




Article

Online Judging Platform Utilizing Dynamic Plagiarism Detection Facilities [†]

Fariha Iffath ¹, A. S. M. Kayes ^{2,*} , Md. Tahsin Rahman ¹, Jannatul Ferdows ¹, Mohammad Shamsul Arefin ^{1,*}  and Md. Sabir Hossain ¹ 

¹ Department of Computer Science and Engineering, Chittagong University of Engineering and Technology, Chittagong 4349, Bangladesh; farihaiffath23@gmail.com (F.I.); tahsinrahman@protonmail.com (M.T.R.); kfoozminus@gmail.com (J.F.); sabir.cse@cuet.ac.bd (M.S.H.)

² Department of Computer Science and Information Technology, La Trobe University, Bundoora, VIC 3086, Australia

* Correspondence: a.kayes@latrobe.edu.au (A.S.M.K.); sarefin@cuet.ac.bd (M.S.A.)

[†] This paper is an extended version of our earlier paper titled “A Framework for Checking Plagiarized Contents in Programs Submitted at Online Judging Systems” published in the International Conference on International Conference on Image Processing and Capsule Networks (ICIPCN 2020).



Citation: Iffath, F.; Kayes, A.S.M.; Rahman, M.T.; Ferdows, J.; Arefin, M.S.; Hossain, M.S. Online Judging Platform Utilizing Dynamic Plagiarism Detection Facilities. *Computers* **2021**, *10*, 47. <https://doi.org/10.3390/computers10040047>

Academic Editor: Paolo Bellavista

Received: 27 February 2021

Accepted: 6 April 2021

Published: 8 April 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Abstract: A programming contest generally involves the host presenting a set of logical and mathematical problems to the contestants. The contestants are required to write computer programs that are capable of solving these problems. An online judge system is used to automate the judging procedure of the programs that are submitted by the users. Online judges are systems designed for the reliable evaluation of the source codes submitted by the users. Traditional online judging platforms are not ideally suitable for programming labs, as they do not support partial scoring and efficient detection of plagiarized codes. When considering this fact, in this paper, we present an online judging framework that is capable of automatic scoring of codes by detecting plagiarized contents and the level of accuracy of codes efficiently. Our system performs the detection of plagiarism by detecting fingerprints of programs and using the fingerprints to compare them instead of using the whole file. We used winnowing to select fingerprints among k -gram hash values of a source code, which was generated by the Rabin–Karp Algorithm. The proposed system is compared with the existing online judging platforms to show the superiority in terms of time efficiency, correctness, and feature availability. In addition, we evaluated our system by using large data sets and comparing the run time with MOSS, which is the widely used plagiarism detection technique.

Keywords: online judging; dynamic plagiarism detection; fingerprint; winnowing; k -gram; rabin-karp algorithm

1. Introduction

Computer Science focuses not only on theoretical knowledge, but also on programming, software operation, and hands-on experiments. The main aim of programming labs is to show the students practical implementation of what they have learned in the classroom. Practical hands-on teaching and learning help students to self-develop their knowledge and have a clearer understanding of the concepts. Generally, in programming labs, students are asked to write computer programs that solve the given problems, and teachers judge them manually. Manual grading involves teachers going through all of the source codes submitted by students. It is time-consuming and challenging work for the teacher. Moreover, judging is not always correct. Sometimes students solve given problems in their ways. Sometimes the programming language that is used by the student is not entirely familiar to the teacher. Sometimes, a subtle bug lies deep inside the code that may have been skipped when judging manually. It is also hard to determine the efficiency of the solution that is submitted by students.

Online judges are becoming increasingly appropriate in various applications. Online judges are designed to accurately assess the source algorithm submitted by users, which is then compiled and evaluated in an environment [1]. They are categorized according to their critical objectives into structures that facilitate the advancement of the educational context by conducting competitive programming competitions and achieving programming challenges. To solve the problems of manual grading in programming labs, automated programming assignment graders, such as online judge programs [2,3] that are used in programming competitions, are adopted in a large number of universities that are developed by themselves. Optil.io [1] is used to compute complex optimization problems. The function of an online judge is beneficial to programming courses. The teachers' workloads in checking the source codes can be significantly lightened, and they can have more time to communicate with their students and answer their questions.

1.1. Motivation

Because of the rapid growth of Internet technologies, almost all sorts of information are now available on World Wide Webs and they are either lawfully or unlawfully publicly accessible. This advancement makes it possible for students to clone source code found in various websites. In the academic field, manually judging students' programming assignments is quite difficult for a judge or a teacher to review all of the source codes to find the plagiarized ones. Therefore, multiple automated systems have been introduced for general texts and source codes to detect plagiarism, such as MOSS [4], JPlag [5], and YAP3 [6]. However, existing systems have many shortcomings, such as conducting programming tasks efficiently, detecting plagiarism efficiently, scoring the codes based on partially corrected answers, good graphical interface, etc. Thus, designing a proper online judging system that involves successful plagiarism detection as well as other functionality has become a great need.

1.2. Problem Statement

This work aims to develop a complete programming problem judging platform to support an efficient plagiarism checker. An online judging platform refers to the system of automatically assessing programming problems. Plagiarism is the act of taking someone else's work or ideas and then signing them off as one's own. Students are expected to accomplish their mission independently, but students often prefer to copy it from their classmates or the internet. More than 5% students have had plagiarism-related encounters, according to scientific research [7]. Because plagiarism is common at universities, the teacher cannot fairly assess each student's work. It is challenging for teachers to identify plagiarism in source codes in contrast to general text plagiarism. If the volume of source codes to review is high, manually comparing each pair of source codes is almost inevitable for them.

1.3. Novelty and Contributions

Main novelty of our work is that it can perform efficient scoring of submitted codes by evaluating the level of accuracy and plagiarism. The contributions of our work can be summarized, as follows:

- We utilize an efficient plagiarism detection technique in our system.
- We develop a technique for evaluating the level of accuracy of each code.
- Our system is able to generate partial scoring of codes.
- We develop an user friendly interface.
- We have compared the efficiency of our system with existing systems.

1.4. Organization the Paper

The rest of the paper is organized, as follows. An overview of the related work for the proposed system is provided in Section 2. In Section 3, we described the system architecture and design of the proposed system. The implementation of the overall system is explained

in Section 4, and the experimental results and performance evaluations are presented in Section 4. The implication and limitations are presented in Section 5. Finally, we conclude the paper in Section 6, and outline the future research directions of this work.

2. Related Work

Stanford University first introduced the online judge system [8] in 1961 to support the evaluate the students coded in ALGOL. The University of Valladolid, Spain Online Judge (UVA) [9] is one of the most popular online judges. It has enormous, diverse kinds of programming questions, many of which are extremely difficult and famous. However, only UVA online judge administrators can create new contests and add new problems. Peking University, China Online Judge (POJ) [10], designed their platform mainly for the training of their students to participate in the International Collegiate Programming Contest. The University of Dhaka developed lightoj [11]. Users can host their contests there. However, lightoj does not support hosting private contests and customizing the grading system. Lightoj is no longer maintained. Accordingly, it does not support many modern programming languages, like C++11, C++14, python, etc. Codemarshal [12], which was built by Muktosoft Ltd., was used by many universities in Bangladesh for programming lab assignments and hosting both online and onsite programming contests. However, unfortunately, codemarshal is no longer maintained. Toph [13] is another Bangladeshi platform that is gaining popularity. It supports dynamic scoring systems. Kattis Online Judge [14] is the current programming contest control system choice for ACM ICPC world finals. It was developed by KTH, Royal Institute of Technology, Sweden in 2005. It supports hosting private contests, but no new problems can be created; only problems that have been already added on the kattis platform can be used. HackerRank is a technology company that focuses on competitive programming challenges for both consumers and businesses [15]. HackerRank supports a custom grading system, where each submission is scored on the accuracy of its output. HackerRank also has a platform for conducting programming labs online, named "HackerRank for Schools" [16]. It supports auto-grading and evaluating the performance of the students. They also have a plagiarism detector that flags any submission >70%, similar to other students. However, this service is not free to use.

The above-mentioned online judges have a significant limitation. It is not possible to host a contest with an active internet connection. However, there exist several platforms for arranging programming contests online. PC2 (Programming Contest Control) is the most common among them [17]. PC2 is the most popular and widely used programming contest control system. California State University developed PC2 in support of programming contest activities. PC2 was used to conduct ICPC until 2008. PC2 requires the IP of the server to be hard-coded into the configuration file of all the clients. Besides, each computer must have a java runtime environment installed. There is no way to provide a soft-copy of the problem sets, which could reduce the costs and ease the updating of any error in the description.

DOMjudge is an automated judging system to run programming contests [18]. DOMjudge is mainly used in ACM ICPC like programming contests. It supports hosting both online and offline contests. However, it is not possible to customize the grading system. Sharif-Judge [19] is a free and open-source programming judging platform. It has several features, including customized grading system, plagiarism detection in source codes, etc. It also provides sandboxing for C, C++ , Java. However, it does not provide sandboxing for modern programming languages, such as python. As a result, there is very low-level security for python in Sharif-Judge. It also does not have the facility of dynamic scoring and partial marking. URI online judge is another online judging platform that is a project developed by the computer science department at Universidade Regional Integrada (URI), Brazil, and it was first introduced at WorldComp'12 [20]. It has several features, including plagiarism detection, as well as the categorization of problems in different categories, so

that students can focus on a specific topic. However, it can not provide partial grading, dynamic scoring & top of that, it cannot be hosted locally.

3. System Architecture and Design

The proposed system comprises three main modules. These are interface development module, storage module, and judging module. All of the modules are interconnected with each other. Figure 1 depicts the overall architecture of the proposed system.

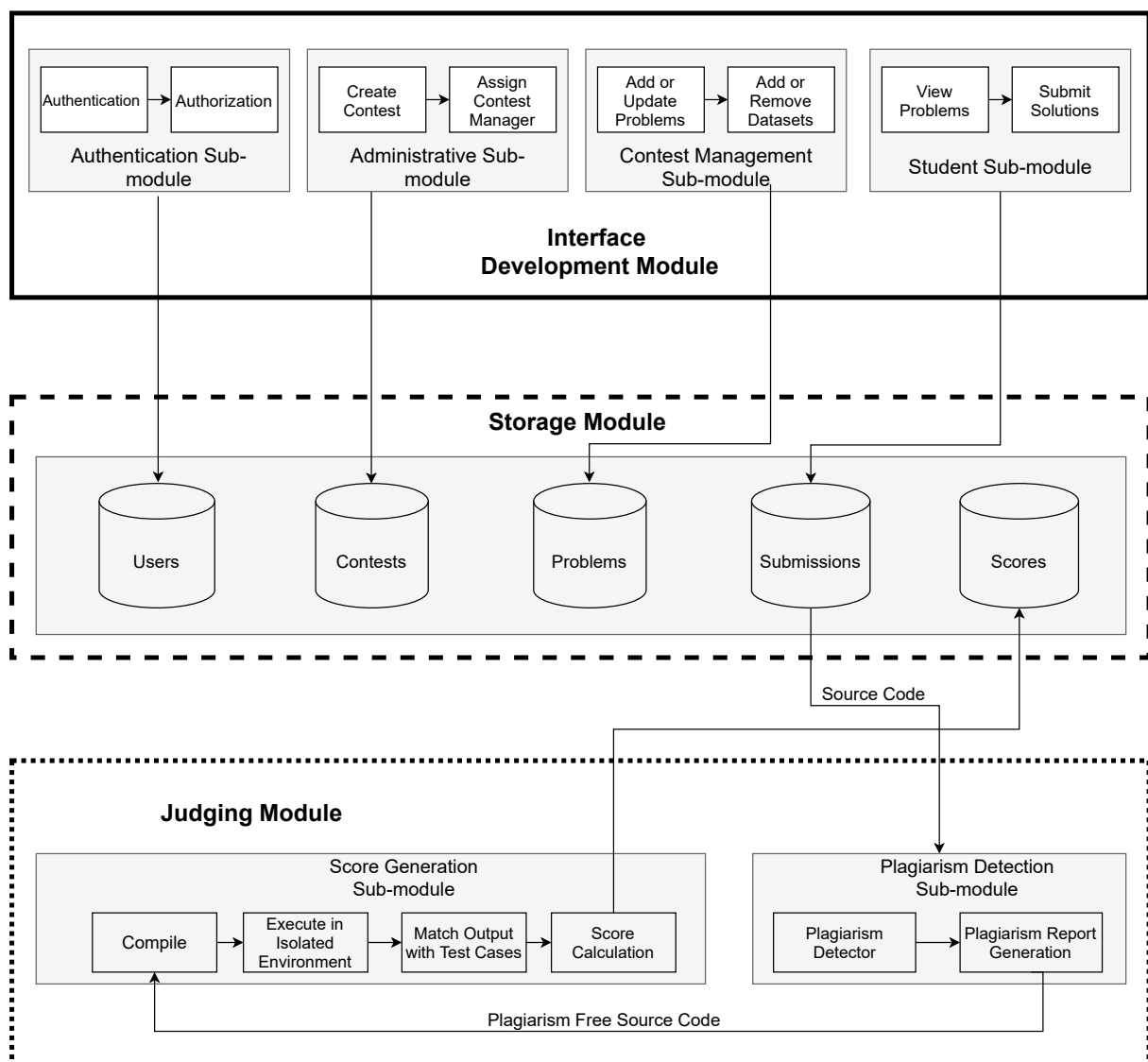


Figure 1. Overall Structure of the Proposed System.

3.1. Interface Development Module

The interface development module consists of the authentication sub-module, administrative sub-module, contest management sub-module, and student sub-module.

3.1.1. Authentication Sub-Module

For each HTTP request, we need to ensure that the current user has correct permissions to access the resource. At first, an HTTP request is sent for the authentication. If the authentication is successful, then the user will be given authorization for accessing the resources. Otherwise, an access denied error will be shown.

3.1.2. Administrative Sub-Module

The administrative sub-module denotes all of the operations performed by the administrator. The administrator has full control over the entire system. The administrator can create new contests, set the contest's name, and set the start time and duration of the contest. In addition, the administrator has the privilege of assigning other contest managers.

3.1.3. Contest Management Sub-Module

The contest management sub-module denotes the contest management procedure. This sub-module provides the facility to create, edit, and update problems and their corresponding data-sets to the authorized users

3.1.4. Student Sub-Module

Students sub-module helps the contestants to view problems and submit solutions within the allowed time. The judging module evaluates the submissions.

The source code submitted by the students is executed on the server. Therefore, we have to ensure that the system is resistant to a broad range of attacks, such as forcing a high execution time, modifying the testing environment, or accessing restricted resources during the solution evaluation process [1]. The most popular methods for avoiding such issues rely on the execution of submitted solutions in dedicated sandboxes that are managed by the online judging system [21], such as virtualization, LXC containers [22], and the Docker framework. Such approaches could significantly increase the safety and reliability of the system [23]. We used docker containers for securely running suspicious programs in our system. For each submission, the server creates a new docker container. The source code is executed in that container. Later, the container is destroyed after the execution finishes.

3.2. Storage Module

For the proposed system, a relational database system is used for storing all of the necessary information. There are five entities in the database: User, Contest, Problem, Submission, and Score. Figure 2 depicts the relationship mapping diagram. The user entity contains information regarding the users. The contest entity holds the problem details provided during the contest. The problem entity is used to store test cases to evaluate the submitted solution. The submission entity contains the source codes that were submitted during the contest period. In addition, score entity is used to store the score of the contestants.

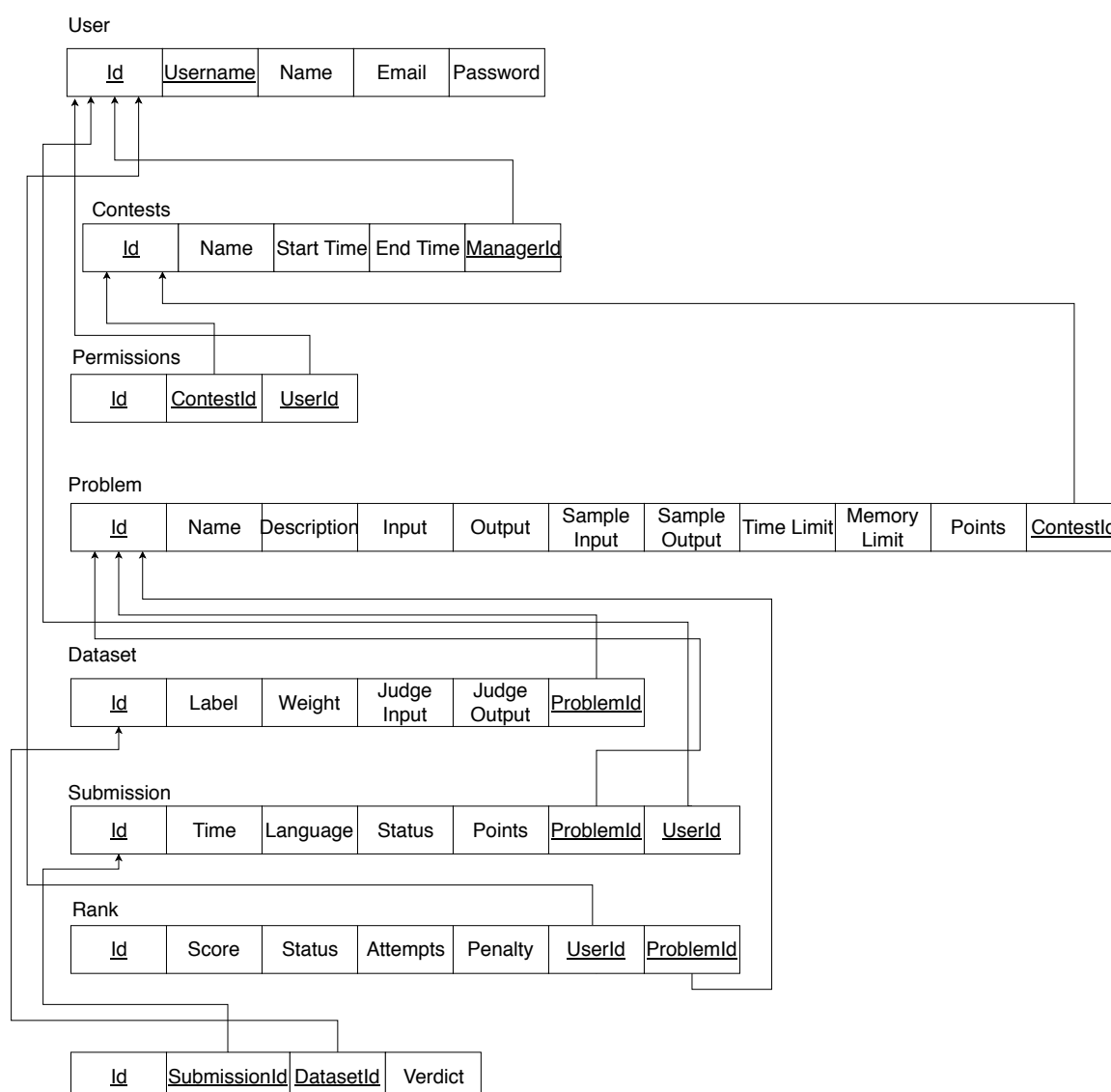


Figure 2. Relation Mapping Diagram.

3.3. Judging Module

The judging module consists two different sub-modules. These are plagiarism detection sub-module and score generation sub-module. The first sub-module checks the plagiarism in the submitted source codes and the other sub-module calculates the score for the submitted source codes. If the the plagiarism module detects a submitted code as a plagiarized one, then it is not passed into the score generation sub-module for further calculation. Every solution that a student submits is executed on the server against a set of secret test cases. The output that is generated by the submitted code is compared with the correct output. If they match, then the student gets the full point for that test case.

3.3.1. Plagiarism Detection Sub-Module

This sub-module of our system is one of the novelties of this proposed work. For programming competitions, plagiarism is the most common method for cheating. Thus, this module was developed with the thought of producing an accurate result. Figure 3 depicts the architecture of the plagiarism module. The first part of the plagiarism module is the scanner. The source codes that will be checked are first passed through the scanner. After that, the scanner converts the codes into token arrays, and later the token arrays are split up into k -gram sub-strings. For further processing, the produced k -grams are converted

into hash values. We have used a winnowing algorithm for comparing the hash values. Each of the steps mentioned above is explained below:

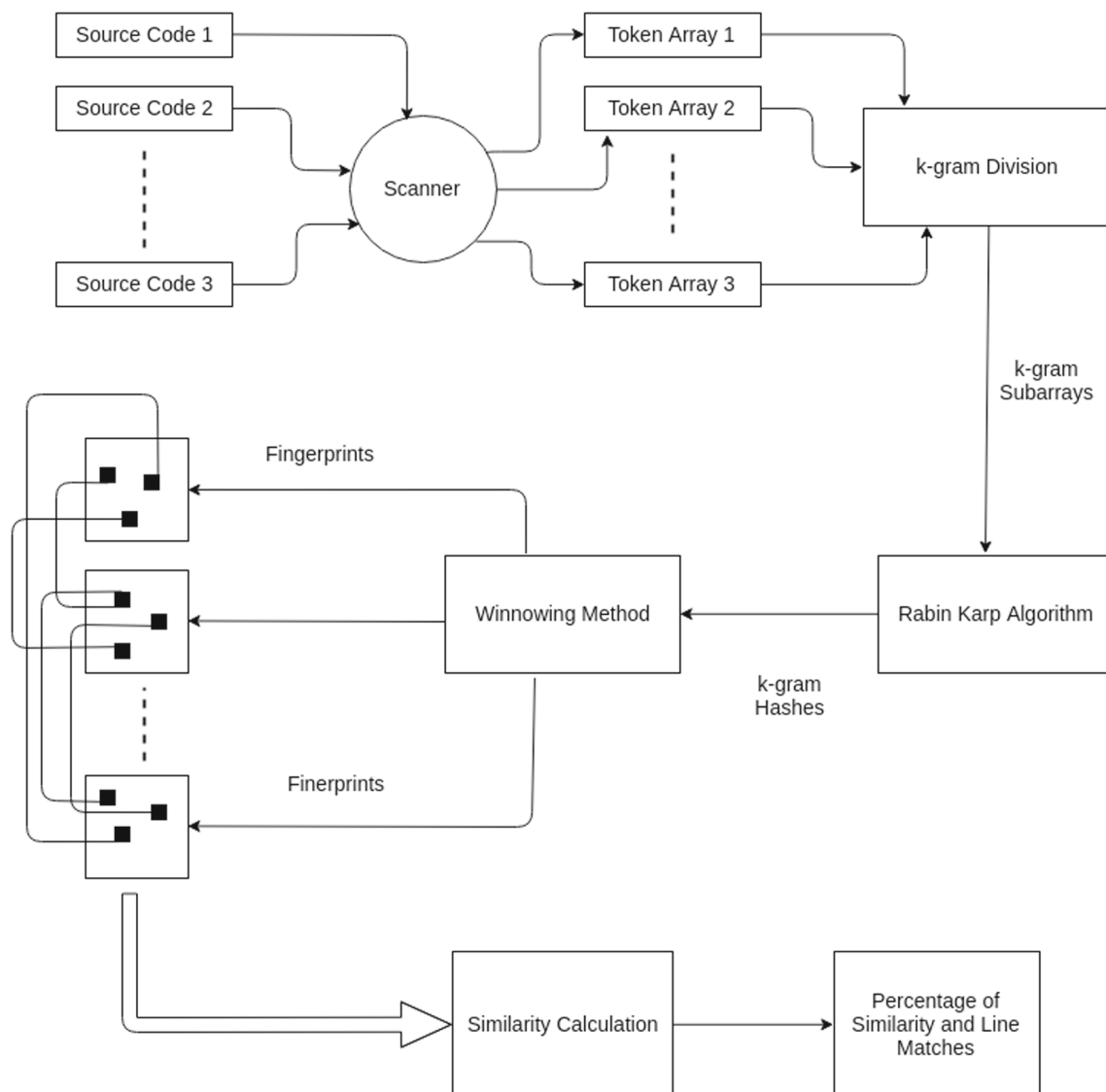


Figure 3. Overall Structure of the Plagiarism Module.

Tokenization

In terms of a stream of integers, called tokens, every input program is passed through a lexical analyzer to create a compact version of its structure. Flex generates the lexical analyzer, so this plagiarism detection method can be easily configured to work with programs in all languages, provided the keywords and reserved words for a particular language. Each token represents an operator, punctuation, keyword, number or string constant, a comment, or an identifier.

k-Gram Hash with Rabin-Karp Algorithm

The token streams are then split up into *k*-gram substrings. These *k*-grams are then converted to some hash value [24] when we want to compare strings of fixed length, *k*. The hash value was determined while using the Rabin-Karp Algorithm. To stop the possibility of overrun, we had to use a mod *MOD*. We obtained the Rabin-Karp value mod *MOD*. This number was very high in order to extend the range of hash values. We also used two prime numbers, since the base and mod were both prime. Algorithm 1 shows the pseudocode for finding *k*-gram hash with the Rabin-karp algorithm.

Algorithm 1 *k*-Gram Hash Value Calculation

```

1: procedure RABIN-KARP(K, B)
2:   for i = 0 to tokenSize − 1 do
3:     if i < k then
4:       hash ← hash · B + token[i]
5:     else
6:       hash ← (hash − token[i − K] · Bk−1) · B + token[i]
7:     end if
8:     if i ≥ k − 1 then
9:       add hash and location to kGramHashList
10:    end if
11:  end for
12: end procedure

```

Winnowing Algorithm

We used a robust winnowing algorithm to compare the source file's chosen fingerprints and choose the minimum hash of a window size *W* [25]. If the minimum hash of this window had been chosen previously, it was discarded. If the minimum hash from the previous window was still present in the current window, we just compared it to the new element. If the sum comes from the current window, the minimum value must be measured from the scratch. Algorithm 2 shows the winnowing algorithm.

Source Code Matching

After we finished selecting fingerprints, they are mapped to each of the programs in which they exist. We also saved the number of occurrence of a fingerprint in a source code. Now, for each source code, for each of its fingerprints, we counted the number of occurrences of the fingerprints in other programs. If fingerprint *f_i* occurs *x* times in program *d_x* and it occurs *y* times in program *d_y*, then we can safely increase the count of matches of program *d_x* with *d_y* by a minimum of *x* and *y*.

Like this, we count the matches with other programs, and then we picked the program with the greatest number of counts. This is the most important and efficient part of our system, which makes it separable with other plagiarism detection systems. In existing systems, pairwise comparisons of source codes are done to find the best match. On the other hand, in our fingerprinting and mapping technique, we calculate the matching results of all source codes at once.

Algorithm 2 Winnowing Algorithm

```

1: procedure FINGERPRINT( $K, W$ )
2:    $minPos \leftarrow 0$ 
3:    $minHash \leftarrow kGramHashList[minPos]$ 
4:   for  $i = 0$  to  $W - 1$  do
5:     if  $minHash \geq kGramHashList[i]$  then
6:        $minPos = i$ 
7:        $minHash = kGramHashList[i]$ 
8:     end if
9:   end for
10:  add  $minHash$  and  $minPos$  to fingerprints
11:  for  $i = W$  to  $kGramHashListSize - 1$  do
12:    if  $minPos = i - W$  then
13:       $minPos = i - W + 1$ 
14:       $minHash \leftarrow kGramHashList[minPos]$ 
15:      for  $j = i - W + 1$  to  $i$  do
16:        if  $minHash \geq kGramHashList[j]$  then
17:           $minPos = j$ 
18:           $minHash = kGramHashList[j]$ 
19:        end if
20:      end for
21:    else
22:      if  $minHash \geq kGramHashList[i]$  then
23:         $minPos = i$ 
24:         $minHash = kGramHashList[i]$ 
25:      end if
26:    end if
27:    add  $minHash$  and  $minPos$  to fingerprints
28:  end for
29: end procedure

```

Another improvement of our system is that our method can find the best match with the program, from which many source codes are copied. If multiple source codes are plagiarized from one particular program, then all of these plagiarized programs are clustered in a single cluster with a reference to the program from which these source codes are copied. As a result, in our system there is no need to compare a new source code with all of these plagiarized source codes. Instead, we simply compare the new codes with the reference code. If this new code is also copied from the reference code, we add the new code to the same cluster. Otherwise, we consider it in a new cluster. We have used a threshold value that is represented in percentage to consider whether a program is plagiarized or not. The threshold value can be set by the contest director for each contest. Based on the result of the level of plagiarism, a program is discarded as plagiarized if it is above the threshold value. However, if the plagiarism is below the threshold value, then the program will be

considered for score generation with the help of score generation sub-module. For each source, a report about plagiarism will also be generated.

3.3.2. Score Generation Sub-Module

This sub-module is responsible for generating scores for the submitted solution. If the source code is not plagiarized, then it is passed into the score generation sub-module for compilation. After compiling the source code, it will be passed into an isolated environment for further execution. Source code may sometimes contain some code that may harm the system. Thus, we kept another source code running environment that is completely separated from our original system. Because of it running in an isolated environment, even if the source code contains any harmful code, it cannot harm the system. In this isolated environment, the source code is executed to generate results for the provided test cases. The program generated output is matched with each of the test cases. In traditional online judges, even for only one wrong output for a test case among a number of test cases, say hundreds test cases, the whole submission is considered “Wrong Answer” i.e., the submission has two states, either accept or reject, even if the mistake is very silly. In order to overcome this problem, in our technique we consider the facility of scoring based of the accuracy level of the submitted source codes. For this purpose, our system uses the equation that is shown in (1). If there is N number of test cases and the submitted solution passes M test cases, the total score is calculated by dividing M by N . Later, this score is normalized by multiplying the result with 100. After the score calculation, it is passed into the storage module for further usage.

$$Score = \frac{M}{N} \times 100 \quad (1)$$

4. Implementation and Experiments

The entire system implementation consists of developing a backend server for handling different issues, such as the execution of the codes, storing the problem sets and test cases, etc., and frontend user friendly interfaces for easy interaction with the system. In addition, implementation requires the integration of plagiarism detection and score generation techniques. All of these were successfully implemented. We have developed all necessary interfaces authentication purpose, administrator, contest management and students. The main components of our system, such as plagiarism detection and score generation, were implemented using C++. It executes the Rabin-Karp, Winnowing and Similarity Calculation. The shell script runs its last command, which passes every token file as the input of the C++ file, and it gives the final result as output. For ensuring uniform distribution, we consider the values of base B and MOD as primes.

The developed system is compared with other well-known online judges. Table 1 shows a detailed comparison of the proposed system with other online judging platforms. Apart from HackerRank [15] and toph [13], all other platforms that are mentioned above do not support partial and dynamic scoring systems. Only URI Online Judge [20], and Sharif Online Judge [19] supports plagiarism detection. Besides some essential management functions, our system supports automatic programming assignment grading, dynamic scoring, and partial grading system. The course teacher can obtain a detailed result on plagiarism. Moreover, like PC² [17], DOMJudge [18], and Sharif-Judge [19], the proposed system can also host contests locally if there is no internet connection available.

Table 1. Comparison With Other Platforms.

Features OJs	Problem Viewing	Plagiarism Detection	Dynamic Scoring	Partial Marking	Host Locally
ine PC ²	No	No	No	No	Yes
ine DOMJudge	No	No	No	No	Yes
ine Sharif-Judge	Yes	Yes	No	No	Yes
ine HackerRank	Yes	No	Yes	Yes	No
ine LightOJ	Yes	No	No	No	No
ine URI OJ	Yes	Yes	No	No	No
ine Toph	Yes	No	Yes	Yes	No
ine CodeMarshal	Yes	No	Yes	Yes	No
ine CUET OJ	Yes	Yes	Yes	Yes	Yes

4.1. Experimental Results for Plagiarism Checker

To check the accuracy of the proposed plagiarism checker, we have performed experiments on some actual source codes. The time efficiency and correctness were a major concern here. Besides, we compared our plagiarism discovery engine's outcome with the commonly used plagiarism identification system MOSS.

4.1.1. Data Set

For plagiarism checker, we have collected data from two distinct sources listed below, and a summary is shown in Table 2.

1. Source Codes from 10 lab tests arranged for first and second-year students. The course name of those two courses is—CSE-142 Structured Programming and CSE-242 Data Structures.
2. Source Codes from 7 practice contests, arranged for competitive programmers of CUET.

Table 2. Summary of Dataset.

Source	Number of Tests/Contests	Total Number of Tasks	Number of Submissions
Lab Assignments	10	24	683
Practice Contests	7	103	784
Total	17	127	1467

In this table, the number and type of data sets are shown. The total number of tasks defines the sum of all of the tasks for all contests. All of these tasks are applied to the plagiarism detection system separately. The number of Submissions defines the total volume of source codes that were submitted in these contests.

4.1.2. Time Efficiency

Time efficiency is one of the essential parts of the plagiarism checking system. Three test runs were done to test the time efficiency of the proposed system. They are mentioned below.

Test Run 1. In the first experiment, we used source codes from the lab test of the introductory course for first-year students. The difficulty level of the provided tasks was easy and it needed fundamental programming knowledge. Table 3 shows that the expected solution and the runtime are deficient.

Table 3. Experiment with Data Set 1.

Total Files	63
Total Bytes	9200
Max. Size of Token Array	419
Execution Time	0.07878 s

Test Run 2. In the second experiment, we used small and introductory programs. However, the number of files increased, which is listed in Table 4.

Table 4. Experiment with Data Set 2.

Total Files	236
Total Bytes	56,100
Max. Size of Token Array	830
Execution Time	1.4243 s

Test Run 3. For experiment three, we selected some programs that are designed to solve an advanced problem used in competitive programming. Table 5 shows the number of files, file size, and execution time.

Table 5. Experiment with Data Set 3.

Total Files	25
Total Bytes	15,800
Max. Size of Token Array	3100
Execution Time	0.0823 s

Test Run 4. Our final experiment is shown in Table 6. In this regard, we gathered 1368 source codes from all of the contests. Although they are from different contests and different tasks, as we are experimenting to determine whether this system is efficient or not, they can be used for this purpose.

Table 6. Experiment with Data Set 4.

Total Files	1368
Total Bytes	912,100
Max. Size of Token Array	4249
Execution Time	182.15 s

Because of the optimization, in the last phase of our algorithm, i.e., to discard programs that were already plagiarized, we found that the comparison mechanism improved its speed over time. As the number of programs to compare decreased over time, the runtime of the plagiarism of the new programs increased simultaneously.

4.1.3. Correctness

To show the correctness of this plagiarism checker system, we can use our last experiment results that are shown in Figure 4. There are 127 tasks in all, and no source code from one task can have a high degree of similarity with codes from another program. If our plagiarism detection method is correct, all of the similarity reports with a high percentage should be linked to the same problem.

We selected approximately 60 pairs of code with more than 90% similarity and manually confirmed that they all originated from the same folder or task. Consequently, the program's correctness is established.

8842407_Cuet_Mystics_E.cpp	8842402_Cuet_Mystics_E.cpp	98.955 percent	35 line
8842471_Cuet_Mystics_U.cpp	8704599_Avij33t_U.cpp	70.149 percent	9 line
8854914_MAMUN_CUET_J.cpp	11986038_pure_amatuer_M.cpp	53.846 percent	30 line
8855045_MAMUN_CUET_J.cpp	8757789_p_factor_J.cpp	54.266 percent	30 line
8855940_MAMUN_CUET_U.cpp	8704599_Avij33t_U.cpp	92.157 percent	8 line
8860377_Frustrated_coder_E.cpp	14193008_Tajir_Hasnain_A.cpp	47.107 percent	18 line
8861238_p_factor_X.cpp	8751455_Emdadul_Hoque_X.cpp	41.418 percent	16 line
8861343_p_factor_U.cpp	8864001_p_factor_V.cpp	56.180 percent	12 line
8863534_Yeasin10A_E.cpp	12081368_Yeasin10A_D.cpp	49.573 percent	7 line
8864001_p_factor_V.cpp	8726836_p_factor_D.cpp	55.932 percent	17 line
8865958_MAMUN_CUET_I.cpp	8756251_p_factor_I.cpp	33.000 percent	38 line
8869688_Frustrated_coder_I.cpp	8704541_Avij33t_I.cpp	34.498 percent	17 line
8871578_nahid3334_E.cpp	8722426_p_factor_B.cpp	47.639 percent	15 line
8873335_Yeasin10A_Q.cpp	8704652_Avij33t_Q.cpp	46.256 percent	43 line
8873926_nahid3334_U.cpp	11913853_Naim_khan_B.cpp	47.126 percent	7 line
8874533_Xaqir_J.cpp	8757789_p_factor_J.cpp	58.156 percent	23 line
8874933_Frustrated_coder_V.cpp	8836620_Frustrated_coder_T.cpp	67.797 percent	8 line
8876128_Xaqir_U.cpp	12859132_Ratul1997_C.cpp	45.283 percent	15 line
8876196_Cuet_Mystics_I.c	11986038_pure_amatuer_M.cpp	38.721 percent	20 line
8876254_Cuet_Mystics_V.cpp	8819156_Cuet_Mystics_W.cpp	58.108 percent	8 line
8877028_Xaqir_V.cpp	12049596_1504054_nahid_O.cpp	39.655 percent	5 line
"Execution Time: ", (double)(clock() - tStart)/CLOCKS_PER_SEC : Execution Time: 182.15			
[all]			

Figure 4. Screenshot for Data Set 4.

4.1.4. Evaluation

We compared our system to the commonly used MOSS to illustrate the proposed method's time efficiency. Table 7 shows the attained result for MOSS and our proposed system. It took MOSS 1 min. 13 s to provide us the result, whereas our system took 1.4 s. Even though MOSS is effective and commonly utilized in academic practices, our plagiarism identification allows for us to use it from a local server and save time on request-response. MOSS's output representation is not well-organized. It displays a similarity report for each pair of source codes uploaded. Our method displays the best match for a particular program. Another issue with MOSS is that, although it is free to use, the details of how it was produced are still unclear. Schleimer et al. [26] published the base method of their system—the winnowing method, which we implemented this system, and, by doing this, we opened the door to work on this project more specifically and solve limitations that MOSS has. Figure 5 depicts the runtime of MOSS on data set 2.

We can also agree that this plagiarism detection system can be extended to any programming competition. Plagiarism detection would work effectively if 120 students in a class take a lab exam or 120 teams competing in a national programming competition, and the number of problems is about 10, and each individual/team submits each problem.

Table 7. Time Efficiency Comparison of the Proposed System.

Method	Time
MOSS	13 s
Proposed System	1.4 s

```

Uploading 12818429_1704015_Nadira_C.c ...done.
Uploading 12825895_1704034_Bibhas_C.c ...done.
Uploading 12826078_1704034_Bibhas_D.c ...done.
Uploading 12826118_1704034_Bibhas_D.c ...done.
Uploading 12826343_1704034_Bibhas_C.c ...done.
Uploading 12877504_1704039_Jawad_C.c ...done.
Query submitted. Waiting for the server's response.
http://moss.stanford.edu/results/758660385
./moss *.c 0.05s user 0.02s system 0% cpu 1:13.99 total
[all]
```

Figure 5. MOSS Runtime for Data Set 2.

5. Implication and Limitations

It is impossible for a judge or an instructor to manually judge the students' source codes in a huge collection of programs to identify the plagiarized ones. When considering this fact, in this work we have developed an online judging platform with dynamic plagiarism detection system. This system is helpful for both academia and industry. Competitive programming is gaining increasing popularity for assessing students and candidates' programming ability. Our proposed method can run both online and offline and provide partial scoring and perform plagiarism detection. These qualities are the essential aspect of an online judging system. However, no stress testing is performed on the system. Thus, it is still unknown how much audience the system will be capable of holding.

6. Conclusions and Future Research

In this paper, we presented a framework for automating the judging procedure of the contestants' solutions. We have built a web-based judging platform where teachers can create assignments for programming labs. Students can view and submit the solutions to these assignments within the given time. Their submission is executed in a completely isolated environment, so they cannot harm the system.

The grading processes of the traditional online judging systems are not adequate for detecting source code plagiarism efficiently. In addition, most of them do not consider partially correct programs. In our approach, we considered both plagiarism detection and partial scoring. In our present version, we do not consider the task of evaluating the quality of source codes, making the evaluation more accurate. In addition, we do not integrate machine learning algorithms for code evaluation and further assessment. We plan to address these issues in the future.

Author Contributions: Specification of the individual contributions: Conceptualization, F.I., M.T.R., J.F. and M.S.A.; investigation, F.I., A.S.M.K., M.T.R., J.F., M.S.A. and M.S.H.; methodology, F.I., A.S.M.K., M.T.R., J.F., M.S.A. and M.S.H.; software, F.I., M.T.R., and J.F.; validation, F.I., A.S.M.K., M.T.R., J.F., M.S.A. and M.S.H.; writing—original draft preparation, F.I., M.T.R., J.F. and M.S.A.; writing—review and editing, F.I., A.S.M.K., M.T.R., J.F., M.S.A. and M.S.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by Directorate of Research and Extension (DRE), Chittagong University of Engineering and Technology (CUET), Bangladesh.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Wasik, S.; Antczak, M.; Badura, J.; Laskowski, A.; Sternal, T. A Survey on Online Judge Systems and Their Applications. *ACM Comput. Surv. (CSUR)* **2018**, *51*, 1–34. [\[CrossRef\]](#)
2. Kurnia, A.; Lim, A.; Cheang, B. Online Judge. *Comput. Educ.* **2002**, *36*, 299–315. [\[CrossRef\]](#)
3. Liu, J.; Zhang, S.; Yang, Z.; Zhang, Z.; Wang, J.; Xing, X. Online Judge System Topic Classification. In Proceedings of the 2018 14th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), Huangshan, China, 28–30 July 2018; pp. 993–1000. [\[CrossRef\]](#)
4. Aiken, A. *A System for Detecting Software Plagiarism*; University of Berkeley: Berkeley, CA, USA, 2005.
5. Prechelt, L.; Malpohl, G. Finding Plagiarisms among a Set of Programs with JPlag. *J. Univ. Comput. Sci.* **2003**, *8*, 1016.
6. Wise, M. YAP3: Improved Detection Of Similarities In Computer Program and Other Texts. *ACM SIGCSE Bull.* **1996**, *28*. [\[CrossRef\]](#)
7. Carter, J. Collaboration or plagiarism: What happens when students work together. *ACM SIGCSE Bull.* **1999**, *31*, 52–55. [\[CrossRef\]](#)
8. Hext, J.B.; Winings, J. An automatic grading scheme for simple programming exercises. *Commun. ACM* **1969**, *12*, 272–275. [\[CrossRef\]](#)
9. University of Valladolid. UVA Online Judge. Available online: <http://uva.onlinejudge.org> (accessed on 1 December 2020).
10. Peking University. PKU Online Judge. Available online: <http://poj.org> (accessed on 1 December 2020).
11. Jan, J.A. Welcome to Lightoj. Available online: <http://www.lightoj.com> (accessed on 1 December 2020).
12. Muktosoft LTD. Welcome to Codemarshal. Available online: <http://algo.codemarshal.org> (accessed on 1 December 2020).
13. Furqan Software. Toph—A Sport Programming Platform. Available online: <https://toph.co/> (accessed on 1 December 2020).

14. KTH Royal Institute of Technology, Sweden. Welcome to the Kattis Problem Archive. Available online: <https://open.kattis.com> (accessed on 1 December 2020).
15. HackerRank. Hackerrank. Available online: <https://www.hackerrank.com> (accessed on 1 December 2020).
16. HackerRank. HackerRank for Schools. Available online: <http://www.hackerrank.com/school> (accessed on 1 December 2020).
17. California State University. CSUS Programming Contest Control (pc²). Available online: <http://pc2.ecs.csus.edu> (accessed on 1 December 2020).
18. Johnson, K.; Eldering, J.; Gerritsen, N. DOMJudge—Programming Contest Jury System. Available online: <http://www.domjudge.org> (accessed on 1 December 2020).
19. Naderi, M.J. Sharif-Judge. Available online: <http://github.com/mjnaderi/Sharif-Judge> (accessed on 1 December 2020).
20. Bez, J.L.; Tonin, N.; Rodegheri, P. URI Online Judge Academic: A Tool for Algorithms and Programming Classes. In Proceedings of the 2014 9th International Conference on Computer Science & Education, Vancouver, BC, Canada, 22–24 August 2014. [CrossRef]
21. Yi, C.; Feng, S.; Gong, Z. A Comparison of Sandbox Technologies Used in Online Judge Systems. In *Mechanical Design and Power Engineering*; Applied Mechanics and Materials; Trans Tech Publications Ltd.: Bäch, Switzerland, 2014; Volume 490, pp. 1201–1204. [CrossRef]
22. Felter, W.; Ferreira, A.; Rajamony, R.; Rubio, J. An updated performance comparison of virtual machines and Linux containers. In Proceedings of the 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Philadelphia, PA, USA, 29–31 March 2015; pp. 171–172. [CrossRef]
23. Merkel, D. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux J.* **2014**, *2014*, 2.
24. Karp, R.M.; Rabin, M.O. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.* **1987**, *31*, 249–260. [CrossRef]
25. Elbegbayan, N. Winnowing, a Document Fingerprinting Algorithm. TDDC03 Projects, Spring 2005, pp. 1–8. Available online: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.130.3186&rep=rep1&type=pdf> (accessed on 1 December 2020).
26. Schleimer, S.; Wilkerson, D.S.; Aiken, A. Winnowing: Local algorithms for document fingerprinting. In Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, CA, USA, 10–12 June 2003; pp. 76–85.