case Study by, André F. M. Batista, Maria G. B. Marietto, Gislene C. O. Barbosa, Robson S. França and Emerson A. Noronha Federal University of ABC – Center of Mathematics, Computation and Cognition Santo André, São Paulo - Brazil

[2] Impact of Multi-Agents in Hospital Environment by, Mijal Mistry,Dr. Dipti Shah

[3] Building a Truly Distributed Constraint Solver with JADE by, Ibrahim Adeyanju School of Computing, The Robert Gordon University, Aberdeen, UK

[4] Designing scenario in Virtual Knowledge Communities using the JADE Framework by, Studienarbeit:von, Julien Subercaze

[5] Compilers - Principles Techniques and Tools by Alfred Aho - Monica Lam- Ravi Sethi- Jeffrey Ullman - Second Edition

[6] Developing Multi-Agent Systems with JADE - Fabio Luigi Bellifemine, Giovanni Caire, Dominic Greenwood

[7] jade tutorial jade programming for beginners, Copyright (C) 2009 Telecom Italia S.p.A.

[8] JADE architecture (www.jade.tilab.com)

[9] Lexical analyzer (http://www.cs.nyu.edu/courses/fall06/G22.2130–001/class–notes.html)

[10] pattern matching algorithms (http://bIochem218.stanford.edu/Projects%202007/Ng.pdf; http://cs.fit.edu/~wds/classes/algorithms/Text/text)

[11] m4 macro for file splitting (http://davin.50webs.com/research/2008/sjcfimsf.html)

[12] Boyer-Moore algorithm (http://www–igm.univ–mlv.fr/~lecroq/string/node14.html#SECTION00140)