Parallel Miss Marple: Threads e Java RMI Aplicados à Verificação de Indícios de Plágio

Ricardo Bianchin Gomes[1], Roseclea Duarte Medina[1]

[1]Programa de Pós-graduação em Informática – Universidade Federal de Santa Maria

[ricardo, rose]@inf.ufsm.br

Abstract. *Plagiarism identification in digital academic texts still has challenges. One of them is the long time necessary for computer programs do the analysis of big digital documents sets. This work reports the improvement of an open-source plagiarism checking tool by applying parallel-distributed computing techniques, using threads and Java RMI, bringing benefits for teachers, professors, evaluators and general users. Performance tests have shown the new version of the program reaches a speedup of 1.99 with the usage of two processing threads, that is, reduction of approximately 50% in analysis time in comparison to the previous version.*

Resumo. *A identificação do plágio em trabalhos escolares ainda possui desafios. Um deles é o longo tempo necessário para que programas de computador realizem a análise de grandes lotes de documentos digitais. Este trabalho relata o processo de aprimoramento de uma ferramenta open-source de busca de indícios de plágio através da aplicação de técnicas de computação paralelo-distribuída, usando-se threads e Java RMI, trazendo benefícios para professores, avaliadores e usuários em geral. Os testes de desempenho apontaram que a nova versão do programa atinge um speedup de 1.99 com o uso de duas threads de processamento, ou seja, redução de aproximadamente 50% do tempo de análise em relação à versão anterior.*

1. Introdução

O plágio é uma prática muito frequente, principalmente em se tratando em trabalhos do meio escolar e acadêmico [Moraes 2004]. *Id* (2004) afirma que "nas universidades brasileiras vem surgindo um novo e perigoso grupo, formado por professores e alunos plagiários". Segundo Arenhardt (2013), a Internet é a fonte de informação mais utilizada, devido à facilidade em se obter resultados de pesquisa, propiciada pelos motores de busca. Esta grande fonte de informações, aliada à falta de experiência na escrita, falta de instrução ou ainda, a má índole, acaba se tornando uma ferramenta que potencializa a questão do plágio, ideia sustentada por Moraes (2004).

Baseado nisso, a detecção de plágio tem se tornado uma tarefa cada vez mais complexa para professores, e de fundamental importância para a educação como um todo. O plágio, além de ilegal, reduz a capacidade do aluno de pensar e escrever [Arenhardt *et al.* 2012]. Por este motivo, a identificação desta prática compõe uma importante etapa para a educação moral e construção da boa índole dos estudantes, em idade escolar. Com base nestes fatos, tornou-se necessário contar com uma ferramenta computacional que auxilie os educadores nesta tarefa, o que levou ao desenvolvimento

DOI: 10.5753/cbie.wie.2016.369

de diversos *softwares*. Um exemplo dessas ferramentas é o Miss Marple [Arenhardt 2013], o qual possui código aberto, e que de acordo com testes apresentados pela autora, possui boa precisão de resultados e um bom tempo de processamento ao se submeter um único arquivo, comparado a outras ferramentas similares.

Entretanto, o tempo necessário para realizar a análise de grandes lotes de documentos digitais, cenário vivenciado por muitos professores, se demonstra demasiadamente longo. Isto se deve à grande quantidade de trabalhos que eles necessitam corrigir em cada uma de suas turmas, e também ao fato de que os arquivos são inspecionados pelos aplicativos de forma sequencial. Esta forma de execução além de causar atrasos aos docentes, demonstra não explorar por completo as arquiteturas mais recentes de processadores multi-core. Portanto, é provável que a adequada exploração de arquiteturas multi-core atuais pode trazer benefícios para o processo de análise de plágio como um todo.

No intuito de mitigar a questão apresentada, este trabalho apresenta o desenvolvimento de uma nova versão da ferramenta, o *Parallel Miss Marple,* a qual possui suporte a execução paralela e distribuída através do uso de *threads* em Java e Java RMI.

A estrutura do texto está disposta da seguinte forma: na seção 2, é apresentado a conceituação do plágio e uma forma de identificá-lo. A seção 3 apresenta a metodologia utilizada para o desenvolvimento do projeto; a seção 4 retrata a arquitetura e os recursos da nova ferramenta desenvolvida; a seção 5 apresenta os resultados obtidos através dos testes da nova ferramenta. Na seção 6, são abordados os trabalhos relacionados e, por último, a seção 7 apresentam-se as conclusões do trabalho.

2. O Plágio e Sua Identificação

O plágio, segundo o dicionário Aurélio, pode ser definido como “tornar seu trabalho ou ideia de outrem”. A grande quantidade de informações disponíveis digitalmente tem potencializado esta prática, uma vez que a pesquisa e cópia foram simplificadas por poucas operações do teclado e mouse [Pertile *et al.* 2010]. Na literatura, é possível encontrar algumas classificações desta prática. Arenhardt *et al.* (2012) *apud* Kirkpatrick (2007) o classificam em três tipos: plágio direto, plágio por referência vaga ou incorreta e plágio mosaico. Destes três tipos, o plágio mosaico se demonstra o mais elaborado e difícil de se detectar, pois tratam-se de modificações realizadas em trechos copiados de um trabalho original, no intuito de camuflar este ilícito.

A forma básica de se identificar plágio em textos digitais é através do emprego de algoritmos de comparação de *strings*. Porém, para se obter resultados mais precisos, podem-se usar técnicas de pré-processamento de texto, ao exemplo da substituição de caracteres acentuados por não-acentuados, remoção de *stopwords* (artigos, advérbios, pronomes, preposições, dentre outros) e aplicação de *stemming* [Álvila e Soares 2013]. Um exemplo de algoritmo de detecção de plágio é o método DIP – Detector de Indícios de Plágio [Arenhardt 2013], [Arenhardt *et al.* 2012], [Pertile 2011], que é apresentado a seguir.

2.1 Método DIP – Detector de Indícios de Plágio

A construção do aplicativo Miss Marple original, programa que deu origem a este trabalho, está baseada no método DIP – Detector de Indícios de Plágio. O funcionamento deste método de análise consiste dos seguintes passos, que são exibidos pela figura 1, a seguir:

- Submissão de arquivos e conversão para texto plano: o usuário seleciona arquivos a serem inspecionados pelo programa, os quais tem seu conteúdo convertido para texto plano. Figuras são desconsideradas pelo processo de análise.
- Pré-processamento: os conteúdos dos arquivos lidos na etapa anterior passam por um estágio de remoção de *stopwords*, substituição de caracteres acentuados e conversão para letras minúsculas. No caso específico do Miss Marple original, existe também nesta etapa a aplicação de *stemming*.
- Tokenização: os parágrafos do texto resultante da etapa anterior são agrupados em *tokens* contendo até 50 palavras, os quais são enviados para pesquisa na Internet utilizando-se a *Google Search Ajax*[1].
- Recuperação dos resultados e cálculo de similaridade: os resultados da busca na Internet são baixados e seus conteúdos são comparados com o texto o qual está sendo analisado.

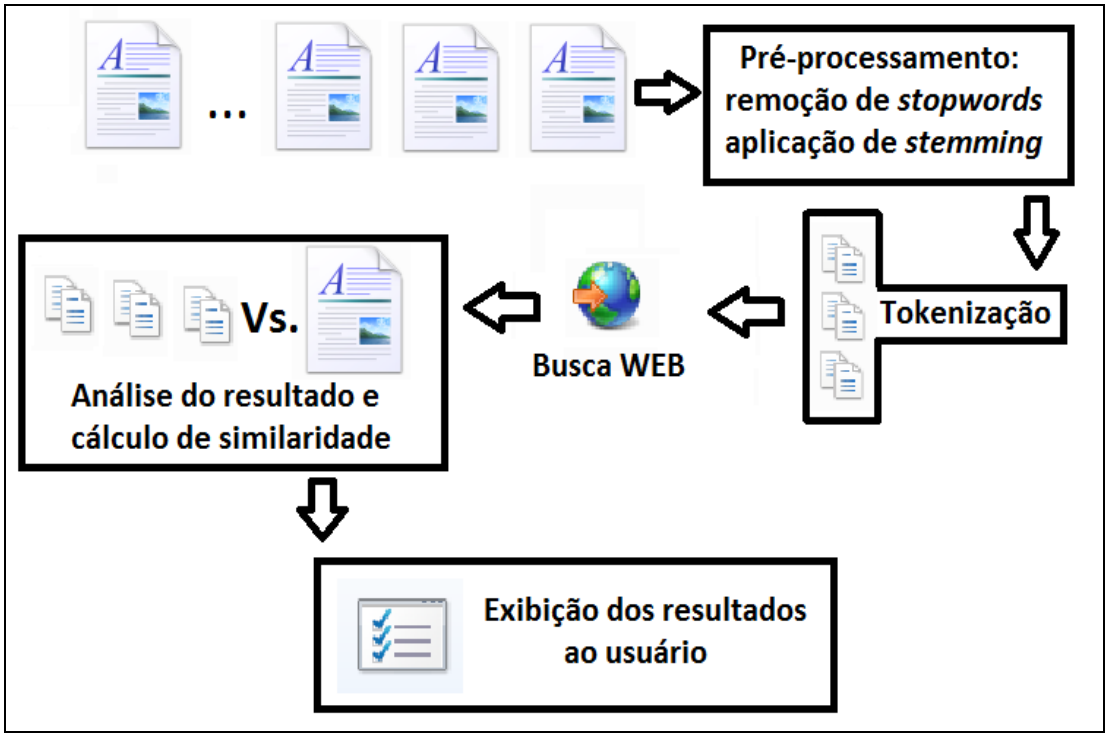


Figura 1. Etapas do método de detecção de plágio. Fonte: próprio autor.

[1] A API Google *Search Ajax* demonstrou-se operacional durante a implementação e realização dos experimentos. Atualmente, encontra-se desativada.

Para o cálculo de similaridade, faz-se uso de um algoritmo chamado *Similaridade Cosseno*. Este algoritmo retorna um percentual de similaridade entre um trecho de texto submetido e o texto recuperado da *web*. Arenhardt (2013) e Pertile (2011) definem 60% como sendo um valor máximo de similaridade aceitável, portanto, valores acima deste limiar são considerados indicativos da ocorrência de plágio. Vale ainda ressaltar que embora o programa possa apontar o indício de plágio em um documento, a confirmação de sua ocorrência deve ser feita pelo usuário.

3. Materiais e Métodos

Este trabalho se originou através da utilização experimental do aplicativo de análise de plágio Miss Marple para a verificação de trabalhos em formato de texto digital. Estando identificados os pontos limitantes e o potencial de aprimoramento, deu-se início à escolha dos materiais e métodos a serem utilizados.

A linguagem de programação escolhida para o desenvolvimento deste trabalho foi o Java, devido ao fato de que o Miss Marple original também ser desenvolvido em Java. Para edição e construção do código, optou-se por fazer uso da ferramenta NetBeans IDE, devido a esta se tratar de uma ferramenta gratuita e bem consolidada. Em seguida, foram elencadas as arquiteturas de paralelismo a serem empregadas no trabalho. Foram escolhidas as arquiteturas multiprocessador e multicomputador, uma vez que estas arquiteturas permitem o uso de *hardware* doméstico, mais acessível ao público-alvo.

No intuito de paralelizar o programa em nível multi-CPU, optou-se por realizar uma implementação baseada em *threads*. Já ao que se refere na paralelização em nível multicomputador, optou-se por utilizar o Java Remote Method Invocation (RMI), para troca de mensagens entre processos.

Para testes e obtenção de resultados dispôs-se de dois computadores: um Core i7 860 2.8GHz, 4 núcleos/8 threads, 8GB de memória RAM, e um Core2 Quad Q6600 4 núcleos, 2.4GHz, 4GB de memória RAM. Foi disposto de uma base contendo 24 artigos em formato PDF, com tamanho variando de 300KB até 3MB. O conjunto de trabalhos foi analisado em seis casos diferentes como seguem:

- Análise de documentos utilizando o programa original, no computador Core i7;
- Análise utilizando a nova versão desenvolvida utilizando-se 1, 2, 4 e 8 *threads* de processamento, no computador Core i7;
- Análise utilizando a nova versão, distribuída entre o computador Core i7 e o Core2 Quad.

Para cada caso de utilização, foram executados três testes. Os tempos totais de execução foram aferidos através do uso da ferramenta *NetBeans Profiler*. Os registros obtidos englobam tempo de processamento e tempo de comunicação (transmissão de dados na rede, busca na Internet). Em cada caso, foi calculado o tempo médio dos três testes, os quais foram comparados. Os resultados podem são abordados na seção 5 deste trabalho.

4. Arquitetura do Parallel Miss Marple

A arquitetura do *Parallel Miss Marple* é similar à arquitetura do Miss Marple original, também fazendo uso do método DIP. Porém, o grande diferencial é que os arquivos submetidos são analisados concorrentemente, ao contrário da versão anterior do programa, que executava a análise de um único arquivo por vez.

A nova versão do programa é composta por *threads* independentes de análise (analisadores), as quais são executadas de forma independente uma da outra e de forma concorrente, ou seja, ao mesmo tempo. Cada *thread* implementa o método DIP, descrito anteriormente, da mesma forma que o programa anterior o fazia. Uma analogia a ser feita é que o *Parallel Miss Marple* pode ser visto, de forma simplificada, como várias instâncias do programa original encapsulado, sendo executadas ao mesmo tempo. A figura a seguir (figura 2) ilustra este paradigma.

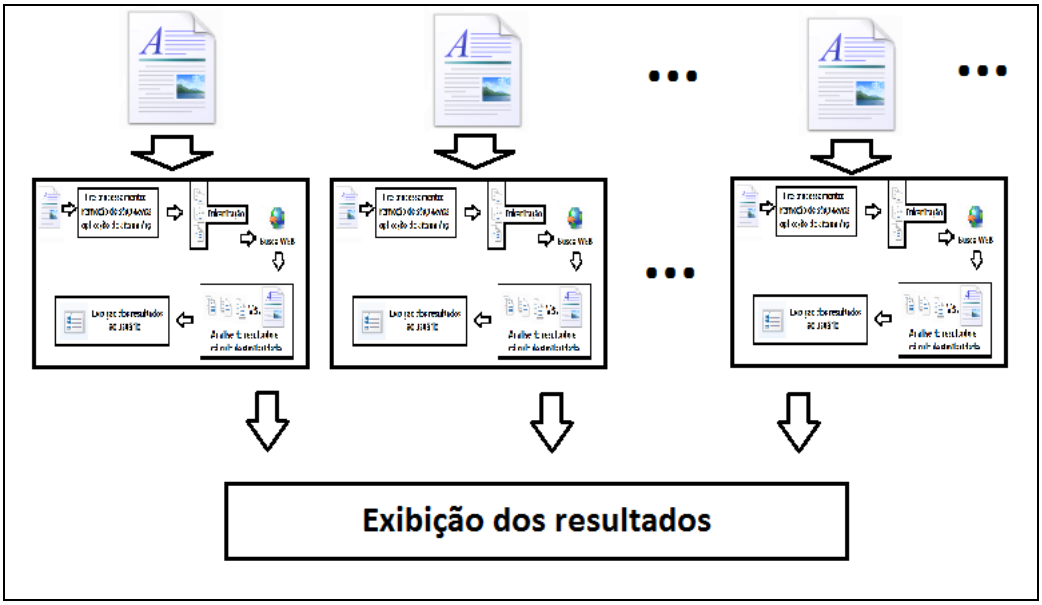


Figura 2. Intâncias do Parallel Miss Marple. Fonte: próprio autor.

Por padrão, o *Parallel Miss Marple* cria tantos analisadores quanto forem os núcleos de processadores disponíveis no computador em que está sendo executado. Ou seja, em um processador *quad-core*, são criados quatro analisadores. Uma vez criadas as *threads*, o programa executa a distribuição de tarefas por dentre elas, utilizando uma algoritmo *round-robin*. A configuração do programa, com relação ao número de *threads* empregadas pode ser editada pelo usuário, através do menu de configurações existente na interface, conforme exibido no campo 1 da figura 3, a seguir. Existem três possibilidades de entrada neste campo:

1. Valor "-1": O número de *threads* criadas é ilimitado, ou seja, o aplicativo cria um analisador para cada arquivo submetido pelo usuário.
2. Valor "0": É o valor padrão da aplicação, onde são criados tantos analisadores quanto forem os núcleos de processamento.
3. Valor inteiro maior do que zero: especifica exatamente quantos analisadores devem ser criados.

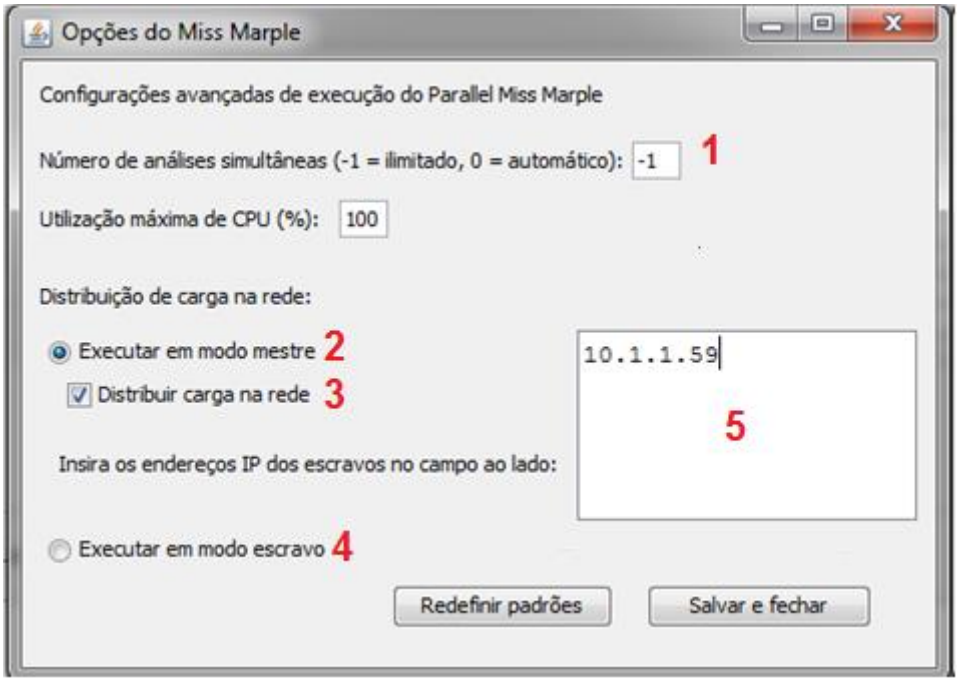


Figura 3. Interface de configuração do Parallel Miss Marple.
Fonte: próprio autor.

O *Parallel Miss Marple* também suporta distribuição de tarefas entre outros computadores na rede, podendo executar várias *threads* em vários computadores em uma mesma rede local. Para isto, o usuário deve executar o programa em modo mestre (item 2 da figura 3), habilitar a distribuição de trabalho (item 3) e informar os endereços dos computadores escravos no campo 5. Nos demais computadores, o programa deve ser executado no modo escravo (item 4).

Executando de forma distribuída, o aplicativo configurado como "mestre" faz a distribuição de tarefas de forma circular dentre todos os computadores executando o *Parallel Miss Marple* na rede. Para isto, é enviada uma cópia dos arquivos atribuídos a cada computador "escravo" via conexão *socket*. Em seguida, o programa mestre executa solicitações de análises remotas através de chamadas RMI, as quais serão processadas em cada um dos computadores escravos. Neste modo de funcionamento, é criada um *thread* para cada documento analisado (semelhante ao modo "ilimitado"). Vale também ressaltar o programa mestre funciona concomitantemente como escravo.

5. Resultados

Conforme descrito previamente na metodologia, realizaram-se testes, onde foram aferidos e registrados os tempos de processamento da nova ferramenta desenvolvida e os da versão antiga, através do NetBeans Profiler. Os resultados obtidos através deste procedimento são exibidos pelo quadro 1 e comparados a seguir.

Quadro 1. Tempos de processamento obtidos, em segundos.

Teste	*Threads*	Tempo 1	Tempo 2	Tempo 3	Média
Miss marple original	1	2248,612s	2133,674s	2196,363s	2192,884s
Parallel Miss Marple	1	2158,585s	2075,857s	2239,077s	2157,839s
Parallel Miss Marple	2	1023,959s	1193,045s	1087,457s	1101,486s
Parallel Miss Marple	4	693,292s	665,558s	703,217s	687,335s
Parallel Miss Marple	8	425,719s	401,502s	443,274s	423,498s
Parallel M. M. Distribuído	24	143,446s	155,260s	157,865s	152,190s

Como pode ser observado através dos dados apresentados no quadro, o desempenho da nova versão do programa (paralelizada), executando com apenas uma *thread*, possui desempenho similar ao da versão original. Entretanto, ao se aumentar o número de *threads*, observa-se redução no tempo necessário para analisar o mesmo número de documentos (total de 24 arquivos, em todos os testes).

Com posse dos tempos de processamento, pode-se também calcular o *speedup* obtido através da versão paralelizada do programa. *Speedup*, segundo Foster (1995), é uma medida de ganho de desempenho de um algoritmo paralelo, dada pela razão entre o tempo de execução sequencial e o tempo de execução paralela de um código, como pode ser visto através da figura 4.

$$\text{Speedup} = \frac{\text{Tempo}_{\text{Sequencial}}}{\text{Tempo}_{\text{Paralelo}}}$$

Figura 4. Fórmula do *speedup* paralelo. Fonte: adaptado de [Ohmann 2013]

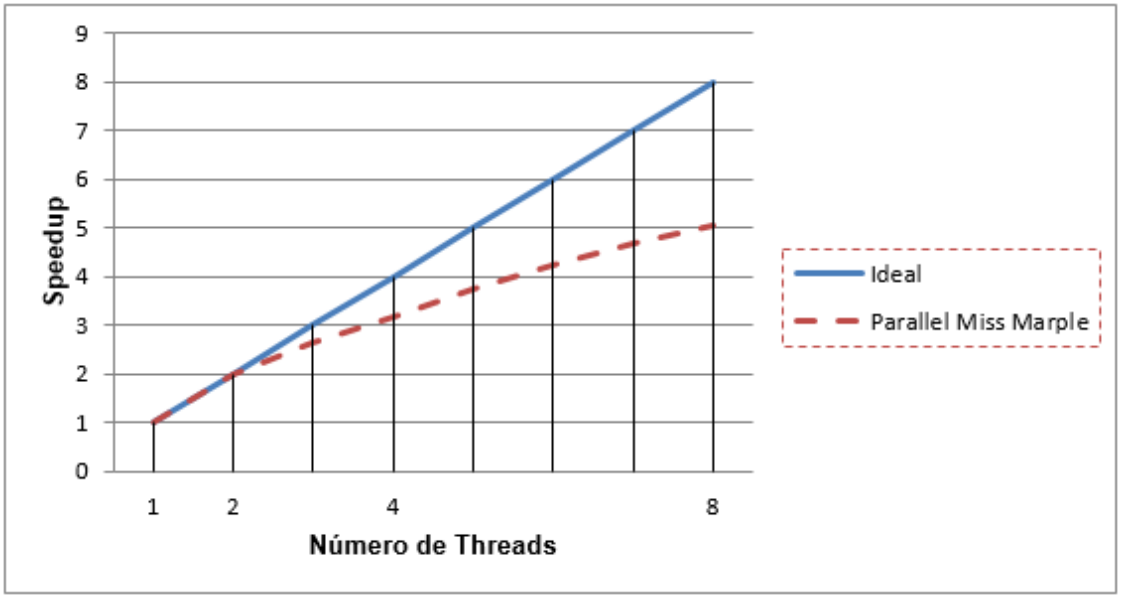
O *speedup* ideal é de ordem linear, ou seja, cresce na proporção 1:1. Ao se dobrar o número de *threads* de processamento, teoricamente, dobra-se o desempenho. Graficamente, o *speedup* ideal é representado pela reta identidade (y = x).

A seguir são apresentados os resultados contendo o *speedup* relacionado ao número de *threads* (quadro 2), e um gráfico comparativo entre o *speedup* ideal e os resultados obtidos através da execução do *Parallel Miss Marple.*

Quadro 2. *Speedup versus* número de *threads*.

Threads	1	2	4	8	24 (Distribuído)
Speedup	1	1,9908	3,1903	5,0702	14,408

Gráfico 1. *Speedup* ideal *versus* obtidos no trabalho.



Ao se analisar os dados do quadro 2 e do gráfico 1, percebe-se um considerável ganho de desempenho através do uso da nova versão da ferramenta. Observa-se que com o uso de duas *threads* de análise, o *Parallel Miss Marple* se situa bastante próximo do gráfico ideal de melhoria de desempenho. Ou seja, com dois analisadores, o código paralelizado é 1,9908 (aproximadamente 2) vezes mais rápido do que o código sequencial (versão antiga do programa). Em outras palavras, o código paralelo executado com duas *threads* de análise se demonstrou aproximadamente 50% mais rápido do que a execução sequencial. Com um número maior de analisadores, obteve-se ainda ganho de desempenho, porém, os ganhos obtidos distanciaram-se da reta de melhoria ideal.

Com relação à precisão dos resultados, o programa paralelizado demonstrou possuir a mesma precisão que a versão antiga, uma vez que o método de análise de documentos (método DIP) permaneceu inalterado. A nova ferramenta possui assim, boa acurácia em suas análises, conforme o método empregado foi demonstrado e comprovado por Arenhardt (2013).

6. Trabalhos Correlatos

Devido ao exposto na introdução, muitos estudos têm sido desenvolvidos no intuito de aprimorar métodos de detecção de indícios de plágio e minimizar o esforço necessário para que os avaliadores realizem esta tarefa. É importante salientar que a análise de plágio também ocorre em outros contextos, além de revisões em documentos de textos científicos/escolares, incluindo até mesmo análise de documentos de código fonte de programas. Através da revisão da literatura, é possível obter trabalhos relacionados a detecção de plágio, os quais são apresentados a seguir.

Em "*Efficient Clustering-based Plagiarism Detection using IPPDC*", Ohmann (2013) apresenta o desenvolvimento de uma ferramenta para detecção de plágio em códigos-fonte, chamada de *Inteligent Parallel Plagiarism Dettection using Clustering*. Neste trabalho, o autor demonstra a preocupação em se produzir uma ferramenta de

análise de plágio otimizada para as arquiteturas multicore atuais. O autor faz uso de *threads* em Java e executa um processo de agrupamento de dados antes do estágio de cálculo de similaridade, obtendo boa precisão de análise e bons tempos de processamento. Diferentemente deste trabalho, o autor realizou testes com diferentes quantidades de arquivos de entrada, obtendo valores de *speedup* diferentes.

Em "*AntiPlag: Plagiarism detection on electronic submissions of text based assignments*", Jiffriya *et al.* (2013) apresentam o desenvolvimento do AntiPlag, uma ferramenta de análise de plágio em documentos de texto digital. De forma similar a este trabalho, o aplicativo desenvolvido desconsidera imagens, e o conteúdo restante é convertido para texto plano. Em seguida é realizada a *tokenização*, onde os *tokens* passam por um algoritmo de cálculo de similaridade, sendo comparados com um documento sabidamente autêntico. O AntiPlag não executa busca na WEB. Também não existe uma abordagem a respeito do emprego de paralelismo para otimizar o processamento de documentos.

Em *"Plagiarism Detection by Identifying the Keyword"s* [Dutta e Bhattacharjee 2014], aborda-se uma forma de classificar documentos como "possivelmente plagiados" a partir de suas palavras-chave. As palavras-chave de um artigo em formato PDF são identificadas e seus sinônimos são pesquisados em um banco de dados. Em seguida, as palavras-chave e seus sinônimos são comparados com um documento previamente conhecido como original. Havendo semelhança entre os termos obtidos da etapa anterior e os termos do documento original, o documento é classificado como "possivelmente plagiado". Vale ressaltar, porém que este método não analisa o documento inteiro, assim como não realiza busca na Internet.

Com base no exposto, o corrente trabalho se diferencia dos demais pelo fato de se tratar de uma ferramenta *open-source* de detecção de indícios de plágio em documentos digitais, que realiza análise de todos os trechos de texto existentes nos documentos, possuindo recurso de buscas na Internet e apresentando suporte a tecnologias de processamento paralelo e distribuído.

7. Considerações Finais

O presente trabalho teve como objetivo o aprimoramento do *software* de busca por indícios de plágio *Miss Marple* [Arenhardt 2013], de forma a mitigar a questão do alto tempo necessário para realizar a análise de grandes lotes de documentos digitais. O objetivo proposto foi alcançado, conforme apontam os dados exibidos na seção 5.

As principais contribuições deste trabalho são: a) redução do tempo total de análise vários documentos; b) suporte a distribuição de tarefas entre outros computadores executando o *software*; c) salvamento das configurações, para posterior utilização em novas execuções do programa.

Através dos resultados dos testes expostos, conclui-se que a divisão de tarefas utilizando-se métodos de paralelismo baseados em multi-CPU e multicomputador demonstra contribuir com processo de análise de plágio em documentos digitais. Com estas tecnologias, a busca de plágio pode ser otimizada, trazendo benefícios para os professores de escolas e de instituições de ensino superior, por meio da redução considerável do tempo necessário para se realizar esta tarefa. Os benefícios ocorrem

principalmente quando se dispõe de uma grande quantidade de arquivos a serem verificados, cenário vivenciado por muitos educadores. Vale ainda ressaltar que a identificação do plágio dentro de um tempo hábil é de extrema importância pois, desta forma, o educador pode tomar as medidas adequadas para minimizar esta prática e assim, aprimorar a construção do intelecto e do caráter idôneo de seus alunos.

Por fim, são sugeridos como trabalhos futuros: a) o desenvolvimento de uma versão com a capacidade de execução em ambientes sem interface gráfica, como aqueles encontrados em servidores; b) a criação de um repositório de arquivos localizado em uma nuvem computacional, de forma a reduzir o número de consultas através da API de busca e c) realização de testes com turmas dentro de instituições de ensino.

Referências

Álvila, R. L., Soares, J. M. (2013) “Uso de técnicas de pré-processamento textual e algoritmos de comparação como suporte à correção de questões dissertativas: experimentos, análises e contribuições”, em Anais do XXIV SBIE.

Arenhardt, C. P. B. (2013) “Miss Marple - Desenvolvimento de Ferramenta Para Auxiliar na Verificação e Detecção de Indícios de Plágio com Base no Método DIP - Detector de Indícios de Plágio”, UFSM.

Arenhardt, C. P. B., Medina, R. D., Pertile, S. L., Gomes, R. B., Trindade, V. L. (2012) “Miss Marple Proposta de Desenvolvimento de Ferramenta de Detecção de Indícios de Plágio com base no Método DIP Detector de Indícios de Plágio”, em Anais do XXIII SBIE.

Dutta, S., Bhattacharjee, D. (2014) "Plagiarism Detection by Identifying the Keywords", em Computational Intelligence and Communication Networks (CICN), p.703-707, 14-16 de Novembro.

Foster, I. (1995) “Designing and Building Parallel Programs”.

Jiffriya, M.A.C., Jahan, M.A.C.A., Ragel, R.G., Deegalla, S. (2013) "AntiPlag: Plagiarism detection on electronic submissions of text based assignments", em Industrial and Information Systems (ICIIS), 2013 8th IEEE International Conference on Industrial and Information Systems, p.376-380, 17-20 de Dezembro.

Moraes, R. (2004) “O plágio na pesquisa acadêmica: a proliferação da desonestidade intelectual”. Disponível em www.faculdadesocial.edu.br/revistas/index.php /dialogospossiveis/article/view/191/146. Acesso em junho/2015.

Ohmann, A. (2013) "Efficient Clustering-based Plagiarism Detection using IPPDC", em College of Saint Benedict and Saind John's University. Disponível em http://digitalcommons.csbsju.edu/cgi/viewcontent.cgi?article=1015&context=honors _theses. Acesso em Julho/2015.

Pertile, S. L., Piovesan, S. D., Lobo, Medina, R. D. (2010) “Agente Integrado a Plataforma MLE-Moodle para Detecção Automática de Indícios de Plágio”, em Anais do SBIE, 2010.

Pertile, S.L. (2011) “Desenvolvimento e Aplicação de um Método para Detecção de Indícios de Plágio”, UFSM.