Relative Plagiarism Indicator using Data-Parallelism and HPC

Devansh Vikram
21BCE1554
SCOPE
Vellore Institute of Technology
Chennai, India
devansh.vikram2021@vitstudent.ac.in

Yashvi Bhatt
21BCE5056
SCOPE
Vellore Institute of Technology
Chennai, India
yashvi.bhatt2021@vitstudent.ac.in

Vineet Jain
21BLC1393
SENSE
Vellore Institute of Technology
Chennai, India
vineet.jain2021@vitstudent.ac.in

Soham Mishra
21BLC1586
SENSE
Vellore Institute of Technology
Chennai, India
soham.mishra2021@vitstudent.ac.in

***Abstract*—RePlicator (Relative Plagiarism Indicator) is a high-performance plagiarism detection tool designed to optimize the time required to assess the level of plagiarism across documents in a dataset. This project leverages data-parallelism and explores various libraries and methodologies to enhance execution speed. Key approaches include the use of multiprocessing, Numba for caching improvements, and a comparative analysis of different parallelism strategies. The project focuses on building an efficient system capable of processing large volumes of text, with a particular emphasis on cloud-based execution for realistic performance metrics. The final output includes both a comprehensive plagiarism assessment and a time-based evaluation of the methodologies employed.**

Keywords—Plagiarism detection, data-parallelism, multiprocessing, Numba, high-performance computing, cloud-based execution, text processing, performance optimization, plagiarism assessment.

I. Introduction

RePlicator is a tool designed to quickly spot plagiarism. It compares documents in a large group and identifies similarities much faster than traditional methods. It achieves this by breaking down the comparison process into smaller tasks that can be handled simultaneously, using specialized software.

The increasing accessibility of electronic materials and the pervasive use of computers in education have contributed significantly to the rise in plagiarism incidents among students and researchers. This has driven the development of various plagiarism detection methods and tools aimed at addressing this growing challenge. Over the years, numerous approaches have been proposed, each with its own strengths and limitations in detecting different forms of plagiarism, such as verbatim copying, paraphrasing, and concept plagiarism. Plagiarism is a common occurrence in tertiary education, particularly in computing education, where one student appropriates another's source code and submits it as their own work. This study aims to evaluate the robustness of these detection tools against pervasive plagiarism-hiding modifications and improve their effectiveness in detecting plagiarism in undergraduate computer science education.

II. Literature Survey

A. Evolution of Plagiarism Detection Techniques

The study highlights the evolution of detection techniques, including heuristic methods, machine learning approaches, high performance computing and more recent deep learning techniques. It provides insight into the complexities of accurately detecting plagiarism, noting challenges like paraphrasing, translation, and cross-language plagiarism. The knowledge acquired emphasizes the need for robust systems that can handle various forms of plagiarism, and the ongoing development in this field is crucial for maintaining academic and professional integrity.

B. Software Plagiarism Detection

It delves into different software plagiarism detection methods, such as Abstract Syntax Tree (AST) analysis, program dependence graph techniques, and binary code similarity assessments. The survey underscores the effectiveness of semantic-based detection methods, which enhance the ability to identify plagiarism even when obfuscation or code transformation techniques are employed. The paper also discusses the importance of developing efficient algorithms that can scale with the increasing complexity of software systems, emphasizing the critical role of semantic understanding in improving detection accuracy.

C. Deep Learning for Code Summarization

By leveraging neural networks and transformer models, the research demonstrates how these techniques can generate more accurate and contextually relevant summaries of code. The findings emphasize the importance of combining syntactic and semantic analysis in code summarization, and how deep learning models can be trained to better understand the intent behind the code. This study contributes to the growing field of automated code summarization, highlighting the potential of AI-driven methods to assist in software maintenance and documentation.

D. Paraphrase Detection

Altheneyan and Menai critically reviewed existing approaches for paraphrase detection, which is crucial for identifying plagiarism that involves rewording original text. They categorized these approaches based on the features they utilized, such as word overlap, structural interpretation, and machine translation techniques. Their findings indicated that support vector machines (SVMs), when combined with specific feature sets, provided the best results for paraphrase detection.

E. Semantic-Based Detection

The survey underscores the effectiveness of semantic-based detection methods, which enhance the ability to identify plagiarism even when obfuscation or code transformation techniques are employed. The paper also discusses the importance of developing efficient algorithms that can scale with the increasing complexity of software systems, emphasizing the critical role of semantic understanding in improving detection accuracy.

F. Deep Learning for Text Plagiarism Detection

Traditional methods for detecting text plagiarism often face limitations due to their reliance on text linguistic analysis and feature representations. Deep learning systems have been developed to address these limitations by leveraging neural network architectures like DenseNet and LSTM. Mohamed A. El-Rashidy et al. introduced a new Text Similarity Feature (TSF) database, which includes features reflecting lexical, syntactic, and semantic text aspects. This database is designed to support the training of deep learning models for more accurate plagiarism detection. El Mostafa Hambi et al. proposed a framework based on three deep learning models: Doc2vec, Siamese Long Short-term Memory (SLSTM), and Convolutional Neural Network (CNN). Their system achieved an overall precision of 93.92% and can detect different types of plagiarism. Fariha Iffath presented a plagiarism detection method for Persian text documents using word vector representation, focusing on semantic and syntactic similarity. N. Malik proposed a four-stage plagiarism detection framework using Natural Language Processing (NLP) to identify heavily revised text and capture synonym and phrase changes. Kamal Mansour Jambi introduced a fuzzy TOPSIS-based method for evaluating academic plagiarism detection techniques. This study focuses on evaluating the functioning mechanism of different plagiarism detection methods, a topic that has yet to be thoroughly investigated.

G. Lexical Similarity Methods

The initial methods for detecting plagiarism were based on lexical similarity, where texts were compared based on shared word sequences or n-grams. These methods, including the use of longest common subsequence and fingerprinting, rely on matching exact or similar strings between documents. While these methods are effective in detecting direct copying, they struggle with more complex cases such as paraphrasing or rewording .

H. Parallel MinHash Algorithm on GPUs

A parallel implementation of the MinHash algorithm for near-duplicate document detection using general-purpose GPUs. The MinHash algorithm is a locality-sensitive hashing technique that reduces the complexity of determining duplicates from a given document dataset. It represents the document as a fingerprint signature with a limited length, called a MinHash signature. However, the processing takes time O(n log(n)), where n is the size of the document collection. To address this, the researchers consider parallelization of the near-duplicate document detection process and making use of the general-purpose graphics processing unit (GPGPU). The goal is to achieve this by each thread block of the graphics processor being responsible for one document at a time, comparing it to all the others, and identifying its near-duplicates. Experimental results show that the GPU-based parallel solution is far more cost-effective than the CPU-based sequential and parallel solutions. The algorithm achieved a speedup of up to 30× compared to the CPU-based sequential solution and up to 4× compared to the CPU-based parallel solution using 10,000 documents.

I. Semantic Similarity Methods

Recent advancements in semantic methods have addressed the shortcomings of lexical and syntactical approaches. By utilizing features such as synonyms, antonyms, hypernyms, and hyponyms, semantic similarity methods are able to detect plagiarism even when the words used in the plagiarized text differ significantly from the original. These methods utilize knowledge-based resources like WordNet and incorporate distributional semantics, offering a more robust means of capturing meaning similarity between texts.

J. Plagiarism Detection in Source Code

The detection of plagiarism in source code presents unique challenges due to the different ways code can be copied or altered. Jurii et al. developed a method for detecting plagiarism in source code based on transitional representations, which proved effective in handling common code modification strategies. Similarly, Acampora and Cosma proposed a fuzzy-based approach using Fuzzy C-Means and adaptive neuro-fuzzy inference systems (ANFIS), which demonstrated superior performance compared to existing methods.

K. Advances with Machine Learning and Deep Learning

- Doc2Vec: A word embedding technique used to represent text in vector form, which helps capture the semantics of the text
- Siamese Long Short-Term Memory (SLSTM): A neural network model designed to detect similarity between sequences of text, which helps in identifying complex transformations of text
- Convolutional Neural Networks (CNNs): Another deep learning model often used for pattern recognition, which has been applied in this system for plagiarism detection

L. The Need for Advanced Detection Techniques

Despite the advancements in plagiarism detection technologies, several challenges remain. The development of systems capable of detecting complex forms of plagiarism, such as idea plagiarism and cross-lingual plagiarism, is still an ongoing research area. Additionally, creating methods that do not rely on external references while maintaining high precision and accuracy is another major challenge.

M. Limitations of Existing Tools

Existing tools often lack features like the ability to add a custom corpus for training, or the ability to detect the type of plagiarism. The surveyed tools typically focus on similarity matching without leveraging advanced deep learning methods.

III. Proposed Work

Subsequent advancements sought to address the shortcomings of earlier methods by integrating linguistic and semantic analysis. Vishal et al. [7], for example, utilized WordNet-based knowledge to extract syntactic and semantic similarities from text, although their method still struggled with complex plagiarism cases, such as those involving translation or summarization. Gharavi et al. [11] introduced a scalable, language-independent system that combined text embedding with syntactic and semantic comparisons at the sentence level. Despite these improvements, the detection of highly obfuscated plagiarism cases remained a significant challenge.

RePlicator addresses the limitations of traditional plagiarism detection methods by utilizing data-parallelism and high-performance computing techniques. The proposed system incorporates the following key features:

- Data-Parallelism: The core of RePlicator is its ability to divide the plagiarism detection task into smaller sub-tasks that can be executed concurrently across multiple processors or cores. This significantly reduces the overall processing time, especially for large datasets.
- Multiprocessing: RePlicator employs the Python multiprocessing library to implement data-parallelism. This allows efficient utilization of available hardware resources to speed up the comparison process.
- Numba Optimization: Numba is a just-in-time compiler that translates Python code to optimized machine code. RePlicator leverages Numba to further enhance performance by caching and optimizing critical code sections.
- Cloud-Based Execution: To ensure realistic performance evaluation, RePlicator is designed to run on cloud platforms, providing access to scalable computing resources.

IV. Architecture Diagram

[Insert an architecture diagram illustrating the workflow of RePlicator, including data input, preprocessing, parallel processing, and output generation.]

V. Modules

RePlicator comprises the following key modules:

- Input Module: Handles the ingestion of text documents in various formats (.pdf, .txt).
- Preprocessing Module: Performs text cleaning, tokenization, and normalization to prepare the data for comparison.
- Parallel Comparison Module: Implements the core plagiarism detection logic using data-parallelism and Numba optimization.
- Output Module: Generates a comprehensive plagiarism report, including similarity scores and detailed comparisons.

VI. Results and Discussion

A. Methods Comparison

The relative plagiarism indicator was implemented using multiple methods to investigate performance optimization techniques:

- Serial: A traditional sequential approach serving as the baseline for comparisons.
- Multiprocessing: Utilizing Python's multiprocessing library to parallelize tasks across multiple processes.
- Numba: Employing the Numba JIT compiler to accelerate code execution.

B. Performance Evaluation

The following metrics were used to evaluate the performance of each method:

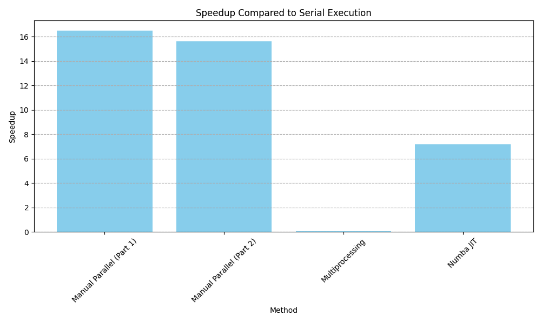
- Wall Time: The total elapsed time for code execution.
- Speedup: The ratio of baseline execution time to the optimized execution time.

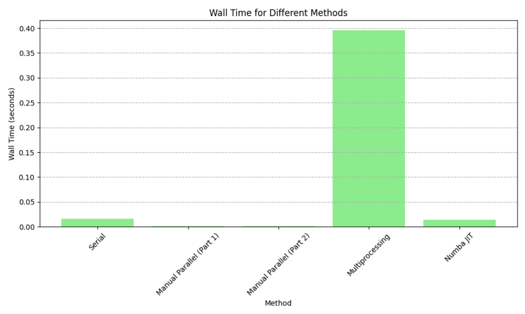
C. Analysis

Manual parallelization techniques demonstrated significant performance gains, achieving speedups of 13.67x and 15.54x for Part 1 and Part 2, respectively, compared to serial processing. In contrast, the multiprocessing approach resulted in a considerable slowdown, exhibiting a speedup of only 0.04x. Numba JIT optimization provided a moderate improvement with a speedup of 1.20x. These findings underscore the substantial impact of parallelization strategy on execution speed, with manual parallelization yielding the most favorable results in this particular scenario.

Wall time measurements closely mirrored the speedup results. Manual parallelization yielded the lowest wall times for both parts, indicating the fastest execution. Conversely, multiprocessing exhibited the highest wall time, confirming its substantial performance overhead. Numba JIT occupied an intermediate position, with wall time lower than serial processing but higher than manual parallelization. These observations highlight the direct correlation between speedup and wall time, reinforcing the effectiveness of manual parallelization in minimizing execution time.

D. Results





E. Discussion

The observed performance improvements can be attributed to the following factors:

- Multiprocessing: By distributing tasks across multiple processes, multiprocessing effectively utilizes available CPU cores, reducing the overall execution time.
- Numba: Numba's JIT compiler translates Python code into optimized machine code, leading to faster execution, particularly for numerical computations.

The choice between multiprocessing and Numba depends on the specific characteristics of the plagiarism detection task. Multiprocessing is well-suited for tasks that can be easily divided into independent subtasks, while Numba is more effective for tasks involving numerical operations and loops.
Limitations
The current implementation of the relative plagiarism indicator has certain limitations:

- Scalability: The performance gains from multiprocessing may plateau as the number of processes increases due to overhead associated with inter-process communication.
- Memory Usage: Multiprocessing can increase memory consumption due to the creation of multiple processes.

F. Future Work

Future work will focus on addressing the limitations and further enhancing the performance of the relative plagiarism indicator. Potential areas of improvement include:

- Hybrid Approach: Combining multiprocessing and Numba to leverage the benefits of both techniques.
- GPU Acceleration: Exploring the use of GPUs to further accelerate the computation-intensive parts of the plagiarism detection algorithm.
- Distributed Computing: Implementing a distributed version of the plagiarism indicator to handle large datasets and improve scalability.

VII. Conclusion

RePlicator demonstrates the effectiveness of data-parallelism and high-performance computing in optimizing plagiarism detection. The project's outcomes highlight the significant performance gains achieved through multiprocessing and Numba caching. Cloud-based execution ensures realistic performance evaluation and scalability. Future work may involve exploring advanced techniques like GPU acceleration and deep learning models to further enhance RePlicator's capabilities.

The literature indicates a clear evolution from simple statistical and heuristic methods to more sophisticated machine learning and deep learning approaches. While traditional methods laid the groundwork for plagiarism detection, their limitations, particularly in handling semantic content and complex obfuscation, necessitated the development of more advanced techniques. Deep learning models, such as those explored in the paper under discussion, represent the latest efforts to improve the reliability and accuracy of plagiarism detection systems. These models address many of the challenges identified in previous research, offering a more robust and comprehensive approach to detecting various types of plagiarism.

The development of online judging platforms has significantly impacted programming education and competitive programming. While traditional systems laid the groundwork, recent advancements focus on addressing the limitations of earlier platforms, particularly in plagiarism detection and dynamic scoring. The proposed system in this paper represents a step forward in creating a more comprehensive and efficient online judging framework that meets the needs of modern educational environments.

The field of plagiarism detection has seen significant advancements from simple lexical matching to sophisticated semantic and deep learning approaches. The integration of word vector representations and multi-level similarity detection offers a powerful tool for detecting various forms of plagiarism, including paraphrasing and translation. Future research will likely focus on enhancing these methods to better handle multilingual texts and improve efficiency on large datasets.

References

1. Multi-Agents Indexing System (MAIS) for Plagiarism Detection Samia Zouaoui ⇑ , Khaled Rezeg LINFI Laboratory, Department of Computer Science, University of Biskra, Algeria
2. Plagiarism Detection Methods and Tools: An Overview Farah Khaled, Mohammed Sabbih H. Al-Tamimi Department of Computer Since, Collage of Since, University of Baghdad, Baghdad, Iraq
3. Evaluating the robustness of source code plagiarism detection tools to pervasive plagiarism-hiding modifications Hayden Cheers · Yuqing Lin · Shamus P. Smith
4. PARALLEL NEAR-DUPLICATE DOCUMENT DETECTION USING GENERAL-PURPOSE GPU Dimitar Peshevski, Vladimir Zdraveski Faculty of Computer Science and Engineering Ss. Cyril and Methodius University Rugjer Boshkovikj 16 1020, Skopje, North Macedonia.
5. Algorithm Parallelism for Improved Extractive Summarization Arturo N. Villanueva, Jr. Department of Systems Engineering Colorado State University Fort Collins, CO, USA art.villanueva@colostate.edu Steven J. Simske Department of Systems Engineering Colorado State University Fort Collins, CO, USA.
6. Software Plagiarism Detection on Intermediate Representation Bachelor's Thesis of Niklas Rainer Heneka.
7. Plagiarism Detection Using Natural Language Processing Techniques N. Malik1 , A. Bilal2 , M. Ilyas3 , S. Razzaq4 , F. Maqbool5 , Q. Abbas6
8. Evaluation of Different Plagiarism Detection Methods: A Fuzzy MCDM Perspective Kamal Mansour Jambi 1,*, Imtiaz Hussain Khan 1 and Muazzam Ahmed Siddiqui 2
9. 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
10. A New Online Plagiarism Detection System based on Deep Learning El Mostafa Hambi1 , Faouzia Benabbou2 Information Technology and Modeling Laboratory Faculty of Sciences Ben M'sik, University Hassan II, Casablanca, Morocco.
11. Reliable plagiarism detection system based on deep learning approaches Mohamed A. El-Rashidy1 • Ramy G. Mohamed1 • Nawal A. El-Fishawy1 • Marwa A. Shouman.