JUAN ANDRÉS MUSSINI

NOVAS ARQUITETURAS PARA DETECÇÃO DE PLÁGIO BASEADAS EM REDES P2P

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para a obtenção do título de Mestre em Informática.

Curitiba PR
Maio de 2008

JUAN ANDRÉS MUSSINI

NOVAS ARQUITETURAS PARA DETECÇÃO DE PLÁGIO BASEADAS EM REDES P2P

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para a obtenção do título de Mestre em Informática.

Área de concentração: *Ciência da Computação*

Orientador: Altair Olivo Santin
Co-orientador: Lau Cheuk Lung

Curitiba PR
Maio de 2008

Pontifícia Universidade Católica do Paraná
Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Informática

ATA DE DEFESA DE DISSERTAÇÃO DE MESTRADO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

DEFESA DE DISSERTAÇÃO Nº 01/2008

Aos 22 dias do mês de fevereiro de 2008 realizou-se a sessão pública de Defesa da Dissertação **"Novas Abordagens para Detecção de Plágio Baseadas em Redes P2P"**, apresentada pelo aluno **Juan Andrés Mussini** como requisito parcial para a obtenção do título de Mestre em Informática, perante uma Banca Examinadora composta pelos seguintes membros:

Prof. Dr. Altair Olivo Santin PUCPR (Orientador)	(assinatura)	Aprovado (aprov/reprov.)
Prof. Dr. Alcides Calsavara PUCPR		Aprovado
Prof. Dr. Lau Cheuk Lung UFSC		Aprovado
Prof. Dr. Luiz Nacamura Junior UTFPR		Aprovado

Conforme as normas regimentais do PPGIa e da PUCPR, o trabalho apresentado foi considerado Aprovado (*aprovado/reprovado*), segundo avaliação da maioria dos membros desta Banca Examinadora. Este resultado está condicionado ao cumprimento integral das solicitações da Banca Examinadora registradas no Livro de Defesas do programa.

Prof. Dr. Alessandro Lameiras Koerich
Diretor do Programa de Pós-Graduação em Informática

Programa de Pós-Graduação em Informática
PPGIa
PUCPR

Rua Imaculada Conceição, 1155 Prado Velho CEP 80215 901 Curitiba Paraná Brasil
Tel.: (41) 3271 1669 Fax: (41) 3271 2121 www.ppgia.pucpr.br

Último sobrenome (seguido de Jr., Filho ou Neto), Primeiro nome e demais sobrenomes.

Título da Dissertação ou Tese. Curitiba, 2001. 88p.

Dissertação (Mestrado) ou Tese (Doutorado) - Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática.

1. Palavra-chave 2. Palavra-chave 3. Palavra-chave 4. Palavra-chave. I. Pontifícia Universidade Católica do Paraná. Centro de Ciências Exatas e de Tecnologia. Programa de Pós-Graduação em Informática II-t.

A mis padres...

Agradecimentos

À minha família, a quem devo tudo. Me deram forças para seguir adiante e sempre estão ao meu lado.

Aos professores e funcionários do PPGIA e da PUCPR em geral, que de alguma forma me ajudaram neste processo.

Ao Edson Kageyama, vulgo Manga, por começar e terminar esta caminhada junto comigo, me incentivando quando necessário.

Ao Fábio Favarim, que mesmo nunca tê-lo conhecido pessoalmente foi essencial na escrita do trabalho.

À Mariana, por dar um toque de classe no português e na escrita do trabalho.

Aos meus amigos, por me distrair nas horas vagas.

Resumo

Hoje em dia a *Internet* se tornou uma referência em aquisição de informações, graças à grande quantidade presente e a sua facilidade de acesso. Mas isto pode ser mal utilizado, já que alguém pode simplesmente acessar uma informação e usá-la como de sua própria autoria. Isto caracteriza uma ato de plágio. Tem sido cada vez mais comum que as pessoas façam isto, e ferramentas para prevenção deste ato são necessárias. A fim de ajudar a lidar com este problema, é apresentado o PeerDetect, um sistema de detecção de plágio elaborado sobre uma infraestrutura P2P. A solução proposta apresenta duas abordagens. A primeira permite ao usuário detectar plágios em nós que pertencem a uma rede P2P, procurando por documentos nesta rede, o que inclui documentos que podem não estar disponíveis na *Internet*. A segunda abordagem é similar ao sistemas de deteção convencionais, os quais detectam plágios na *Internet*. Esta abordagem também utiliza redes P2P para distribuir o trabalho da busca entre os nós participantes. As soluções foram testadas utilizando diferentes documentos, com tamanhos variados, e redes com diferentes número de nós. Os resultados foram satisfatórios, provando ser um bom mecanismo de deteção de plágio.

Palavras-chave: detecção de cópia, plágio, P2P.

Abstract

Nowadays the Internet has become a reference on information retrieval, thanks to its large amount and easiness to access it. But this can be misused, as one can simply access an information and take it as his own authorship. This constitutes an act of plagiarism. It has become increasingly common for people to do this, and tools to prevent this are in need. In order to help to deal with this problem, this paper presents the PeerDetect, a plagiarism detection system built over a Peer-to-Peer (P2P) infrastructure. The proposed solution offers two approaches. The first allows users to detect plagiarism in peers that belongs to a P2P network, searching through documents that may not be available on the Internet. The second approach is similar to conventional detection systems that detects plagiarism on the Internet, but this approach also uses P2P networks to distributed the effort of doing this search. As presented in this paper, the solution has shown itself as a strong mechanism for plagiarism detection.

Keywords: copy detection, plagiarism, P2P.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

P2P	*Peer-to-Peer*
DHT	*Distributed Hash Tables*
LCS	*Longest Common Subsequence*
COPS	*COpy Protection System*
SCAM	*Stanford Copy Analysis Mechanism*
RFM	*Relative Frequency Model*
URL	*Universal Resource Locator*
XML	*Extensible Markup Language*
RDT	*Resource Description Format*
W3C	*World Wide Web Consortium*
PKI	*Public Key Infrastructure*
SOAP	*Simple Object Access Protocol*
WSDL	*Web Services Description Language*

UDDI	*Universal Description, Discovery and Integration*
RPCs	*Remote Procedure Calls*
HTTP	*Hypertext Transfer Protocol*
NATs	*Network Address Translation*
TTL	*Time to live*
API	*Aplication Programming Language*
GPL	*GNU Public License*

Capítulo 1

Introdução

O advento e a expansão da *Internet* trouxeram benefícios inquestionáveis para a socialização da informação. A *Internet* tem sido essencial na vida de muitas pessoas, já que facilita o acesso a uma vasta quantidade de informações. Esta facilidade contribui para uma distribuição eficiente do conhecimento humano e sua aquisição [13]. No entanto, esta socialização da informação também trouxe problemas. No ambiente acadêmico, e até mesmo nas empresas, tem havido casos de pessoas ou companhias que tomam posse de informações adquiridas da *Internet*, obtendo autoria de forma fraudulenta [14], caracterizando o plágio.

Segundo o dicionário Larousse, plágio é definido como "1. Ação de copiar obras alheias, apresentando-as como de sua própria autoria. 2. Imitar trabalho alheio". Plágios podem ser feitos com diferentes intenções. Na academia, professores e pesquisadores têm dificuldade em determinar a autoria de trabalhos acadêmicos e científicos apresentados por seus alunos. Nas empresas, uma companhia pode tomar informação de outra a fim de obter mais lucro sem ter de investir na sua criação. Na maioria das vezes, o que pode ser preocupante, a determinação da autoria passa à segunda instância, tendo que se confiar na honestidade das pessoas e das empresas. O plágio é um problema sério, pois consiste na apropriação criminosa da propriedade intelectual de outra pessoa, a fim de se obter uma vantagem econômica, social ou profissional. Isto faz do plágio um tópico importante a ser lidado e muito comum atualmente.

Ocorrências de plágios têm aumentado nos últimos anos [14], e ferramentas para fazer cópias cresceram na mesma proporção, desde uma simples busca em um indexador de conteúdo, como o Google [15], a páginas que oferecem trabalhos já prontos [16, 17]. Muitos professores, especialmente aqueles que não são de áreas relacionadas à computação, não possuem informação sobre técnicas de detecção de fraudes.

Comumente é utilizada a checagem manual, onde um professor seleciona frases aleatórias do documento e as utiliza para procurar por documentos iguais em uma ferramenta de busca. No entanto, esta checagem manual consome muito tempo e é um processo laborioso, fazendo com que não seja amplamente praticado.

Nos dias atuais, algumas ferramentas têm surgido [18, 2] a fim de automatizar este processo. Estas ferramentas geralmente indicam qual o grau de similaridade entre dois documentos. Caso este grau seja considerado alto, uma checagem manual por um especialista, como por exemplo, um professor, é necessária. Esta checagem manual é essencial, já que não existe um sistema de detecção que garanta um grau de similaridade de 100%.

Desta maneira, é proposto um sistema de detecção de plágio em documentos escritos em linguagem natural chamado PeerDetect, o qual se utiliza de duas abordagens diferentes para a detecção de plágio.

A primeira (abordagem centralizada) difere da maioria dos sistemas de detecção de plágio, pois seu repositório é formado por documentos de usuários para procurar por cópias. A maioria dos sistemas somente utiliza documentos disponíveis na *Internet*, o que significa que o documento deve estar disponível publicamente de alguma maneira na *Internet*. No entanto, muitos documentos somente estão disponíveis nos computadores pessoais. Desta forma, a probabilidade de encontrar uma cópia de um documento é reduzida. PeerDetect possibilita utilizar, como fonte de dados, os computadores pessoais de vários usuários conectados na rede *Peer-to-Peer* (P2P). Estes computadores atuam como nós na rede P2P. Inicialmente, o algoritmo de detecção de plágio extrai informações sobre o documento analisado. Esta informação é comparada a outros documentos na rede P2P com a mesma área de interesse. Documentos que possuem informações similares são transferidos para o nó que iniciou a busca e, após a transferência, um segundo e mais eficiente algoritmo de detecção de plágio é aplicado. Este algoritmo é executado localmente nesse nó. Os resultados deste segundo passo são apresentados para o usuário, para que seja feita uma avaliação mais a fundo.

A segunda (abordagem descentralizada) utiliza a *Internet* como fonte de dados e redes P2P como uma forma de distribuir o trabalho de procurar na *Internet* entre os nós. Quando os nós ingressam na rede P2P, eles se assinalam como nós disponíveis para trabalhar. No momento em que um nó recebe um documento a ser procurado, ele passa a ser um nó dono. Ele analisa o tamanho deste documento e o número de nós disponíveis na rede. Ele então define em quantas partes o documento irá dividir-se e qual será a relação nó/parte do documento. Cada parte é enviada ao nó responsável designado a essa parte. Estes nós fazem uma busca simples utilizando

um mecanismo de busca na *Internet* e então publicam as páginas encontradas. O nó dono coleta estas respostas e as apresenta ao usuário para avaliação.

Um exemplo desta abordagem são cópias evidentes feitas diretamente da *Internet*. Este é o caso da maioria dos alunos que para resolver uma tarefa simplesmente procuram a resposta na *Internet* e, quando encontrada, copiam como sendo de sua autoria.

O restante desta dissertação está organizada da seguinte maneira. O Capítulo 2 apresenta os conceitos de detecção de plágio e os trabalhos relacionadosas. O Capítulo 3 apresenta as arquiteturas P2P, incluindo as as *Distributed Hash Tables* (DHT). O Capítulo 4 descreve o sistema de detecção de plágio proposto, e o Capítulo 5 seus testes e resultados. Finalmente, o Capítulo 6 conclui a dissertação.

Capítulo 2

Detecção de Plágio

2.1 Introdução

Pode-se dividir a detecção de plágio em dois tipos: detecção de plágio em códigos fontes e em documentos de texto. Para detecção em códigos fontes, existe a técnica de contagem de atributos e técnica de métrica de estruturas (conforme [19]).

Há mais literatura referente a plágio em códigos fontes do que em documentos escritos. Em partes, isso pode acontecer devido ao fato de que detectar plágio em código fonte é mais simples do que em linguagens naturais. A gramática completa da linguagem de programação pode ser definida e especificada. No entanto a linguagem natural é muito mais complexa e ambígua.

Este capítulo foi baseado nas informações presentes em [3].

2.2 Plágio em código fonte

Whale [20] cita os métodos mais comuns utilizados pelos alunos para disfarçar programas copiados:

- Alterar comentários.
- Alterar tipos de dados.
- Alterar identificadores.
- Adicionar variáveis e declarações redundantes.

Parker e Hamblen [1] declaram que as transformações feitas de um programa a outro podem variar de muito simples a muito complexas, conforme Figura 2.1, de Faidhi e Robinson [21].

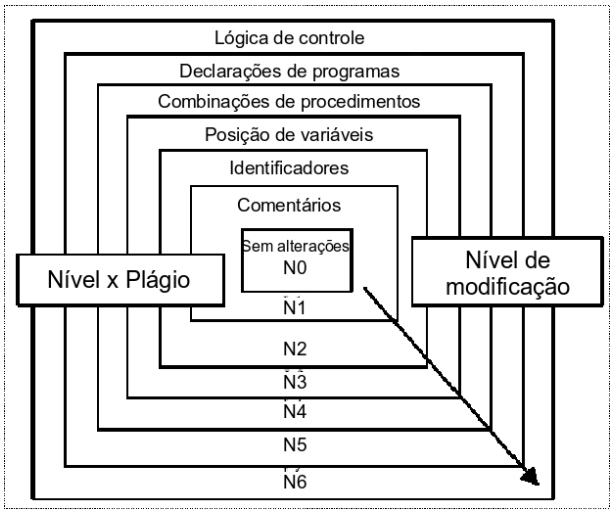

Figura 2.1: Espectro de plágio em programas [1]

O primeiro software de detecção foi um programa de contagem de atributos desenvolvido por Ottensein em 1976. Este programa foi desenvolvido para detectar trabalhos acadêmicos quase idênticos ou idênticos. O programa utilizava as métricas de ciência de software de Halstead para contar operandos e operadores para módulos ANSI-FORTRAN.

As métricas sugeridas por Halstead eram:

- $n1$ - número de operadores únicos.
- $n2$ - número de operandos únicos.
- $N1$ - número total de ocorrências de operadores.
- $N2$ - número total de ocorrências de operandos.

Pares de programas com $n1$, $n2$, $N1$ e $N2$ idênticos eram considerados similares.

Em 1980, foi desenvolvido por Robinson e Soffa outro mecanismo de detecção que combinava novas métricas com as antigas de Halstead a fim de aumentar a eficácia da detecção. O sistema foi chamado de ITPAD e consistia em três passos: análise léxica, busca na estrutura do programa por características e análise dessas características.

A idéia deste sistema era quebrar o programa em blocos, desenhar dois gráficos para representar a estrutura de cada programa dos alunos, gerar uma lista de atributos baseado na análise léxica e estrutural e então comparar pares desses programas através da contagem dessas características.

Em 1981, Donaldson propôs o sistema de métricas de estrutura como um método adicional para melhorar o desempenho do já utilizado contagem de atributos. Eles identificaram técnicas simples que estudantes novatos utilizam para esconder cópias, como renomear variáveis, reordenar declarações, alterar o formato de declarações entre outros.

A detecção deste sistema é feita através da comparação da estrutura de dois programas. Inicialmente, o programa analisa cada trabalho e armazena informações sobre certos tipos de declarações. As declarações mais relevantes para a estrutura são marcadas com um caractere único. Cada trabalho é então representado por uma *string* de caracteres. Caso esta representação seja parecida ou igual, então o par de programas comparados é marcado como similar. As estruturas recentes mais conhecidas são o YAP3, Plague e JPLAG.

Estas estruturas seguem um mesmo princípio básico – a *tokenização*. Nesta fase, diferentes partes do código são substituídas por um *token* pré-definido e consistente. Cada trabalho é então representado por uma série de *tokens*. Os *tokens* de cada documento são comparados para detectar documentos similares.

Várias comparações detalhadas [20] [22] de algoritmos de contagem de atributos e de estrutura têm sido realizadas. As conclusões demonstram que os métodos de contagem de atributos utilizados individualmente são mecanismos de detecção com resultados piores do que os algoritmos de comparação de estruturas.

Conforme [23], um sistema que tenha algoritmos de comparação sofisticados é naturalmente mais complexo de ser implementado, muito provavelmente exigindo que o trabalho examinado seja totalmente analisado (*parsed*). O investimento para desenvolver tal algoritmo é alto, fazendo com que seja justificável num caso de software comercial, para detectar casos mais graves como violação de direitos autorais. Num contexto educacional, o esforço feito pelos alunos para esconder as cópias é limitado, conforme descrito no Capítulo 1. Um bom algoritmo de detecção de plágio em documentos acadêmicos tem de ser eficaz e suficientemente confiável com uma alta taxa de sucesso.

2.3 Plágio em documentos escritos

A identificação de plágio em textos escritos cabe normalmente ao professor ou tutor. Caso estejam familiarizados com o estilo de escrever do aluno, eles podem ser capazes de identificar irregularidades no trabalho se comparada a outros trabalhos do mesmo só que mais antigos ou até mesmo identificar vocabulários e linguagens diferentes utilizadas. Inicialmente, estas características podem identificar um plágio em potencial. Outras características suspeitas de plágios em documentos escritos são [3]:

- Uso de vocabulário: comparação de vocabulário com vocabulário conhecido. Quanto maior a diferença, ou seja, quanto mais palavras novas o documento possuir, menor a probabilidade de cópia.
- Mudança de vocabulário: caso o vocabulário utilizado mude constantemente dentro de um mesmo texto, isto pode indicar um caso de cópia.
- Texto incoerente: se o texto não é consistente, pode ser o indício de cópia.
- Pontuação: a pontuação varia muito de texto para texto. Se ela for similar em dois documentos, pode ser um caso de cópia.
- Quantidade de similaridade entre textos: quanto maior a similaridade de termos comuns como nomes, definições, maior a suspeita.
- Erros de gramática comuns: como erros de gramáticas comuns não são prováveis de acontecer em textos revisados, a presença destes em dois documentos separados pode indicar uma cópia.
- Estrutura do texto: quando dois textos possuem estruturas similares, como parágrafos ou seqüência de seções e capítulos.
- Seqüências longas de texto conhecido: texto conhecido (frases famosas, por exemplo) sem referências.
- Ordem de similaridade entre textos: se dois textos de documentos diferente possuem seqüências similares com somente algumas caracteríticas diferentes (palavras, pontuação).

- Dependência de certas frases e palavras: um autor prefere utilizar certas palavras em particular.

- Preferência no uso de sentenças longas ou curtas: cada autor prefere utilizar sentenças longas ou curtas.

- Capacidade de leitura (*Readability*) do texto escrito: utilizando métricas como o índice *Gunning FOG*, uma pontuação é dada ao documento. É difícil que dois autores diferentes tenham a mesma pontuação.

- Referências pendentes: referências que aparecem no texto mas não na bibliografia.

Para tudo isso é necessário determinar precisamente o estilo do autor. Uma boa forma de se fazer isso pode ser vista em [24].

Para determinar o estilo de escrita de um autor, seria necessário utilizar algumas das características citadas acima. Técnicas de estatísticas também foram definidas em [25], que envolvem contar a freqüência que algumas características ocorrem, elaborando assim um perfil de escrita. Estas incluem:

- Tamanho médio de sentenças (palavras).

- Tamanho médio de parágrafos (sentenças).

- O uso de voz passiva (expressado em porcentagem).

- O número de preposições (porcentagem em relação ao número total de palavras).

- A freqüência de palavras de função[1] utilizadas.

Existe também uma série de palavras e frases que são comuns em vários documentos diferentes, como nomes, datas, localizações, termos específicos e termos comuns.

Em [26], são descritas várias formas de plágio em linguagens naturais:

- Copiar diretamente da fonte: copiar sem colocar aspas indicando que é uma citação.

- Parafrasear: reescrever uma sentença com as próprias palavras, mas sem utilizar aspas indicando que é uma citação.

[1] Palavras com baixo significado léxico: artigos, preposições, pronomes, etc.

- Submeter trabalho alheio: copiar o trabalho de outra pessoa, com passagens idênticas.
- Não referenciar a fonte: novas informações que são apresentadas às pessoas que não são familiares com o assunto devem ser referenciadas apropriadamente.
- Cópia da *Internet*: a cópia de um conjunto de parágrafos de uma variedade de meios eletrônicos e colocadas todas juntas para fazer um documento.

Há ainda mais uma série de exemplos de plágio, como utilizar sinônimos para escrever uma mesma frase, mudar a ordem da frase, redução de um parágrafo em uma frase, entre outros.

2.4 Ferramentas existentes de detecção de plágio em documentos escritos

Baseado na lista fornecida em [3], fez-se uma pesquisa das principais ferramentas de detecção de plágio presentes. Além delas outras foram encontradas e apresentadas. A seguir, são apresentadas as ferramentas existentes voltadas para detecção de plágio em documentos escritos.

2.4.1 Plagiarism.org / Turnitin.com / iThenticate.com

Estes talvez sejam os maiores serviços *online* atualmente. Todos os três pertencem à mesma empresa (iParadigms), porém, possuem público-alvo diferente. A página Plagiarism.org [2] oferece uma gama de informações em geral a respeito de plágio, com estatísticas e opiniões dos autores. As outras duas páginas são mecanismos de detecção de plágio, uma voltada para área acadêmica e outra para área comercial.

Para detecção de plágio, primeiramente se faz uma impressão digital do documento submetido para análise, esta impressão é comparada com uma base de documentos e ao mesmo tempo é feita uma busca na *Internet*. No término da análise, é realizado um relatório customizado. São pesquisadas duas similaridades: palavras substituídas e frases adicionadas.

Para substituição de palavras, uma escala de 0 (sem similaridades) e 1 (cópia total) é utilizada. A distribuição adotada segue a Figura 2.2.

A linha cinza é o limite para que um documento seja considerado cópia. Quanto maior a diferença entre os documentos comparados, mais baixo o índice. Para ter uma noção da

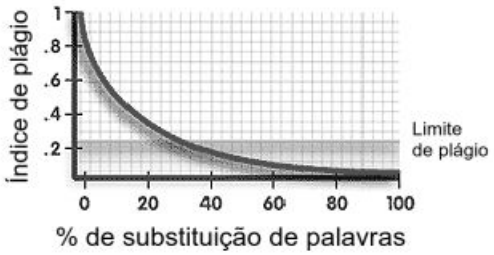

Figura 2.2: Gráfico de índice de plágio X porcentagem de palavras substituídas [2]

performance, os dois gráficos 2.3 e 2.4 apresentam os resultados de uma detecção. Em 2.3, o documento submetido é cópia de um documente presente na base, sem qualquer outro indício de falso positivo. Pode-se ver que o índice de plágio referente a esse documento é o único que excede a linha do limite de plágio. O restante dos documentos oscila sem ultrapassar a linha. Em 2.4, aproximadamente metade das palavras foram alteradas no documento, entretanto foi marcado como suspeito. Neste caso o gráfico apresenta o índice de plágio referente a esse documento ultrapassando levemente o limite de plágio. O restante dos documentos pernamece como no gráfico anterior.

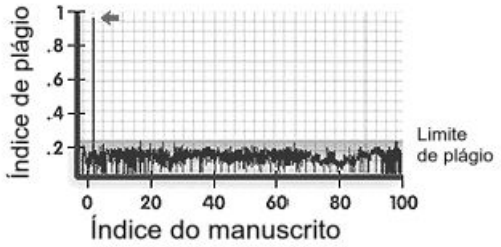

Figura 2.3: Performance do plagiarism.org [2]

Para adição de frases, a distribuição segue a Figura 2.5.

Para casos onde o índice é 1, significa que nenhuma palavra foi adicionada, ou seja, a cópia é idêntica. Mesmo tendo todo outro documento adicionado, ainda é caracterizado um plágio, mesmo que em menor grau.

Um caso onde se encontra uma cópia exata do documento é demonstrado no gráfico 2.6. No gráfico 2.7 , o documento é copiado parcialmente de duas fontes diferentes, aumentando o nível de ambas.

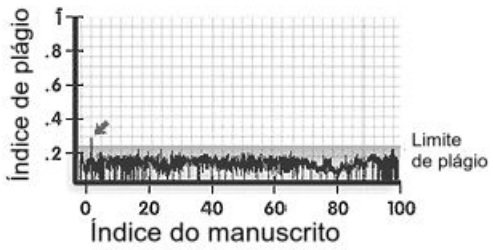

Figura 2.4: Performance do plagiarism.org [2]

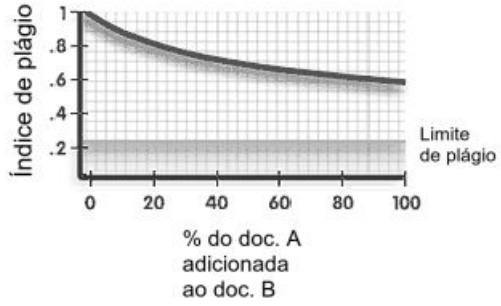

Figura 2.5: Gráfico do índice de plágio X porcentagem de cópia de duas fontes [2]

Há ainda um mecanismo de redução de buscas, onde documentos que abordam assuntos diferentes não são comparados.

Não foi possível obter informações sobre preço, já que ele é estabelecido sob acordo.

2.4.2 EVE2

O EVE2 é voltado para a área acadêmica e utiliza a *Internet* para procurar documentos similares ao submetido. Este precisa ser no formato texto, Microsoft Word ou Corel Draw Word Perfect. O sistema retorna uma lista dos endereços encontrados, junto com um grau de similaridade. O trabalho passa então, a ser manual. O sistema teoricamente utiliza uma série de técnicas que diminui o retorno de falsos positivos.

O custo do EVE2 é $30 por licença e declara ser melhor que o plagiarism.org.

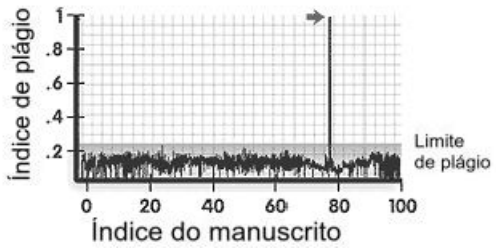

Figura 2.6: Performance do plagiarism.org [2]

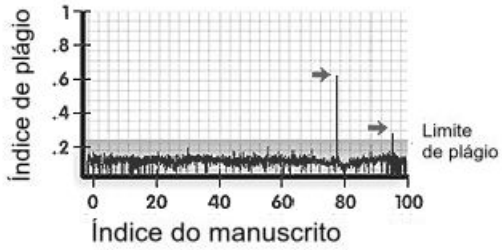

Figura 2.7: Performance do plagiarism.org [2]

2.4.3 CopyCatch

O CopyCath é um sistema voltado a universidades para detecção automática de plágio. O sistema é um melhoramento de uma série de ferramentas de análise lingüística desenvolvidas para ajudar lingüistas forenses na busca de textos curtos. Este trabalho foi realizado por um grupo na Universidade de Birmingham.

A similaridade de documentos neste caso é determinada pela semelhança léxica entre pares de documentos, analisando o vocabulário utilizado e semelhança de frases entre os textos.

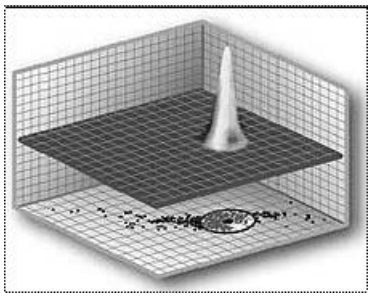

Figura 2.8: Gráfico representando abordagem por conteúdo [2]

O programa lê um conjunto de documentos, compara um ao outro e fornece o número de palavras em comum. O grau de similaridade é baseado no número de ocorrências de palavras no oposto às ocorrências de vocabulário baseado na hipótese de que as palavras substituídas não será feita novamente, conseqüentemente vestígios do original não irão aparecer. Onde freqüências similares apareçam, a probabilidade de que se trate de uma cópia também surge. Palavras interessantes abordadas são as *hapax legomena* (aquelas que só aparecem uma vez no texto). Os autores divulgam que semelhança de até 40% é normal, e que algo acima de 70% precisa ser analisado mais detalhadamente. Susan Finlay em [27] demonstra que textos escritos independentemente podem ter semelhança de 50% ou mais onde vocabulários similares sejam utilizados, mas é utilizado em proporções diferentes pelos autores. Nestes casos, os *hapax legomena* podem ser utilizados para discriminar entre casos de plágio e semelhantes por casualidade.

Conforme descrito em [28], "CopyCatch se utiliza da estrutura inerente de um texto coerente da seguinte forma: uma dissertação de 1000 palavras, por exemplo, pode ser dividida em 550 palavras de função e 450 palavras léxicas. As 450 palavras léxicas serão baseadas em um vocabulário entre 200 e 300 palavras. Estimando uma probabilidade baixa de que 70% das 200 palavras sejam utilizadas somente uma vez. Para transformar um documento para outro utilizando sinônimos seria necessário então, alterar 140 palavras para tal. Comparando textos produzidos independentemente o número de palavras hapax é freqüentemente menor que 10, então, há muito trabalho para fazer e ainda ter sentido."

Os resultados do programa demonstram que o número de *tokens* em cada texto é equivalente ao grau de semelhança. A hipótese que o CopyCatch utiliza é que se a taxa de palavras usadas com menor freqüência diferencia textos portanto as próprias palavras devem ser diferentes mesmo quando respondendo à mesma pergunta. Estudantes que respondem à mesma pergunta, demonstram esse fato onde a similaridade de hapax legomena é muito baixa.

O CopyCatch também utiliza a ferramenta de ligação *abridgment*, baseada na coesão léxica para reduzir a quantidade de texto comparada através da remoção de frases irrelevantes e preserva o significado do texto.

2.4.4 Glatt

A Glatt Plagiarism Services Inc. provê três ferramentas básicas para ajudar a detectar e deter plágio: *Teaching Program*, *Screening Program* e *Self-Detection Program*.

O *teaching program* é um tutorial desenhado para ilustrar em que consiste um plágio, com exemplos de plágios diretos e indiretos, e demonstra como uma referência deve ser feita corretamente para evitar-se o plágio de outros trabalhos. O tutorial é voltado a acadêmicos e também apresenta formas de parafrasear sem plagiar.

O *screening program* é o programa utilizado pelos professores para detectar plágio nos trabalhos dos alunos. O programa funciona na premissa que todos possuem seu estilo de escrita e padrão lingüístico. Ele utiliza uma técnica desenvolvida pelo jornalista Wilson Taylor (em [29]) chamada de procedimento *cloze*. Esta técnica focaliza-se em questões de redundância na língua Inglesa e funciona através da remoção de toda palavra em uma passagem escrita. O Glatt, em particular, remove cada cinco palavras de um trabalho suspeito e o aluno é interrogado para fornecer as palavras ausentes. Como presume-se que a pessoa conheça seu próprio estilo de escrita instintivamente, o tempo estimado para preencher as ausências deve ser relativamente rápido. O programa também utiliza uma combinação do número certo de palavras retiradas, o tempo utilizado para preencher as ausências e uma série de outros fatores para calcular a Pontuação de Probabilidade de Plágio. Glatt alega que em dez anos, não houve nenhuma acusação falsa devido à alta quantidade de testes realizados.

A última ferramenta portanto, é o programa de auto-detecção, utilizada pelo próprios escritores para obter uma visão do seu próprio estilo de escrita para evitar o plágio. O programa está disponível *online* ou para compra. Ele é simples e solicita ao usuário que digite um pedaço de texto. Cada cinco palavras são removidas e o usuário preenche as ausências. A porcentagem de acerto das palavras digitadas é divulgada no final.

2.4.5 YAP3

A série YAP concentra-se principalmente na detecção de plágio em código fonte, como visto anteriormente, contudo, a última versão, YAP3 ([30]), foi adaptada para ser utilizada com a língua Inglesa, no entanto poucos testes têm sido feitos. O maior problema é a quantidade limitada de textos disponíveis para análise de plágio.

O YAP3 é a terceira extensão do YAP que utiliza um algoritmo novo chamado de *Karp-Rapin Greedy-String-Tiling* (ou RKS-GST) ([31]) utilizado para comparar pares de *strings* a procura de transposição de *substrings*. O algoritmo é similar ao *Longest Common Subsequence* (LCS) (Maior Subseqüência em Comum) [32] e *Levenshtein Distance* (Distância de Levenshtein), ambas as soluções para comparar a similaridade entre duas *strings*. O problema com

ambos os métodos é que preservam a ordem, ou seja, se uma parte do texto é movida os algoritmos tratam isto como linhas novas, em vez do mesmo parágrafo em um lugar diferente.

A abordagem utilizada pelo YAP não faz uma análise (*parse*) de toda a linguagem, porém compara *tokens* de *strings* elaboradas de palavras chaves retiradas dos lexemas da linguagem. Isso é muito útil quando se trata de línguas faladas, visto que é quase impossível obter uma análise de toda a linguagem.

O sistema funciona da seguinte maneira:

- Um gerador de *tokens* é utilizado para analisar um conjunto de textos e então, determina o conjunto léxico (*lexicon*) utilizado para gerar as sentenças de *tokens*.

- Um analisador elimina todos os números, palavras formadas por uma ou duas letras, substantivos e todas as palavras "comuns", as quais, são definidas em uma lista específica. Esta lista é estendida utilizando variações simples como, por exemplo se a palavra *keep* está na lista, as palavras *keeps* e *keeping* também estarão.

O restante das palavras são variadas utilizando o reconhecedor PC-Kimmo e as regras e conjuntos léxico do Englex10 [2].

O gerador de *tokens* e *lexicons* resultantes foram aplicados para uma série de documentos utilizando o YAP3, todavia não foi encontrados nenhum caso de plágio.

2.4.6 COPS

O *COpy Protection System* (COPS) (Sistema de proteção de cópias) é um protótipo experimental funcional de um sistema de detecção de cópia que pode ser utilizado em uma biblioteca digital ([33]). O trabalho é uma pequena parte do projeto de biblioteca digital da universidade de Stanford, que tem como objetivo disponibilizar um repositório sofisticado de informações. A função do COPS no projeto é detectar cópias exatas ou parciais na biblioteca devido à facilidade com que informações podem ser copiadas e plagiadas. Ele é referenciado tecnicamente como um serviço de detecção de cópias e trabalho somente com detecção de cópias entre textos em formato Tex, DVI ou troff. Os documentos são convertidos para ASCII antes de serem registrados e de se detectar documentos similares (ver Figura 2.9). A sistema procura por documentos com sobreposições significantes assim como cópias exatas.

[2] Ambos estão disponíveis pelo *Consortium for Lexical Research*, http://clr.nmsu.edu

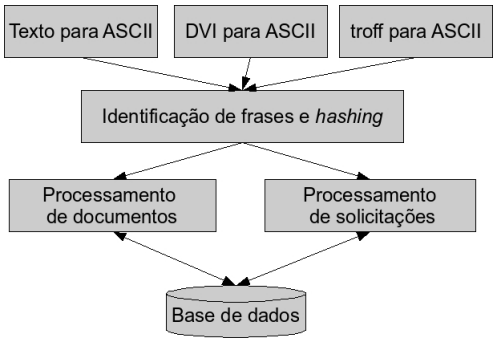

Figura 2.9: Principais módulos do COPS [3]

Os documentos são quebrados em sentenças, denominadas unidades, as quais são agrupadas em seqüências de sentenças chamadas *chunks* (nacos). As sentenças são armazenadas em um servidor de registros que é simplesmente uma grande tabela *hash*. Os nacos de documentos são comparados com aqueles de outros documentos no repositório, a fim de que sejam detectadas sobreposições. Caso os documentos possuam um número pré-definido de sentenças similares, é caracterizada uma violação. Posteriormente, uma pessoa verifica a violação para determinar o problema.

O sistema tem muito em comum com o *Siff*, um programa projetado por Udi Manber para encontrar arquivos similares em um sistema de arquivos (visto em [34]). Esta técnica envolve selecionar algumas palavras como base e calcular *checksums* de certa janela de caracteres para comparação.

As possíveis violações que podem ocorrer entre documentos incluem plágio de uma pequena quantidade de sentenças, cópia exata do documento, e estágios intermediários.

Os autores do COPS admitem que plágios são difíceis de serem detectados e que a intervenção humana é necessária. O COPS implementa o *Ordinary Operation Tasks* (OOTs - Tarefas comuns de operação) para fazer testes de plágio, subconjunto e sobreposições que podem ser implementados eficientemente.

Testes preliminares do COPS utilizavam noventa e dois documentos técnicos em Latex, DVI e ASCII, consistindo em aproximadamente sete mil e trezentas palavras e quatrocentas e cinqüenta sentenças. Os documentos formaram nove conjuntos de tópicos e os resultados demonstraram que o COPS conseguiu separar os documentos de forma correta, apesar de discrepâncias terem sido encontradas nas sentenças em comum para todos os documentos. Os autores oferecem melhorias para diminuir a quantidade de problemas.

2.4.7 SCAM

O *Stanford Copy Analysis Mechanism* (SCAM) (Mecanismo de análise de cópia de Stanford) foi elaborado a partir das experiências obtidas no desenvolvimento do COPS por Narayanan Shivakumar e Hector Garcia-Molina em [35]. O sistema é projetado para detectar plágios, cópias, extratos e grande similaridade em documentos de bibliotecas digitais. SCAM ficou conhecido em 1995 quando o sistema assinalou corretamente treze casos de plágios confirmado pelos autores originais por parte de um estudante, após suspeitas na submissão de documentos para a conferência EURO PAR de 1995.

O SCAM apresenta a estrutura ilustrada na Figura 2.10, um servidor de detecção de cópia genérico com um repositório de documentos registrados. Os documentos a serem registrados são divididos em nacos e inseridos no repositório. Novos documentos são divididos com o mesmo tamanho e comparados com os documentos existentes.

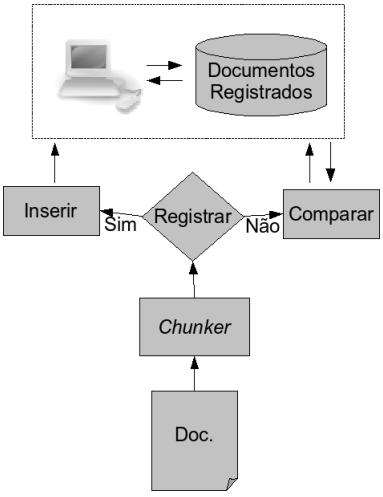

Figura 2.10: Servidor de detecção de cópias genérico [3]

A maior diferença entre o SCAM e COPS, é que o SCAM utiliza um esquema baseado em palavras em vez de sentenças. O problema com a comparação de sentenças, é que sobreposições de partes de sentenças não são detectadas. Outros problemas do COPS são: a detecção de sentenças com figuras, equações e abreviações confundindo o sistema e a performance, já que a detecção de sobreposições envolve muitas varreduras aleatórias na tabela *hash*.

Os documentos são divididos em palavras (unidades), as quais são agrupadas para formar *chunks*. Estes nacos são inseridos no repositório em uma estrutura de índices invertidos e utilizados para comparar com novos documentos, conforme Figura 2.11. O SCAM utiliza

palavras como nacos para comparação desta forma permitindo ao sistema a detecção de sobreposição parcial de sentenças. SCAM utiliza uma derivação do *Vector-Space Model* para medir similaridade em documentos. Esta é uma medida popular de aquisição de informação (*information retrieval* - IR) e funciona através do armazenamento da freqüência de palavras no documentos em um vetor. Estes vetores são posteriormente comparados procurando-se sua similaridade utilizando uma medida como a *dot product* ou co-seno. No caso do *dot product*, dado um vetor de freqüências normalizadas $V(R) =< 1/3, 1/3, 1/3, 0, ... >$ e outro vetor $V(S1) =< 1/2, 1/2, 0, 0, ... >$, a similaridade ente R e S1 seria $sim(R, S1) = 1/3 * 1/2 + 1/3 * 1/2 = 1/3$. Para a similaridade de co-seno, é calculada por $\frac{\sum a}{}$.

O valor resultante de similaridade é marcado, caso ultrapasse um valor pré-definido. Problemas com ambos os métodos apontados em [36] deram lugar ao desenvolvimento de uma nova medida de similaridade chamada de *Relative Frequency Model* (RFM) (Modelo de freqüência relativa).

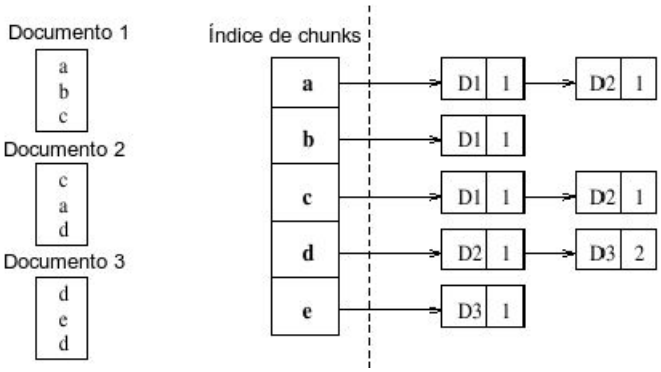

Figura 2.11: Mecanismo de armazenamento de índices inverso [3]

O RFM utiliza as freqüências relativas das palavras como indicadores de palavras similares e combina o processo com a medida de similaridade do coseno. Inicialmente, um conjunto de palavras que tenham um número de ocorrências similares em cada documento é definida. Este conjunto é a medida para um subconjunto de teste que determinar se D1 é um subconjunto de D2. O teste é assimétrico, portanto, o conjunto de similaridades é o maior valor do conjunto D1 comparado com D2 e um valor do subconjunto de D2 comparado com D1.

Testes foram realizados com mil duzentos e trinta e três artigos do Netnews e comparado consigo mesmos para detectar sobreposições. Os resultados foram comparados com o COPS. Os valores de sobreposição foram então comparados com textos examinados manualmente para determinar casos de plágio, subconjunto, cópia ou relacionado. Os textos poderiam ficar em

mais de uma categoria. Os valores de sobreposição eram posteriormente comparados com os resultados do julgamento humano e os resultados demonstraram que o SCAM funciona melhor que o COPS na detecção de sobreposições apesar do número de falso positivos ser maior.

2.4.8 dSCAM

O dScam ([37]) é uma versão distribuída do SCAM, onde cada cliente possui um conjunto de informações do conteúdo presente em cada base de dados. Dado um documento suspeito, o dScam utiliza esta informação para filtrar todas as bases que não contenham qualquer documento que não seja relevante, conseqüentemente, a busca pode focalizar-se nos lugares restantes.

Os mecanismos utilizados pelo dScam são dois: Descoberta e Extração.

A descoberta trata da identificação das bases de dados que sejam relevantes, sendo que qualquer base que contenha até mesmo um único documento relacionado deve ser selecionada. Este mecanismo está baseado na abordagem GIOSS, na qual se faz a coleta de informações das bases de dados possíveis de serem analisadas. Isto pode incluir, por exemplo, informação de quão freqüente é a aparição de termos em documentos em uma base. Esta informação é muito menor que um índice completo da base e que o SCAM precisa conhecer para fazer a detecção. Portanto, baseado nestas informações, dSCAM pode ignorar bases que não contenham documentos que sejam considerados cópias pelo SCAM.

São consideradas duas técnicas de descoberta: conservadora e liberal. Na conservadora, somente são ignoradas bases de dados caso seja certo de que o SCAM não irá considerar nenhum documento presente como cópia. Neste caso, não são produzidos nenhum falso negativos (documentos que não são marcados como suspeitos, mas que na verdade são cópias), mas pode produzir falsos positivos (documentos que são marcados como suspeitos, mas que na verdade não são cópias). Os dados presentes nas informações são analisados de forma pesquisar todos as bases que tenham documentos suspeitos.

A técnica liberal pode deixar de encontrar bases com documentos plagiados. No entanto, isto não é comum. A escolha no final depende da quantidade de recursos disponíveis e de quão exaustiva a busca tem de ser. Neste caso, a busca pode concentrar-se em palavras mais raras, em vez de todas as palavras, ou em vez de utilizar modelos determinísticos, utilizar modelos probabilísticos, onde se calcula a probabilidade da base de dados terem um documento suspeito.

O outro mecanismo do dSCAM, a extração, cuida da estratégia de gerar automaticamente uma busca que retorne cópias em potencial para posterior análise.

Depois do dSCAM ter escolhido o conjunto de bases a serem analisados, deve ser extraído o conjunto de cópias em potencial dessa base. Para que esse procedimento seja efetuado, é escolhido um pequeno conjunto de palavras que não deixarão de marcar nenhuma cópia. Este conjunto é determinado pela quantidade que a palavra pode incrementar o grau de similaridade.

Os resultados demonstram que o dSCAM tem melhor performance quando ele considera somente 10% de todas as palavras dos documentos, aquelas que são as mais raras. Estas funcionam como palavras características de um documento, facilitando a escolha da base de dados.

2.4.9 CHECK

Como na maioria dos modelos apresentados, o CHECK [4] mantém uma base de dados com documentos registrados que são utilizados na comparação com novos documentos. No entanto, CHECK tem duas diferenças. Primeiramente, em todos os sistemas anteriores, com exceção do dScam, não há preocupação em filtrar as buscas, conseqüentemente muitas comparações exaustivas e desnecessárias são realizadas, consumindo tempo e recursos. A segurança dos sistemas anteriores é fraca no que se refere a pequenas modificações, pois podem ser feitas em cada sentença a fim de burlar o mecanismo de detecção.

CHECK difere-se ainda na aplicação das técnicas de aquisição de informação. Elas são aplicadas primeiro para filtrar os candidatos de plágios. Este processo é aplicado recursivamente em diferentes níveis de granularidade desde seções, subseções, parágrafos até sentenças. A comparação entre documentos é baseada em palavras-chaves que teoricamente identificam o significado semântico do documento.

Seus autores reconhecem as diferenças entre comparar plágio entre linguagens naturais e de computação. Segundos eles, programas de computadores são bem estruturados e preservam a árvore analisada do programa original mesmo se modificado. No entanto, plágio em texto escrito é muito mais difícil, já que um documento pode preservar a semântica do original, mas mesmo assim ter muito mais diferenças que um programa de computador como, por exemplo, ter sua árvore PARSE alterada.

O CHECK utiliza-se de pesos em palavras chaves na árvore analisada do documento para ter melhor representação com maior resistência à modificação simples no documento. Até o momento o CHECK somente reconhece documentos do tipo LATEX.

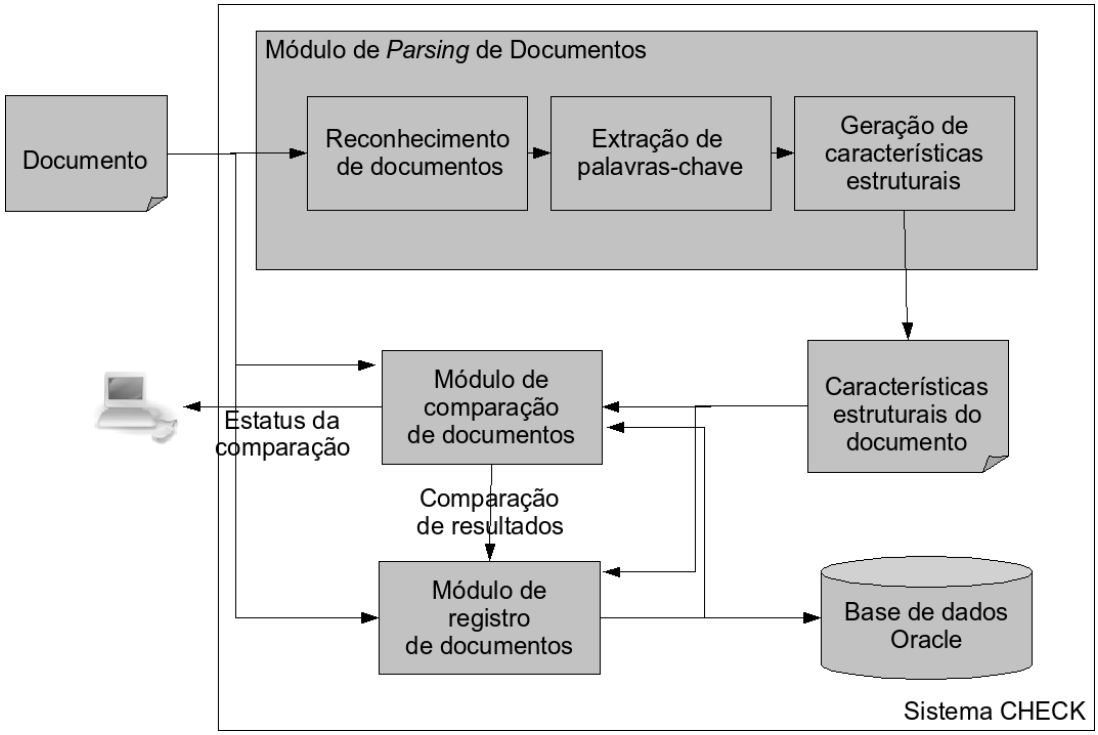

Figura 2.12: Arquitetura do CHECK [4]

Conforme demonstrado na Figura 2.12, o CHECK funciona da seguinte maneira:

- Reconhecimento de documentos: um reconhecedor de LATEX analisa o documento e cria uma árvore do documento, conforme exemplo da Figura 2.13. Esta estrutura é semelhante a uma árvore que se assemelha a estrutura do documento em vários níveis de abstração, como seção, subseção e parágrafo. Para cada uma das palavras, uma forma de lematização (*lemmatisation*)[3] é aplicada onde se converte plural em singular, verbos para sua forma infinitiva e todos os sufixos assim como adjetivos e advérbios para suas formas fundamentais.

- Extração de palavras chaves: As técnicas de aquisição de informação são utilizadas para extrair palavras que melhor descrevem a semântica do documento. Estas são categorizadas em palavras de classe aberta (substantivos, verbos, adjetivos e advérbios) e classe

[3]utilizar a forma de uma palavra conforme consta em um dicionário, glossário, índice.

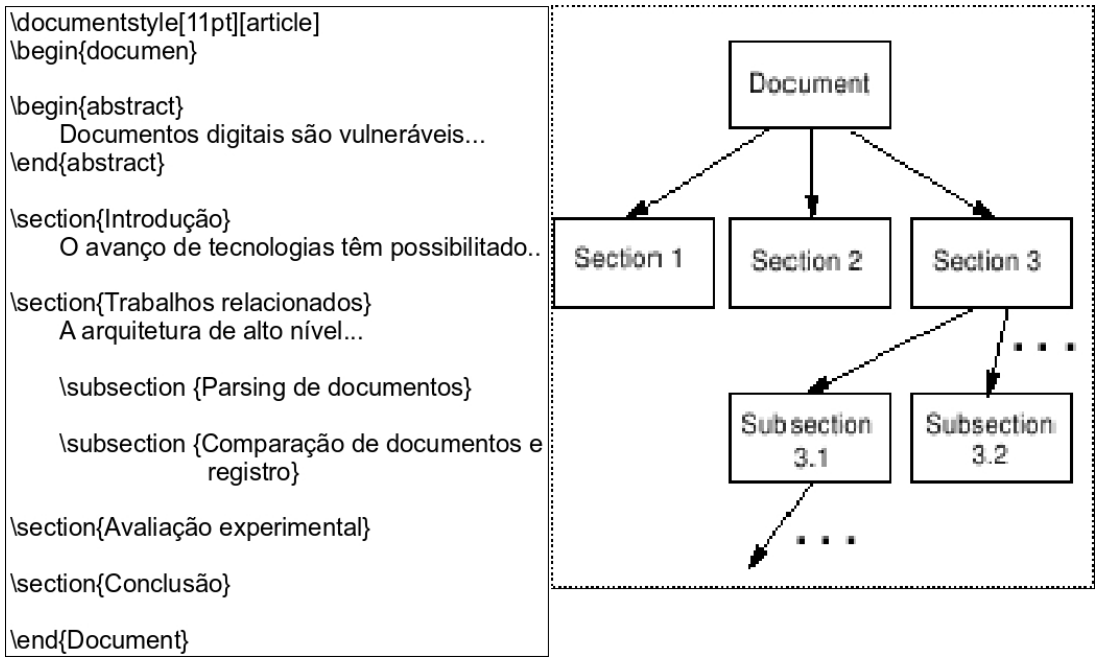

Figura 2.13: Exemplo de documento e sua árvore correspondente [4]

fechada (preposições, pronomes, conjunções e interjeições). Algumas heurísticas como comandos de formação no LATEX são utilizadas para ajudar extração de palavras-chaves.

- Gerar características de estrutura: para cada documento, uma Característica Estrutural é gerada. Está é a estrutura do documento integrada com o conjunto de palavras-chaves extraídas. As palavras-chaves têm pesos atribuídos nos níveis de seção, subseção e parágrafos.

O resultado deste processo é uma árvore para cada documento com os pesos das palavras-chaves atribuídos para cada nó da árvore. Estas representações dos documentos são adicionadas em um servidor de registros que pode ser usado para compara novos documentos adicionados ao sistema. A similaridade entre os documentos novos e existentes é medida utilizando o dot product entre os vetores normalizados representando as palavras-chaves de um documento. Esta comparação é repetida para cada nível de abstração do documento, porém, somente se a comparação prévia gera um grau de similaridade maior que um nível predeterminado, evitando-se assim comparações desnecessárias de níveis mais baixos de abstração entre documentos não relacionados.

Testes foram feitos com quinze documentos técnicos e foram avaliadas os seguintes cenários:

- Identificação de documentos idênticos (documentos comparados com eles mesmos): ele se mostrou capaz de detectar todas os parágrafos copiados, no entanto em 10% dos casos eram falsos positivos.

- Comportamento com documentos com assuntos similares, mas sem cópia: CHECK foi capaz de detectar que os documentos eram originais. Todos os processos de comparação pararam no nível 1, demonstrando que ele é capaz de determinar que dois documentos estão descrevendo assuntos similares, porém, são suficientemente diferentes como para cessar a análise rapidamente.

- Identificação de cópias reais: a precisão se mostrou ser de 100%, todos os parágrafos detectados eram de fato plágio. No entanto, alguns parágrafos plagiados não foram detectados, taxa que variou de 50% a 90%. Isto é explicado pelo limite predeterminado estabelecido: quanto maior o limite, maior a similaridade para que seja passada para o próximo nível.

- Comportamento com documentos plagiados, todavia, com assuntos diferentes: neste caso, ele não detectou nenhum documento plagiado, já que comparações em documentos com assuntos diferentes não são feitas, ou não são levadas a fundo.

2.4.10 DetectIt

Um problema apontado pelos autores do DetecIt, Elif Tosun e Ben Wellington [5], em relação ao uso de ferramentas de detecção de plágio na *Internet*, é que a maioria é paga. Segundo, se diferentes professores em uma universidade utilizam serviços distintos, eles não detectarão cópias entre eles. E por último, armazenar os trabalhos dos alunos em uma base de dados *online* pode ser considerada como uma violação da propriedade intelectual, já que os documentos são repassados para uma terceira pessoa.

Para os autores, a solução ideal é manter um registro de todos os documentos escritos em todo o mundo para que no momento em que um novo documento é submetido, este seja comparado com todos os documentos nesta base. Logo, quanto mais usuários, maior a base. Esta base, devido ao tamanho, passaria a ser distribuída. O DetectIt foi desenvolvido seguindo estas características. Ele é um sistema ponto a ponto, desenvolvido em Java, tendo como base o Tapestry (visto em 3.5.2).

O problema com o Tapestry é que não se pode fazer buscas baseado em características gerais de um objeto. Cada objeto recebe um GUID e a busca é feita sob estes GUIDs. Para se conseguir um melhor método de busca, é utilizado o ATA (Approximate Text Addressing - Endereçamento de Texto Aproximado), que permite buscas em propriedades de documentos similares.

A forma de comparação utilizada é a combinação de *fingerprints* do documento. Uma *fingerprint* é um conjunto de caracteres de tamanho fixo n. Se dois documentos possuem os mesmos n caracteres em seqüência, então eles terão a mesma impressão digital representando estes n caracteres. O DetecIt toma todas as impressões digitais de n *bits* de um arquivo, e armazena uma pequena parte destes, dependendo de seus valores. Foi escolhido pelos autores armazenar as impressões digitais que são divisíveis por duzentos e cinqüenta e seis. Isto significa menos que 5% de um documento com 10.000 caracteres. Com isto, a performance aumenta. O algoritmo utiliza f impressões, onde f é o número de impressões digitais escolhidas em um documento. Se qualquer destas impressões aparecer em outro documento, também será divisível por duzentos e cinqüenta e seis, e é provável que sejam parte de f impressões digitais desse documento.

Há diversos parâmetros envolvendo impressões digitais em documentos. A granularidade é o tamanha do caracteres de uma impressão digital (n). Com baixa granularidade, encontram-se muitos falsos positivos. A Figura 2.14 demonstra isso. O número de impressões digitais utilizadas é outro parâmetro, definido como f. Quanto maior é f, pior a performance, já que cada impressão digital equivale a uma busca no Tapestry. E por último o limite (*threshold*), definido como T, que é o percentual de impressões digitais que devem combinar para que os dois documentos sejam considerados similares. Quanto menor o valor de T, maior a chance de que as similaridades sejam apenas coincidências, conforme demonstra Figura 2.15.

Os testes feitos com o DetectIt demonstraram que o sistema é adequado para as necessidades do usuário. Os valores utilizados nos testes foram $n = 100$, $f = 100$ e $T = 10\%$. A performance relata apenas os tempos de geração de impressão digital, e não de detecção de plágio.

2.4.11 Detecção automática de plágio utilizando a *Internet*

A idéia apresentada nesta proposta é a de utilizar ferramentas de buscas na *Internet* como o Google para procurar textos de trabalhos de alunos. A questão é que partes do documento

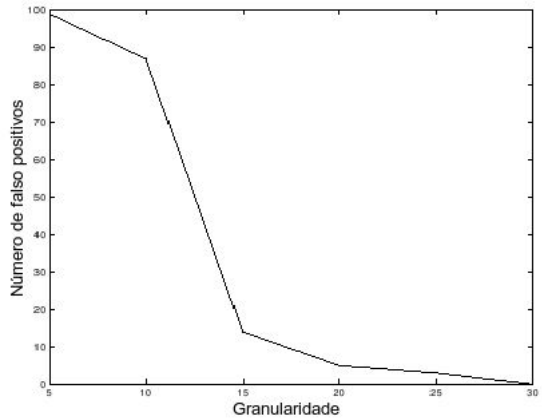

Figura 2.14: Efeito da granularidade nos números de falso positivos [5]

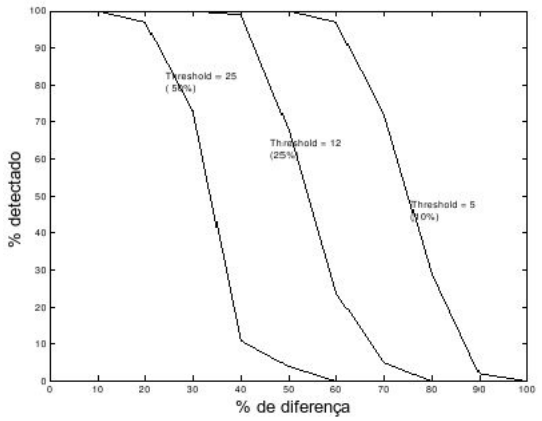

Figura 2.15: Efeito do limite (*threshold*) na detecção de documentos similares [5]

devem ser escolhidas para serem procuradas no Google. As opções variam de utilizar uma janela de n palavras no documento até escolher aleatoriamente partes de sentenças no texto. Três técnicas foram desenvolvidas e avaliadas.

- Busca exaustiva

  Esta seja talvez a melhor e mais eficiente forma de detectar trabalhos copiados. Esta técnica separa cada trabalho em cada uma de suas sentenças e submete as oito primeiras palavras de cada sentença que tenha pelo menos oito palavras. Isto limita o número de buscas para cerca de cem por documento. O número oito foi escolhido, pois provocava o menor número de falso positivos.

  Após ter coletado todas as sentenças de cada documento, elas são enviadas como buscas no Google e os resultados são salvos e, posteriormente investigados. Caso alguma sen-

tença indica alguma *Universal Resource Locator* (URL), o documento que contém esta sentença é marcado como suspeito. Uma investigação humana a partir daí determina se realmente trata-se de um caso de plágio.

- Limitando a busca

  Utilizou-se um conjunto de aproximadamente quatrocentos e oitenta documentos para avaliar. Cada um tinha aproximadamente dez páginas. Com a busca realmente exaustiva, sem a distinção das oito primeiras palavras mencionadas anteriormente, houve cerca de um milhão e quatrocentas mil buscas. Com a distinção, houve cerca de cinqüenta e cinco mil. Os autores declaram que um próximo passo seria encontrar técnicas mais inteligentes para reduzir mais ainda o número de sentenças procuradas.

- Utilizando recursos lingüísticos

Há vários recursos lingüísticos superficiais mencionados. Estes envolvem geralmente cálculos baseados no número de palavras por sentenças e a distribuição de sílabas em cada palavra. Estes recursos não possuem qualquer influência no contexto da sentença, contudo, determinam efetivamente o grau de leitura de um trabalho.

O intuito é determinar o grau de leitura de uma sentença e a somente submeter aqueles que possuem um valor abaixo de um certo nível. O menor valor é utilizado pois isto significa que a sentença com pouco valor não é uma sentença comum, logo, a probabilidade de encontrar duas sentenças incomuns é baixa.

Neste caso, foi escolhido a Nota de Nível de Flesch-Kincaid (*Flesch-Kincaid Grade Level*). A Tabela 2.1 apresenta a sua fórmula.

Tabela 2.1: Nota de Nível de Flesch-Kincaid

(.39 x ASL) + (11.8 x ASW) - 15.59
onde:
ASL = tamanho médio de sentença ASW = número médio de sílabas por palavra

Cada sentença é processada utilizando o comando UNIX *style*. Cada pontuação é armazenada e as sentenças são ordenadas de acordo com sua nota. Todas as sentenças com nota menor que dez são escolhidas para serem procuradas no Google. Como anteriormente, os resultados são armazenados para uma vistoria humana.

A avaliação da ferramenta foi feita com quatrocentos e oitenta documentos. Os resultados estão apresentados na Tabela 2.2.

Tabela 2.2: Resultados coletados

Tipo da busca	Número de buscas	Número de possíveis acertos
Busca exaustiva	55.522	4
Busca limitada	18.152	2

Percebe-se que com uma busca limitada obteve-se um resultado um pouco pior, todavia, o número de buscas foi bastante reduzido.

2.5 Conclusão

Foram apresentados os principais conceitos relacionados à detecção de plágio atualmente. Estes são divididos em detecção de plágio em código fonte e documentos escritos. O primeiro é muito mais explorado que o segundo, devido a certa facilidade em relação ao outro. Inúmeras ferramentas utilizam os conceitos apresentados neste capítulo de diversas formas. Estas ferramentas estão apresentadas nos trabalhos relacionados.

Apesar de estas técnicas apresentadas serem eficientes, é sempre necessária a intervenção humana para ser ter certeza de que o caso marcado como suspeito é de fato um plágio e não um mal entendido. Todavia, quanto menos requer-se a intervenção humana, mais eficiente o algoritmo é considerado.

Foram apresentadas também as principais ferramentas de detecção de plágio existentes, as quais foram abordadas somente as de detecção de plágio em documentos escritos em linguagem natural. Pode-se verificar que este campo não é de nenhum modo inexplorado, porém, o número destes aplicações que utiliza uma abordagem P2P é baixo. A única que explicitamente pode ser considerada uma aplicação P2P é o DetectIt.

Quanto às ferramentas existentes, teve-se certo interesse em relação ao Check, já que demonstrou-se eficiente e compartilha algumas idéias similares à nossa proposta.

No próximo capítulo, será apresentado qual é proposta, baseado em todo o conteúdo visto até aqui.

Capítulo 3

Arquiteturas P2P

3.1 Introdução

Neste capítulo será apresentado o sistema P2P e suas arquiteturas. Um sistema P2P é um sistema distribuído cujos nós participantes têm papéis simétricos, o que significa que todo nó possui as mesmas capacidades e responsabilidades. Desta maneira, cada nó pode atuar simultaneamente como cliente e servidor para os outros nós na rede. Contrastando esta arquitetura, sistemas convencionais como cliente-servidor têm uma clara distinção entre os clientes que fazem requisições e consumem recursos, e servidores que oferecem serviços. As principais vantagens de uma arquitetura P2P são escalabilidade, tolerância a falhas e a ausência de gargalo nos recursos dos servidores [6].

3.2 Localização em redes P2P

Um dos serviços mais importantes das redes P2P é a descoberta de recursos. Este serviço permite aos usuários encontrar nas redes P2P o recurso que necessitam, tais como serviços, arquivos, entre outros. De forma geral, o resultado consiste em uma lista de nós onde o recurso pode ser encontrado. Depois que o recurso é encontrado, o nó que iniciou a busca pode acessar diretamente o dono do recurso. Basicamente, há duas abordagens para a localização de conteúdo em sistemas P2P: descentralizado e híbrida. Na abordagem descentralizada, cada nó na rede é tratado igualmente e não há controle sobre eles. Na abordagem híbrida, existe ao menos um nó de controle que possui um papel autoritário na rede. A Figura 3.1 apresenta sete possíveis tipos de arquiteturas. Suas principais características serão explicadas a seguir.

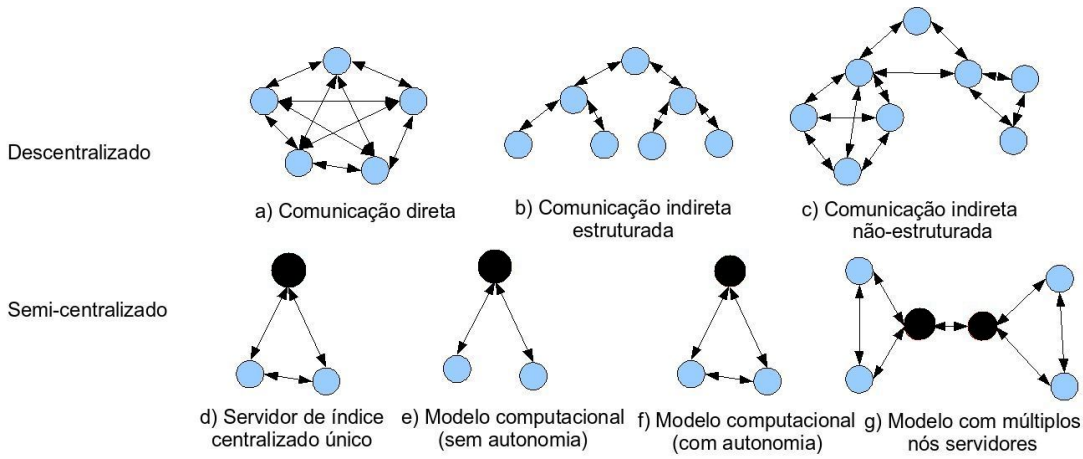

Figura 3.1: Modelos de arquiteturas P2P (servidores centrais em preto) [6]

3.2.1 Abordagem descentralizada

Baseado na Figura 3.1, três tipos de modelos descentralizados são definidos:

Comunicação direta

Na comunicação direta, todos os nós são autônomos. Isso pode significar que eles são independentes e/ou inteligentes, por exemplo, mas o principal é que todos tenham um mesmo papel para a rede, ou seja, que não haja um nó central.

Graças a essa autonomia dos nós, dados e cálculos podem ser distribuídos por toda a rede, sem exceção. Esse "repasse"de informações é feito de nó em nó, já que todo nó pode se comunicar diretamente com qualquer outro nó na rede. As informações não necessitam passar por algum nó em especial, pois, como dito anteriormente, não há nó central. Este modelo está ilustrado na Figura 3.1a.

Comunicação indireta estruturada (hierárquica)

Este tipo de comunicação é muito semelhante à comunicação direta. A maior diferença é que os nós são organizados em uma maneira estruturada, onde cada nó ainda pode se comunicar com todos os nós na rede, mas alguns de forma direta e o restante de forma indireta. As outras características permanecem - todos os nós são autônomos e têm um mesmo papel para a rede.

As DHTs fazem parte deste modelo. As DHTs são um tipo de rede P2P com comunicação indireta estruturada recente. A Seção 3.5 é dedicado especialmente a este modelo. Este modelo está ilustrado na Figura 3.1b.

Comunicação indireta não-estruturada (grafo)

Este modelo também é conhecido como modelo de inundação, visto que os nós são organizados em uma maneira não-estruturada, como, por exemplo, uma rede de grafos. Cada nó pode se comunicar de forma direta somente com alguns nós, mas de forma indireta pode se comunicar com todos os outros. Cada requisição de um nó é enviada para todos os nós diretamente conectados, os quais enviam para os nós diretamente conectados a eles, e assim sucessivamente até que a requisição seja respondida ou que ocorra o número máximo de encaminhamentos (tipicamente 5 a 9) (modelo representado na Figura 3.2).

Como nas comunicações anteriores, todos os nós são autônomos e cumprem um mesmo papel para a rede, ou seja, não há nó central. Mas neste caso, um nó pode não se comunicar com todos os outros nós. Devido à natureza não-estruturada um nó pode não estar ciente de quais outros nós existem na rede.

Este modelo é utilizado pelo Gnutella [38] e requer alta capacidade dos enlaces de comunicação para proporcionar desempenho razoável, apresentando problemas de escalabilidade quando o objetivo é alcançar todos os nós de uma rede. Contudo, o modelo é eficiente em comunidades e redes limitadas [39]. Este modelo está ilustrado na Figura 3.1c.

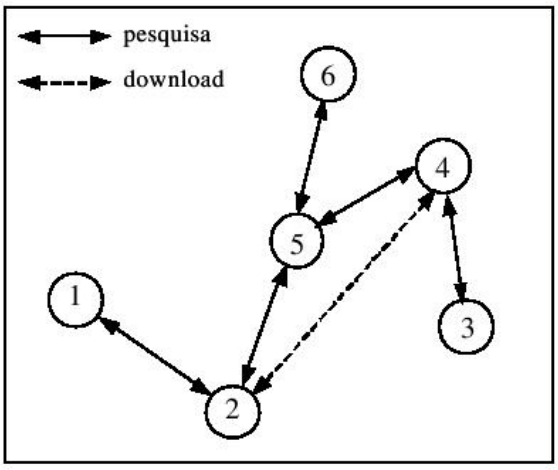

Figura 3.2: Modelo de Inundação [6]

3.2.2 Abordagem híbrida

Baseado na Figura 3.1, quatro tipos de modelos híbridos são definidos:

Servidor de índice centralizado único

Neste modelo, um único servidor mantém um catálogo dos recursos que podem ser acessados por todos os nós clientes. Ele é utilizado comumente como um ponto de referência para os dados existentes na rede. O restante dos nós (os nós "clientes") se comportam de forma autônoma onde todos têm um mesmo papel para a rede como anteriormente.

Quando um nó ingressa no sistema, ele contata o nó central e envia a ele uma lista de todos os seus recursos disponíveis para os outros nós. Para localizar o recurso, um nó cliente envia uma solicitação ao servidor central, que, por sua vez, executa uma busca na sua base de dados e responde com a lista de nós que possuem o recurso desejado. Após fazer esta solicitação ao servidor central, ele inicia uma comunicação direta com os outros nós a fim de obter o recurso desejado (Figura 3.3).

Tal aspecto pode gerar limites de escalabilidade ao modelo, uma vez que requer servidores maiores à medida que o número de requisições aumenta e mais espaço para armazenamento conforme a quantidade de usuários cresce. O Napster [40] utiliza este modelo, que está ilustrado na Figura 3.1d.

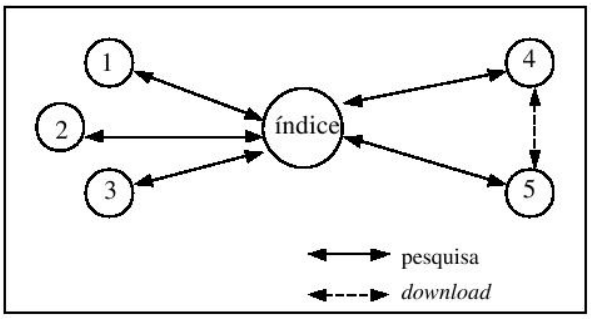

Figura 3.3: Modelo de índices centralizado [6]

Modelo computacional (sem autonomia)

Neste modelo, uma dependência maior surge em torno do servidor central. Os nós clientes deixam de ser autônomos e possuem um mesmo papel para a rede. Todavia, todos eles

se comunicam somente pelo nó central. Este servidor central controla e gerencia a distribuição de dados/processos para os nós clientes.

Pode-se visualizar que um único ponto de falha pode trazer conseqüências para toda a rede. Este modelo está ilustrado na Figura 3.1e.

Modelo computacional (com autonomia)

A diferença deste modelo com o anterior é a autonomia dos nós clientes. Eles são autônomos incluindo a capacidade de cada nó cliente de operar inteligentemente, solicitar/enviar dados ao servidor central ou a outros nós clientes. A comunicação também passa a depender menos do servidor central. A referência inicial ainda pode depender dele, porém a partir dessa referência a comunicação é direta. Este modelo está ilustrado na Figura 3.1f.

Modelo com múltiplos nós servidores

Este modelo é uma extensão dos modelos D e E (servidor de índices centralizado único e modelo computacional com autonomia) da Figura 3.1, mas envolvendo o uso de mais de um servidor central. Estes nós servidores podem se comunicar entre si. Este modelo está ilustrado na Figura 3.1g.

3.3 Modelos de Aplicação

Uma rede P2P pode ser utilizada para diversas finalidades:

- Comunicação instantânea, como o Microsoft Netmeeting [41] e o Skype [42].
- Compartilhamento de arquivos, como o Napster e Gnutella.
- Trabalho colaborativo, ou groupwares, projetados para melhorar a produtividade de indivíduos com objetivos e interesses comuns. O Lotus Notes [43] e o Groove [44] são exemplos destas aplicações.
- Aplicações computacionais, onde uma carga de trabalho é distribuída em vez de ser processada por um nó somente. Geralmente um nó servidor faz a distribuição e os clientes não precisam se comunicar entre si. Um exemplo é o BOINC [45].
- Jogos.

- *Multicasting*, como por exemplo, a distribuição de *streams* multimídia.
- Mecanismos de busca.

A Tabela 3.1 faz uma comparação entre aplicações existentes. Para a coluna “Arquitetur usada”, as letras estão relacionadas com as arquiteturas apresentas na Figura 3.1.

Tabela 3.1: Aplicações P2P

	Natureza do sistema	Arquitetura usada (Figura 3.1)	Mecanismo de endereçamento
Napster	Compartilhamento de arquivos	D	Nome de usuário: mapeamento de endereço por *ip*
Freenet	P2P Completo. Busca e compartilhamento de informações	C	Nó mantém lista de *ip*'s
ICQ	Mensageiro	D	Nome de usuário: mapeamento de endereço por *ip*
SETI@home	Computacional	E	Atualização de *ip* programada
Gnutella	P2P Completo. Busca de informações e protocolo de compartilhamento	C	Nó mantém lista de *ip*'s
AIM	Mensageiro	D	Nome de usuário: mapeamento de endereço por *ip*

3.4 Ferramentas utilizadas para desenvolvimento de aplicações P2P

Com o rápido crescimento de aplicações utilizando a infra-estrutura P2P, protocolos e padrões têm sido adaptados para trabalhar com protocolos e padrões e com plataformas de desenvolvimentos, vistos nas seções seguintes.

3.4.1 Protocolos e padrões

- *Extensible Markup Language* (XML) e padrões de metadados - XML e a padronização de metadados é um desenvolvimento crucial para que as redes P2P sejam adotadas por empresas. Uma camada consistente de abstração provê uma plataforma consistente para desenvolver aplicações. O *Resource Description Format* (RDT) (formato de descrição de recursos), uma recomendação do *World Wide Web Consortium* (W3C) para definir o conteúdo de páginas web em XML, oferece um framework generalizado para todos os tipos de metadados. Estes metadados da RDT oferecem uma rica semântica na qual aplicações P2P podem ser baseadas.

- Protocolo de segurança e confiança - Se nós entre empresas diferentes devem trabalhar entre si, é necessário que seja de forma segura. Pode ser utilizada ou não uma infra-estrutura de chave pública global *Public Key Infrastructure* (PKI). Uma falta de uma PKI global significa que a confiança deve ser estabelecida em comunidades menores na qual os membros devem colaborar freqüentemente. Além de identidade, há também a preocupação com confidencialidade e integridade de dados. SSL e IPSec foram implementados em diversas plataformas e oferecem uma boa infra-estrutura para isto.

- *Web services* - A utilização de tecnologias de *Web Services*, como o Microsoft .NET, SunONE, IBM Services Toolkit e o Sun Web Services Developer Pack, que permite a comunicação de aplicações utilizando protocolos baseados no XML como o *Simple Object Access Protocol* (SOAP), faz com que aplicações publiquem serviços umas às outras, acessíveis pela *Internet*. Padrões como o *Web Services Description Language* (WSDL) e o *Universal Description, Discovery and Integration* (UDDI) oferecem uma linguagem para descrever tais serviços e encontrá-los na *Internet*.

3.4.2 Plataformas de desenvolvimento

Atualmente, algumas plataformas de desenvolvimento podem ser utilizadas para ajudar na construção de aplicativos P2P, como o JXTA, .Net e o Groove.

- JXTA

  Inicialmente, quando alguns protocolos e aplicações P2P começaram a ser utilizados, eles não interagiam entre si. Com a intenção de resolver este problema, a Sun Microsystems

anunciou a plataforma JXTA [46], que foi concebida a fim de padronizar um conjunto comum de protocolos abertos permitindo que qualquer dispositivo na rede se comunique e colabore na maneira das redes P2P. Os protocolos JXTA são definidos como um conjunto de mensagens XML.

Os protocolos JXTA estabelecem uma rede superposta virtual por sobre a *Internet*, permitindo que nós interajam diretamente e se auto-organizem independentemente da topologia e conectividade de sua rede. Desta maneira, muitas redes virtuais AD-HOC podem ser criadas e mapeadas dinamicamente a apenas uma rede física. JXTA define um conjunto mínimo de requisitos para que os nós formem e ingressem a redes P2P virtuais. Sendo assim, é possível que desenvolvedores de aplicações definam a topologia de rede que mais convém aos requisitos das suas aplicações. JXTA foi desenhado para ser independente de linguagem de programação, de sistemas operacionais e de protocolos de rede.

Com o JXTA, desenvolvedores podem assumir que tudo o que é necessário para interação de forma P2P se encontra na plataforma, da mesma forma que a *Java Virtual Machine* oferece uma infra-estrutura e funcionalidade na qual se pode iniciar o desenvolvimento de uma aplicação. O objetivo é oferecer aos desenvolvedores P2P suficientes funcionalidades nas quais eles possam desenvolver suas aplicações P2P. Desta forma, os desenvolvedores podem ser concentrar em fazer aplicações sem se preocupar com as necessidades de baixo nível.

- .NET

  O .NET é uma plataforma que disponibiliza classes projetadas para ajudar no desenvolvimento de aplicações P2P. Com uma grande variedade de modelos de aplicação presentes na plataforma .NET, o desenvolvedor pode projetar qualquer tipo de aplicação P2P. Há quatro modelos mais importantes, os quais são apresentados a seguir.

  1. Os *Web Services* são utilizados principalmente para efetuar registro, descoberta e procura de conteúdo entre os nós. Isto é efetuado através da escrita de uma classe específica para escutar requisições e responder com informações úteis.
  2. As *Windows Forms* oferecem as ferramentas apropriadas para desenvolver rapidamente e facilmente aplicações com interface gráfica. Podem ser utilizadas para enriquecer a aplicação com uma interface amigável para o usuário.

3. Com *Web Forms* é possível implementar um sistema onde o usuário pode visualizar nas aplicações P2P informações inseridas pelos desenvolvedores. Esta informação varia desde dados sobre o sistema até propagandas de como utilizar o serviço. Estes dados podem ser enviados para a plataforma do usuário em forma de código HTML.
4. O *Service Process* (processo de serviço) pode ser utilizado para implementar um servidor de descoberta de longa duração, porém, com suporte a protocolos diferentes do HTML. Em casos onde o mecanismo de descoberta não esteja utilizando HTML, um processo de serviço escuta ao mesmo protocolo.

- Groove

  O Groove oferece o Groove Development Kit (GDK), uma plataforma gratuita, contudo, é necessária uma licença para os produtos criados com ele. Seus programas dependem da plataforma de desenvolvimento da Microsoft, pois o Groove foi desenvolvido utilizando o Microsoft Component Object Model (COM).

  Este programa pode ser utilizado para criar tanto aplicações centralizadas quanto descentralizadas, devido à sua abordagem híbrida. Ele também disponibiliza alguns serviços:

  – Armazenamento de mensagens off-line.
  – Serviços de interface gráfica.
  – Gerenciamento de dados, permitindo manipular facilmente dados sincronizados.
  – Compartilhamento de espaço, possibilitando acessar ferramentas (de um grupo) em tempo de execução.
  – Serviço de identificação, que permite recuperar (localizar) usuário que estão *online*.
  – Ferramentas de publicação, que permitem disponibilizar, criar, refinar e testar as ferramentas desenvolvidas.

  O Groove permite, portanto, uma prototipação rápida de aplicações P2P utilizando linguagens script como VBScript ou JavaScript.

3.5 *Distributed Hash Tables*

Distributed Hash Tables (Tabelas de *Hash* Distribuídas) são uma classe de sistemas distribuídos descentralizados que executam as funções de uma tabela *hash*. Esta tabela *hash*

armazena duplas da forma chave, valor e os valores são procurados através da chave. Na DHT, as chaves são distribuídas entre os vários nós. Desta maneira, tanto o armazenamento quanto a busca são distribuídos entre os nós participantes. DHT são desenhados para suportar constantes ingressos, saídas e falhas dos nós. Isto permite que suportem um grande número de nós. DHTs podem então ser utilizadas para a implementação de uma diversa variedade de aplicações P2P, tais como compartilhamento de arquivo, sistema de arquivos distribuídos, serviços de nome de domínio, *cache* web cooperativo e mensagens instantâneas.

Ao contrário dos mecanismos de busca P2P convencionais (centralizado e por inundação), a DHT utiliza um mecanismo de busca baseado em chaves mais estruturado, a fim de obter tanto a descentralização do Gnutella quanto a eficiência de resultados do Napster. No entanto, a DHT tem uma desvantagem, pois somente suporta buscas com resultado exato (utilizando a chave), ao invés de busca por palavras-chave. Esta funcionalidade pode ser feita em uma camada acima da DHT [47].

As quatro primeiras infra-estruturas baseadas em DHT (CAN [9], Chord [7], Pastry [8] and Tapestry [12]) foram introduzidas na mesma época em 2001. Desde então esta área de pesquisa é bastante ativa.

As principais características da DHT são resumidas nas seguintes propriedades:

- Descentralização: os nós compõem o sistema coletivamente sem coordenação central;
- Escalabilidade: o sistema deve funcionar de forma eficiente com qualquer quantidade de nós (milhares ou milhões);
- Tolerância a falhas: o sistema deve ser confiável mesmo com nós constantemente ingressando, saindo ou falhando.

3.5.1 Estrutura das *DHTs*

As DHTs são entendidas mais facilmente quando separadas em dois componentes: o esquema de particionamento de chaves, que divide a posse das chaves entre os nós, e a rede sobreposta, o qual conecta o nó e permite que se encontre o dono de uma dada chave. Esta decomposição foi sugerida em [48] e [49]. É importante ressaltar que diversos DHTs não seguem a linha apresentada aqui.

O seguinte exemplo explica o funcionamento de um DHT. Suponha-se que o tamanho da chave é um conjunto de 160*bits*. Para armazenar um arquivo com um nome e dados no

DHT, encontra-se o índice (*hash*) SHA1 do nome do arquivo, produzindo uma chave de 160*bits* denominada k. Envia-se então (k, dados) para qualquer nó participante no DHT. A mensagem é encaminhada de nó em nó na rede sobreposta até que se chega ao único nó responsável pela chave k (de acordo com o particionamento de chaves), onde (k, dados) é armazenado. A mensagem é roteada novamente pela camada sobreposta para o nó responsável por k, que irá então responder com os dados armazenados.

A seguir, são descritos o particionamento de chaves e a rede sobreposta mais detalhadamente.

Particionamento de chaves

A maioria dos DHTs utiliza alguma variante de *hashing* para mapear chaves a nós. Esta técnica utiliza a função $\delta(k1, k2)$ que define uma noção abstrata da distância entre a chave $k1$ para a chave $k2$. Cada nó é associado a uma chave única denominada identificador (id). Um nó com id i possui todas as chaves cujo i é o id mais próximo, medido de acordo de acordo com δ.

Na DHT Chord, por exemplo, trata-se as chaves como pontos em um círculo e $\delta(k1, k2)$ é a distância partindo em sentido horário em volta do círculo de $k1$ a $k2$. O espaço circular é divido em segmentos contínuos cujas pontas são identificadores de nó. Se $i1$ e $i2$ são ids adjacentes, então o nó com id $i2$ possui todas as chaves que estão entre $i1$ e $i2$.

Uma função *hash* consistente tem como característica o fato que uma remoção e adição de um nó altera somente o conjunto de nós possuído pelos nós com ids adjacentes, e deixa todos os outros nós inalterados. Em contrapartida, em uma tabela *hash* tradicional, todo conjunto de chaves precisa ser modificado com a entrada ou saída de um nó. Como cada alteração corresponde a uma largura de banda sendo consumida graças aos movimentos de objetos guardados de um nó para outro, minimizar esta reorganização aumenta a tolerância a entrada e falha de nós (em inglês *churn*).

Rede Sobreposta

Cada nó tem um conjunto de ligações para outros nós. Este conjunto de ligações forma a rede sobreposta, onde um nó escolhe seus vizinhos de acordo com a estrutura da topologia da rede na qual ele pertence.

Dada uma chave k, um nó qualquer tem um vizinho que possui a chave k ou tem uma ligação para um outro nó que esteja mais perto a k em termos de distância de particionamento de chaves descrita acima.

A topologia da rede deve garantir basicamente duas premissas. Primeiramente, o número de saltos deve ser baixo em qualquer rota (denominado dilatação da rede), a fim de atender aos pedidos de forma rápida. E segundo que o número de vizinhos por nó (denominado grau) seja também baixo, a fim de ter um *overhead* baixo. Isto significa que para se ter pouca dilatação é necessário ter um grau maior.

3.5.2 Implementações de *DHTs*

A seguir, serão descritas as principais implementações do protocolo DHT. São eles: Chord, Bamboo/OpenDHT, Bunshin, CAN, DKS System, Kademlia, Pastry e Tapestry.

Chord

O projeto Chord tem como objetivo construir sistemas distribuíveis escalonáveis, simétricos e robustos utilizando as premissas de uma rede P2P. Ele é completamente descentralizado e pode encontrar dados utilizando somente $log(N)$ mensagens, onde N é o número de nós no sistema. O mecanismo de busca do Chord é comprovadamente robusto quando se trata de entrada e falhas freqüentes de nós. O Chord está sendo desenvolvido no MIT.

Topologicamente o Chord lembra um anel. No Chord, cada nó armazena somente um subconjunto de todas as chaves, o que aumenta a confiabilidade da rede. O Chord pode ser utilizado para implementar diferentes serviços, incluindo serviços de busca de informação distribuída. O projeto foi desenvolvido com seis objetivos:

- Escalonável – significa que o sistema pode continuar funcionando eficientemente a medida em que a rede cresce.
- Disponibilidade – significa que o sistema pode funcionar mesmo no momento em que a rede se divide ou alguns nós falham.
- Balanceamento de carga – significa que os pares chave/valor são distribuídos igualmente no sistema.
- Dinamicidade – significa que o sistema pode suportar mudanças rápidas na topologia da rede.

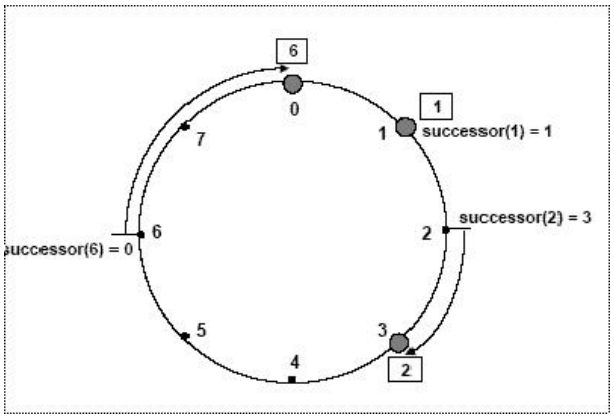

Figura 3.4: Visão geral da arquitetura Chord [7]

- Atualizabilidade – significa que os dados podem ser atualizados utilizando o algoritmo DHT.

- Localização de acordo a proximidade – significa que se o valor da busca é armazenado em um nó próximo ao nó que originou a buscam então os dados devem ser obtidos desde esse nó, em vez de percorrer grandes distâncias.

Quando o sistema está sob grande demanda, é possível que a rede sobreposta se particione. Isso acontece quando um grande número de nós deixam a rede simultaneamente quase dividindo a rede em duas subredes que tem uma conexão limitada à outra, ao menos na teoria. O Chord foi desenhado para que estas divisões de redes não afetem a usabilidade da rede toda.

O Chord também mantém uma tabela de localidades em cada nó. Esta tabela funciona como um *cache* e contém os nós que recentemente foram encontrados pelo Chord. O *cache* é basicamente uma tabela de roteamento que mapeia identificadores de nós com os seus endereço *ip* e portas correspondentes. Caso sejam armazenadas informações dos nós, este *cache* pode ser usado para acelerar a busca por chaves, já que nós mais próximos topologicamente são melhores que nós mais distantes. Isso é decidido baseado no *round trip time* do nó destino.

À nível de aplicação, o Chord é dividido em servidor e cliente, onde ambos rodam no mesmo nó da rede e são implementados como *Remote Procedure Calls* (RPCs) e suportam atém 10000 requisições por segundo.

Na Figura 3.4, temos uma visão geral da arquitetura.

Utilizando o protocolo de busca Chord, as chaves dos nós são organizadas em círculo. Este não pode ter mais de $2m$ nós. O círculo pode ter id/chaves variando de 0 a $2m - 1$. Na Figura 3.4 temos $m = 3$.

Na Figura 3.4 os círculos maiores representam nós. Os pontos negros representam chaves. Ids e chaves são obtidos utilizando a *consistent hashing*. O algoritmo SHA-1 é a função base para a *consistent hashing*.

Cada nó tem um sucessor e um predecessor. O sucessor de um nó ou chave é o próximo nó no círculo movendo-se em sentido horário. O predecessor é o próximo movendo-se sentido anti-horário.

Pastry

Cada nó Pastry tem um id único de 128*bits*. O conjunto de ids dos nós é distribuído uniformemente – isto pode ser alcançado, por exemplo, baseando o id do nó em uma *hash* da chave pública do nó ou do endereço *ip*. Dada uma mensagem e uma chave, o Pastry a roteia ao nó com o id que é numericamente mais perto da chave. Assumindo uma rede com N nós, Pastry pode rotear para qualquer nó em menos de $\lceil log_{2^b} N \rceil$ passos em média (b é um parâmetro de configuração cujo valor é usualmente 4). Com seguidas falhas de nós, a entrega é garantida a não ser que $l/2$ ou mais nós com ids adjacentes falhem simultaneamente (l é um parâmetro inteiro par cujo valor usualmente é 16).

As tabelas necessárias em cada nó Pastry têm apenas $(2^b - 1) * \lceil log_{2^b} N \rceil + l$ entradas, onde cada entrada mapeia um id ao endereço *ip* associado. Além disso, após a falha ou ingresso de um novo nó, as invariantes em todas as tabelas de roteamento podem ser recuperadas trocando-se $O(log_{2^b} N)$ mensagens.

Em relação à roteamento, ids e chaves são geralmente uma seqüência de dez dígitos com base $2b$. A tabela de roteamento (Figura 3.5) de um nó é organizada em $\lceil log_{2^b} N \rceil$ linhas com $2b - 1$ entradas cada. A distribuição uniforme dos *id* assegura um povoamento equilibrado do espaço de *ids*. Desta forma, somente $\lceil log_{2^b} N \rceil$ níveis são povoados na tabela de roteamento. Cada entrada na tabela de roteamento se refere a um de muitos nós cujo id tem o prefixo apropriado. Entre tais nós, o mais próximo ao atual (de acordo com uma métrica de proximidade escalar, tal como o *round trip time*) é escolhido.

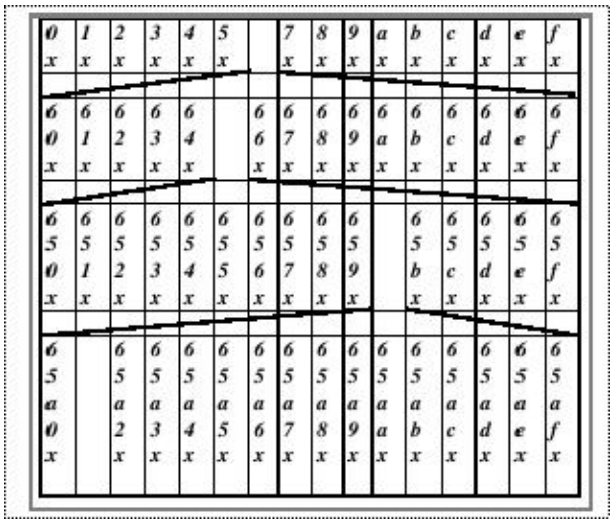

Figura 3.5: Tabela de roteamento de um nó Pastry [8]

Além da tabela de roteamento, cada nó mantém o endereço *ip* dos nós no seu conjunto de folhas (*leaf set*). Este conjunto pode ser, por exemplo, o conjunto de nós com os $l/2$ ids numericamente mais perto em relação ao id do nó atual.

A Figura 3.6 mostra o caminho de uma mensagem de exemplo. Em cada passo do roteamento, o nó atual normalmente encaminha a mensagem para um nó cujo id compartilhe com a chave um prefixo que é pelo menos um digito maior que o prefixo que a chave compartilha com o nó atual. Se tal nó não é encontrado na tabela de roteamento, a mensagem é encaminhada para o nó cujo id compartilhe um prefixo com a chave do mesmo tamanho que a do nó atual. Tal nó deve existir no conjunto de folhas a não ser que a) o id do nó atual ou seus vizinhos imediatos sejam numericamente mais pertos à chave, ou b) $l/2$ nós adjacentes no conjunto de folhas tenham falhado.

Inserção e falha de nós

Um ponto chave na arquitetura do Pastry é como manter com eficiência e dinamicidade o estado do nó, ou seja, a tabela de rotas, o conjunto de folhas e o conjunto de vizinhos, na presença de falhas de nós, recuperação de nós, e inserção de nós.

Resumidamente, um nó inserido com um id recém escolhido X pode inicializar seu estado através do contato com um nó próximo A (de acordo com a métrica de proximidade) e solicitar a A que encaminhe uma mensagem especial utilizando X como chave. Esta mensagem é roteada para o nó existente Z com id numericamente mais próximo a X. X então obtém o

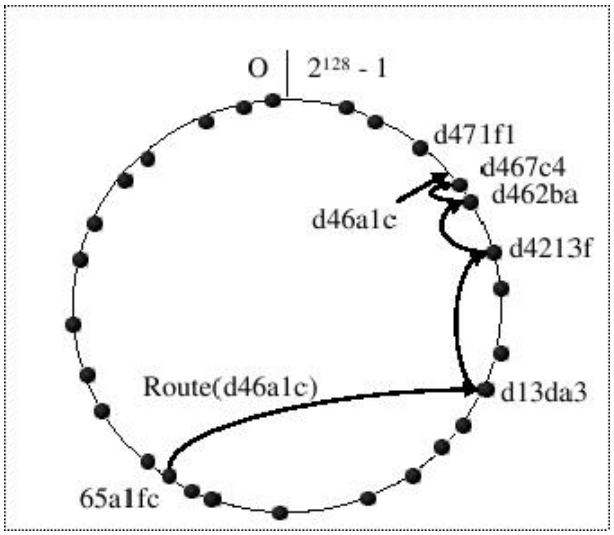

Figura 3.6: Roteamento de uma mensagem no Pastry [8]

conjunto de folhas de Z. Pode-se mostrar que utilizando esta informação, X pode inicializar de maneira correta o seu estado e notificar nós que precisam saber da sua inserção.

Para tratar falhas em nós, nós vizinhos no espaço de ids que estão ciente um do outro devido ao fato de estarem um no conjunto de folhas do outro, periodicamente trocam mensagens do tipo *keep alive*. Se um nó não responde durante um período T, presume-se que ele tenha falhado. Todos os membros do conjunto de folhas do nó que falhou são então notificados e eles atualizam seus conjuntos de folhas. Um nó que tenha se recuperado contata os nó de acordo com o último conjunto de folhas conhecido, obtém o conjunto de folhas atual, atualiza o seu próprio conjunto e então notifica os membros do conjunto da sua presença. Entradas nas tabelas de roteamento que referem-se a nós que tenha falhado são reparados pouco a pouco.

Bamboo/OpenDHT

O OpenDHT é um projeto que utiliza o algoritmo Bamboo como base. O Bamboo é baseado no Pastry, com uma melhoria no tratamento de entrada e saída de nós.

Uma dificuldade ao prover uma infra-estrutura DHT compartilhada é desenhar uma interface que atenda às necessidades de uma boa variedade de aplicações para justificar a distribuição compartilhada. O OpenDHT trata desse problema de duas formas. Primeiramente, uma interface coloca/retira (*put/get*) faz com que a escrita de aplicações seja simples, porém ainda possua uma grande quantidade de aplicações de armazenamento. E segundo, o uso de uma bi-

blioteca no cliente chamada ReDiR permite que interfaces mais sofisticadas sejam feitas sobre a base da interface coloca/retira.

A interface do OpenDHT coloca/retira atende à necessidade de uma grande gama de aplicações, desde armazenamento do Cooperative File System (CFS) [50] a mensagens instantâneas.

Os objetivos do projeto da interface coloca/retira são os seguintes:

Primeiro, aplicações OpenDHT simples devem ser simples de se escrever. A vantagem de uma DHT compartilhada está na sua maioria em quão fácil é de se utilizar. O OpenDHT pode ser acessado utilizando tanto Sun RPC sobre TCP ou XML RPC sobre *Hypertext Transfer Protocol* (HTTP), facilitando deste modo, o uso desde linguagens de programação e funcionando desde *firewalls* a *Network Address Translation* (NATs). Um programa Python que lê uma chave e valor desde a console e os coloca no DHT tem somente nove linhas, e o programa complementar que retira tem somente onze.

Segundo, o OpenDHT não deveria restringir a escolha de chave. Esquemas anteriores de autenticação de valores armazenados em uma DHT necessitavam de uma relação em particular entre o valor e a chave sobre a qual está armazenada. Já se sabe de aplicações que possuem condições de escolha de chave que são incompatíveis com tais restrições. Não é aconselhável impor restrições similares em aplicações futuras.

Terceiro, o OpenDHT deve prover autenticação para clientes que a necessitam. Um cliente pode desejar verificar se uma entidade autorizada escreveu o valor sob uma chave em particular ou proteger seus próprios valores de serem sobrescritos por outros clientes. Como apresentado a seguir, certos ataques não podem ser prevenidos sem um esquema de autenticação no DHT. Como o objetivo neste caso é a simplicidade, a autenticação é uma opção e não uma necessidade.

A forma atual de distribuição do OpenDHT atende aos dois primeiros objetivos apresentados (simplicidade e escolha de chaves) e possui em parte, suporte ao terceiro (autenticação).

CAN

CAN significa Rede de Conteúdo Acessível. A topologia da CAN é um espaço cartesiano de n dimensões. O espaço de coordenadas é do tipo lógico e não necessariamente assemelha-se a uma coordenada física espacial. É sempre particionado inteiramente entre todos os nós no sistema, cada um possuindo uma parte dele. Quando um nó junta-se a rede sobre-

posta, alguma partição reservada é dividida pela metade para alocar espaço para o novo nó. Um nó é vizinho de outro nó se seus espaços de coordenadas se encontram no eixo $d-1$. No caso de se utilizar duas dimensões, as partições no espaço das coordenadas teriam de ser encontrar em um eixo. A Figura 3.7 representa esta estrutura.

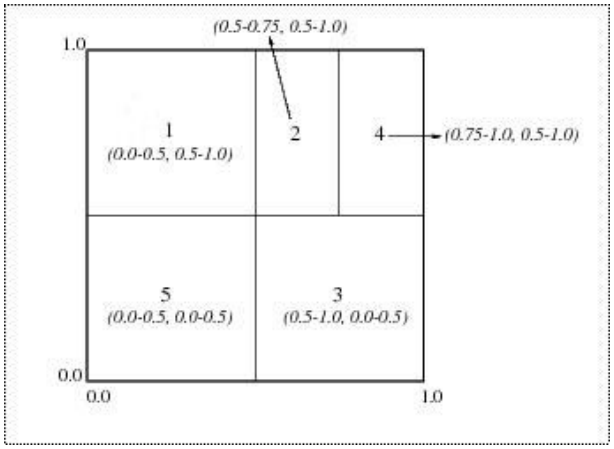

Figura 3.7: DHT CAN [9]

Cada nó no sistema CAN armazena uma parte da tabela de índices e também guarda informação sobre os nós vizinhos que estão próximos a ele. A proximidade neste caso é medida em saltos.

CAN pode ser usada para implementar gerenciamento de armazenamento em larga escala e sistemas de recuperação. Pelo menos um protótipo de implementação do CAN é conhecida: pSearch é um serviço de busca semântica baseada em conteúdo que utiliza uma variante hierárquica da CAN, chamada eCAN [51].

Bunshin

O cenário onde o Bunshin se situa é o projeto Planet [10]. O Bunshin surgiu da necessidade de criar serviços de persistência em Dermi [52]. O Dermi é um *middleware* de objetos distribuídos construído em cima de Pastry, e em um primeiro momento utilizava o DHT PAST para seu serviço de diretório descentralizado. Como o PAST não possuia uma implementação confiável e mudanças eram difíceis de fazer, foi implementado um DHT próprio.

A arquitetura do Bunshin pode ser dividida em 3 camadas. A camada de aplicação propriamente dita e a camada de serviços que se constrói em cima desta. A Figura 3.8 ilustra esta divisão.

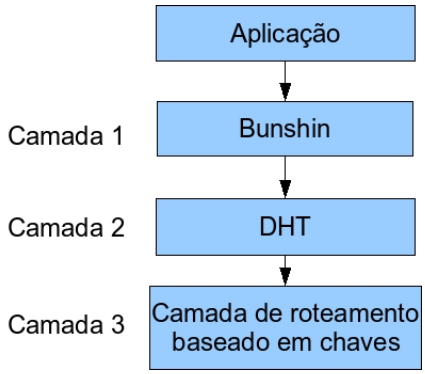

Figura 3.8: Divisão em 3 camadas do Bunshin [10]

As características principais do sistema são as seguintes: sistema de replicação ativo, esquema de *caching* adaptativo, diferentes modos de persistência, valores multicampo e estrutura multicontexto. Estas características serão descritas brevemente a seguir.

Replicação

Para poder decidir quais serão os nós que guardarão as réplicas de uma chave/valor, o nó responsável escolhe seus vizinhos mais próximos, obtendo a lista de sucessores da rede sobreposta. Assim assegura-se que caso o nó responsável falhe, seu sucessor imediato terá os dados que precisa. Além disso, caso a rede sobreposta utilizada agrupe os vizinhos por proximidade, como é o caso do Pastry, as operações que envolvam estes vizinhos serão mais eficientes.

Os candidatos a guardar as réplicas são obtidos através do método *replicaSet()*, oferecido pela rede sobreposta, o qual retorna a lista dos nós sucessores ordenados de uma chave. Esta chave também é utilizada para informações de entrada ou saída dos nós, através do método *update()*. As seguintes provas são feitas para prevenir a perda de mensagens ou eventos:

- Comprovar se todas as chaves armazenadas atualmente por um nó ainda pertencem a ele, mediante a comprovação de que o nó atual é o primeiro que retorna o método *replicaSet()*. Se descobrir-se que algum par chave/valor não é mais de sua responsabilidade ele é reinserido na rede, chegando assim ao seu novo dono.

- Comprovar se os nós nos quais há réplicas armazenadas ainda estão ativos, assim como comprovar se o número de réplicas de cada chave é igual ao número desejado. Caso alguma dessas condições não seja atendida, significa que é necessário realizar mais répli-

cas e portanto, serão escolhidos os candidatos necessários e serão realizadas as réplicas faltantes.

- Comprovação de que os nós dos quais se obtém réplicas ainda estão ativos. Caso contrário, o procedimento descrito inicialmente é realizado novamente. Neste caso, o critério a seguir pelo novo dono é guardar o par chave/valor com uma versão mais nova, sendo muito importante neste ponto o controle de versões.

O controle de versões segue uma política atemporal, ou seja, a data em que o valor foi atualizado não é levada em consideração, já que não se pode garantir que todos os pontos estão sincronizados temporalmente. Para isso, a técnica a seguir é a de designar um número de atualização sobre uma cópia viva do sistema, sendo este, um procedimento fácil de reconhecer se uma réplica nos envia um valor anterior ao que já se tem. Caso se trate de uma cópia sem número de versão, significa que é uma nova atualização.

Caching

A cópia de chaves/valor proporciona duas características importantes que são confiabilidade e performance. Para melhorar a performance, utiliza-se a técnica de *caching*.

Alguns nós podem começar a sofrer sobrecargas temporárias de requisições de clientes próximos. Para solucionar este problema, utiliza-se o sistema de *caching* adaptativo. O uso desses *caches* é dinâmico e adaptativo segundo o número de requisições em um espaço de tempo sobre certa chave/valor, proporcionando a criação de cópias temporárias que não mantém a estrutura ocupada.

Para que o número de acessos a um nó diminua e um balanceamento de carga seja efetuado, o dono da chave/valor envia uma cópia para que seja armazenada no *cache* do nó imediatamente anterior a partir de onde se recebe o número máximo de requisições. Desta forma, mantém-se uma cópia temporária no *cache* e o dono deixará de receber requisições desse nó durante o tempo que o *cache* durar.

Caso o nó vizinho que possui o *cache* também seja altamente requisitado, este também ativará um *cache* no nó vizinho interessado. O objetivo é que a cópia seja propagada nos *caches* dos nós mais interessados no momento, de forma que nenhum nó seja saturado por ser o dono de uma chave/valor muito popular, mas que ativem-se cópias próximas dos nós que fazem as requisições.

Na Figura 3.9 podem-se visualizar as requisições '1' de U1 e U2 acabam por saturar o nó O, que faz *cache* em C1. Como C1 também fica saturada, esta faz *cache* em C2.

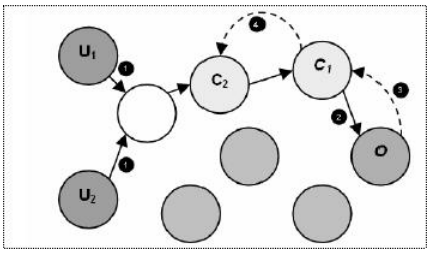

Figura 3.9: Uso de cache no Bunshin [10]

Modos de persistência

O Bunshin foi projetado para que seja simples de se extender e portanto, é compatível com diferentes modos de persistência:

- Memória.
- Disco.
- FileMapping.

O modo de persistência em memória é apropriado para os dados replicados ou em *cache* e também para os dados de aplicações de teste. O modo padrão é o modo de persistência em disco, armazenando todos os dados em um determinado diretório.

O mais interessante é o *FileMapping* que baseando-se no armazenamento em disco, têm mecanismos que permitem mapear os arquivos no estado original na qual foram inseridos no Bunshin. Este modo de persistência é muito útil para aplicações que necessitam acessar os arquivos diretamente. Um exemplo é o *Snap*, que é um portal descentralizado construído sobre uma rede P2P estruturada, que dá suporte a aplicações web, onde uma delas utiliza o Bunshin como serviço de persistência do servidor web e onde são inseridos documentos, os quais precisam ser acessados por um cliente web.

Multicontexto e multicampo

O Bunshin não é um DHT convencional, no sentido de estrutura chave/valor, já que proporciona a possibilidade de poder tratar os dados como se fossem uma tabela *hash*. Desta maneira, para uma chave podemos ter tantos valores quanto forem desejados, identificados por uma subchave denominada *field*.

Outra forma, com nível de abstração mais elevado, é poder visualizar um nó não somente como um recipiente único, mas sim de tantos quanto forem necessários. Portanto, caso

a aplicação precise ter diferentes recipientes de armazenamento, ela não está obrigada a instanciar uma aplicação Bunshin para cada um deles e somente será necessário indicar onde se quer trabalhar.

DKS System

O *multicast* é tratado geralmente em duas abordagens quando diz respeito a infra-estruturas baseadas em DHT [11]. Na primeira (SCRIBE), uma árvore *multicast* é mantida por cada grupo, por alguns nós na infra-estrutura sobreposta. Uma vantagem dessa abordagem é que a raiz da árvore *multicast* pode ser utilizada para controle de acesso. A maior dificuldade no entanto, é que há possibilidade de gargalos no nó raiz por onde todas as mensagens *multicast* devem passar. Na segunda abordagem (CAN), os membros de grupos podem auto-organizar-se em uma rede sobreposta similar a rede sobreposta sobre a rede P2P. Essa abordagem é provida da vantagem de que somente grupos de membros mantêm informações sobre o grupo. Gargalos somente ocorrem em nós iniciais e não necessariamente por causa do *multicast*. No entanto, controle de acesso pode ser difícil neste caso.

No sistema DKS, o *multicast* é obtido em $DKS(N, k, f)$ redes sobrepostas. A arquitetura se baseia no CAN, sendo que a maior vantagem em relação a este é que cada grupo é construído de forma a atender os requisitos de tamanho máximo de grupo, grau de tolerância a falha e custo de manutenção. Membros de grupos de *multicast* auto-organizam-se em uma instância de $DKS(N, k, f)$, que é criada e mantida exatamente como a rede sobreposta. Mensagens *multicast* são disseminadas eficientemente graças a um algoritmo de correção de *broadcast*, que assegura que cada processo da camada de aplicação receba cada mensagem exatamente uma vez, mesmo com a presença de informação de rotas equivocadas no nível da infra-estrutura.

A forma como os grupos são criados é descrita a seguir. Sendo O uma instância de $DKS(N, k, f)$, chamada de rede sobreposta subjacente. Sendo n um nó de O e assumindo-se que o nó n quer criar um novo grupo identificado por g. Então, o nó n primeiro determina as características do grupo g. As quais incluem, (i) kg: a paridade do grupo. Este parâmetro serve para a construção da topologia da rede sobreposta representando o grupo g. (ii) Ng: uma potência de kg que especifica o número máximo de membros que o grupo g pode ter. Este número serve para determinar o anel lógico em que os membros do grupo são mapeados. (iii) fg: o parâmetro de tolerância a falhas dentro do grupo g. (iv) Hg: a função *hash* usada para mapear nó de O para o anel lógico de tamanho máximo Ng. Então, uma vez que esses parâmetros tenham

sido configurados, o nó n se insere como o primeiro nó do $DKS(Ng, kg, fg)$ representando o grupo g. As características do grupo g, e o endereço do nó n são disponibilizados para que outros nós de O interessados em g possam se juntar ao grupo.

A entrada, saída ou falha de um membro de um grupo é tratada exatamente como na rede sobreposta subjacente. No entanto, uma política tem que ser adotada para se tratar das colisões, já que o tamanho do grupo pode ser relativamente pequeno. No modelo atual, a política adotada é *first-come-first-in*.

O princípio do grupo *multicast* nas redes sobrepostas do tipo $DKS(N, k, f)$ está apresentado na Figura 3.10. Na figura, a) mostra uma instância de $DKS(N = 32, k = 2, f = 3)$, servindo como a rede sobreposta subjacente, onde somente nós com identificadores 4, 11, 15, 19, 25 e 28 estão presentes. b) mostra um $DKS(N = 16, k = 4, f = 2)$ representando um grupo *multicast* com identificador $g = 4$, onde os nós 28, 25, 19 e 15 de O são membros. Devido à função *hash* do grupo $g = 4$, denominada $H4$, cada um dos nós 28, 25, 19 e 15 recebem um novo identificador referente a este grupo *multicast*. Por exemplo, o nó 28 recebe 1 como identificador do grupo $g = 4$. c) mostra uma $DKS(N = 8, k = 2, f = 1)$ representando o grupo $g = 2$ onde os nós 4, 11, 15 de O são membros.

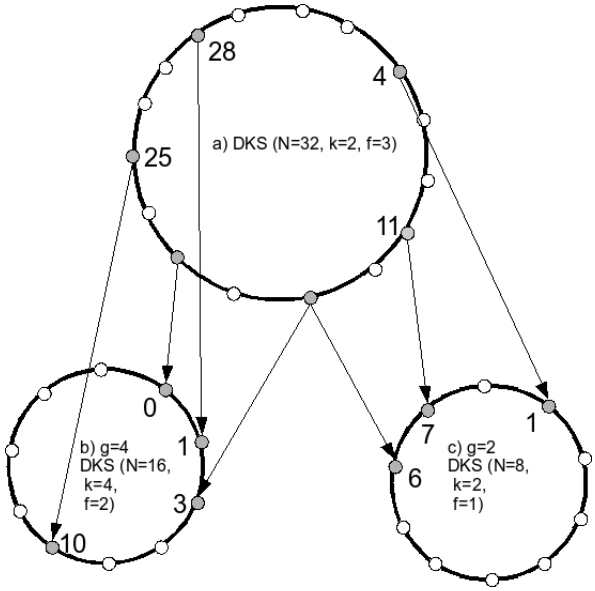

Figura 3.10: Dois grupos baseados em um $DKS(32, 2, 3)$ [11]

O maior desafio em se fazer um *multicast* em uma rede sobreposta $DKS(N, k, f)$ é alcançar um encaminhamento ótimo de mensagens *multicast*. Em uma rede $DKS(N, k, f)$, para o nó n, o espaço de identificação (anel) é visível somente de $log\, k(N)$ níveis. A cada nível,

o nó n tem uma visão restrita do espaço de identificação, que consiste em k partes disjuntas. O nó n é responsável por uma parte e o nó n mantém informações sobre os responsáveis das outras partes. Isto forma a informação de rota do nó n.

Assumindo-se que todo nó tem a informação de roteamento correta, a idéia basicamente é a seguinte: Quando um n necessita enviar uma mensagem de tipo *broadcast* denominada *msg*, o nó n envia *msg* para cada nó responsável, n', incrementando o intervalo, In, cujo responsável é o nó n'. Ao receber a mensagem, o nó n' envia *msg* para cada responsável que ele conhece entre o intervalo In', incrementando um subintervalo de ln'. Devido a essa divisão dos intervalos, não há mensagens repetidas. Este processo é repetido até que todos os nós tenham sido considerados.

Este algoritmo funciona bem quando todos os nós possuem informações corretas de rotas. Contudo, devido à dinamicidade da rede sobreposta, a informação de rotas fica desatualizada. Em $DKS(N, k, f)$, isto é tratado pela técnica de correção por uso.

Kademlia

O Kademlia foi desenhado por Petar Maymounkov e David Mazières. Existem duas variantes do Kademlia: o Kashmir e o SharkPy, ambos feitos em *python*, mas aparentemente não estão em desenvolvimento. Os nós do Kademlia se comunicam utilizando UDP. No Kademlia, cada nó tem um identificador de 160*bits*. Cada mensagem que um nó transmite inclui o seu id, de modo que o receptor salve a existência do remetente caso seja necessário.

As chaves também são identificadas com 160*bits*. Para publicar e encontrar pares do tipo (chave, valor), o Kademlia baseia-se na noção de distância entre dois identificadores. Dados dois identificadores de 160*bits*, x e y, o Kademlia define a distância entre eles como sendo a disjunção exclusiva (XOR), interpretada como um inteiro, $d(x, y) = x \oplus y$.

Um nó que quer juntar-se a uma rede deve primeiramente passar pelo processo de inicialização. Nesta fase, o nó precisa saber o endereço *ip* de outro nó (fornecido pelo usuário, ou por uma lista armazenada) que já seja participante da rede Kademlia. Caso o nó iniciante nunca tenha participado da rede, ele registra um id aleatório único. Ele utiliza este id até que ele saia da rede.

Quando procura-se por uma chave, o algoritmo explora a rede em diversos passos, e em cada um deles aproxima-se cada vez mais da chave que se deseja, até que o nó solicitado retorne

o valor, ou nenhum nó mais próximo seja encontrado. O número de nós contatados durante a busca depende do tamanho da rede.

Assim como a métrica em círculo do Chord, o XOR é unidirecional. Para cada ponto x e distância $\Delta > 0$, há exatamente um ponto y tal que $d(x, y) = \Delta$. Devido ao fato de ser unidirecional, assegura-se que todas as buscas pela mesma chave convirjam para um mesmo caminho, independente do nó original. Assim, fazendo *cache* de pares de chaves e valores pelo caminho diminui-se a formação de pontos de processamento alto (*hot spots*). Como o Pastry e diferente do Chord, a topologia XOR também é simétrica $(d(x, y) = d(y, x)$ para todo x e y.

Cada nó possui uma lista de triplas (Endereço *ip*, porta UDP, id do nó) para nós de distância entre $2i$ e $2i + 1$ dele mesmo, para cada $0\, i\, 160$. Esta lista é denominada *k-bucket*.

O Kademlia possui um protocolo próprio com quatro RPCs: *ping*, *store*, $find_node$ e $find_value$. O *ping* testa um nó para verificar se está *online*. O *Store* faz com que o nó armazene um par (chave, valor) para que seja acessado depois. O $Find_node$ recebe um id de 160*bits* como parâmetro. O membro retorna uma tripla (Endereço *ip*, porta UDP, id do nó) para os k nós conhecidos que estão mais próximos do id de destino. Estas triplas podem partir de um simples *k-bucket*, ou podem partir de múltiplos *k-buckets* se o *k-bucket* mais próximo não estiver cheio. Neste caso, o recipiente RPC deve retornar k itens.

Em todas as RPCs, o recipiente deve possuir um id aleatório de 160*bits*, o que dificulta um pouco a falsificação de endereço.

Tapestry

O Tapestry é a infra-estrutura de roteamento do OceanStore [53]. O Tapestry assume que cada nó e objeto do sistema pode ser reconhecido com identificadores únicos, representados por um conjunto de dígitos. Como no Pastry, os identificadores são distribuídos uniformemente no espaço de nomes. Os identificadores de objeto são conhecidos como identificadores únicos globais ($GUIDs$). Isto significa que toada busca tem um $GUID$ de destino único que em uma dada instância é traduzido para o id de um nó. Para um conjunto de dígitos $\propto$, $\mid \propto \mid$ representa o número de dígitos nesse conjunto.

O Tapestry herda sua estrutura básica do esquema de localização de dados de PPR [10]. Como no esquema do PRR, cada nó Tapestry contém ponteiros para outros nós (conexões vizinhas), assim como os mapeamentos entre os $GUIDs$ de objetos e id dos nós de servidores de armazenamento. Buscas são roteadas de nó em nó entre conexões vizinhas até que um pon-

teiro de objeto apropriado seja encontrado, e neste ponto, a busca é encaminhada por meio de conexões vizinhas para o nó de destino.

Basicamente, o algoritmo do Tapestry é este: quando um nó deseja divulgar que possui certo objeto, é enviada uma mensagem de publicar para o nó raiz único do objeto. Em cada salto da rede, incluindo o próprio nó raiz, um ponteiro para a localização do objeto é salvo. Quando outro nó quer localizar o objeto, é enviada uma mensagem de localização para o nó raiz do objeto. Cada nó durante o percurso da mensagem verifica se ele tem um ponteio para o objeto. Caso possua, a busca de localização é encaminhada diretamente para o nó que publicou o objeto; caso contrário, a busca continua pelo caminho para o nó raiz do objeto, onde é garantido que um ponteiro existe. Quando objetos são replicados e múltiplos nós publicam o mesmo objeto, cada nó mantém uma lista de todos os ponteiros que recebeu para cada objeto, ordenados pela distância de rede daquele nó até o publicador. Ele irá direcionar buscas daquele objeto para o publicador mais próximo.

As Figuras 3.11 e 3.12 demonstram este processo.

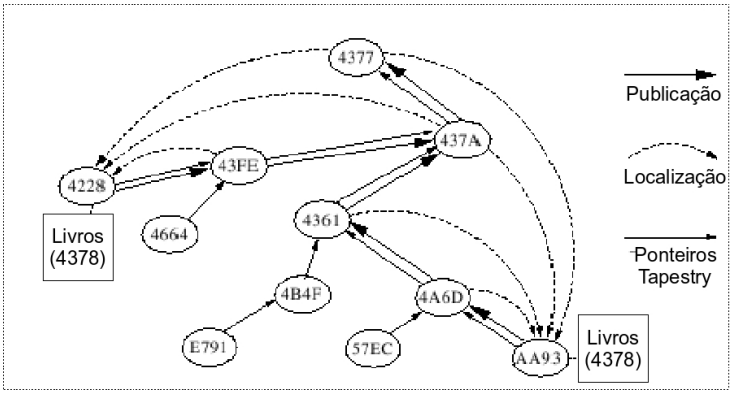

Figura 3.11: Publicação de um objeto Tapestry [12]

A estrutura de roteamento Tapestry

A estrutura de roteamento do Tapestry é uma rede sobreposta entre os nós participantes. Cada nó possui uma conexão para uma série de vizinhos que compartilham prefixos com o seu id. Desse modo, nós com prefixo $\propto$ estão restritos a nós que compartilham prefixos com $\propto$. Conexões entre vizinhos são nomeadas de acordo com o número de nível, que é um número maior que o número de dígitos no prefixo compartilhado ($| \propto | + 1$). A Figura 3.13 apresenta uma parte da arquitetura de roteamento. Para cada ponteiro vizinho dianteiro do nó A para o nó B, haverá um ponteiro vizinho traseiro de B para A.

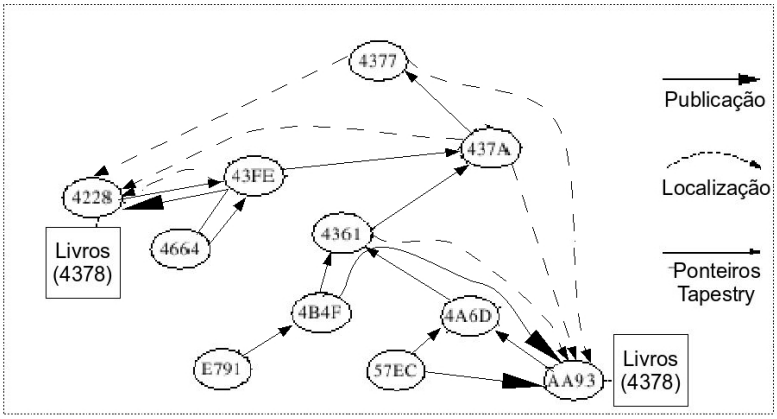

Figura 3.12: Rota para o objeto tapestry [12]

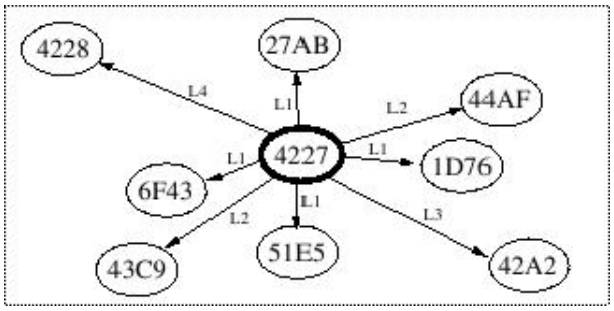

Figura 3.13: Estrutura de roteamento Tapestry [12]

3.6 Conclusão

Foram apresentadas as principais características da tecnologia P2P. Dentre os modelos apresentados estão o modelo centralizado e o híbrido, cada um com suas diferentes versões.

Diversas considerações devem de ser levadas em conta no momento de projeção do aplicativo, já que os sistemas P2P possuem diversas características que podem influenciar no desempenho da aplicação, como escalabilidade, por exemplo.

O uso do P2P cresce a cada momento e o número de aplicações que utiliza essa tecnologia vem crescendo. Programas como o Skype já fazem parte do nosso cotidiano. Foram apresentadas outras aplicações e as principais tecnologias atuais.

Devido ao desenvolvimento atual das DHTs, que possuem um foco especial dentro dos vários tipos de redes P2P, o próximo capítulo trata das diferentes tecnologias existentes.

Capítulo 4

Proposta

4.1 Introdução

Neste capítulo, é apresentado um sistema de detecção de plágio em documentos escritos em linguagem natural chamado **PeerDetect**. Este sistema é baseado nas idéias do Web2Peer [54]. O Web2Peer é uma infra-estrutura para publicação e recuperação de páginas *web* em uma rede P2P. PeerDetect apresenta duas abordagens. Elas diferenciam-se principalmente na fonte de dados utilizada na busca de plágio e também no algoritmo utilizado para realizar esta busca. Como a primeira abordagem centraliza o trabalho de detecção de plágio em um nó somente, foi chamada de abordagem centralizada. Na segunda abordagem este trabalho é dividido entre nós presentes na rede, portanto foi chamada de abordagem descentralizada.

Vale ressaltar quando nos referimos a linguagem naturais estamos nos referindo às línguas faladas por seres humanos e usadas como instrumento de comunicação, diferentemente daquelas que são linguagens formais construídas (linguagens de programação). Dentre as linguagens naturais, os testes foram feitos utilizando as línguas portuguesa e inglesa. Não há empecilhos em utilizar as arquiteturas apresentadas aqui em outras línguas, desde que a codificação da linguagem seja ocidental.

4.2 Abordagem centralizada

A abordagem centralizada do PeerDetect possibilita a utilização, como fonte de dados, de computadores pessoais conectados através de uma rede P2P. Todos os usuários na rede são nós que podem publicar documentos e também podem utilizar esta rede como uma fonte de da-

dos para procurar por possíveis plágios nos documentos disponíveis em outros nós. PeerDetect utiliza as vantagens de redes P2P através protocolos JXTA para publicar e procurar documentos similares.

O JXTA possui seu próprio protocolo para a pesquisa de recursos, neste caso, documentos escritos em linguagens naturais. Porém, não é utilizado, visto que a busca é baseada em uma abordagem por inundação (Seção 3.2). Esta abordagem resulta em um tempo de resposta alto e em certas ocasiões, um recurso disponível na rede pode não ser encontrado devido ao escopo limitado da busca. Para superar estas características, foi utilizado um serviço de indexação baseado em tabelas *hash* distribuídas (DHT) (Capítulo 3.5), a qual provê respostas mais rápidas e determinísticas, se comparadas com o mecanismo JXTA. A DHT cria uma relação entre a localização do documento, dentro da rede P2P, e uma ou mais palavras-chave. A abordagem centralizada do PeerDetect possui duas funcionalidades principais – publicar documentos disponíveis nos nós e procurar por plágios de um documento.

4.2.1 Publicação de documentos

A publicação de documentos é um processo importante, onde se permite que outros nós tomem conhecimento da existência de um documento. Para publicar um documento, é necessário definir palavras-chave de acordo com o seu assunto, onde cada uma delas será utilizada para indexá-lo na DHT. Elas também serão utilizadas no processo de verificação para que seja descoberto na DHT quais os documentos relacionados a estas palavras-chave.

Estas palavras que possuem relação com o documento, são determinadas por um algoritmo específico (algoritmo inicial). Este algoritmo tem como característica a rapidez em extrair informações simples que diferem um documento do outro. Muitos algoritmos existentes podem ser utilizados, como apresentado na Seção 2.4.

Para evitar o problema de haver comparações entre documentos de contextos diferentes (por exemplo, um documento sobre Compucação@SistemasDistribuídos comparado com um da Medicina@Patologia) o PeerDetect define o conceito de comunidade de nós. A comunidade de nós é formada por nós que possuem documentos dentro de uma área de interesse específica. Sendo assim, ao publicar o documento, o usuário necessita especificar uma série de atributos, ou seja, um conjunto de palavras-chave, que define a comunidade de nós do documento. Estes atributos são: área de conhecimento do documento, linha de pesquisa, tópico e assunto. A

Figura 4.1 apresenta a arquitetura do PeerDetect, onde é mostrado o Peer 0 passo-a-passo no processo de publicação e na seqüência, cada passo é explicado.

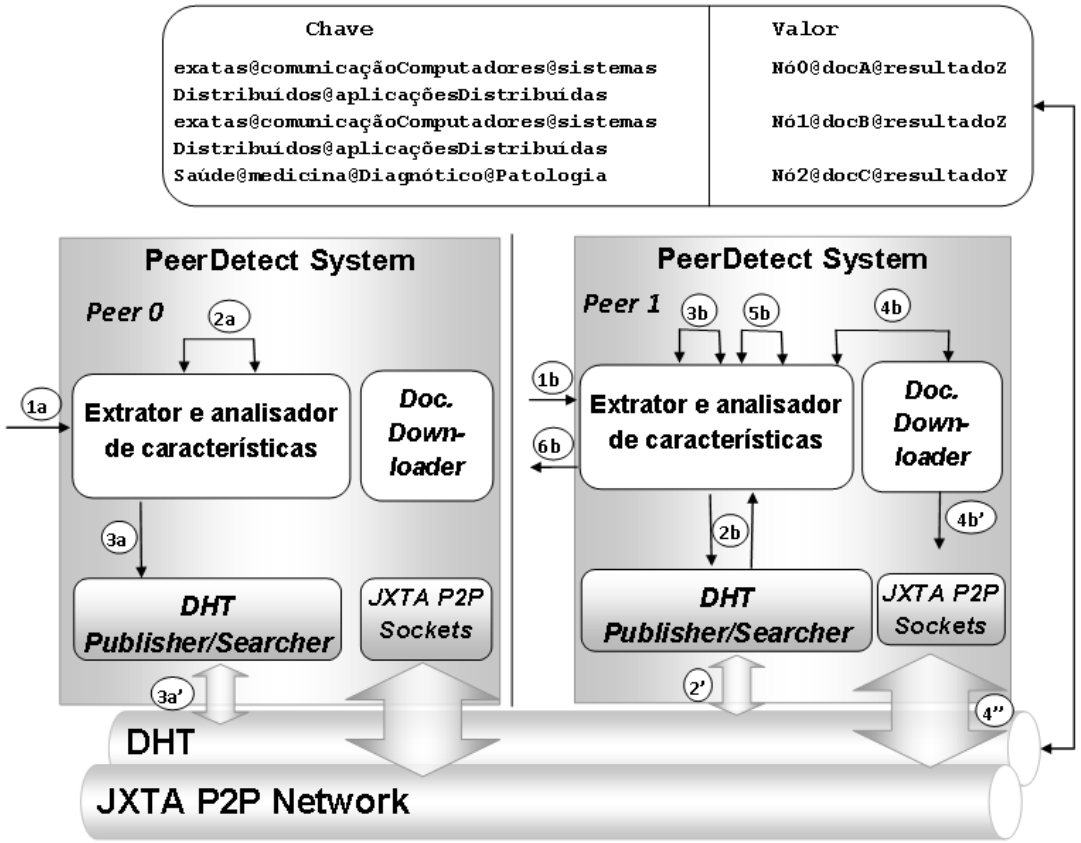

Figura 4.1: Processo de publicação (Peer 0) e de verificação (Peer 1) para a abordagem centralizada

1. a) O usuário define a comunidade de nós ao especificar a área do documento, linha de pesquisa, tópico e assunto. Mais adiante, o usuário necessita indicar qual o algoritmo inicial que será usado no passo seguinte;

2. a) O módulo "Extrator de características" aplica o algoritmo inicial no documento e uma lista de palavras-chave relacionadas a ele é gerada ($docKeywords$);

3. a) O par $\langle chave, valor \rangle$ é inserido na DHT. A $chave$ é composta pela concatenação dos atributos que definem a comunidade de nós ($area$, $linha$, $topico$ e $assunto$). O $valor$ é composto pela concatenação do identificador do nó ($peerId$), nome do documento ($docId$), identificador do algoritmo inicial ($algId$), e a lista de palavras-chave ($docKeywords$). A Tabela 4.1 é uma representação do par $\langle chave, valor \rangle$ inseridos na

DHT. A partir deste passo, o documento é disponibilizado na rede P2P, e pode ser encontrado pelos outros nós utilizando o mecanismo de busca disponibilizado pela DHT.

chave	valor
$area@linha@topico@assunto$	$peerId@docId@docKeywords$

Tabela 4.1: Estrutura de armazenamento utilizada na DHT na abordagem centralizada

4.2.2 Verificação de documentos

A verificação de documentos permite a um usuário verificar se um documento é plágio. O Peer 1 da Figura 4.1 apresenta passo a passo o processo de verificação e na seqüência, cada passo é explanado.

1. b) O usuário define a comunidade de nós que quer realizar a busca por documentos plagiados. Ele realiza este procedimento especificando a área, linha de pesquisa, tópico e assunto do documento. O usuário também deve indicar qual é o algoritmo inicial que será utilizado no passo 3b;

2. b) O PeerDetect utiliza os atributos dos documentos para formar a chave $area@linha@topico@assunto$ e executa uma busca na DHT utilizando esta chave. Esta busca é feita utilizando a interface "Publisher/Searcher" da DHT. Ela resulta em uma lista ($docList$) indicando todos os documentos na mesma comunidade de nós. Cada item nesta lista possui a seguinte informação: o identificador do nó $peerId$, o nome do documento $docId$, identificador do algoritmo inicial ($algId$) e a lista de palavras-chave ($docKeywords$) relacionadas ao documento;

3. b) O módulo "Extrator de características" aplica o algoritmo inicial ao documento sendo verificado e gera uma lista de palavras-chave sobre o documento ($docCheckKeywords$). O PeerDetect utiliza estas palavras-chave para determinar quais documentos da $docList$ são similares. Este processo consiste basicamente em comparar as $docCheckKeywords$ com as $docKeywords$ para cada item na $docList$. Todos os documentos cujas $docKeywords$ sejam similares em 80% são colocados em outra lista ($docList2$);

4. b) O módulo "Document Downloader" faz com que o PeerDetect receba todos os documentos indicados na $docList2$, utilizando a rede JXTA.

5. b) O módulo “Analisador de características” executa uma comparação de todos os documentos recebidos, utilizando um algoritmo mais eficiente a fim de determinar o grau de similaridade do documento sendo verificado. Uma lista ($plagiarismList$) com os documentos mais similares é gerada.

6. b) Finalmente, o PeerDetect informa ao usuário a $plagiarismList$. Posteriormente, o usuário pode executar uma verificação manual dos documentos na lista para certificar-se da veracidade dos resultados.

4.3 Abordagem descentralizada

A abordagem descentralizada do PeerDetect utilizando as vantagens tanto da *Internet* quanto das redes P2P. Em relação à *Internet*, ela é a fonte de dados utilizada para comparar os documentos submetidos para verificação. A *Internet* possui inúmeros documentos, e mecanismos de indexação como o *Google* são capazes de varrer uma grande porção destas informações em poucos segundos.

O PeerDetect utiliza o *Google* para procurar por documentos com frases similares. Entretanto, um documento muito extenso demoraria muito para ser verificado caso fosse procurado frase por frase. Isso significaria uma grande quantidade de trabalho para um único computador, e é por este motivo que a utilização de redes P2P é proposta. As partes de documentos a serem procuradas são distribuídas entre os nós participantes a fim de dividir a carga de trabalho.

Todos os nós na rede comunicam-se através da DHT, a qual é utilizada como um “quadro de mensagens”, onde todo nó publica seu estados e suas tarefas. A abordagem descentralizada consiste em dois processos diferentes: a publicação de tarefas e a verificação de tarefas.

4.3.1 Publicação de tarefas

Este processo é executado pelo nó cujo dono submeteu um documento para busca, chamado de $ownerpeer$. Este nó, primeiramente calcula o tamanho do documento submetido e procura na DHT quantos nós estão disponíveis para trabalhar. Com esta informação o nó distribui as tarefas para cada nó. As tarefas consistem simplesmente em partes do texto. O nó líder então aguarda pelas análises.

1. a) Usuário submete um documento para ser analisado ao nó líder (nó 0). Este documento é tratado pelo módulo “Gerenciador de tarefas”;

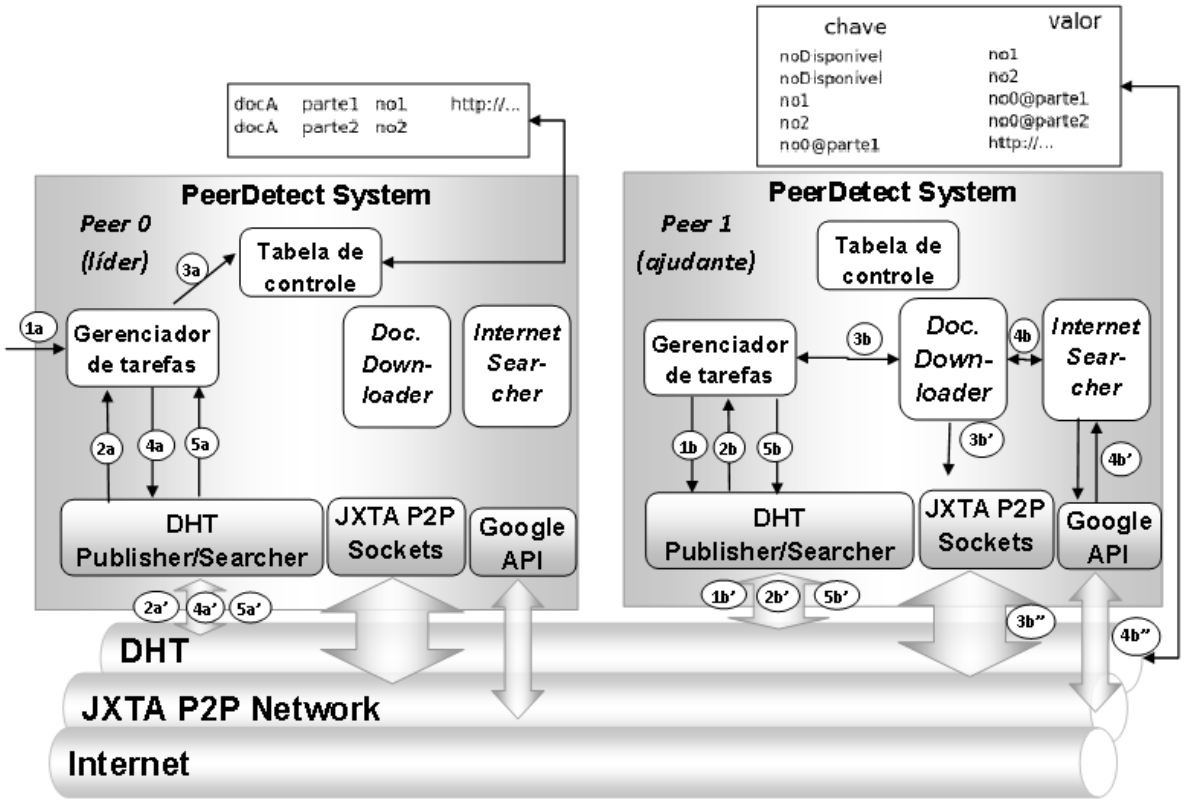

Figura 4.2: Publicação (Peer 0) e verificação (Peer 1) de tarefas na abordagem descentralizada

2. a) O nó líder procura na DHT por nós ajudantes disponíveis. Esta busca é feita procurando-se pela chave $noDisponivel$ na DHT (linha 1 da Tabela 4.2). A DHT retorna a lista de identificadores dos nós;

3. a) O nó líder distribui partes do documentos para cada nó disponível e atualiza a tabela de controle com esta informação. A distribuição é feita simplesmente dividindo o número de páginas pelo número de nós disponíveis;

4. a) O nó líder publica as informações das tarefas na DHT utilizando a estrutura apresentada na linha 2 da Tabela 4.2 e a interface "DHT Publisher";

5. a) O nó líder procura periodicamente por respostas utilizando a interface "DHT Searcher". Esse procedimento é realizado procurando-se pela concatenação do seu identificador ($idDono$) e o identificador da parte do documento publicado ($parteDoc$) (linha 3 da Tabela 4.2). Em caso de respostas, atualiza-se a tabela de controle.

O PeerDetect utiliza *Time to live* (TTL) ao publicar uma informação. O TTL é a quantidade de tempo que uma informação fica disponível na DHT antes de ela expirar e ser removida.

	chave	valor
1	$noDisponivel$	$idNo$
2	$idNoDisponivel$	$idDono@parteDoc$
3	$idDono@parteDoc$	$resultado$

Tabela 4.2: Estrutura de armazenamento utilizado na DHT na abordagem descentralizada

A informação das tarefas a serem realizadas, publicadas pelo nó líder, possuem um TTL médio. Desta forma, quando um nó ajudante que possuir tarefas atribuídas a ele sair da rede, estas tarefas não ficarão demasiado tempo tem serem resolvidas. Assim que o TTL expirar, o nó líder republica essas tarefs assinalando elas a um nó ajudante disponível.

O nó líder utiliza uma tabela local chamada $TabeladeControle$ para manusear a informação das tarefas que estão sendo realizadas. Quando o nó líder distribui as tarefas entre os nós ajudantes disponíveis, ele atualiza a tabela a fim de manter-se informado de qual nó trabalhará em qual parte do documento. Após todos os nós ajudantes terem retornado os resultados, o nó líder atualiza a tabela com esta informação.

$idDono$	$parteDoc$	$idNoDisponivel$	$resultado$

Tabela 4.3: Tabela de controle do nó líder

4.3.2 Verificação de tarefas

Os nós ajudantes, cujos donos não submeteram nenhum documento, possuem outro papel. A primeira ação é publicar na DHT o seu estado: $noDisponivel$ (estrutura apresentada na linha 1 da Tabela 4.2). Esta informação tem um TTL baixo, já que ela deve ser o mais precisa possível para que o nó líder, ao receber a lista de nós disponíveis, não assinale uma tarefa a um nó que agora está ocupado. Para que isso ocorra as informações de estado expiram rapidamente e necessitam, caso o nó ainda esteja disponível, ser submetidas novamente. Após a realização deste processo, os nós ajudantes periodicamente buscam a DHT, procurando por tarefas que o nó líder possa ter submetido. Se o nó encontra seu próprio identificador na DHT, ele recebe a tarefa utilizando a estrutura JXTA e a executa, procurando no *Google* por frases. Caso a busca traga resultados, o nó publica a informação utilizando a estrutura apresentada na linha 3 da

Tabela 4.2, a qual possui tem um TTL alto, já que não há motivo para que estas informações seram retiradas a curto prazo.

O Peer 1 da Figura 4.2 apresenta o processo de verificação de tarefas passo a passo e em seguida, cada um deles é explanado.

1. b) Ao entrar na rede P2P, o nó ajudante (nó 1) faz uma auto publicação na DHT utilizando a interface "DHT Publisher", informando que está disponível. Ele realiza este processo publicando a palavra-chave $noDisponivel$ juntamente com seu identificador ($idNoDisponivel$) (linha 1 da Tabela 4.2);

2. b) O nó ajudante periodicamente procura por seu $idNoDisponivel$ na DHT. Ele procura por tarefas que um nó líder (nó 0) possa o ter atribuído;

3. b) Quando houver uma tarefa, o nó faz o *download* das partes, via JXTA;

4. b) O nó ajudante realiza a busca das frases na Internet utilizando a API de acesso ao Google. O usuário pode definir qual o tamanho (quantas palavras) da frase. A busca é feita utilizando-se aspas, o que significa que o Google irá procurar por frases exatas;

5. b) Os resultados são publicados utilizando a estrutura apresentada na linha 3 da Tabela 4.2.

4.4 Conclusão

Foram apresentadas as abordagens propostas neste estudo. São duas propostas que abordam o combate ao plágio de forma diferente. A primeira, utiliza a rede P2P como fonte de dados e como mecanismo de publicação e busca. A segunda, utiliza a *Internet* como fonte de dados e as redes P2P para distribuir as tarefas de buscas.

Pode-se afirmar que a primeira abordagem encontra plágios em documentos privados, disponibilizados por usuários do sistema, e a segunda, encontra plágios mais evidentes, copiados diretamente da *Internet*. Ambas retornam ao usuário um grau de similaridade que deve ser conferido manualmente.

No próximo capítulo, serão apresentados alguns detalhes da implementação, bem como os resultados dos testes realizados.

Capítulo 5

Implementação e avaliação dos resultados

5.1 Introdução

As abordagens apresentadas no Capítulo 4 foram implementadas em um protótipo utilizando como base a proposta apresentada em [54] e a linguagem *Java*. Foram realizados diversos testes para verificar o desempenho da proposta. Este capítulo trata das tecnologias utilizadas na implementação e o resultado desses testes.

5.2 Detalhes da implementação

O intuito foi desenvolver um protótipo funcional para verificar as abordagens apresentadas. Uma boa ferramenta de detecção de plágio deve ser, além de tudo, útil, o que significa que a ferramenta deve ao menos apontar para a direção correta, para que um especialista obtenha a menor quantidade de falsos positivos (indicação de plágio quando na verdade não é). É preciso também apresentar os resultados em um tempo aceitável. Em outras palavras, a ferramenta deve funcionar de forma correta em termos de acertos e eficiente no que se diz respeito ao tempo.

- *Linguagem de Programação* – Como a proposta aqui apresentada é embasada na arquitetura vista em [54], utilizou-se a mesma linguagem de programação – *Java*. Por tratar-se de uma linguagem multiplataforma, o sistema pode ser executado tanto em *Linux* quanto em *Windows*.

- *Mecanismo de busca* – A princípio, o sistema utilizava o *Yahoo* [55] como mecanismo de busca para a abordagem descentralizada. Contudo, a busca desta empresa não é realizada

em documentos em formato PDF. Este fato diminuiria consideravelmente a quantidade de resultados, optou-se por utilizar o *Google*. Foi utilizada a *Aplication Programming Language* (API) *Java* disponibilizada pela empresa para desenvolvimento.

- *DHT* – A DHT empregada foi a *OpenDHT*, apresentado na Seção 3.5.2. O serviço disponibiliza APIs para programação em *Java*, o que facilitou a sua utilização.

- *PlanetLab* – O *PlanetLab* [56] é um grupo de computadores disponibilizados a fim de serem utilizados como plataforma de testes de diversas aplicações, entre elas a DHT. Foi utilizada pelo PeerDetect como base de testes.

- *Formato de Arquivo* – O formato de arquivo dos documentos escritos em linguagem natural é o PDF. Para tornar-se possível a realização dessa leitura foi necessário utilizar a API PDFBox [57] (para manusear documentos deste tipo) e o FontBox (para tratar caracteres diferentes, devido à internacionalização de documentos).

- *Máquinas de testes* – Os testes foram realizados utilizando um AMD Turion 1.6 GHz com 1 GB ram como nó 0 e dois AMD Athlon XP 2.6 GHz com 512 MB ram como nós 1 e 2 em uma rede a 100Mb/s.

5.2.1 Testes da abordagem centralizada

Tempo de publicação

A primeira avaliação da abordagem centralizada foi o *Tempo de publicação*, o que consiste no tempo de extração do algoritmo inicial (passo 2a da Figura 4.1) e o seu tempo de publicação (passo 3a da Figura 4.1).

Os testes foram realizados com a utilização de documentos em formato PDF com diferentes números de páginas. Esses documentos possuem características apresentadas na Tabela 5.1. O algoritmo inicial utilizado foi o algoritmo apresentado em [27] (Seção 2.4.3). Este algoritmo detecta as chamadas *hapax legomena*. Os resultados são apresentados na Figura 5.1.

Pode-se visualizar que o tempo de extração é relativo ao tamanho do documento, porém, isso não está diretamente relacionado ao tempo de publicação, visto que este tempo depende da quantidade de *hapax legomena* encontradas no documento. Considera-se satisfatório o tempo de seis segundos para extração e catorze para a publicação de um documento de 4.7MB.

	Páginas	*hapax legomena*	Tamanho
Documento 1	1	77	38KB
Documento 2	6	655	83KB
Documento 3	13	944	450KB
Documento 4	90	473	733KB
Documento 5	93	1590	4.7MB

Tabela 5.1: Documentos utilizados para testes na abordagem centralizada

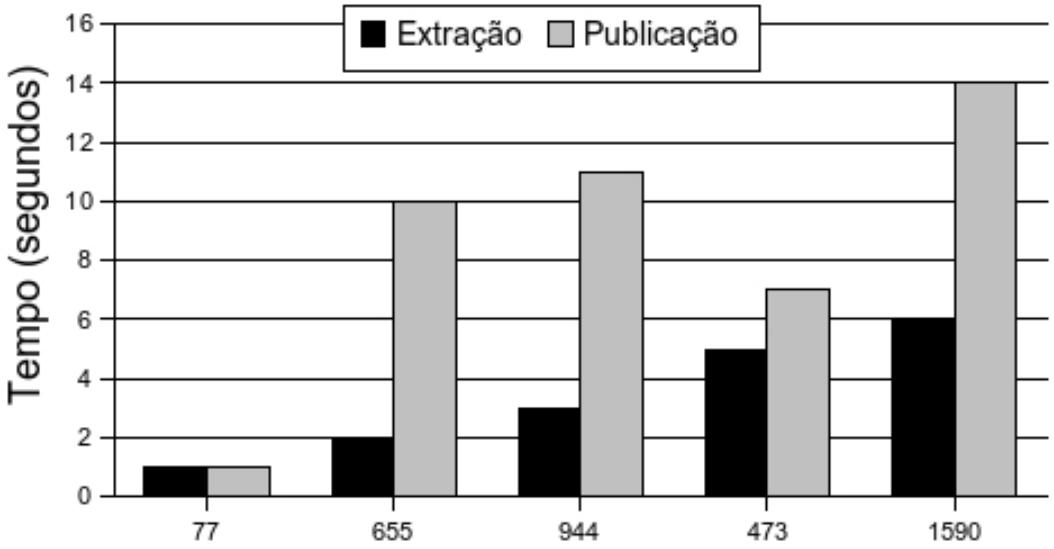

Figura 5.1: Tempo de publicação total na abordagem centralizada por Hapax Legomina

Tempo de verificação

Outra avaliação executada foi relacionada ao tempo de verificação. Esse tempo, consiste na busca (passo 2b da Figura 4.1), *download* (passo 4b da Figura 4.1) e a aplicação de um algoritmo mais eficiente (passo 5b da Figura 4.1), que foi a utilização de uma adaptação do SIM (Seção 2.4). Os resultados estão apresentados na Figura 5.2.

O tempo de busca permanece basicamente o mesmo em todos os casos. O tempo de *download* varia de acordo com o tamanho do documento. Contudo, dois minutos para um documento de 4.7 MB pode ser considerado um tempo aceitável. A execução do algoritmo eficiente foi baixa, dada a característica simples do algoritmo. Porém, acredita-se que ao utilizar outros tipos de algoritmos mais trabalhosos, o tempo aumente.

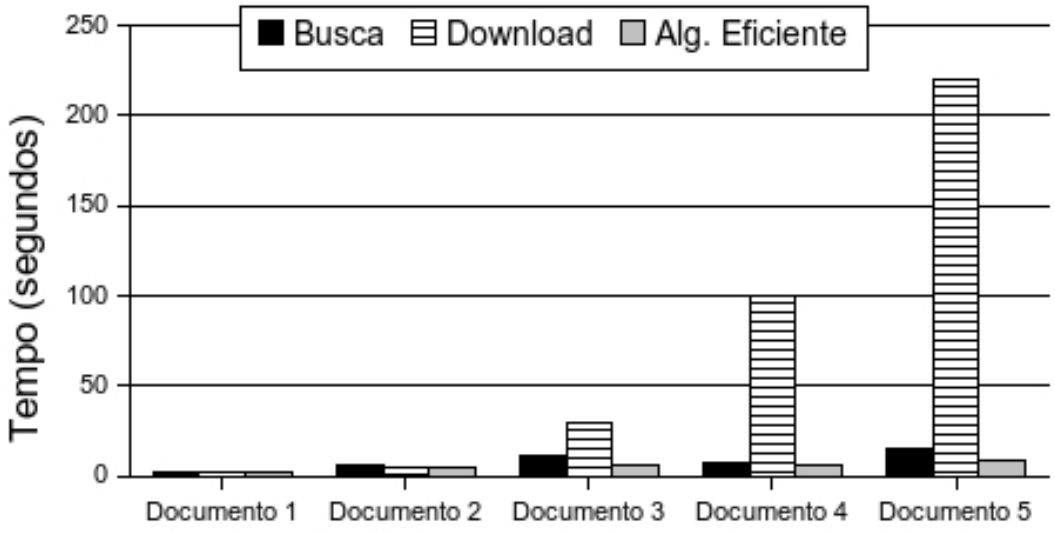

Figura 5.2: Tempo de verificação total na abordagem centralizada

Percentual de respostas corretas

Finalmente, foi testado o percentual de respostas corretas. Esta última avaliação demonstra a eficiência da detecção de plágio. Para tal, foram copiadas partes exatas de um documento em outro. Este processo foi realizado com dez pares de documentos. O PeerDetect detectou todas as cópias, entretanto, quanto menor a parte copiada para outro documento, menor o grau de certeza apontado pelo PeerDetect. Os resultados estão apresentados na Tabela 5.2. Os graus de similaridade apresentados podem ser considerados baixos, porém vale ressaltar que o menor indício de similaridade deve ser conferido.

5.2.2 Testes da abordagem descentralizada

Na abordagem descentralizada, foram realizados testes para medir o tempo total de execução e percentual de acertos (respostas corretas) dos resultados. A definição de tamanho da frase pelo usuário é um fator importante, visto que quanto menor o tamanho da frase, maior a quantidade de falso positivos encontrados. Testes demonstraram que tamanho de frase com quinze palavras é suficiente para resultados satisfatórios.

A Tabela 5.3 apresenta o número de buscas executadas por um único computador que utilizou o PeerDetect com dois e três nós. Foram utilizados três documentos para os testes. A quantidade de buscas consiste no número de frases no documento com quinze palavras. Devido

tamanho documento1 (palavras)	tamanho documento2 (palavras)	Nr. de palavras copiadas	Similaridade %
3661	2217	10	0
3661	2217	50	4
12207	6094	100	2
8048	6094	400	10
12207	16094	2000	20
3661	16094	10	0
16094	2217	50	0
12207	8048	100	2
8048	3661	400	10
16094	16094	2000	100

Tabela 5.2: Testes de grau de similaridade na abordagem centralizada

à divisão do número de páginas pelo número de nós, o número de buscas por nó aumenta ou diminui dependendo de quantas frases de quinze palavras há na página. Os valores na Tabela 5.3 são um média do número de buscas feitas por cada nó.

Tamanho do documento (páginas)	Computador único	2 nós	3 nós
13	42	28	22
22	46	23	13
93	390	174	82

Tabela 5.3: Número de buscas (frases) por nó

A Figura 5.3 apresenta o tempo que levou-se para que fosse concluída a busca em cada caso. Este tempo é calculado desde o momento em que o nó principal recebeu o documento até o momento em que ele recebe as respostas dos nós assistentes e as publica ao usuário. Quanto maior o documento, aumenta-se o tempo e o número de buscas. Entretanto, a adição de um nó assistente auxilia na diminuição do tempo total aproximadamente em cinqüenta por cento. O tempo está relacionado ao número de buscas que cada nó precisa executar. O que significa que

Formato da frase	Inteira (42 palavras)	26 palavras	16 palavras	13 palavras	9 palavras	5 palavras
Com aspas	127	18.500	44.000	47.000	210.000	1.720.000
Sem aspas	1	1	1	1	4	6

Tabela 5.4: Resultados da busca com e sem aspas

caso o documento seja pequeno, não haverá grandes vantagens em possuir muitos nós dividindo a tarefa.

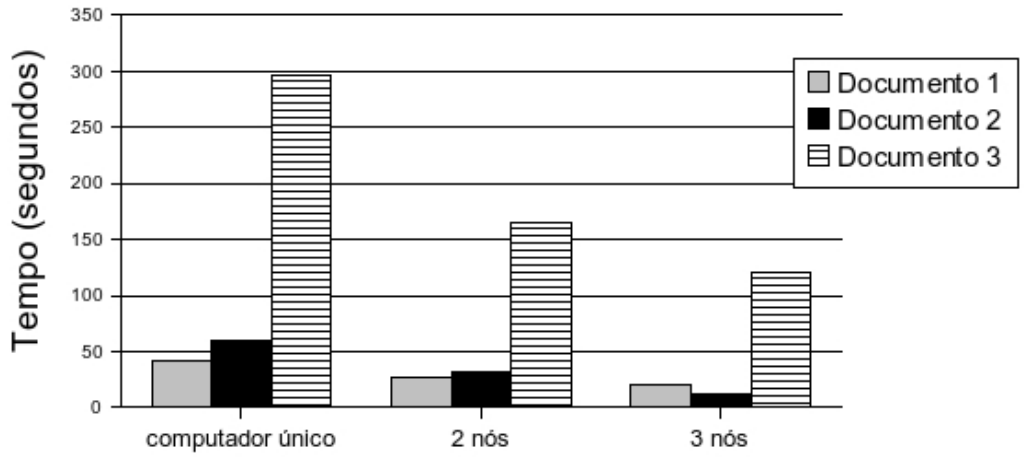

Figura 5.3: Tempo total de execução da abordagem descentralizada

Pode-se visualizar que no momento em que há dois nós em vez de somente um, o tempo não é simplesmente a metade, visto que há o tempo de publicação das tarefas e suas respostas (*overhead*).

Determinando o formato e tamanho da busca

Vamos supor que o autor do plágio tenha copiado a frase . Procurando o Google por esta frase, os resultados encontrados estão apresentados na Tabela 5.4. Entre estes, alguns deles não possuem nenhuma relação com a busca, apenas possuem suas palavras pelo texto. Estes resultados não podem ser considerados plágios. Os resultados na qual a frase aparece extamente da mesma forma que na busca caracteriza um plágio. A tabela 5.5 apresenta a porcentagem de acerto entre os resultados apresentados. Pode ser visto que o uso de aspas garante um 100% de acerto, utilizando um tamanho de frase razoável, por exemplo 13 palavras. O teste consiste em usar 10 frases diferentes.

Formato da frase	Inteira (42 palavras)	26 palavras	16 palavras	13 palavras	9 palavras	5 palavras
Com aspas	0.7	0.005	0.002	0.002	0.001	0.0004
Sem aspas	100	100	100	100	25	17

Tabela 5.5: Relevância dos resultados com e sem aspas

Baseado em este tamanho de frase, duas abordagens foram testadas mais a fundo para a busca Descentralizada. A primeira utiliza aspas na busca e a segunda não.

A primeira consiste em selecionar as cinco palavras que são mais freqüentes no texto e também as cinco palavras palavras menos freqüentes no teto. As frases na qual estas palavras faziam parte eram selecionadas e então procuradas no Google. Para ilustrar esta busca, em um dado documento as palavras mais freqüentes foram:

- império
- patrícios
- poder
- Roma
- para

E algumas das frases na qual a primeira palavra faz parte são:

- "ira do patriciado. Mas o império etrusco já estava em decadência,"
- "O Egito foi o último império Mediterrâneo a ser conquistado, sendo"
- "restabeleceu a paz em todo império. Distribuiu terras aos soldados, obrigou"

Estas frases foram todas procuradas no Google utilizando aspas. Buscas que retornaram mais de 1.000 resultados foram ignoradas. Os resultados estão apresentados na Tabela 5.6.

Há basicamente nenhuma diferença em utilizar frases com as palavras mais freqüentes do que com as menos freqüentes. Todas elas resultam em uma média de um resultado por busca. Assim, o uso de aspas é por si só determintante. Os documentos um e cinco, como são trabalhos de assuntos comuns (um é sobre dicas de Windows XP e o outro sobre o filme Titanic), tiveram uma grande quantidade de resultados.

	Mais freqüentes	Menos freqüentes
Documento1	178 buscas com 9644 resultados no total - média de 54 resultados por busca: 6	3 buscas com 518 resultados no total - média de 172 resultados por busca
Documento2	44 buscas com 50 resultados no total - média de 1 resultado por busca	17 buscas com 21 resultados no total - média de 1 resultado por busca
Documento3	23 buscas com 71 resultados no total - média de 3 resultados por busca	1 busca com 2 resultados no total - média de 2 resultados por busca
Documento4	95 buscas com 116 resultados no total - média de 1 resultado por busca	4 buscas com 4 resultados no total - média de 1 resultado por busca
Documento5	6 buscas com 1655 resultados no total - média de 275 resultados por busca	0 buscas com 0 resultados no total
Documento6	119 buscas com 128 resultados no total - média de 1 resultado por busca	5 buscas com 6 resultados total - média de 1 resultado por busca

Tabela 5.6: Resultado das buscas utilizando aspas

	Mais freqüentes	Menos freqüentes
Documento1	25.500	7
Documento2	23.300	0
Documento3	281	0
Documento4	146.000	0
Documento5	3	0
Documento6	527	1

Tabela 5.7: Resultados das buscas sem o uso de aspas

Os testes sem a utilização de aspas consiste em selecionar as trinta palavras mais comuns e as trinta menos comuns no text. Estas palavras foram procuradas no Google sem aspas. Os resultados estão apresentados na Tabela 5.7.

Os resultados demonstram que utilizando as palavras mais freqüentes temos um grande número de resultados, dificultando a inspeção manual. Isto pode ser explicado pelo simples fato que as palavras comuns são mais prováveis de serem encontradas em uma grande quantidade de documentos. Se não houver filtros de restrições (como as aspas), os resultados podem ser vatos. Utilizando as palavras menos freqüentes não há certeza de resultados, variando entre zero e poucos, provavelmente devido ao mecanismo de indexação do Google. Porém, a relevância dos resultados, caso exista, é alta. Assim, acreditamos que o não uso de aspas não é uma maneira confiável de se encontrar plágio.

5.3 Teste verídico

O primeiro teste verídico do PeerDetect foi realizado com uma professora na mesma universidade em que o sistema foi criado. Um dos autores tomou conhecimento de que ela havia descoberto que uma de suas alunas havia copiado um texto da *Internet* sem qualquer modificação. A professora evidenciou por conta própria duas fontes diferentes. O autor então, solicitou a ela o empréstimo desse trabalho para que fosse testado com a utilização da abordagem descentralizada do PeerDetect. Os resultados foram muito positivos e o sistema não descobriu somente as duas fontes encontradas pela professora, como também detectou outras duas fontes que a aluna havia utilizado para copiar outra parte do texto. A professora ficou muito satisfeita com o sistema, e espera que ele seja implantado como ferramenta oficial da universidade.

5.4 Conclusão

Neste capítulo foram apresentadas as tecnologias utilizadas para implementação da proposta. Foram realizados testes tanto na abordagem centralizada quanto na descentralizada, onde ambos obtiveram resultados satisfatórios.

Capítulo 6

Conclusão e Trabalhos Futuros

O principal objetivo deste trabalho é oferecer um sistema de detecção de plágio eficiente, o qual oferece duas abordagens de detecção de plágio. A abordagem centralizada consiste em usar documentos utilizados por documentos em redes P2P como fonte de dados para procurar por cópias fraudulentas. A abordagem descentralizada utiliza a *Internet* como fonte de dados, porém também utiliza a rede P2P para balancear o trabalho de busca. Ambos os métodos utilizam um conjunto de tecnologias e mecanismos, como DHT e JXTA. A abordagem centralizada possui a vantagem de procurar em documentos indisponíveis na *Internet*, enquanto a abordagem descentralizada possui a vantagem de fornecer uma resposta rápida para plágios evidentes.

O PeerDetect será lançado sob a licença *GNU Public License* (GPL), no momento em que houver um versão estável com um interface amigável. Os autores estão dispostos a usar o PeerDetect como ferramenta oficial de detecção de plágio na Universidade, onde a cultura anti-plágio não é difundida e muitos professores desconhecem formas de como detectar plágio. A Universidade PUCPR [58] utiliza um sistema colaborativo de ensino denominado Eureka [59], através do qual professores e alunos possuem aulas virtuais. O sistema provê diversas funcionalidades, como entrega de trabalhos *online*, o que permite aos alunos enviar seus trabalhos via navegador e aos professores que os recebam também via navegador. Isto considera-se uma boa fonte de dados para o PeerDetect. Desta forma, caso o PeerDetect seja utilizado em sistemas similares em outras universidades, cada uma delas se tornaria um nó na rede P2P disponibilizando seus documentos.

Alguns trabalhos futuros podem ser realizados para melhoria e complementos da arquitetura proposta, como:

- Integração com diversos algoritmos de detecção de plágio, na qual o usuário escolheria via interface gráfica qual opção ele deseja utilizar;

- Integração de outros mecanismos de buscas além do *Google*, possibilitando ao usuário a escolha;

- Mecanismos de controle de acesso, para impossibilitar o acesso a base de dados por usuários maliciosos;

- Em [5] é mencionada uma preocupação com a propriedade intelectual dos documentos acadêmicos. A preocupação diz respeito ao fato do autor do documento (o aluno por exemplo) não autorizar a distribuição do documento para outras pessoas que não o seu professor. Ao fazer isto, dispondo do documento e enviando-o para outros servidores para que seja comparado com outros documentos como a maioria dos sistemas convencionais faz, estaria-se violando a propriedade intelectual do aluno. Dada esta preocupação, é possível adicionar a possibilidade do usuário escolher não passar a segunda fase na busca descentralizada. Desta forma o documento não seria copiado para uma análise mais eficiente, e as informações finais se baseariam somente nos resultados do primeiro algoritmo.

Referências Bibliográficas

[1] Alan Parker and James O. Hamblen. Computer algorithms for plagiarism detection. *IEEE Transactions on Education*, 32(2):94–99, 1989.

[2] www.turnitin.com, 2008.

[3] Paul Clough. Plagiarism in natural and programming languages: an overview of current tools and technologies. 2000.

[4] CHECK: A document plagiarism detection system. dscam: Finding document copies accross multiple databases. *In Proceedings of ACM Symposium for Applied Computing*, 1997.

[5] Ben Wellington Elif Tosun. Detecit: a peer-to-peer plagiarism detection system. *Corant Institute, NYU*, 2003.

[6] University of Athens ATC Engineering, Lancaster University. Comprehensive survey of contemporary p2p technology.

[7] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.

[8] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218, 2001.

[9] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. Technical Report TR-00-010, Berkeley, CA, 2000.

[10] Plaxton C . G . Rajaraman R . Andricha A . W. Accessing nearby copies of replicated objects in a distributed environment. *9th Annual Symp. on Parallel Algorithms and Architectures - ACM*, pages 311–320, 1997.

[11] dks.sics.se, 2008.

[12] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

[13] Phillip G. Armour. The five orders of ignorance. *Communications of the ACM*, 43(10):17–20, 2000.

[14] Kate Zernike. With student cheating on the rise, more colleges are turning to honor codes. *New York Times*, Novembro:10–11, 2002.

[15] www.google.com, 2008. Google Search Engine.

[16] www.zemoleza.com.br, 2008.

[17] www.10emtudo.com.br, 2008.

[18] www.canexus.com/eve/index.shtml, 2008.

[19] Xin Chen Brent Francia Ming Li Brian Mckinon Amit Seker. Shared information and program plagiarism detection. 2003.

[20] Whale G. Identification of program similarity in large populations. *The Computer Journal*, 33(2):140–146, 1990.

[21] J. A. e S. K. Robinson Faidhi. An empirical approach for detectin program similarity within a university programming environment. *Computers and Education*, 11(1):11–19, 1987.

[22] Kristina L. Verco and Michael J. Wise. Software for detecting suspected plagiarism: Comparing structure and attribute-counting systems. *1st. Australian Conference on Computer Science Education*, 1996.

[23] Michael Luck Mike Joy. Plagiarism in programming assingments. *IEEE Transaciotns on Education*, 42(2):129–133, 1999.

[24] Tony McEnery and Mike Oakes. Authorship identification and computational stylometry. *Internal report, Department of Linguistics, Lancaster University*, 1998.

[25] Asbjorn Aakjaer Philip Sallis and Stephen MacDonell. Software forensics: old methods for a new science, in proceedings of software engineering: Education and practice (se:e&p'96). *IEEE Computer Society Press*, 1996.

[26] The university of Kentucky. Plagiarism: definitions, examples and penalties. *http://www.chem.uky.edu/Courses/common/plagiarism.html*, 1998.

[27] Susan Finlay. Copycatch. *Masters Dissertation, University of Birmingham*, 1999.

[28] Paul Clough. Identifying re-use between the press association and newspapers of the british press. *Department of Computer Science, University of Sheffield, Internal Report*, 2000.

[29] Wilson Taylor. Cloze procedure: A news tool for measuring readibility. *Journalism Quarterly*, 1953.

[30] Michael J. Wise. Yap3: improved detection of similarities in computer programs and other texts. *SIGCSE'96, Philadelphia, USA*, 1996.

[31] Michael J. Wise. String similarity via greedy string tiling and running karp-rabin matching. *Dept. of CS, University of Sydney*, 1993.

[32] Charles E. Leiserson Thomas H. Cormen and Ronald L. Rivest. Introduction to algorithms. *MIT Press*, 1990.

[33] J. Davis S. Brin and H. Garcia-Molina. Copy detection mechanisms for digital documents. *Proceedings of the ACM SIGMOD Annual Conference*, 1995.

[34] Udi Manber. Finding similar files in a large file system. *USENIX*, 1994.

[35] N. Shivakumar and H. Garcia-Molina. Scam: a copy detection mechanism for digital documents. *Proceedings of 2nd International Conference in Theory and Practice of Digital Libraries (DL'95)*, 1994.

[36] N. Shivakumar and H. Garcia-Molina. Building a scaleable and accurate copy detection mechanism. *Proceedings of 1st ACM Conference on Digital Libraries (DL'96)*, 1996.

[37] H. Garcia Molina Luis Gravano Narayanan Shivakumar. dscam: Finding document copies accross multiple databases. *Stanforf University*.

[38] Clip2. The gnutella protocol specification. 2000.

[39] João Rocha Marco Domingues Arthur Callado Eduardo Souto Guthemberg Silvestre Carlos Kamienski Djamel Sadok. Peer-to-peer: Computação colaborativa na internet.

[40] Sean Fanning. Napster: a p2p file sharing. 1999.

[41] www.microsoft.com, 2008.

[42] www.skype.com, 2008.

[43] www.lotus.com, 2008.

[44] office.microsoft.com/en-us/groove/FX100487641033.aspx, 2008.

[45] boinc.berkeley.edu, 2008.

[46] B. Traversat, M. Abdelaziz, M. Duigou, J. Hugly, E. Pouyoul, and B. Yeager. *Project JXTA Virtual Network*, 2002.

[47] Hari Balakrishnan and et al. *Looking up Data in P2P Systems*.

[48] Moni Naor and Udi Wieder. Novel architectures for p2p applications: The continuos-discrete approach. *Proc. SPAA*, 2003.

[49] Gurmeet Singh Manku. Dipsea: A modular distributed hash table. *PhD Thesis*, 2004.

[50] F. Dabek M. F. Kaashoek D. Karger R. Morris and I. Stoica. Wide-area cooperative storage with cfs. *In SOSP*, 2001.

[51] Mahalingam Tang, Xu. Psearch: Information retrieval in structured overlays. *ACM SIGCOMM Computer Communication Review*, 33(1), 2003.

[52] García P. e Skarmeta A.F. Pairot C. Dermi: A new distributed hash table-based middleware framework. *IEEE Internet Computing*, 8(3):74–84, 2004.

[53] John Kubiatowicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.

[54] Heverson Borba Ribeiro, Lau Cheuk Lung, Altair Olivo Santin, and Neander Larsen Brisola. Web2peer: A peer-to-peer infrastructure for publishing/locating/replicating web pages on internet. In *ISADS '07: Proceedings of the Eighth International Symposium on Autonomous Decentralized Systems*, pages 421–428, Washington, DC, USA, 2007. IEEE Computer Society.

[55] www.yahoo.com, 2008.

[56] www.planet-lab.org, 2008.

[57] www.pdfbox.org, 2008.

[58] www.pucpr.br, 2008.

[59] eureka.pucpr.br, 2008.