UM CRAWLER PEER-TO-PEER BASEADO EM AGENTES

Marcelo de Mattos Mayworm

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Geraldo Bonorino Xexéo, D. Sc.

Prof. Jano Moreira de Souza, Ph.D.

Prof. Geraldo Zimbrão da Silva, D. Sc.

Prof. Renato Fontoura de Gusmão Cerqueira, Ph.D.

RIO DE JANEIRO, RJ - BRASIL

SETEMBRO DE 2007

MAYWORM, MARCELO

Um crawlers peer-to-peer baseado em agentes. [Rio de Janeiro] 2007

XIII, 130 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2007)

Dissertação - Universidade Federal do Rio de Janeiro, COPPE

1. Busca e Recuperação de Informação
2. Web Crawlers
3. Agentes de Software
4. Máquinas de Busca

I. COPPE/UFRJ II. Título série

À minha ***mãe***, exemplo de vida, inspiração espiritual para os que a cercam, sensibilidade natural, o meu muito obrigado por ter sempre apoiado as minhas iniciativas e por termos juntos vencido tantos obstáculos ...

Ao meu ***pai***, que tanto me ensinou, que tanto me orientou, que com certeza também foi o responsável pelas lições da vida, obrigado pela força e saiba que o tenho eternamente em meu coração ...

Ao meu ***irmão***, por tudo o que representa e pelo exemplo de amizade e alegria de vida ...

A todos vocês dedico este trabalho.

A você ***Dani***, por tudo o que significa em minha vida, alegrando-a, energizando-a; por sua pureza, preciosidade, ternura e amor; por tanta força e positividade depositadas nesse meu trabalho; com carinho, o meu eterno agradecimento e dedicação.

Agradecimentos

Todos, com certeza, tiveram um papel fundamental para que este trabalho se concretizasse, sendo muitos os nomes que deveriam ser relacionados para não incorrer em injustiças.

Devo contudo lembrar algumas pessoas que, com certeza, influenciaram muito o meu desenvolvimento acadêmico.

Carla Valle, amiga e professora, sempre apaixonada pelo o que faz, mostrando e ensinando que o aprendizado não tem limites, e seremos eternos estudiosos, pelo resto de nossas vidas.

Geraldo Xexéo, orientador e professor. Sem dúvida poucos tem a experiência que esse mestre, com generosidade, compartilha, divulgando o seu conhecimento, ensinando e guiando.

Jano Moreira de Souza, chefe e professor da linha de Banco de Dados, uma pessoa convicta, que sempre compartilhou entusiasticamente de várias idéias, e como fruto disto, nasceram vários trabalhos importantes.

Dani, esposa e amiga, obrigado por toda paciência e ajuda neste projeto de vida.

Gil, Leo, Mutaleci e Barros, obrigado pela energia e tempo investidos neste meu trabalho.

Agradeço, também, aos muitos amigos temporariamente abandonados durante o tempo que vivi este meu trabalho, sei que vocês têm a consciência das prioridades que coloquei em minha vida.

Por fim, aos professores que tive durante minha vida, com certeza cada um de vocês contribuíram com a viabilização e realização deste sonho.

A todos, o meu muito obrigado.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

UM CRAWLER PEER-TO-PEER BASEADO EM AGENTES

Marcelo de Mattos Mayworm

Setembro/2007

Orientador: Geraldo Bonorino Xexéo

Programa: Engenharia de Sistemas e Computação

Existem várias iniciativas na comunidade científica para produzir *crawlers* mais eficientes, contudo, ainda persistem os problemas quanto à utilização de recursos computacionais e a identificação de servidores hospedeiros de páginas Web próximos aos *crawlers*. Nessa dissertação é apresentado um sistema multiagente para realizar o processo de captura de páginas da Web. Este sistema utiliza um mecanismo de distribuição e balanceamento de carga desenvolvido sobre uma plataforma *peer-to-peer*, visando localizar os servidores mais próximos dos *crawlers* distribuídos, permitindo que estes atuem de forma colaborativa através das informações coletadas ao longo de suas atividades, buscando otimizar o uso de recursos computacionais.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

A CRAWLER PEER-TO-PEER BASEAD ON AGENTS

Marcelo de Mattos Mayworm

September/2007

Advisor: Geraldo Bonorino Xexéo

Department: Systems and Computer Engineering

Although the scientific community has been able to design increasingly more efficient web crawlers, the challenge posed both by the use of computational resources and the identification of web page host servers in the neighborhood of these crawlers still remains. This dissertation introduces a multi-agent system capable of capturing pages on the web. This system deploys a load distribution and balancing mechanism developed on a peer-to-peer platform with the aim of locating servers in the neighborhood of distributed web crawlers, thereby allowing the latter to perform collaboratively based on the information gleaned from their activities. The ultimate purpose is to optimize the use of computational resources.

Índice

Índice de Figuras

Índice de Tabelas

Capítulo 1 – Introdução

A busca e localização de dados textuais na Internet é hoje um campo consolidado que ganha complexidade e eficiência com o passar dos anos. Cada vez mais as máquinas de busca investem não somente no processo de indexação e extração de informações contidas nos arquivos capturados na Internet, mas também no processo de captura e localização dessas informações. Brin & Page (1998) já tinham relatado que milhões de páginas são criadas e atualizadas diariamente na Internet, o que torna o processo de captura um desafio constante para as máquinas de busca.

1.1 – Motivação

1.1.1 – A Fartura de informação

Vivemos na era da explosão de informação. Apesar da Internet ser um canal de informação recente, se comparado com outros meios historicamente utilizados pela humanidade, a Internet cresce em grande velocidade e mantém-se como o principal recurso de pesquisa para seus usuários. Segundo Lyman & Varian (2003), é estimado um total de 170 *Terabytes* de informações dispostas na Internet.

A Internet é uma crescente coleção de documentos: de acordo com (O'Neill, Lavoie et al., 2003), em 2002 existiam algo em torno de 9 milhões de domínios conhecidos, com mais de 3,6 bilhões de páginas. Atualmente segundo ISC (2007) são 500 milhões de domínios existentes.

1.1.2 – Busca e recuperação de informação e *Web search*

Busca e recuperação de informação é a área da ciência da computação que tem como foco a obtenção de informação sobre um determinado assunto, a partir de uma coleção de dados (Baeza-Yates & Ribeiro-Neto, 1999). Segundo Baeza-Yates & Ribeiro-Neto (1999), os sistemas de busca e recuperação de informação devem de algum modo "interpretar" o conteúdo da informação dos itens em uma coleção e construir um ranque de acordo com o grau de relevância da pesquisa realizada pelo usuário. Esta interpretação do conteúdo do documento envolve extração sintática e semântica da informação de um texto.

Uma visão do ambiente Web mostra cinco elementos principais que refletem a escala e dinamismo de suas informações, e mostra apenas como ele é hostil em relação aos tradicionais algoritmos de busca e recuperação de informação. São estes os elementos:

- Volume de informação: Segundo Lyman & Varian (2003) a Internet cresce 30% ao ano, onde milhões de novas páginas são criadas diariamente.
- Um índice infinito: Páginas dinâmicas na Web são 100 vezes mais numerosas do que páginas estáticas (Brin & Page, 1998). Sites dinâmicos podem teoricamente criar uma nova página para cada permutação de dados a partir do banco de dados, o qual a informação é extraída, efetivamente criando um ilimitado número de páginas Web indexadas (Eiron, McCurley et al., 2004).
- O aumento de índices ultrapassados: Todos os índices tornam-se ultrapassados se o conteúdo muda após terem sido indexados, assim a Web é extremamente dinâmica e apresenta grandes desafios a serem considerados. Máquinas de busca devem entretanto, escolher entre estar constantemente investindo no poder computacional para direcionar seus *crawlers*, arriscando os seus índices a se tornarem ultrapassados, ou diminuir a escala de seus *crawlers* para um conjunto menor na Web.
- *Spamming*: O ambiente da Web sofre de *spamming*, no qual autores de páginas Web deliberadamente enganam um algoritmo de ranque das páginas Web com truques freqüentes, independente do conteúdo da página para a necessidade do usuário.
- Conteúdo textual: Conteúdo textual existente nas páginas Web difere radicalmente do que era achado nos sistemas antigos de busca e recuperação de informação. Qualquer coisa que pode ser publicada é publicada, sem qualquer forma de controle, classificação, ou censura. Entretanto não podemos assumir o conteúdo de um documento somente sobre seu texto, nem podemos depender de meta dados embutidos nos títulos das páginas ou elementos HTML, pois eles podem também ser direcionados para *spamming*.

A solução destas situações tem vindo na forma de *Web Search*, o qual é visto como um campo específico dentro da busca e recuperação de informação. Seu foco é na pesquisa puramente em ambiente Web, usando chaves de propriedades Web, tais como ancora e *hyperlinks*, para determinar a relevância e usar modelos específicos nas pesquisas realizadas pelos usuários.

1.1.3 – Pesquisando na Web e Web *crawling*

Um dos principais componentes incluído nas máquinas de busca é o *crawler* ou *spider* ou robô.

Crawlers são aplicações especializadas que coletam páginas da Web para serem indexadas. Eles iniciam suas tarefas por meio dos *hyperlinks* de uma página e obtém as páginas para os quais *hyperlinks* apontam (Brin & Page, 1998). As páginas vinculadas através dos *hyperlinks* serão visitadas e armazenadas em um momento subseqüente. Os *crawlers* determinam quais os *hyperlinks* seguintes serão obtidos a partir das características da máquina de busca, máquinas de busca verticais por exemplo, têm seus *crawlers* focados somente em páginas de um tópico específico, rastreando profundamente o conteúdo de um site. Máquinas de busca horizontais, em contraste, tem seus *crawlers* rastreando mais superficialmente, porém abrangendo um número maior de sites diferentes. Os *crawlers* analisam sintaticamente URL's, HTML e textos contidos em cada página Web, e devem evitar riscos, tais como identificadores únicos gerados dinamicamente e colocados dentro de URL's, o que forma um infinito conjunto de URL's que referenciam o mesmo documento, e armadilhas, as quais são aplicações escritas para gerar um infinito número de páginas para aumentar significativamente o ranque de um site. De acordo com Heydon & Najork (1999), múltiplos *crawlers* frequentemente operam em paralelo, utilizando um eficiente algoritmo de agendamento de tarefas, determinando qual processo de *crawler* ira obter páginas na Web.

1.2 – Problema

Como é possível prover mais eficiência no processo de captura de páginas na Internet, visando a proximidade entre os servidores hospedeiros de páginas e os *crawlers*, buscando a otimização de recursos computacionais utilizados?

1.3 – Hipótese

A implementação de um *crawler* colaborativo organizado em uma plataforma *peer-to-peer* capaz de identificar servidores hospedeiros de páginas Web que estejam próximos a este durante o processo de captura das páginas, permitindo a colaboração entre os agentes de software formadores do mesmo, aumentará o desempenho do processo de captura de páginas na Internet. Colaborará com o *framework* COPPEER (Miranda, Xexéo et al., 2006), como sendo a primeira aplicação a rodar na rede mundial utilizando-o.

1.4 – Organização

As principais etapas previstas para essa pesquisa são a identificação e o levantamento dos *crawlers* já relatados na literatura, um estudo sobre estratégias de distribuição e balanceamento de carga para *crawlers* distribuídos, e a realização de um estudo observatório para avaliação das estratégias relatadas nesse trabalho.

O Capítulo 2 faz uma revisão sobre trabalhos propostos para *crawlers* e descreve as atividades conceituais realizadas por estes, como a organização de uma arquitetura. Ao final deste capítulo são apresentados diversos exemplos de *crawlers* já desenvolvidos.

O Capítulo 3 continua com um estudo focado em *crawlers* colaborativos e distribuídos, buscando explicar a organização das atividades paralelas e as estratégias existentes.

O Capítulo 4 descreve a proposta de *crawlers* colaborativos, apresentando os requisitos principais e requisitos técnicos como a utilização de uma plataforma *peer-to-peer*. Este capítulo apresenta uma visão geral da arquitetura do *crawler* e dos agentes que o constituem, definindo também o mecanismo de distribuição e balanceamento de carga utilizada pelos *crawlers* colaborativos.

O Capítulo 5 apresenta uma implementação de referência, isto é, um protótipo de uma aplicação multiagente visando demonstrar as funcionalidades e estratégias propostas nesse trabalho. Tal capítulo descreve a plataforma *peer-to-peer* utilizada, a implementação do mecanismo de distribuição e balanceamento de carga, os desafios encontrados, e os pacotes e as classes utilizadas na implementação dos agentes.

O Capítulo 6 descreve um estudo observatório de uso dos *crawlers* colaborativos e como o processo de captura ocorreria na Internet para busca de páginas.

Por fim, o Capítulo 7 apresenta a conclusão deste trabalho, apontando os objetivos alcançados e indicando as perspectivas para trabalhos futuros.

Capítulo 2 – Revisão bibliográfica

Nesse capítulo são revisados os trabalhos correlatos a essa dissertação. Na seção 2.1 são apresentadas estratégias de captura utilizadas por *crawlers*. A seção 2.2 provê uma visão geral das características dos *crawlers* no ambiente da Web, na seção 2.3 são apresentados os principais tipos de *crawlers*, a seção 2.4 descreve arquiteturas de *crawlers* e a seção 2.5 apresenta alguns exemplos de *crawlers*.

2.1 – Estratégias de captura

Como visto na seção "1.1.3 – Pesquisando na Web e Web *crawling*", *crawlers* são aplicações especializadas que coletam páginas da Web para serem indexadas. Diferentes métodos podem ser utilizados pelos *crawlers* durante o processo de captura de páginas.

Um dos principais desafios relacionados a Web é o de localizar páginas com conteúdo relevante. Como existem poucas páginas importantes no meio de uma enorme quantidade de outras menos relevantes (tendo como base algumas métricas tais como o algoritmo de *PageRank* apresentado por Lawrence, Sergey *et al.* (1999), tamanho da página, número de referências, entre outras), apenas tomar uma URL aleatoriamente não é suficiente para uma boa pesquisa. Para muitas aplicações, páginas com pouco ou nenhum conteúdo significante, devem ser excluídas. Com isso, segundo Henzinger, Heydon *et al.* (2000) é importante estimar a importância de cada página, mesmo tendo somente informações parciais sobre as mesmas. A seguir são abordados os dois principais métodos para captura.

2.1.1 – Captura vertical

Envolve obter as páginas exclusivamente por nomes de domínios. Os sistemas de domínios induzem a uma estrutura hierárquica, onde a captura vertical pode ser realizada nos diferentes níveis dessa estrutura. Quando capturas verticais são realizadas nos níveis mais altos, elas podem selecionar partes por países, tais como: .br, .it, .cl, as quais são imaginadas serem coesas em termos de: linguagem e história. Outra forma menos coerente é a seleção de níveis de domínio mais gerais, tais como: .edu.br ou .com.br. Quando capturas verticais são realizadas em um segundo nível,

elas podem escolher um conjunto de páginas produzidas por membros de uma mesma organização, por exemplo: ufrj.edu.br, cos.ufrj.br e if.ufrj.br) (Baeza-Yates, Castillo et al., 2005).

2.1.2 – Crawler focado

A primeira introdução sobre *crawlers* focados foi relatada por Bra *et al.* (1994), e subsequentemente estudada por Chakrabarti *et al.* (1999). Os *crawlers* que seguem essa estratégia são designados seletivamente a capturar páginas com conteúdo relevante para tópicos de interesse, utilizando a estrutura de *hyperlinks*.

Um método baseado em *crawlers* focados inicia-se a partir de uma lista de URL's pertencentes a um determinado tópico. Ele estima a probabilidade de cada *hyperlink* candidato subseqüente aos *hyperlinks* da lista conduzir o *crawler* para capturar páginas de conteúdo relevante, e deve priorizar a ordem de captura dessas páginas com base na probabilidade calculada, podendo ou não excluir os *hyperlinks* candidatos com baixa probabilidade de visita. Evidentemente, as âncoras dos *hyperlinks*, palavras contidas nas URL's e a relevância do conteúdo das páginas são tipicamente explorados no momento de estimar o valor de um *hyperlink* candidato.

Em Chakrabarti *et al.* (1999) é usado um grafo de *hyperlink* baseado em "vizinhos de entrada" documentos que apontam para outros documentos destino, através de citação e "vizinhos de saída" documentos que são apontados por outros documentos, através de citação, para algumas classificações. De acordo com Chakrabarti *et al.* (1999), um *crawler* focado pode adquirir páginas relevantes, enquanto um *crawler* padrão rapidamente indexa um grande número de páginas irrelevantes e perde seu rumo, mesmo esses *crawlers* tendo iniciado a partir da mesma lista de URL's.

Talvez um dos pontos mais críticos de avaliação para um *crawler* focado é mensurar a taxa de relevância de uma página capturada, e como efetivamente filtrar páginas irrelevantes para que o *crawler* não desvie seu rumo e não perca tempo no processo de localização e captura de páginas relevantes. Seria interessante avaliar a relevância das páginas, com base no julgamento de um ser humano. Entretanto, isso torna-se impraticável visto que esse julgamento teria que ser realizado sobre centenas de milhares de páginas.

Sendo assim, o uso de um classificador automático que analise a coleção de páginas torna-se imprescindível. Chakrabarti *et al.* (1998) usa uma extensão do

classificador *naive-Bayesian*, chamado *Rainbow*, para determinar a relevância de um tópico em um documento obtido.

Esse classificador constrói seu modelo interno, podendo determinar um tópico de uma página obtida. Por exemplo, como um tópico em uma taxonomia com um alto valor de probabilidade. Dada uma página, o classificador retorna uma lista ordenada de todas as classes de nomes e a pontuação relevante da página em cada classe. Portanto, o classificador é responsável por determinar o tópico para cada página. Adicionalmente ele determina a próxima URL a ser visitada, assumindo a relevância da página como sendo um indicador de relevância de seu vizinho. Isso é chamado de *radius-1 hypothesis*, onde se uma pagina "u" é um exemplo de tópico, e "u" aponta para "v", então a probabilidade de "v" ser um tópico é maior do que a probabilidade aleatória da escolha de uma página na Web ser um tópico.

Com base em *crawler* focado, Chakrabarti *et al.* (1999) define dois modelos de aproximação, *hard-focus* e *soft-focus*. *Hard-focus* ocorre caso o classificador identifique uma página obtida como *off-topic*, onde o *crawler* não visita as URL's extraídas dessa página, ou seja, ele desencoraja o *crawler* a visitar as URL's. Por exemplo, se a pontuação retornada pelo classificador para a página obtida não classificar a página como um tópico a ser alcançado ou se a pontuação é menor do que o requerido, o *crawler* conclui que esta página é *off-topic* e não visita seus *hyperlinks*. Em contrapartida a aproximação *soft-focus* é menos restritiva, nessa aproximação, o *crawler* recebe do classificador a relevância da página obtida e adiciona essa pontuação a todas as URL's extraídas da página. Com isso o *crawler* adiciona essas URL's a fila de prioridades no processo de busca. Essa aproximação *soft-focus* é muito parecida com o PageRank (Lawrence, Sergey et al., 1999) inicial, onde a pontuação obtida para a página era distribuída para suas URL's, porém com cálculos diferentes.

Para Altingovde & Ulusoy (2004) *crawlers* focados apresentam como ponto fraco o fato de não proverem suporte para *tunneling*, ou seja, o classificador não aprende que o caminho de acesso a uma página *off-topic* pode eventualmente levar a páginas *on-topic*. Por exemplo, no caso de procura por artigos de redes neurais, o usuário deve encontrá-los provavelmente através de *hyperlinks* de páginas de centro de pesquisas de universidades. Classificadores "básicos" podem atribuir uma baixa relevância para essas páginas, e portanto desperdiçar a oportunidade de encontrar páginas *on-topic*. Altingovde & Ulusoy (2004) apresentam a idéia de usar regras para

guiar o *crawler*, regras que o *crawler* aprenderá ao longo das extrações de conteúdo das páginas capturadas.

A extração de regras se baseia em heurísticas para determinar relacionamentos tais como: uma página em uma classe A refere-se a páginas em uma classe B com probabilidade *p*. Durante o processo de *focused crawling*, é pedido ao classificador para classificar uma página em particular que já tenha sido obtida. De acordo com a classificação recebida, a pontuação é computada indicando a probabilidade total de atingir um tópico a partir dessa página. Então o *crawler* insere as URL's extraídas dessa página em uma fila de prioridades com a pontuação computada. O mecanismo de pontuação baseado em regras proposto por Altingovde & Ulusoy (2004) é independente das similaridades de páginas focadas, mas ainda assim depende da probabilidade que uma classe de uma página de se referir à classe a ser buscada.

Finalmente o *crawler* pode computar todas as regras para um particular conjunto de tópicos a partir de regras armazenadas em um banco de dados, antes do inicio do processo de captura. Isso permite representar o grafo das regras. O *crawler* desenvolvido por Altingovde & Ulusoy (2004) utilizou a taxonomia dos dados de um diretório público da Web, o DMOZ (1998). Este diretório tem como característica a categorização e organização dos *hyperlinks* em tópicos. Estes tópicos seguem uma estrutura lógica e são subdivididos por detalhamentos. Por exemplo a categoria esportes tem a subdivisão futebol que por sua vez pode ser divida em profissional e amador. Normalmente os diretórios Web são feitos por pessoas como o caso do DMOZ (1998), diferente das máquinas de busca que usam *crawlers*. Altingovde & Ulusoy (2004) seguiram a idéia de alguns trabalhos já existentes utilizando um classificador próprio para determinar a relevância das páginas.

2.1.3 – Captura horizontal

Envolve um critério de seleção que não se baseia em nomes de domínios. Nesse caso existem duas formas para capturar os dados: o uso do *log* de transações no *proxy* de um grande provedor de internet, ou através do uso de *crawler*. Ocorrem vantagens e desvantagens em cada uma, o uso de *proxy* facilita a localização das páginas mais populares, porém é impossível controlar o tempo de um novo acesso (Cho, 2001), pois este depende dos usuários que acessam a Internet através do *proxy*. Com a utilização de um *crawler* o período para revisitar as páginas populares precisa ser estimado, entretanto, o período de revisita pode ser ajustado.

Em capturas horizontais, percursos randômicos podem ser utilizados para obter um conjunto, no qual páginas são forçadamente visitadas com probabilidade proporcional aos seus valores de *PageRank* (Lawrence, Sergey et al., 1999), sendo assim, segundo Henzinger, Heydon *et al.* (1999) espera-se que a captura seja imparcial. Boldi, Codenotti *et al.* (2002) relatam que utilizando este método de captura, espera-se que os servidores hospedeiros de páginas Web não sofram sobrecarga com as diversas requisições feitas pelos *crawlers*, pois durante o processo de extração de *hyperlinks* das páginas capturadas um domínio diferente será encontrado e passará a ser requisitado por estes *crawlers*. Outra importante vantagem nas visitas são as requisições aos servidores não serem constantes, dessa forma, naturalmente ocorre um controle de acesso durante as visitas realizadas, evitando sobrecarregar os servidores hospedeiros das páginas.

Segundo Najork & Wiener (2001) *crawlers* que utilizam capturas horizontais tendem localizar páginas de alta qualidade durante os primeiros estágios do processo de captura, porém com decorrer do tempo de execução a qualidade das novas páginas localizadas se deteriora. É especulado que captura horizontal é uma boa estratégia para *crawlers*, pois a maioria das páginas de alta qualidade contém diversos *hyperlinks* para outros domínios, e portanto estes *hyperlinks* serão localizados primeiro de qualquer maneira. Baeza-Yates, Castillo *et al.* (2005) informam que captura horizontal está próxima de ser a melhor estratégia de navegação durante os primeiros momentos do processo de captura.

Alguns *crawlers*, como o apresentado por Boldi, Codenotti *et al.* (2002) utilizam captura horizontal, porém a profundidade da busca realizada pelos *crawlers* é limitada, incentivando uma maior cobertura no número de domínios identificados. A Figura 1 mostra os caminhos percorridos por um *crawler* utilizando os métodos de captura apresentados.

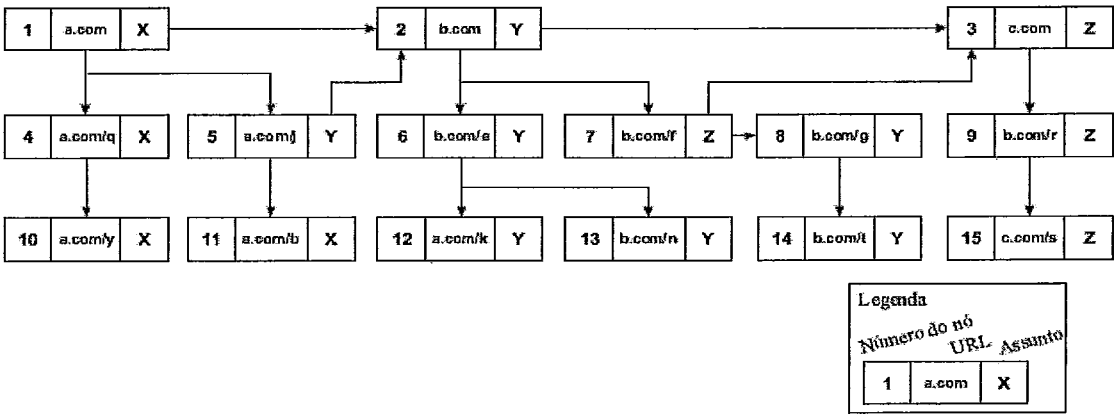


Figura 1 – Uma estrutura hipotética de Web

A Figura 1 apresenta o *crawler* percorrendo diferentes caminhos cada um destes orientados por diferentes métodos de captura. O método de captura horizontal faz uma busca em largura, gerando o seguinte caminho: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15. O método de captura vertical faz uma busca em profundidade gerando o caminho: 1, 4, 10, 5, 11, 2, 6, 12, 13, 7, 8, 14, 3, 9, 15. O método de captura focado simples (em Y), partindo de 2 gera o caminho 2, 6, 12, 13, não encontrando 8, 14 e 5. Já o método de captura focado estendido (em Y) partindo de 2, gera o caminho 2, 6, 12, 13, e escolhe 7 para tentar, 8, 14 e 3.

2.2 – Características sobre Web *crawlers*

2.2.1 – Políticas de visita

Existem muitos *crawlers* em atividades simultaneamente na Web, caracterizando uma grande carga de atividades na rede. Cada um desses *crawlers* é capaz de visitar vários servidores Web diariamente, e capturar um grande número de URL's, executando a captura de arquivos de diferentes tamanhos e tipos. É sabido da extrema importância dos *crawlers* para as máquinas de busca, porém, os mesmos no momento de captura das URL's podem causar potenciais sobrecargas de acesso aos servidores Web, através do consumo de recursos de máquina e largura de banda de rede, causados pelo excessivo número de requisições submetidas.

Algumas sugestões sobre boas práticas em política de visita, foram dadas por Koster (1995). Dentre as boas práticas descritas por Koster (1995), destacam-se: agendar o acesso dos *crawlers* para ser realizado entre intervalos de tempo, não visitar espaços não permitidos e intercalar as visitas entre servidores. Outra adição para a solução de políticas de visita, foi a definição do protocolo de exclusão de robôs

(Koster, 1996), sugerido para os administradores de *sites* indicarem quais partes dos servidores não deveriam ser acessadas, e também complementando o protocolo de exclusão de robôs, foram criadas as *meta tags robot tag* (Koster, 1996) com o intuito de permitir aos autores das páginas HTML informarem aos *crawlers*, se a página que esta sendo visitada pode ser indexada ou usada para garimpar mais *hyperlinks*.

Na arquitetura de um *crawler*, questões como consumo de largura de banda e recursos de CPU dos servidores a serem requisitados, precisam ser levadas em consideração. Isso significa que um *crawler* deve agir de forma a evitar ataques involuntários aos servidores. Mais precisamente, um *crawler* deve seguir as definições do protocolo de exclusão de robôs (Koster, 1996), de forma a limitar o número de subseqüentes requisições e estipular o tempo de um minuto entre cada requisição submetida para um mesmo servidor, garantindo boas práticas em políticas de visitas.

A utilização de política de visita varia entre diversos tipos de *crawlers*, levando em consideração as suas implementações, estratégias de navegação e possivelmente contexto de atuação, como por exemplo tópicos de interesse. Em Cho & Garcia-Molina (2003), caso o *crawler* tenha que executar uma requisição cujo servidor é o mesmo da requisição executada anteriormente, são utilizados 10 segundos como intervalo de tempo. Najork & Wiener (2001) insere duas modificações no método de captura horizontal, usada na pesquisa e no processo de baixar URL's por seu *crawler*, de forma a manter uma política de visita. São elas:

- Não é permitido mais de uma conexão HTTP aberta simultaneamente para um mesmo servidor Web. Segundo Najork & Wiener (2001) essa modificação se faz necessária para evitar o processo de baixar diversas páginas da Web paralelamente de um mesmo servidor, visto que em torno de 80% dos *hyperlinks* existentes na páginas Web apontam para o mesmo servidor de origem da mesma;
- Caso o *crawler* leve *t* segundos para executar o processo de baixar um arquivo de um dado servidor Web, então ele aguardará outros *t* segundos antes de requisitar o servidor Web novamente. Essa modificação não é estritamente necessária, mas distancia a possibilidade de sobrecarga nos servidores Web, por parte do *crawler*,

respeitando uma política de visita como descrito na seção "4.3.4 – Política de Visita".

2.2.2 – Política de atualização.

Crawlers projetados para máquinas de busca não podem somente visitar todas as páginas Web existentes na Internet uma única vez e armazenar seus conteúdos, eles precisam registrar as atualizações sofridas por estas páginas e até mesmo o seu desaparecimento. Com isso os *crawlers* devem continuamente, tanto quanto possível, atualizar todo o conteúdo que tenham armazenado, visto que as páginas Web sofrem constantes mudanças. Isso significa que estratégias adequadas devem ser implementadas para que os *crawlers* possam determinar quando uma página Web deverá ser revisitada, tendo como base dados estatísticos sobre a freqüência de mudança de uma página, ou outra forma similar de informação.

Existem duas principais formas de detectar e capturar páginas atualizadas. A primeira é baseada no modelo *push* (Olston & Widom, 2002; Olston, 2003), onde as fontes de dados (exemplo: sites da Web) são cooperativas e enviam suas atualizações para um repositório local do *crawler*. Apesar de este modelo funcionar bem para mecanismos de *cache*, seu desempenho se deteriora quando expandido à grandes aplicações, incluindo a Web, pois receberá continuamente um grande número de requisições. O modelo *pull* (Cho & Ntoulas, 2002) é um outro modelo onde o *crawler* periodicamente verifica as fontes de dados para detectar e obter as páginas. Esse modelo é mais adequado para grandes aplicações como a Web, pois ele controla a ação de visitar as páginas Web, e é utilizado em diversas máquinas de busca (Brin & Page, 1998; Cho & Garcia-Molina, 2000; Lawrence & Giles, 2000). A maioria das máquinas de busca costuma utilizar políticas simples de agendamento de visita, como *round-robin* (Brin & Page, 1998; Lawrence & Giles, 1998; Heydon & Najork, 1999; Cho & Garcia-Molina, 2000). Isso garante que todas as páginas serão capturadas em intervalos regulares de tempo.

Entretanto, como o aumento das aplicações, o desempenho destes agendamentos diminui, e a *roud-robin* falha em perceber as mudanças nas páginas. Para solucionar esse problema, Cho & Garcia-Molina (2003) introduziu um método chamado Freqüência de Mudança (CF), para estimar a freqüência de atualização nas páginas Web. Baseado no passado histórico de mudanças das páginas Web, o método CF estima a freqüência de atualizações e decide quando visitar novamente as páginas.

As políticas baseadas em freqüência provam ser mais favoráveis, se a freqüência de atualização pode ser computada corretamente em detalhes (Cho, 2001), o que é uma tarefa difícil de conseguir.

Por outro lado, Cho & Ntoulas (2002) propôs uma política de amostragem para *crawlers*, na qual um pequeno número de páginas Web de cada site é primeiro capturado, e estimada a freqüência de atualizações que estas sofrem neste site. Baseado nesta estimativa são alocados para cada site, recursos dos *crawlers* para a captura das páginas Web. Diferentemente de CF, o método de amostragem realiza suas capturas puramente baseadas em amostragens tomadas em um ciclo corrente. Sendo assim o método de amostragem não mantém o acompanhamento de históricos de atualizações anteriores das páginas. Entretanto, CF consegue um melhor desempenho e resultado em uma longa execução, embora sua variação de tempo entre as páginas alteradas e inalteradas diminua com o passar do tempo, tornando-se obrigatório a captura de páginas que raramente mudam.

Olston (2003) apresenta uma política de agendamento para revisitar as páginas existentes em um repositório local, com base nas pesquisas realizadas na máquina de busca, qualificando as páginas existentes e caracterizando uma métrica chamada *User-Centric*. Com base nessa métrica ocorrerá um processo incremental de *crawling*, que tem como objetivo priorizar a atividade de baixar páginas com o maior índice de qualidade, mensurado de acordo com funções definidas por Olston (2003). Estas funções levam em consideração dados utilizados pelos usuários nas pesquisas realizadas na máquina de busca, proporcionando a atualização nas páginas existentes em seu repositório local.

Com o objetivo de minimizar o número de páginas desatualizadas dentro do repositório da máquina de busca, Cho & Garcia-Molina (2003) se baseiam no modelo SBR (*Staleness-Based Refreshing*), o qual foca unicamente em determinar com qual freqüência cada página deverá ser atualizada. Pandey & Olston (2005) apresentam o modelo EBR (*Embarrassment-Based Refreshing*), o qual objetiva minimizar o nível de "constrangimento" para a máquina de busca, onde esse constrangimento é identificado quando ao visitar uma página retornada pela pesquisa realizada na máquina de busca, o usuário descobre que a página visitada não tem relevância na sua pesquisa. Olston (2003) segue o esquema *user-centric refreshing* o qual é parametrizado através de um mecanismo de pontuação, onde para cada página "p" é

atribuído um valor prioritário P(p,t), o qual pode variar com base no tempo. O agendamento de revisita se baseia nesse valor prioritário.

2.3 – Tipos de crawlers

Essa seção descreve os principais tipos de *crawlers* existentes e relatados até o momento nos trabalhos referenciados por essa dissertação.

2.3.1 – Crawlers paralelos

Com o crescimento natural da Web, torna-se cada vez mais difícil sua catalogação como um todo ou uma porção significante através da aplicação de um *crawler* baseado em um processo único. Ou seja, as atividades realizadas pelo *crawler* e seus componentes não podem ser paralelizadas e/ou escaladas, o que causaria uma degradação no processo de captura de páginas.

Para Cho & Garcia-Molina (2002) os principais desafios de *crawlers* paralelos são: sobreposição, qualidade e largura de banda para comunicação. Sobreposição ocorre quando múltiplos processos são executados em paralelo para capturar páginas e diferentes processos capturam as mesmas. Qualidade torna-se desafio quando frequentemente um *crawler* objetiva capturar as páginas importantes primeiro, buscando maximizar a qualidade das páginas capturadas. Entretanto, em *crawlers* paralelos cada processo não tem ciência da porção da Web capturadas até o momento por um outro processo. Largura de banda para comunicação torna-se desafio quando para evitar sobreposição e aumentar a qualidade das páginas capturadas, os processos que executam os *crawlers* necessitam periodicamente comunicar-se para coordenar o trabalho uns dos outros.

Segundo Cho & Garcia-Molina (2002), *crawlers* paralelos apresentam algumas importantes vantagens sobre *crawlers* que trabalham em um único processo, tais como: escalabilidade, dispersão de carga e redução de carga de rede. Com o enorme tamanho da Web, é necessário escalonar os processos de *crawlers*, ou seja, torna-se imperativo executar *crawlers* paralelamente, onde simplesmente não é possível atingir certas taxas de captura de páginas com a execução de um único processo de *crawler*. A aplicação do conceito de dispersão de carga de rede em *crawlers* paralelos proporciona a captura de páginas regionalizadas por redes, por exemplo, os processos que estejam sendo executados no Brasil, somente procurarão

páginas brasileiras, além disso ainda é possível proporcionar a redução de carga de rede, pois possivelmente os *crawlers* estarão atuando em redes locais.

Cho & Garcia-Molina (2002) apresentam diferentes organizações na arquitetura para processos paralelos:

- Intra-site parallel *crawler*
    - Quando os processos executam em uma mesma rede e comunicam-se através de um canal de alta velocidade como por exemplo uma *LAN*(*Local Area Network*).
- *Crawler* distribuído
    - Processos que executam geograficamente distantes, conectados através da Internet. Por exemplo, um processo que executa nos Estados Unidos somente captura páginas americanas.

Outro ponto importante mostrado por Cho & Garcia-Molina (2002), é a coordenação dos diversos processos de *crawler* que executam em paralelo:

- Independente
    - Todos os processos capturarão páginas sem nenhuma coordenação. Nesse cenário, deverá ocorrer sobreposição nas páginas capturadas, podendo não ser relevante essa sobreposição caso os processos de *crawler* iniciem suas atividades a partir de diferentes URL's.
- Atribuição dinâmica
    - Quando existe um coordenador central para dividir logicamente a Web em pequenas frações e atribuir cada uma dessas partições aos diferentes processos.
- Atribuição estática
    - Quando a Web é divida e atribuída para cada processo de *crawler* antes do início das atividades de captura.

Em Cho & Garcia-Molina (2002) também são definidas formas de interação entre os diversos processos de *crawler*:

- *Firewall*
    - Nessa forma cada processo captura as páginas dentro de sua partição da Web, e não segue qualquer *hyperlink* que aponte

para uma partição de outro processo. Todos os *hyperlinks* que apontam para partições de outros processos são ignorados. Cho & Garcia-Molina (2002) apresenta alguns resultados que quando um pequeno número de processos rodam em paralelo, um *crawler* usando a forma *firewall* poderá prover uma boa abrangência de captura, desde que os processos iniciem com diferentes conjuntos de URL's. Portanto a forma *firewall* pode não ser uma boa escolha caso diversos processos estejam sendo executados paralelamente, pois a falta de comunicação entre estes processos pode resultar em fatias da Web não serem atingidas.

- *Cross-over*
    - Primariamente cada processo captura as páginas dentro da sua partição da Web, porém quando identifica algum *hyperlink* extraído de uma página que aponta para uma parte externa à sua partição, o *crawler* o "segue" continuando seu trabalho de captura. Para Cho & Garcia-Molina (2002) essa forma é mais eficiente à forma *firewall*, porém incursa em sobreposição.
- *Exchange*
    - Ocorre quando temporariamente cada processo se comunica com outros processos para trocar os *hyperlinks* inter-partição, ou seja, *hyperlinks* que apontam para outras partições. É importante saber que esses processos não seguem os *hyperlinks* inter-partição. Para os processos baseados nessa forma, uma interessante técnica para minimizar a troca de URL's é a comunicação *batch*. Utilizando essa técnica, o processo de *crawler* em lugar de transferir um *hyperlink* inter-partição imediatamente após ser descoberto, aguarda até obter uma coleção de URL's e as envia através de um processamento *batch*. A comunicação *batch* tem várias vantagens sobre a comunicação incremental. Em primeiro lugar porque a sobrecarga de comunicação diminui, e em segundo porque o número de URL's trocadas diminuirá. Após seus experimentos,

Cho & Garcia-Molina (2002) concluíram que partições baseadas em site *hash* reduzem significativamente a sobrecarga de comunicação, comparadas ao esquema URL *hash*.

Para a criação das partições da Web, Cho & Garcia-Molina (2002) apresentam três diferentes formas:

- *URL hash*
    - Tendo como base o valor gerado a partir de uma função *hash*, será atribuído para cada processo uma página. Nesse esquema, páginas existentes em um mesmo site poderão ser atribuídas a diferentes processos. Entretanto, a localidade de um *hyperlink* não é refletida na partição, ocorrendo diversos *hyperlinks* inter-partição.
- *Site hash*
    - Ao invés de usar uma função *hash* para gerar o valor a partir de uma URL, o valor será gerado a partir do domínio. Com isso as páginas serão alocadas para a mesma partição, entretanto os *hyperlinks* que apontam para outros sites serão alocados em partições diferentes. É possível perceber como essa forma terá um número significativamente menor de inter-partição em relação à forma "URL *hash*".
- Hierárquico
    - Essa forma ao contrário de utilizar valores *hash* para dividir as partições, toma como base os tipos de domínio, por exemplo: domínios ".com" e ".net" tornam-se duas partições, e os tipos restantes de domínios tornam-se uma terceira partição.

É importante notar que a qualidade de um *crawler* paralelo pode ser inferior a de um *crawler* que trabalhe em um processo único, pois diversos tipos de métricas dependem de uma estrutura global da Web, como por exemplo, a métrica *backlink count*. Para isso cada processo existente em um *crawler* paralelo deve conhecer somente páginas que este capturou, portanto, terá uma visão inferior das páginas importantes, em relação a um *crawler* que trabalhe em processo único. Para evitar esse tipo de problema na qualidade das páginas capturadas, os processos que

trabalham em paralelo necessitam periodicamente trocar informações sobre páginas importantes. Por exemplo: se a métrica *backlink count* é importante, um processo deve periodicamente notificar outros processos de como muitas páginas em sua partição contêm *hyperlinks* que apontam para páginas de outras partições. Essas notificações geram uma sobrecarga na comunicação através das mensagens trocadas, que é definida como a média do número de URL's que apontam para partições diferentes da partição onde se encontra a página que continha as devidas URL's.

A granularidade por paralelismo é uma escolha natural por diversos motivos. Um site opera em um ambiente físico homogêneo e políticas de acesso para os *crawlers* são geralmente consistentes para todas as páginas dentro de um site, não apresentando variações entre as páginas. De fato a maioria dos sites da Internet são geralmente protegidos por simples conjuntos de políticas de visita, especialmente pertinentes a localização de páginas que não devem ser capturadas. Em um *crawler* monolítico, lidar com diversos conjuntos de política de visita, pode retardar seriamente o desempenho de execução. Segundo Cho, Garcia-Molina et al., 2004 algumas razões que favorecem os *crawlers* paralelos em relação aos *crawlers* monolíticos são:

- Tamanho da estrutura de dados de uma URL
    - Dada uma escala de um *crawler*, os dados armazenados nas filas de URL's a serem visitadas e URL's para serem revisitadas são grandes. Assumindo que um *crawler* monolítico captura um bilhão de páginas e a média do tamanho de uma URL são 40 *Bytes*, o *crawler* precisa gerenciar 40 *Gigabytes* de dados nas filas de URL's. Geralmente o tamanho da memória principal dos computadores é menor que 40 *Gigabytes*. Com isso, um *crawler* monolítico necessitara armazenar as URL's em disco, e executar operações específicas para lidar com as URL's. Executando diversos *crawlers* independentes, pode se evitar o uso de operações acessando discos para lidar com as URL's, pois cada *crawler* irá lidar somente com a lista de URL's de um determinado pedaço da Web, o qual ele seja o responsável.
- Independência

- Crawlers paralelos precisam de um nível baixo de coordenação, pois estes trabalham independentes uns dos outros. Essa independência provê facilidade para a execução de múltiplos *crawlers* paralelamente em múltiplos computadores. Com interesse em aumentar a capacidade de captura de páginas Web, basta simplesmente iniciar mais *crawlers* nos mesmos ou em outros computadores. Um *crawler* paralelo em contraste a um *crawler* monolítico, requer cuidados na arquitetura e implementação de concorrência.

Cho, Garcia-Molina *et al.* (2004) relatam o problema de páginas isoladas durante o processo de captura, onde um *crawler* paralelo não segue os *hyperlinks* que apontam para páginas web pertencentes a uma porção da Web diferente da sua. Isso proporciona a perda de algumas páginas durante o processo de captura. A Figura 2 ilustra este problema. Dois sites *S*1 e *S*2 são respectivamente capturados pelos *crawlers* *C*1 e *C*2, e a página *d* é somente acessada através do site *S*2, e não será capturada pelo *crawler* *C*2.

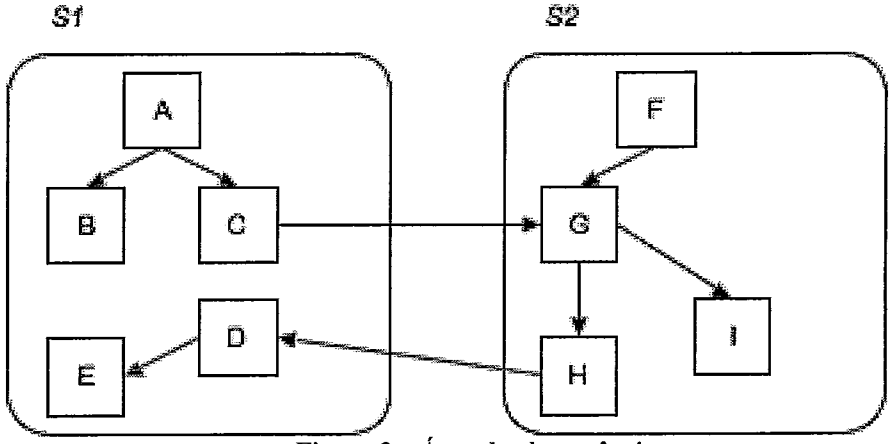


Figura 2 – Área de abrangência

Com o objetivo de não perder esse tipo de página, Cho, Garcia-Molina *et al.* (2004) usam um retrato da Web que foi obtido em um momento anterior do processo de captura. Com essa estratégia, se uma página existente nesse retrato da Web contém uma ligação para a página d, mas a página d não existe no retrato da Web, d será adicionada a lista de URL's para serem capturadas no próximo processo de captura.

2.3.2 – Crawlers componentizados

A necessidade de poder escalar Web *crawlers*, torná-los tolerantes a falhas e adaptáveis a diversas aplicações de *crawlings*, contribuiu para a definição de *crawlers* componentizados. Shkapenyuk & Suel (2002) apresenta uma estrutura de *crawler* distribuído concebida em dois principais componentes, denominados aplicação de *crawling* e sistema de *crawling*. A aplicação de *crawling* decide qual página será capturada, a partir de páginas capturadas anteriormente ou a partir de uma URL de ponto de partida, e delega ao sistema de *crawling* sua captura. O sistema de *crawling* eventualmente captura as requisições e direciona o resultado para a aplicação de *crawling*, para análise e armazenamento. O sistema de *crawling* gerencia tarefas, como: política de visitas, controle de velocidade, resolução de *DNS*(*Domain Name System*), enquanto a aplicação implementa estratégias de captura, tais como, captura vertical, horizontal e *crawler* focado.

A arquitetura de um *crawler* componentizado e distribuído é definida por Shkapenyuk & Suel (2002), de forma a particionar o *crawler* em componentes especializados e todos esses componentes proporcionam a aplicação de *crawling* a execução em diferentes máquinas e diferentes sistemas operacionais, podendo ainda ser replicada para aumentar o desempenho. Conforme mencionado anteriormente, o *crawler* distribuído está dividido em dois principais componentes – aplicação de *crawling* e sistema de *crawling*. O sistema de *crawling* consiste em um gerenciador de *crawler*, um ou mais mecanismos responsáveis por baixar páginas da Web e um ou mais mecanismos capazes de resolver *DNS*. Esses componentes apóiam a distribuição da aplicação de *crawling*.

O gerenciador de *crawler* é responsável por receber a requisição de uma URL a qual tem um nível de prioridade, e direcioná-la para algum componente disponível para efetuar a captura, enquanto trata das regras de análise de política de visita e velocidade de captura (tempo de acesso aos servidores Web). Para a solução definida por Shkapenyuk & Suel (2002), a comunicação é realizada de duas formas: através de *sockets* para pequenas mensagens, e via sistema de arquivos (*Network File System*) para grandes *streams* de dados. O uso de *NFS* (*Network File System*) permite um planejamento mais flexível e o ajuste de desempenho através do redirecionamento e particionamento entre discos. Por último, o gerenciador de *crawler* notifica a aplicação de *crawling* sobre as páginas que foram capturadas e estão disponíveis para

processamento. O gerenciador também exerce o papel de balancear a carga entre os mecanismos responsáveis por baixar as páginas da Web, através de um monitoramento, ajustando a velocidade de captura conforme necessário. Um dos principais objetivos de Shkapenyuk & Suel (2002) foi conceber um sistema capaz de aumentar seu desempenho de execução através da adição máquinas de baixo custo, usando-as para executar componentes adicionais (gerenciador de *crawl*, mecanismos para baixar páginas da Web e mecanismos capazes de resolver *DNS*). É utilizada uma função de *hash*, para criar partições e definir cada componente responsável por processar e capturar cada URL existente nas partições. Se durante o processo de extração das URL's um componente encontrar uma URL's que aponte para uma outra partição, então essa URL é simplesmente direcionada para a partição responsável por seu cuidado.

Um dos problemas apontado por Shkapenyuk & Suel (2002) é o crescimento acelerado do armazenamento de URL's em memória, e este ir além da capacidade da memória utilizada pelo *crawler*. São citadas algumas soluções como *Bloom filter*, *lossless compression* e estrutura de *disk-resident*. Shkapenyuk & Suel (2002) utilizaram técnicas de operação *off-line*, onde as URL's mantidas na memória principal estão em uma estrutura de árvore rubro-negra, e quando essa estrutura alcança um determinado tamanho, uma lista organizada de URL's é armazenada em meio persistente.

A utilização de *DHT* (*Distributed Hash Table*) para coordenar e distribuir tarefas durante o processo de *crawling* entre os nós participantes em um *crawler* distribuído, apresenta pontos interessantes em sobrecarga de comunicação, *throughput*, e balanceamento de carga. Loo, Krishnamurthy *et al.* (2004) mostra esses pontos, sem levar em consideração questões como persistência e indexação. Diferente do *crawler* paralelo, o qual utiliza um coordenador central que despacha as URL's para serem capturadas paralelamente através de um conjunto de nós, ou o uso de esquemas baseados em *hash*, o *crawler* distribuído apresentado por Loo, Krishnamurthy *et al.* (2004) é construído para funcionar sobre quaisquer *DHT's*. Para Loo, Krishnamurthy *et al.* (2004) o potencial de *crawlers* distribuídos é imenso, tais como:

- Personalização

- Usuários podem customizar suas próprias pesquisas para *crawlers* e executa-las através de serviços.

- *Throughput*
  - A junção de diversos nós faz com que o serviço de *crawling* torne-se mais escalável do que serviços centralizados.

- *Crawlers* generalizados para redes *peer-to-peer*
  - Possibilitar a construção de *crawlers* não somente para a Web, mas construir *crawlers* genéricos, capazes de executar pesquisas e capturas sobre estruturas de grafos distribuídos em diversos tipos de redes (Internet, *peer-to-peer*, dentre outras).

As seguintes metas foram consideradas por Loo, Krishnamurthy *et al.* (2004) no momento de definição do seu *crawler* distribuído:

- Facilidade de customização por parte do usuário
  - O usuário pode indicar onde o *crawler* inicia seu trabalho, definir predicados que filtram páginas capturadas e também controlar em que páginas são capturadas.

- Facilidade de Composição
  - Modelado de forma componentizável para utilização de funcionalidades já implementadas por outros projetos de pesquisas, principalmente para processamento de páginas Web.

- Escalabilidade
  - Conforme o número de nós aumenta, o *throughput* (número de *Bytes* baixados por segundo) dos *crawlers* aumenta.

- Bom visitante
  - Crawlers podem facilmente sobrecarregar os servidores Web, caso não tenham um controle efetivo de seu mecanismo de acesso a estes. Para isso a importância da existência de um controle de "frenagem", o qual provê a freqüência com a qual os servidores Web deverão ser acessados.

As tarefas existentes nos processos de *crawling* apresentado por Loo, Krishnamurthy *et al.* (2004) iniciam quando um plano de pesquisa é executado, e a partir de um conjunto de URL's o *crawler* extrai os *hyperlinks* existentes nas páginas

capturadas, e os publica em uma DHT. Esse plano de pesquisa é enviado para todos os nós no momento de inicialização. Cada nó analisa sua DHT local para identificar novos registros a serem capturados, quando um novo registro é identificado, o processo de análise informa-o para o processo de busca. Cada *hyperlink* capturado a partir dos registros é adicionado novamente à DHT local. Um interessante mecanismo existente, é a customização do *crawler* através da adição de filtros especificados pelo usuário, por exemplo, para limitar o *crawler* dentro do domínio "ufrj.br", pode ser executada uma comparação de substring na URL a ser capturada, para somente seguir URL's que contenham o termo "ufrj.br".

Para evitar a sobrecarga de comunicação na publicação dos *hyperlinks*, Loo, Krishnamurthy *et al.* (2004) utilizam a alternativa de criar partições baseadas em domínios (*hash site*), de forma a dedicar cada nó a um servidor Web. Com o intuito de minimizar a comunicação sobre a DHT, esse esquema tem um único ponto de controle para acessar cada servidor Web, provendo uma forma fácil de executar o controle de "frenagem", sem a necessidade de coordenar um grande conjunto de nós.

Um processo de redirecionamento é implementado por Loo, Krishnamurthy *et al.* (2004), provendo uma simples técnica que permite um *crawler* direcionar um trabalho atribuído a ele para outro *crawler*. O redirecionamento funciona perfeitamente com o esquema de partição baseado em domínio, permitindo que uma página seja capturada por um nó diferente. O processo de redirecionamento pode implementar diversas otimizações. Por exemplo, um determinado nó pode decidir redirecionar uma URL's para um outro nó, caso sua latência de rede durante o acesso seja alta para o servidor Web. O redirecionamento também torna-se interessante para ambos os casos de distribuição de carga entre os diversos nós e limitar o número de acesso aos servidores através do controle de "frenagem". Um controle para o número máximo de redirecionamentos para cada URL também foi implementado. Uma sobrecarga de comunicação extra sobre a DHT é gerada. (Loo, Krishnamurthy et al., 2004) também relata a existência de diversas propostas de DHT's para balanceamento de carga com o uso de servidores virtuais e outras técnicas.

2.3.3 – Crawlers *em peer-to-peer*

A utilização de *peer-to-peer* apresenta diversos atrativos, como por exemplo, a utilização de recursos computacionais, pois esses recursos crescem conforme aumenta os números de usuários. Outro atrativo é o fato de ser uma arquitetura auto-

organizável, não necessitando de administração adicional para escalar, e por último ser tolerante a falhas. Sistemas *peer-to-peer* diferem um dos outros em termos de como eles trocam dados ou distribuem tarefas, além de como fazem a localização de informações através da rede.

Singh, Srivatsa *et al.* (2003) apresentam um processo de *crawling* totalmente descentralizado em *peer-to-peer*, sem a utilização de um coordenador central. Todos os nós trabalham independentes sem qualquer comando de um coordenador para decidir o que fazer, e os dados são mantidos e distribuídos através de rede. A tarefa de *crawling* é dividida entre os nós e utiliza uma *DHT* (*Distributed Hash Table*) para trocar informações entre os nós. Segundo Singh, Srivatsa *et al.* (2003), a opção de distribuir as tarefas de *crawling* através da Internet também proporcionou a oportunidade de aproximar geograficamente os *crawlers* aos servidores Web, resultando em um ganho significativo no aumento de velocidade durante o processo de *crawling*. Alguns desafios para modelar um *crawler* descentralizado são apresentados por Singh, Srivatsa *et al.* (2003):

- Divisão de serviços
    - Essa questão é mais importante para *crawlers* descentralizados do que para *crawlers* monolíticos. *Crawlers* distribuídos capturam distintas porções da Web durante todo o tempo em que estão em atividade, porém é importante criar otimizadores baseados em distribuição geográfica não somente para distribuir as porções da Web entre os nós utilizando os valores *hash* dos domínios, como também utilizando uma política de regionalização, que ocorre caso um nó *A* identifique que uma determinada URL a ser capturada pertença a um domínio que esteja sobre responsabilidade de um nó *B*. Então, através de um processamento *batch*, o nó *A* envia essa URL para o nó *B* (Singh, Srivatsa et al., 2003).
- Verificação de duplicidades
    - Mesmo considerando que cada nó captura distintas porções da Web, eles ainda encontram URL's e conteúdos duplicados. No caso da não existência de somente um repositório de URL's capturadas e conteúdos extraídos, os nós precisam manter uma

> comunicação entre eles e decidir qual a próxima URL a ser capturada. É extremamente importante manter esse custo de comunicação o menor possível, devido existência de um grande número de nós em um processo de *crawler peer-to-peer*. Torna-se interessante o uso de protocolos de comunicação baseados em *DHT*, onde estes conseguem o tempo de pesquisa em torno de O(log *N*), onde *N* é o número total de nós.

O protocolo de comunicação entre os nós utilizado por Singh, Srivatsa *et al.* (2003) é fundamentalmente baseado em *DHT*, e segue uma implementação simples, contendo dois tipos de chaves – URL's e conteúdo de páginas. Um nó é responsável pela captura de uma URL e pelo conteúdo da página, isso significa que o nó tem a informação da localização do conteúdo. Porém o processamento do conteúdo não necessariamente precisa ser realizado pelo mesmo nó. Um localizador de URL, o qual utiliza um protocolo de comunicação pode ser feito da seguinte forma: primeiro é obtido o valor *hash* do nome do domínio da URL, em seguida o protocolo executa a pesquisa desse valor e retorna o endereço IP do nó responsável por esta URL. É importante perceber que a escolha resultará em um único domínio a ser capturado por um nó no sistema.

Para verificar a existência de páginas duplicadas, Singh, Srivatsa *et al.* (2003) utilizou o conteúdo da página como chave. Para isso, o valor *hash* do conteúdo da página é obtido, e uma pesquisa é iniciada para localizar o nó responsável por esse conteúdo.

Em uma natureza de sistema *peer-to-peer*, o protocolo deve gerenciar dinamicamente as operações de entrada e saída de nós no sistema. Sempre quando um nó se juntar ao sistema são atribuídas responsabilidades para ele, no contexto de *crawler*, o novo nó passa a ser responsável por domínios que serão direcionados para ele. Os nós que estavam com a responsabilidade sobre esses domínios anteriormente são requisitados para enviar as listas de URL's já capturadas para o novo nó. Quando um nó *P* quiser deixar a rede, ele envia todas as URL's e conteúdos de páginas já capturadas para outros nós. Entretanto se um nó *P* falhasse, ele não seria capaz de enviar qualquer informação para outros nós, sendo perdidas temporariamente as informações gerenciadas pelo nó *P*. Esse problema pode ser contornado através da replicação de informação. Todo domínio *D* é mapeado para múltiplas chaves, as quais são gerenciadas por distintos nós.

No contexto de *crawlers*, a utilização de *bloom filters* possibilita a obtenção da informação sobre uma URL, se esta tem sido capturada ou não. Com o uso *de bloom filters* é possível evitar o armazenamento de todas as URL's em memória, deixando somente as URL's que estão sendo capturadas (Singh, Srivatsa et al., 2003).

Existem três mecanismos que tornam o trabalho apresentado por Singh, Srivatsa *et al.* (2003) um sistema *peer-to-peer*:

- Nós em Sociedade
    - Permite os nós comunicarem-se diretamente com outros nós para distribuir tarefas ou trocar informações. Um novo nó pode juntar-se ao sistema estabelecendo contato com um nó existente na rede.
- Protocolo
    - Esse segundo mecanismo inclui o particionamento dos serviços de *crawling* e o algoritmo de localização. Todos os nós participantes no processo de *crawling* e qualquer nó podem submeter uma nova URL para ser capturada. Quando uma nova URL é encontrada por um nó, este primeiro determina qual nó será o responsável por executar o processo de *crawling* desta URL. Isso é possível através da operação de localização provida pelo sistema.
- *Crawling*
    - O terceiro mecanismo é o processamento da URL. Cada tarefa de executar o processo de *crawling* para uma URL é atribuído a um nó com um identificador que corresponde ao domínio da URL. Baseado em um identificador, URL's são capturadas e seus nós responsáveis podem migrar essa tarefa para outros nós caso seja necessário.

A arquitetura de cada nó do sistema descrito por Singh, Srivatsa *et al.* (2003) é composta por três principais elementos:

- Armazenamento
    - Esse elemento contém todos os dados estruturados que cada nó terá que manter. São eles:
        - *Crawl-Jobs*

- É a lista de URL's que ainda não foram capturadas pelo nó.
- *Processing-Jobs*
  - É a lista de URL's que ainda serão processadas pelo nó.
- *Seen-URLs*
  - É a lista de URL's que já foram capturadas.
- *Seen Content*
  - É a lista de páginas que já tenham sido processadas por algum nó na rede.
- *Routing Table*
  - é utilizado para rotear a localização das pesquisas para o nó certo, e conter informações sobre um pequeno número de nós no sistema.

- *Workers*
  - Esse elemento consiste do módulo que captura as páginas da Web e as processa extraindo as URL's. Esse elemento é formado por dois segmentos:
    - *Downloaders*
      - O *Downloader* procura um domínio a partir da lista de *Craw-Jobs* e pesquisa por um nó que poderá capturar o domínio mais rápido.
    - *Extractor*
      - O *Extractor* obtém uma página a partir da lista *Processing-Jobs* e extrai as URL's existentes.
- Interface
  - Esse elemento forma a interface de cada nó, gerenciando as informações com o sistema. Essa comunicação segue os seguintes cenários:
    - Envio de URL's
      - Distribuição de URL's para os nós responsáveis.
    - Verificação de páginas duplicadas

- Verificar a existência de páginas já capturadas entre os nós.

O trabalho realizado por Singh, Srivatsa *et al.* (2003) mostra um *crawler* descentralizado e distribuído em uma rede *peer-to-peer*. Foi descrita toda a estrutura do sistema e técnicas adotadas.

2.4 – Arquitetura de *crawler*

Diversas arquiteturas de *crawler* são descritas na literatura, porém a maior parte tem como inspiração a arquitetura definida por Heydon & Najork (1999), que divide as tarefas de *crawling* em diferentes tarefas, e estas tratam separadamente de módulos específicos. A separação de tarefas por Heydon & Najork (1999) deu origem aos seguintes componentes: *Frontier*, *DNS Resolver*, componente de processamento e componente para baixar páginas da Web.

2.4.1 – Visão conceitual da arquitetura de um crawler

A Figura 3 mostra a arquitetura conceitual de um *crawler*.

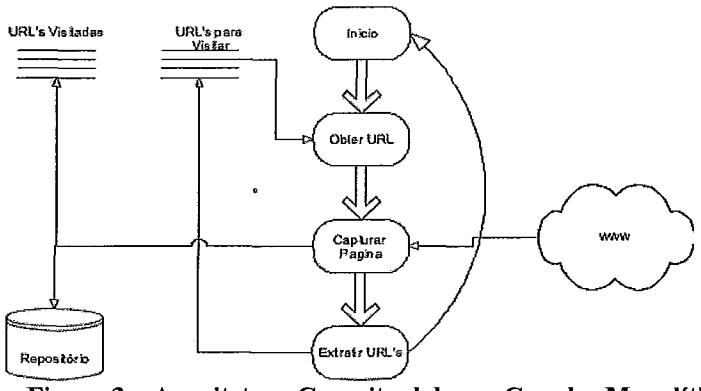


Figura 3 – Arquitetura Conceitual de um Crawler Monolítico

As setas representam o procedimento operacional que o *crawler* segue. O *crawler* mantém duas estruturas de dados para gerenciar as URL's: URL's a serem visitadas e URL's visitadas. As URL's visitadas mantêm a lista de páginas capturadas, isso é importante, pois evita com que o *crawler* realize a baixa de uma mesma página da Web varias vezes durante um mesmo ciclo de atuação. A fila de URL's a serem visitadas contém a lista de páginas a serem capturadas futuramente. O conteúdo inicial da fila chamada de URL's a serem visitadas, é conhecida como lista de sementes. Essa lista é expandida a todo momento. Antes do *crawler* executar a

primeira vez, a lista de sementes é especificada manualmente, ou obtida de alguma fonte. A lista inicial deve cobrir um conjunto variado de sites.

O *crawler* itera sobre a lista de URL's a serem visitadas, obtendo uma URL, capturando a página web correspondente, armazenando a página no repositório e colocando a URL dentro da fila de URL's visitadas. Conceitualmente o *crawler* extrai todos os *hyperlinks* de cada página capturada, converte todos os *hyperlinks* relativos em *hyperlinks* absolutos e verifica se alguns dos novos *hyperlinks* já foram capturados. Caso alguns dos *hyperlinks* já tenha sido capturado, este será descartado, ou será adicionado à fila de URL's a serem visitadas. Esse processo é repetido até o *crawler* ter consumido todas as URL's existentes na fila de URL's a serem visitadas.

No caso de falha na captura de URL, o *crawler* retorna com a URL para a fila de URL's a serem visitadas. O *crawler* deve cancelar a operação de captura no caso de tentar repetidamente capturar um determinada URL e não obter sucesso, pois isso pode representar problemas de conexão com o servidor Web ou até mesmo uma armadilha para o *crawler*.

Uma arquitetura de *crawlers* paralelos pode ser implementada através da inicialização de diversos processos para diferentes sites, ou capturando diversos sites dentro de um único processo. Uma estratégia híbrida desses dois é opcional. Conforme ilustrado nas Figuras 4 e 5.

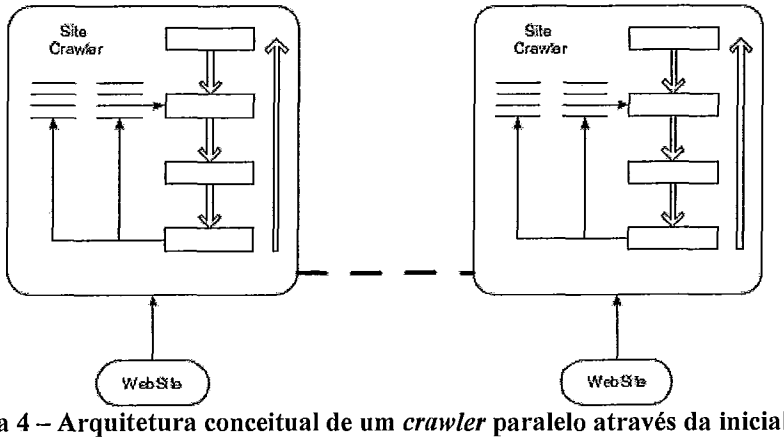


Figura 4 – Arquitetura conceitual de um *crawler* paralelo através da inicialização de diversos processos para diferentes sites

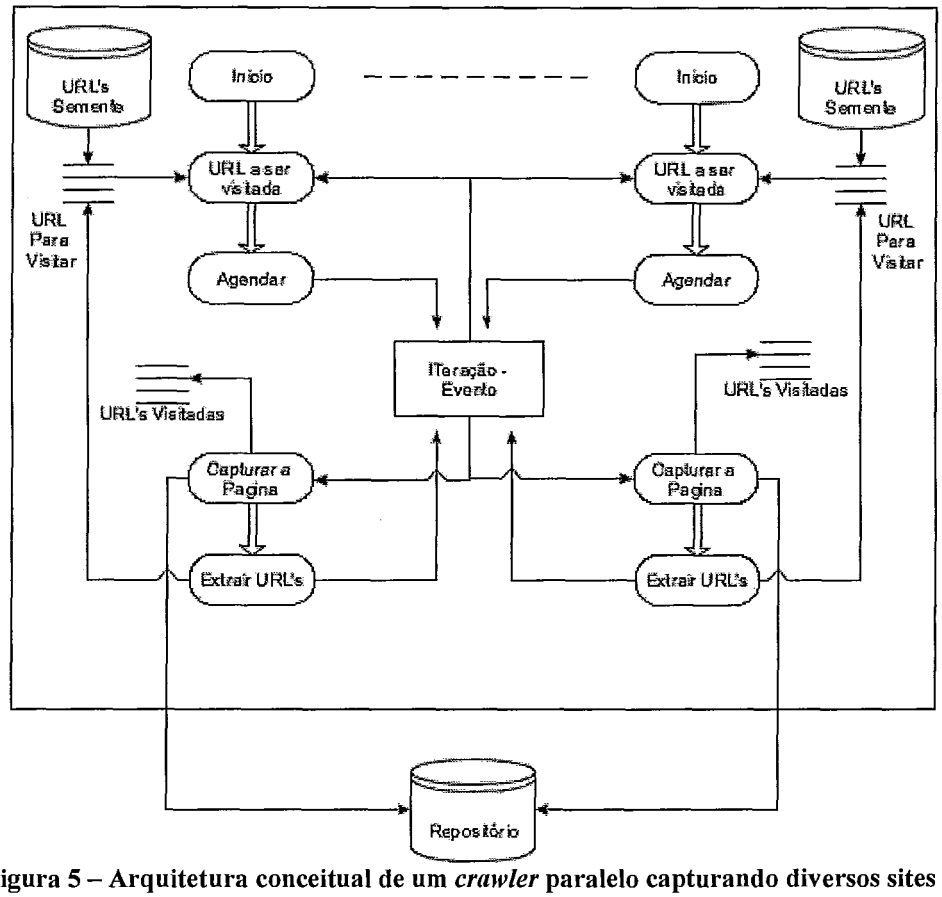


Figura 5 – Arquitetura conceitual de um *crawler* paralelo capturando diversos sites dentro de um processo único

2.4.2 – *Mercator* e sua arquitetura

Criado por Heydon & Najork (1999), o *Mercator* é um crawler modular, cujo o processo de *crawling* é realizado por múltiplas *threads*, onde cada *thread* repetidamente realiza os passos necessários para realizar a captura das páginas e processá-las. O primeiro passo é remover uma URL absoluta que inicia com "http" do *Frontier*. As configurações padrão incluem os protocolos *HTTP(HyperText Transfer Protocol)* e *FTP(File Transfer Protocol)*. Seguindo o mesmo esquema da URL, é escolhido o protocolo apropriado para realizar a captura da página. Logo após, é executado o componente que baixa a página da Web e a envia para um mecanismo de *cache*, o *RIS* (*Rewind Input Stream*), isso evita uma URL ser lida diversas vezes. O *RIS* é uma espécie de abstração de *I/O* (*Input/Output*) que é inicializada por uma *stream* de entrada e que permite ao conteúdo das *streams* ser lido mais de uma vez. Após a página ser adicionada ao *RIS*, a *Thread* executa *o Content-Seen*, o qual determina se a página foi visitada anteriormente. Caso tenha sido visitada anteriormente, a página não é processada e a *Thread* remove a URL do *Frontier*.

Toda página capturada tem um tipo *MIME* (*Multipurpose Internet Mail Extensions*) vinculado, de forma a associa-la ao componente de processamento. Um componente de processamento é basicamente uma abstração do processamento de páginas as quais já foram capturadas, como por exemplo, extrair *hyperlinks*, contar as TAG's encontradas no HTML, coletar estatísticas sobre imagens, entre outros.

Existe uma instância separada de cada componente de processamento por *Thread*. Por padrão associa-se o componente de processamento ao tipo *MIME* para que seja possível extrair os *hyperlinks* existentes em uma página. Cada *hyperlink* é convertido em URL absoluta. Caso a URL passe por um filtro apresentado por Heydon & Najork (1999), a *Thread* realizará uma verificação para averiguar se a URL foi visitada anteriormente, e caso não tenha sido, esta será adicionada ao *Frontier*.

A estrutura de dados que contem todas as URL's que serão visitadas chama-se *Frontier*. Essas URL's ficam organizadas na forma de fila. Atualmente o tamanho do *Frontier* de um *crawler* que esteja obtendo dados a partir da Web, pode ser muito grande. Para isso URL's são armazenadas em disco. Para minimizar o custo de leitura e escrita em disco, a fila mantém um tamanho fixo em memória, aproximadamente 600 URL's, e o resto fica armazenado em disco. O *Frontier* é um componente adicionado ao *Mercator*. Ou seja, pode ser facilmente trocado por outras implementações (Heydon & Najork, 1999).

O *Mercator* (Heydon & Najork, 1999) permite que um mesmo documento seja processado por vários componentes de processamento. O *RIS* lê e armazena no *cache* o conteúdo do arquivo fornecido pela URL, armazenando os documentos pequenos (até 64 *Kilobytes*) na memória enquanto documentos maiores são armazenados em um arquivo no disco.

Sendo cada vez mais comum URL's diferentes armazenarem o mesmo conteúdo na Web, fazendo com que os *crawlers* capturem e processem o mesmo documento mais de uma vez, o *Mercator* (Heydon & Najork, 1999) utiliza o *Content Seen Test*, para decidir se o documento foi processado anteriormente. Isso reduz significativamente o número de documentos que serão processados e até mesmo capturados, através da identificação de URL's repetidas. O *Mercator* (Heydon & Najork, 1999) também disponibiliza uma maneira customizável de controlar as URL's que serão visitadas.

Antes de adicionar as URL's no *Frontier*, a *Thread* consulta um componente de filtro. Se a URL for válida, ela entra na fila. O *Mercator* (Heydon & Najork, 1999) possui uma coleção de diferentes filtros os quais facilitam a restrição de URL's por domínios, prefixos, ou tipo de protocolo e também a possibilidade de computar conjunção, disjunção, ou negação de outros filtros. Usuários podem também fornecer seus próprios tipos de filtros. Um outro ponto interessante apresentado por Heydon & Najork (1999) foi a redução do gargalo existentes na maioria dos *crawlers*, referente à resolução de *DNS*. Antes de acessar um servidor Web, o *crawler* dever usar o *DNS* para mapear o nome do domínio em endereço IP, e *Mercator* (Heydon & Najork, 1999) aliviou esse gargalo através da criação de um mecanismo de *cache*.

2.5 – Exemplos de *crawlers*

Atualmente existem diversas publicações e implementações de *crawlers*. Essa seção apresenta alguns importantes *crawlers*, os quais tiveram publicações realizadas sobre seu funcionamento.

2.5.1 – RBSE

Criado por Eichmann (1994), o RBSE foi o primeiro *crawler* publicado. Foi baseado em dois programas: O primeiro, "*spider*", que mantinha uma fila em uma base de dados relacional e o segundo programa, "*mite*", que é um navegador de Internet ASCII modificado o qual baixava as páginas web.

2.5.2 – WebCrawler

Criado por Pinkerton (2000), foi usando para construir o primeiro índice de páginas de uma porção da WEB. Era baseado em *lib-www* para baixar as páginas, e outro programa para realizar a interpretação e ordenar as URL's, de forma que pudesse posteriormente realizar uma busca utilizando a estratégia *breadth-first*.

2.5.3 – World Wide Web Worm

Criado por McBryan (1994), foi um *crawler* usado para construir um simples índice de título de documentos e URL's. Esse índice pode ser explorado usando comandos *grep* Unix.

2.5.4 – Google Crawler

Criado por Brin & Page (1998) é descrito com alguns detalhes, mas a referência é somente sobre uma versão mais nova da arquitetura a qual é baseada em C++ e Phyton. O *crawler* está integrado com o processo de indexação, pois a interpretação do texto foi feita para uma indexação usando *full-text* e também para extração de URL's. Existe um servidor de URL o qual envia uma lista de URL's para ser visitada pelo *crawler*. Durante a interpretação, as URL's encontradas são passadas para um servidor que verifica se a URL já foi visitada, caso não tenha sido, a URL é adicionada a fila no servidor de URL.

2.5.5 – CobWeb

Criado por Altigran, Eveline *et al.* (1999) usa uma central "*scheduler*" a qual é responsável por gerenciar o tempo entre uma visita e outra em um mesmo site e por uma série de coletores distribuídos. Os coletores realizam a interpretação das páginas capturadas e envia as URL's extraídas para o "*scheduler*", o qual é o responsável por enviá-las aos coletores. O *scheduler* usa a estratégias *breadth-first* com uma política de acesso, para evitar sobrecargas nos servidores Web. Este *crawler* foi escrito em Perl.

2.5.6 – *Mercator*

Conforme detalhado na seção "2.4.2 – *Mercator* e sua Arquitetura", o *Mercator* foi criado por Heydon & Najork (1999), é um *crawler* modular escrito em Java. Baseia-se no uso de "*protocol modules*" e "*processing modules*". *Protocols modules* estão relacionados em como adquirir as páginas HTTP, e *processing modules* estão relacionados em como processar as páginas Web. O padrão do *processing module* somente interpreta as páginas e extrai novas URL's, mas outros *processing modules* podem ser utilizados para indexar o texto das páginas, ou capturar estatísticas da Web.

2.5.7 – WebFontain

Criado por Edwards, McCurley *et al.* (2001), é um *crawler* distribuído similar ao *Mercator*, entretanto, foi escrito em C++. Tem características de uma máquina "controladora" a qual coordena uma série de máquinas "escravas".

2.5.8 – PolyBot

Criado por Shkapenyuk & Suel (2002), é um *crawler* escalável o qual pode baixar centenas de páginas por segundo. O sistema é flexível o bastante, de forma a poder utilizar diferentes estratégias. *PolyBot* executa em uma rede *Solaris* ou estações Linux, podendo ser escalado através da adição de máquinas. Este *crawler* foi escrito em C++ e Phython, o qual é composto por um "gerenciador de *crawl*", um ou mais mecanismos para baixar páginas da Web e um ou mais mecanismos capazes de resolver os nomes dos servidores. As URL's coletadas são adicionadas a uma fila no disco e processadas posteriormente. A política de acesso considera segundo e terceiro níveis de domínio, por exemplo, www.ufrj.br e www2.ufrj.br são domínios de terceiro nível, pois domínios de terceiro nível são hospedados pelo mesmo servidor Web.

2.5.9 – WebBase Crawler

O WebBase *crawler* (Cho, Garcia-Molina et al., 2004) faz parte de um projeto experimental desenvolvido na universidade de Stanford. O *crawler* esta distribuído através de módulos, agindo em paralelo. Esse *crawler* atua constantemente na otimização de recursos, políticas de visita, independência entre os processos paralelos de captura e resolução de *DNS*.

2.5.10 – FAST Crawler

Construído por Risvik, M. *et al.* (2002) ele executa em uma arquitetura distribuída na qual cada máquina possui um organizador de documentos responsável por manter uma fila destes a serem capturados por um processador, o qual os armazena em um subsistema.

2.5.11 – WebRACE

Criado por Dikaiakos & Zeinalipour-Yazti (2001), é um módulos de *crawling* e *caching* implementado em Java, e usado como parte de uma sistema mais genérico chamado eRACE. O sistema recebe requisições de usuários para baixar paginas da Web, então o *crawler* atua em parte como um *proxy* inteligente. O sistema também manipula requisições para "assinaturas" de páginas Web que devem ser monitoradas: quando as páginas mudam, elas devem ser visitadas novamente pelo *crawler* e o

assinante deve ser notificado. O maior diferença do WebRace é que, enquanto muitos *crawlers* começam com uma semente, WebRace vem constantemente recebendo novas URL's para realizar a captura de páginas.

2.5.12 – Ubicrawler

Criado por Boldi, Codenotti *et al.* (2002), é um *crawler* distribuído escrito em Java e não tem um processo central. É composto de um número de agentes idênticos. A função a qual é responsável por distribuir as páginas é calculada usando uma função de *hash* dos domínios. Isso significa que uma página não é visitada por agentes diferentes, a não ser que ocorra uma falha no sistema de coordenação dos agentes. Este *crawler* foi planejado para obter uma grande escalabilidade e para ser tolerante a falhas.

2.5.13 – WIRE

Desenvolvido por Baeza-Yates & Castillo (2002) e utilizado na Universidade de Pisa, Itália, é um *crawler* escalável, totalmente configurável, possui uma bom desempenho, e distribuído através de licença de software livre. Foi escrito em C/C++ e projetado para trabalhar com grandes volumes de documentos e manipular milhares de requisições *HTTP* simultaneamente.

Capítulo 3 – Sistemas *peer-to-peer* e o particionamento da Web

O conceito de *crawlers* colaborativos consiste em um grupo de nós, onde cada um realiza um processo de *crawling*, sendo este responsável por uma porção específica da Web, ou participando de um conjunto de nós responsáveis por uma porção. Uma das grandes dificuldades na implementação de *crawlers* distribuídos, os quais colaboram uns com os outros, é a escolha de uma estratégia para particionar a Web entre os nós que realizam o processo de *crawling*. Uma estratégia de partição precisa levar em consideração questões como minimizar a superposição entre as atividades dos nós. *Crawlers* distribuídos não podem operar inteiramente independentes.

Colaboração é necessário para evitar esforços duplicados em diversos pontos, como reduzir ou eliminar o número de páginas visitadas por mais de um *crawler*. Em um caso extremo, sem a existência de colaboração, todos os *crawlers* deveriam executar capturas idênticas, onde cada página seria visitada na mesma ordem. Porém, *crawlers* distribuídos sobre uma rede *peer-to-peer*, devem focar em subconjuntos da Web, onde cada um destes é fruto de uma ação para setorizar a Web, atribuindo-os aos diversos *crawlers* existentes. Colaboração também é necessário para identificar e lidar com conteúdos duplicados Cho, Shivakumar *et al.* (2000). Frequentemente diversas URL's podem ser utilizadas para referenciar o mesmo site, em alguns casos, um grande conjunto de paginas será encontrado repetidamente durante o processo de *crawling*, e o *crawler* deve evitar processar cada copia na sua totalidade.

3.1 – Particionamento da Web

Crawlers distribuídos superam importantes limitações dos tradicionais sistemas de *crawlers* baseados em um processo único. Dividir a Web entre diversos nós que representam *crawlers* em uma rede *peer-to-peer*, poderá trazer uma gama de vantagens para o processo de *crawling* em conjunto, tais como: evitar a captura de páginas duplicadas, evitar o processamento de páginas duplicadas, realizar o balanceamento de carga entre os nós (consumo de banda e de CPU), capturar as

páginas próximas aos nós, dentre outras. A seguir são descritas duas estratégias de particionamento.

3.1.1 – Particionamento através do uso da função de *hash*

A estratégia mais popular para implementar o particionamento do espaço da Web entre os *crawlers*, é através da computação de uma função de *hash* sobre URL's e o conteúdo das páginas. Quando um *crawler* extrai uma URL de uma página capturada, sua representação é primeiro normalizada, em seguida converte-a em uma URL absoluta. Uma função de *hash* é computada sobre a URL normalizada, a qual é alocada para um ou *n crawlers*. Se o *crawler* alocado esta em um nó remoto, a URL é transferida para o nó. Uma vez que a URL esteja no nó correto, ela deve ser atribuída à fila de URL's pendentes de captura. Similarmente cada *crawler* computa uma função de *hash* sobre o conteúdo de cada página capturada. Apesar de diversas transferências entre os *crawlers* não poderem ser evitadas, os *crawler* devem manter informações das URL e das páginas processadas, enviadas anteriormente, de forma a evitar transferências desnecessárias.

A função de *hash* não necessita que a URL seja computada por completo, por exemplo, a função pode ter como base o nome do domínio, garantindo com isso que todas as URL's de um mesmo domínio sejam alocadas para um mesmo *crawler* para serem capturadas (Cho, 2001), permitindo que o acesso ao servidor seja melhor controlado, evitando consumo que contrarie às políticas de visita, conforme explicado na seção "4.3.4 – Política de Visita". Similarmente, para o conteúdo da página, a função de *hash* estar baseada no conteúdo da URL normalizada, permite páginas duplicadas ou com conteúdos similares serem alocadas a um mesmo *crawler*.

Assumindo que o valor processado por uma função de *hash* para conteúdo de uma página especifica é independente do valor processado por uma função de *hash* para a página que a referência. Chung & Clarke (2002) propuseram a colaboração entre *crawlers* baseada em tópicos, através do uso de um classificador de textos, onde as páginas seriam alocadas a cada nó. A partir do conteúdo de uma página, o classificador determina uma ou mais categorias para a mesma. Cada categoria é associada a um *crawler*. Quando o classificador aloca uma página para um nó remoto, o *crawler* a transfere para o nó alocado. Uma colaboração entre *crawlers* orientada a tópicos deve ser vista como um conjunto de vários *crawlers* focados que particionam a Web entre eles.

3.1.2 – Particionamento geográfico

O espaço da Web também pode ser dividido através de uma estratégia de localidade, baseando-se no cálculo da localização geográfica de um *site*. Em um ambiente de *crawlers* distribuídos, diferentes sub-espaços da Web são atribuídos para cada *crawler*. Cada sub-espaço é um escopo geográfico contendo *sites* dentro do mesmo escopo. A principal informação utilizada para atingir o balanceamento de carga entre os *crawlers* distribuídos é a localização dos servidores Web, onde é assumido que uma página Web esta altamente correlacionada com a posição geográfica dos servidores. Diversos experimentos têm apresentado que resultados de baixar as páginas Web e troca de informação entre os *crawlers* tem uma significativa melhora a partir da introdução do uso de particionamento geográfico, conforme Exposto, Macedo *et al.* (2005).

Gao, Lee *et al.* (2006) propuseram uma extensão à estratégia de particionamento geográfico, onde definiram uma estratégia de *crawler* focados e colaborativos, distribuídos geograficamente considerando aspectos como os endereços das páginas Web, conteúdo das páginas Web, conteúdo das âncoras dos *hyperlinks*, entre outros. Esses *crawlers* tem como foco capturar páginas Web sobre conteúdos específicos das regiões geográficas que os *crawlers* estejam. Os *crawlers* focados e sensitivos a localização geográfica são responsáveis pela captura de páginas Web, pertinentes a um sub-conjunto de informações que envolvam cidade/estado. *Crawlers* em geral são responsáveis pela captura de páginas desassociadas do contexto cidade/estado. A utilização do endereço das páginas Web para atribuir a responsabilidade de captura a um determinado *crawler*, é possível caso um endereço contenha informações sobre cidade/estado na formação da URL, onde será direcionada diretamente para um *crawler* responsável. Outra forma para atribuir a responsabilidade de captura a um determinado *crawler*, é através do uso do endereço *IP*, sendo possível localizar a posição geográfica na qual os servidores Web estejam. As colaborações baseadas em endereço *IP*, exploram essa informação para controlar o quão distante está uma página Web a ser capturada de um *crawler*.

Isso ocorre através da obtenção do endereço *IP* da página Web a ser capturada, a partir desse IP pode ser utilizada uma ferramenta para mapear a localização geográfica do servidor Web, por exemplo cidade/estado, e com isso atribuir a captura ao *crawler* responsável pelo sub-espaço da Web o qual contenha a localização

geográfica identificada. Uma ferramenta para mapear a localização geográfica a partir do endereço *IP* e que o uso de sua *API* (*Application Programming Interface*) encontra-se disponível gratuitamente, pode ser acessada em *hostip.info*.

Os experimentos realizados por Gao, Lee *et al.* (2006), mostram a inexistência de superposição para estratégias de distribuição de *crawlers* baseadas em endereço *IP* e função de *hash,* ou seja, não existem páginas Web repetidas para serem capturadas por *crawlers* diferentes. Isto é uma questão importante em relação ao desempenho. Também é mostrada a baixa taxa de sobrecarga em comunicação entre os *crawlers* para estratégias de distribuição de *crawlers* baseadas em endereço *IP*.

A distribuição de *crawlers* reduz eficientemente o gargalo de redes na captura de páginas Web para as máquinas de busca e consequentemente eleva a qualidade dos resultados apresentados. Distribuir *crawlers* utilizando um mecanismo que facilite a delegação de páginas Web para cada *crawler* que esteja mais próximo em termos de velocidade no processo de baixar as páginas, proporciona um melhor uso dos recursos de rede. A idéia de *proximidade* e *localidade* são importantes e esses dois tipos de distâncias não são sempre análogas, pois proximidade age em termos da distância de rede (latência) enquanto localidade age em termos físicos, ou seja, distância geográfica. Segundo Papapetrou & Samaras (2004) máquinas de busca modernas ao invés de tentar reduzir a latência de rede para cada página Web que tentam capturar, abrem múltiplas *threads* de *crawlers* que concorrem sobre a mesma requisição *HTTP*. Entretanto essa estratégia não é perfeita, pois não evita o gargalo de rede, ao contrário de *proximidade*, a qual baixa as páginas Web mais rápido. Papapetrou & Samaras (2004) utilizaram *RIR's* (*Regional Internet Registries*) para calcular a *proximidade,* que são organizações sem fins lucrativos cuja tarefa é gerenciar endereços *IP* organizados em regiões. A idéia básica implementada por Papapetrou & Samaras (2004) é executar a delegação de cada URL para o *crawler* distribuído mais próximo, levando em consideração dados coletados através das *RIR's*. Com isso é assumida uma baixa latência de rede para o processo baixar as páginas Web, pois os *crawlers* distribuídos estão atuando em classes de endereço *IP*, sendo organizados em hierarquias em conjunto com as URL's a serem capturadas.

3.2 – Localização dos nós

O problema de localização é simples de reportar: dado um item *X* armazenado em algum conjunto de nós dinâmicos no sistema, localize-o. Este problema é um

importante entre os problemas de sistemas distribuídos, sendo crítico em sistema *peer-to-peer* (Balakrishnan, Kaashoek et al., 2003).

Uma abordagem é manter um servidor central que mapeie todos os nós distribuídos, porém essa abordagem apresenta problemas de escalabilidade e recuperação de falhas, pois o servidor é um ponto central de falha. A abordagem tradicional para conseguir escalabilidade é usar hierarquia. A desvantagem dessa abordagem é que a falha ou remoção de um nó raiz ou um nó suficientemente alto na hierarquia pode ser catastrófico. Essas abordagens são exemplos de estruturas de localização, onde cada nó tem um conjunto bem definido de informações sobre os outros nós do sistema. A vantagem dos métodos de localização dessas estruturas, é que estes garantem que os dados podem ser localizados no sistema, uma vez que tenham sido armazenados.

Segundo Balakrishnan, Kaashoek *et al.* (2003) para superar os problemas de tolerância a falhas desses esquemas, alguns sistemas *peer-to-peer* desenvolveram a noção de simetria nos algoritmos de localização. Diferente de hierarquia, nenhum nó é mais importante do que outro no processo de localização, e cada nó é envolvido somente em uma pequena fração no caminho da pesquisa no sistema. Um recente grupo de algoritmos *peer-to-peer*, incluindo os de: Ratnasamy, Francis *et al.* (2001) - CAN -, Stoica, Morris *et al.* (2003) - Chord -, Maymounkov & Mazieres (2002) - Kademlia - e Rowstron & Druschel (2001) - Pastry -, são ambos estruturados e simétricos. Isso os permite oferecer garantias de funcionamento enquanto simultaneamente não tornam-se vulneráveis caso ocorram falhas em nós. Todos esses algoritmos mencionados são implementações de *DHT* (*Distributed Hash Table*).

A utilização de *DHT* é um mecanismo atraente para localização de nós distribuídos. O principal requisito é que dados são identificados usando chaves numéricas e que cada nó estará preparado para armazenar chaves para cada outro nó. Uma *DHT* implementa apenas uma operação: *lookup(key)* que fornece a identificação (exemplo: endereço *IP*) de um nó corrente responsável pela chave fornecida. Qualquer aplicação que queira publicar uma informação através de um nome único, converteria o nome para uma chave numérica usando uma função de *hash* como por exemplo SHA-1, e então executaria *lookup(key)*.

O publicador enviaria a informação para ser armazenada no nó resultante. Qualquer nó que desejar ler a informação armazenada bastará converter o nome para uma chave numérica e chamar *lookup(key)*, e pedir o resultado do nó para uma cópia

da informação. Um sistema completo de armazenamento teria que cuidar da replicação, *caching,* autenticação entre outros; esses aspectos estão fora do escopo inicial do problema de localização.

Para implementar *DHT's*, os algoritmos de localização necessitam prover as seguintes características:

- Mapeamento de chaves para nós em uma estrutura de balanceamento de carga
    - Todos os algoritmos de localização fazem isso essencialmente da mesma forma. Nós e chaves são mapeados utilizando uma função de *hash* em uma *string* de dígitos. Uma chave gerada com a função de *hash* para uma *string s* fornecida é então atribuída para o nó mais próximo, esse nó é o sucessor numérico para *s*.
- Redirecionando a localização de uma chave para o nó apropriado
    - Qualquer nó que recebe uma pesquisa para uma determinada chave deve estar habilitado para redirecionar a chave para o nó cujo ID é mais próximo. Isto garantirá que a pesquisa eventualmente chega no nó mais próximo. Para que isso seja conseguido, cada nó mantém uma tabela de roteamento com um pequeno número de outros nós cuidadosamente selecionados. Se o ID da chave é maior do que o nó corrente, pode ser feito um redirecionamento para o nó que é maior que o nó atual, entretanto, ainda menor do que o ID da chave.
- Construindo tabelas de roteamento
    - Para redirecionar localização de mensagens, cada nó precisa conhecer sobre outros nós. Para prover o primeiro redirecionamento seguindo a regra de obter o nó mais próximo numericamente para o ID da chave, cada nó deve conhecer seu sucessor. Esse sucessor é um nó válido para receber o redirecionamento de qualquer chave que tenha o ID maior do que o ID do nó corrente.

Existem diferenças substanciais entre a forma como cada algoritmo constrói e mantém suas tabelas de roteamento e a forma como cada nó entra ou deixa a rede. A

diferença mais importante entre esses algoritmos é a estrutura de dados que utilizada como uma tabela de roteamento para prover O(log *N*) pesquisas. Stoica, Morris *et al.* (2003) mantém uma estrutura de dados capaz de agir com uma tabela auxiliadora. Cada nó apresentado em Maymounkov & Mazieres (2002) ou Rowstron & Druschel (2001) mantém uma estrutura de dados baseada em arvore. Ratnasamy, Francis *et al.* (2001) utilizam uma estrutura multidimensional baseada em coordenadas cartesianas, onde cada nó mantém uma tabela de roteamento de todos os seus vizinhos numa coordenada de espaço.

3.2.1 – CAN (*Content Addressable Network*)

Ratnasamy, Francis *et al.* (2001) utilizam um espaço de coordenadas cartesianas com *d* dimensões para qualquer *d* fixo para implementar a *DHT*. O espaço de coordenadas é dividido em retângulos conhecidos como zonas. Cada nó no subsistema é responsável por uma zona, onde estes nós são identificados pelos limites de suas zonas. Uma chave é mapeada diretamente para um ponto no espaço das coordenadas de uma zona, sendo armazenada pelo nó o qual zona contém as coordenadas do ponto. A Figura 6 mostra um CAN de duas dimensões com seis nós.

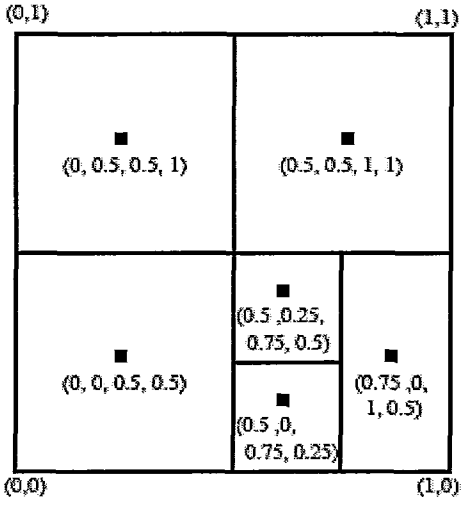


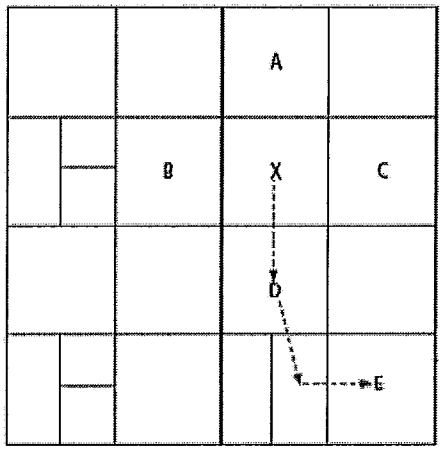
Figura 6 - CAN com seis nós em duas dimensões

Cada nó mantém uma tabela de roteamento de todos os seus vizinhos no espaço de coordenadas. Dois nós são vizinhos se suas zonas compartilham um hiperplano de dimensão $d - 1$. Uma mensagem do CAN contém as coordenadas de destino. Usando as coordenadas de vizinhos, um nó direciona a mensagem em direção ao seu destino usando um simples redirecionamento guloso para o nó vizinho que esta mais próximo do destino. CAN tem um desempenho de $O(d.N1/d)$. Os espaços de

coordenadas virtuais são utilizados para armazenar os seguintes pares **(chave, valor)**: para armazenar um par (C, V), chave *C* é determinantemente mapeada para um ponto *P* em um espaço de coordenadas usando uma função de *hash*. O protocolo de localização recupera uma entidade correspondente a uma chave *C*, onde qualquer nó pode aplicar uma função de *hash* para mapear *C* diretamente para o ponto *P* e recuperar o valor *V* correspondente de *P*.

Um novo nó que se junta ao sistema deve ter sua própria porção de espaço de coordenada alocada. Isso pode ser obtido através da divisão de zonas de nós já existentes no sistema, uma zona é dividida ao meio, sendo uma metade do nó responsável por esta zona e a outra metade para o novo nó. O novo nó solicita através de mecanismos do CAN o endereço *IP* de algum nó existente no sistema. O novo nó randomicamente escolhe um ponto P e envia um pedido para se juntar a rede do CAN. Cada nó existente utiliza o mecanismo de roteamento do CAN para despachar a mensagem até que esta alcance o nó que detém a zona ao qual o ponto P pertence. Uma vez localizado o nó que detém a zona desejada, este divide essa zona e atribui uma das metades ao novo nó.

Quando um nó deixa a rede do CAN, imediatamente um algoritmo de transferência de administração de zona, garante que um dos vizinhos do nó que saiu da rede obtenha o controle da zona. O nó atualiza seus vizinhos imediatamente, para que todos tenham ciência das novas coordenadas. Todos os nós no sistema enviarão atualizações de seus estados para garantir que todos os seus vizinhos estarão cientes das mudanças e atualizarão em seu próprio conjunto de vizinhos. A Figura 7 ilustra o roteamento do caminho de um nó X para apontar o nó *E* e a junção de um nó *Z* à rede CAN.



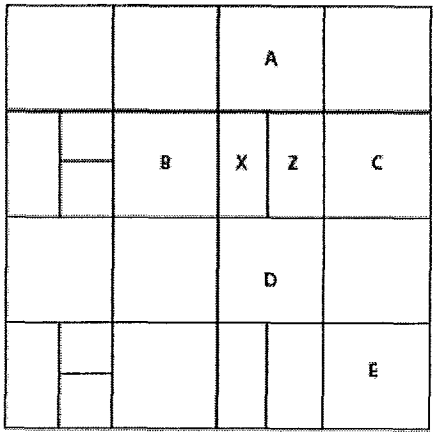


Figura 7 - Exemplo do espaço do CAN antes e depois da junção do nó Z

Para um espaço de dimensão *d* particionado em *n* zonas iguais, a média do tamanho do roteamento do caminho é $(d/4) \times (n^{1/d})$ passos e os nós individualmente mantêm uma lista de $2.d$ vizinhos. Desde que existam diferentes caminhos entre dois pontos no espaço, quando um ou mais nós vizinhos falham, um nó ainda pode rotear através de outros bons caminhos.

Segundo Ratnasamy, Francis *et al.* (2001) CAN pode ser utilizado em larga escala por sistemas de armazenamento. Esses sistemas requerem eficientes inserções e recuperações de conteúdos em grandes armazéns de dados distribuídos em rede com um mecanismo de indexação escalável.

3.2.2 – Chord

Stoica, Morris *et al.* (2003) apresentam um protocolo de localização distribuído para controle de entrada e saída de nós do sistema. Esse esquema descentralizado tende a balancear a carga no sistema, desde que cada nó receba aproximadamente o mesmo número de chaves e exista pouca movimentação destas quando nós entram ou saem do sistema.

Cada nó no Chord tem um identificador único de *m-bit*, obtido através de uma função de *hash* do endereço *IP* do nó e um índice virtual de nós. Na visão do Chord os identificadores estão ocupando um espaço circular. As chaves são mapeadas para os identificadores existentes no espaço do Chord, através da aplicação de uma função de *hash* para um identificador de *m-bit*. Chord define o nó responsável por uma chave para ser o sucessor do identificador da chave. O sucessor de um identificador *j* é o nó com o menor identificador, porém maior ou igual a *j*, tanto quanto no "*hash* consistente" apresentado por Karger, Lehman *et al.* (1997).

O "*hash* consistente" permite que os nós entrem e deixem a rede com o mínimo de movimentação de chaves. Para manter o mapeamento correto de sucessor quando um nó *n* passa a fazer parte da rede, certas chaves previamente atribuídas para o sucessor de *n*, passam a ser atribuídas a *n*. Quando um nó *n* deixa a rede, todas as chaves que estavam atribuídas a ele passam a ser de responsabilidade de seu sucessor.

Um nó existente no Chord utiliza duas estruturas de dados para executar a localização de chaves/nós: uma lista de sucessores e uma tabela de indicação. Somente a lista de sucessores é requerida para correções, então o Chord é cuidadoso em manter sua precisão. A tabela de indicação acelera o processo de localização, mas não necessita ser precisa, sendo assim o Chord é menos agressivo em manter essa tabela.

Todo nó do Chord mantém uma lista de identidades e endereço *IP* de seus *r* sucessores imediatos no anel do Chord. O fato de qualquer nó conhecer seu sucessor significa que um nó pode sempre processar uma localização corretamente: se a chave desejada está entre o nó e seu sucessor, o último nó é o sucessor da chave; de outra forma a localização pode ser redirecionada pelo sucessor , o qual move a localização diretamente para perto de seu destino.

Um novo nó *n* descobre quem são seus sucessores quando passa a fazer parte do anel do Chord, pedindo para um nó existente no anel executar uma pesquisa para os sucessores de *n*, então *n* solicita por um sucessor a sua lista de sucessores. A maior complexidade envolvida com a lista de sucessores está na notificação de um existente nó quando um novo nó deve ser seu sucessor.

Executar o esquema de localização somente com a lista de sucessores iria requerer uma media de *N/2* mensagens trocadas, onde *N* é o número de nós. Para reduzir o número de mensagens para *O(log N)*, cada nó mantém uma tabela auxiliadora com *m* entradas. A i^{th} entrada na tabela no nó *n* contém a identificação do primeiro nó que sucede *n* pelo menos 2^{i-1} no circulo de identificadores. Portanto cada nó conhece a identificação de nós no intervalo exponencial de 2 a partir de sua própria posição no círculo de identificadores. Um novo nó inicializa sua tabela de identificação pesquisando nós existentes. Nós existentes no Chord cuja tabela de identificação ou lista de sucessores deveriam fazer referência a um novo nó, descobrem sobre este através de atualizações periódicas utilizando o mecanismo de localização.

A Figura 8 mostra o pseudocódigo descrito por Stoica, Morris *et al.* (2003) para localizar o sucessor do identificador *id*. O laço principal é o *find_predecessor*, o qual envia a lista de nós precedentes para uma sucessão de outros nós; cada pesquisa remota procura na tabela de outros nós por nós ainda próximos de *id*. Pois os registros existentes na tabela de indicação apontam para nós em um intervalo da potência de 2 em torno do anel. Cada iteração atribuíra para *n'* um nó no anel localizado no meio do caminho entre o atual *n'* e o *id*. Desde que a lista de nós precedentes nunca retorne um identificador maior do que o *id*, este processo nunca irá além do sucessor correto. Isso deve terminar antes, especialmente se um novo nó recentemente juntou-se a rede com um identificador menor que *id*, neste caso a checagem para $id \notin (n', n'.sucessor]$ garante que *find_predesssor* persiste até ele achar o par de nós que circundam o *id*.

```
n.find_successor(id)
  n' = find_predecessor(id);
  return n'.successor;

n.find_predecessor(id)
  n' = n;
  while (id ∉ (n', n'.successor])
    n' = n'.closest_preceding_finger(id);
  return n';

n.closest_preceding_finger(id)
  for i = m downto 1
    if (finger[i].node ∈ (n, id))
      return finger[i].node;
  return n;
```

Figura 8 - Pseudocódigo para localizar o nó sucessor de um identificador.

A Figura 9 (adaptada de Stoica, Morris *et al.* (2003)) descreve o anel do Chord com $m = 6$, onde m é o número de registros na tabela de indicação. Esse anel em particular tem dez nós e armazena cinco chaves. O sucessor do nó 10 é o nó 14, logo a chave 10 será alocado no nó 14. Similarmente se um nó fosse se juntar com um identificador 26, ele armazenaria a chave com identificador 24 pertencente ao nó com identificador 32.

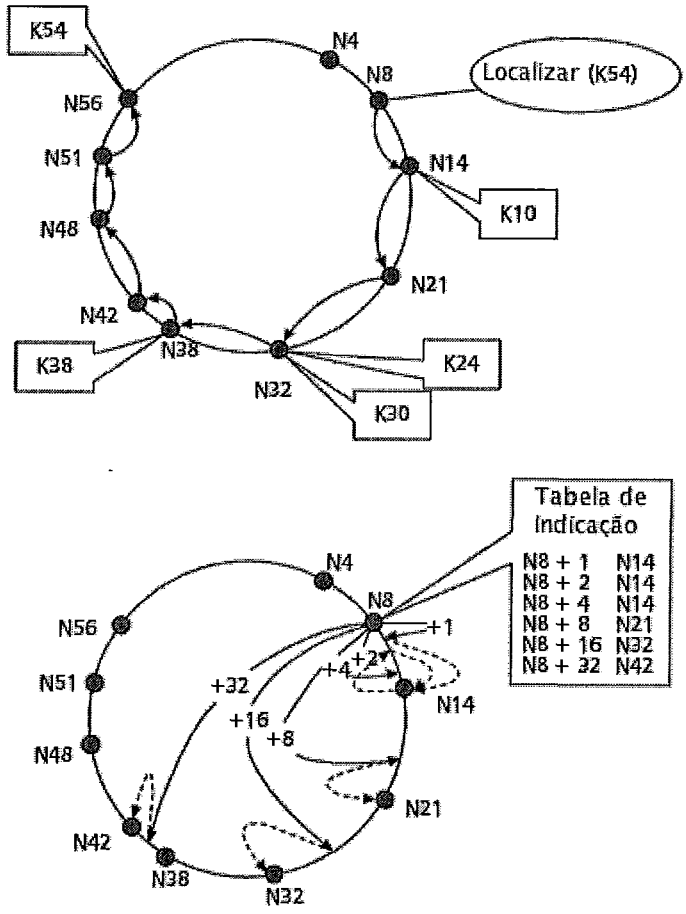


Figura 9 – Formação do Chord

O anel do Chord mostrado na figura acima, está consistindo de dez nós e cinco chaves. É apresentado o caminho percorrido por uma busca partindo do nó 8 para localizar a chave 54.

3.2.3 – Pastry

Rowstron & Druschel (2001) apresentam uma solução que tem se tornado a base de muitos sistemas *peer-to-peer*, onde os nós são armazenados em uma estrutura de árvore.

Pastry faz uso de prefixos no roteamento, para construir uma cobertura de rede descentralizada e auto-organizável. Para cada nó no Pastry é atribuído um identificador de 128 *bits*. O identificador é usado para dar a posição do nó em um espaço circular, com um intervalo de 0 até $2^{128} - 1$, este identificador é atribuído aleatoriamente quando o nó se junta ao sistema e está uniformemente distribuído em um espaço de 128 *bits*. Para uma rede de N nós, Pastry direciona uma chave para o nó numericamente mais próximo desta, em menos de $log_B N$ passos inferior a uma

operação normal (onde $B = 2b$ e b está configurado como um parâmetro com valor igual a 4). Os identificadores e chaves são considerados uma seqüência de dígitos com base B. O Pastry direciona mensagens para o nó cujo identificador é numericamente mais próximo de uma chave informada.

Cada nó no Pastry mantém uma tabela de rotas, um conjunto de vizinhos e um conjunto de nós filhos. Uma tabela de rotas de um nó é definida com $log_B N$ registros, onde cada registro mantém $B - 1$ números de entrada. O $B - 1$ números de entrada em um registro n da tabela de rotas, cada entrada faz referência a um nó cujo identificador compartilha com o identificador corrente os primeiros n dígitos. Cada entrada na tabela de rotas contém o endereço IP de nós cujo identificador tem um prefixo apropriado, e foram escolhidos com base na métrica de proximidade aplicada pelo Pastry.

Com o valor de $b = 4$ e 10^6 nós, uma tabela de rotas contém uma media de 75 entradas e o número de passos esperados para direcionar um nó a outro igual a 5. O conjunto de vizinhos, M, contém o identificador e endereço IP de $|M|$ nós que estão próximos ao nó local. Cada nó n mantém um conjunto de filhos L, o qual é o conjunto de $|L|/2$ nós mais próximos de n e maiores que n, e o conjunto de $|L|/2$ nós mais próximos de n e menores que n. A Figura 10 a seguir apresenta o estado de um nó.

Identificador 10233102

Filhos	menores	maiores	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Tabela de Rotas

-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Vizinhos

13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

Figura 10 - Rowstron & Druschel (2001) apresentam o estado de um nó.

Quando um nó com identificador X se junta ao sistema ele precisa iniciar suas tabelas e notificar os outros nós de sua presença. Para este nó fazer parte da rede

precisa conhecer o endereço de um nó qualquer para contato, então uma pequena lista de contatos baseada em uma métrica de proximidade (exemplo: *RTT – Response Time Trip*) é fornecida pelo Pastry, fazendo com que o novo nó possa escolher um contato qualquer de forma aleatória. Com isso o nó de identificador *X* passa a conhecer um nó de identificador *A*. O nó *X* solicita ao nó *A* que envie uma mensagem especial de junção à rede, com a chave igual a *X*. O Pastry direciona a mensagem de junção para um nó *Z* cujo identificador é numericamente próximo a *X*. Em resposta ao pedido de junção, os nós *A, Z* e todos os outros nós encontrados no caminho de *A* até *Z* enviam suas tabelas de estado para *X*. Finalmente *X* informa para os nós que devem estar cientes sobre sua presença. Isso garante que *X* inicializa com os valores apropriados e que todos os outros nós tenham suas tabelas atualizadas. Como o nó *A* foi eleito para estar topologicamente próximo ao novo nó *X*, o conjunto de vizinhos de *X* é inicializado com o conjunto de vizinhos de *A*.

O procedimento de rotas existente no Pastry é mostrado através de um pseudocódigo por Rowstron & Druschel (2001), exibido na Figura 11. O procedimento é executado quando uma mensagem com chave *D* chega a um nó com identificador *A*. Algumas notações são importantes para o entendimento do pseudocódigo:

- R^i_j: a entrada na tabela de rotas *R* na coluna *i*, $0 \leq i < 2^b$ e linha *l*, $0 \leq l < [128/b]$
- L_i: o *i*-th identificador mais próximo no conjunto de filhos *L*, $-[|L|/2] \leq i \leq [|L|/2]$, onde índices negativos/positivos indicam identificadores menores/maiores do que o identificador atual, respectivamente.
- D_l: o valor do digito *l* na chave *D*
- *Shl(A,B)*: o tamanho do prefixo compartilhado entre *A* e *B* em dígitos..

```
if (L_{-⌊|L|/2⌋} ≤ D ≤ L_{⌊|L|/2⌋}) {
    forward to L_i, s.th. |D − L_i| is minimal;
} else {
    Let l = shl(D, A);
    if (R_l^{D_l} ≠ null) {
        forward to R_l^{D_l};
    }
    else {
        forward to T ∈ L ∪ R ∪ M, s.th.
            shl(T, D) ≥ l,
            |T − D| < |A − D|
    }
}
```

Figura 11 - Rowstron & Druschel (2001) apresentam um pseudocódigo para o algoritmo de rotas do Pastry.

3.3 – Balanceamento de carga entre os nós

Os sistemas *peer-to-peer* citados implementam *DHT's* de forma a distribuir objetos entre nós através da escolha de identificadores gerados randomicamente ou a partir de determinadas regras, para os objetos. Na estrutura desses sistemas cada item armazenado é mapeado para um único identificador. Todo o espaço é dividido entre nós e cada nó é responsável por armazenar todos os itens que são mapeados para um identificador dentro da sua porção de espaço. Portanto os sistemas consistem de duas funções, uma para armazenar cada item associado a um identificador *put(ID, item)* e outra para obter um item associado a um identificador *get(ID)*.

Enquanto algoritmos *peer-to-peer* são simétricos, ou seja, todos os nós tem o mesmo direito no protocolo, sistemas *peer-to-peer* podem ser altamente heterogêneos. Um sistema *peer-to-peer* como o exemplo dos *crawlers* distribuídos geograficamente, deve consistir de nós localizados em servidores que atuam em redes de baixa velocidade conectados à servidores através de redes de banda larga.

Questões como balanceamento de carga são pontos críticos para tornar as operações eficientes em sistemas *peer-to-peer*. Um problema fundamental é a distribuição de itens para serem armazenados entre os nós que formam o sistema *peer-to-peer*. Um dos pontos da utilização de *DHT* é o balanceamento de carga. Todas as implementações de *DHT* fazem algum esforço para balanceamento de carga, geralmente através da ação de tornar aleatórios os endereços da *DHT* associados com cada item, ou fazendo com que cada nó seja responsável por um espaço eqüilátero na *DHT*. Stoica, Morris *et al.* (2003) apresentam um exemplo desta abordagem, onde randomicamente através de um função de *hash* os nós passam a ser responsáveis por pequenos intervalos de espaço do anel, enquanto o mapeamento randômico dos itens, significa que somente um número limitado de itens são direcionados para os "pequenos" intervalos dos anéis de responsabilidade dos nós.

Porém, esse esforço para balanceamento de carga pode falhar. Por exemplo, um típico particionamento aleatório de espaço de endereços entre os nós não é completamente balanceado. Alguns nós terminam com uma porção maior de endereços, portanto recebem uma porção maior de itens aleatoriamente distribuídos.

Um outro exemplo de falha ocorre quando as *DHT* criam uma estrutura homogênea sobre a rede, ignorando a heterogeneidade natural dos sistemas *peer-to-peer*.

3.3.1 – Balanceamento de carga em sistemas *peer-to-peer*

Rao, Lakshminarayanan *et al.* (2004) apresentam três esquemas simples de balanceamento de carga, são estes: Um-para-Um, Um-para-Muitos e Muitos-para-Muitos. Estes esquemas diferem primariamente na quantidade de informação usada para decidir como reorganizar a carga entre os nós. Rao, Lakshminarayanan *et al.* (2004) utilizam o conceito de servidores virtuais para balanceamento de carga. Um servidor virtual atua como um nó na *DHT*, mas cada nó físico pode ser responsável por mais de um servidor virtual. Por exemplo, Stoica, Morris *et al.* (2003) apresentam cada servidor virtual sendo responsável por uma região continua de identificadores de espaço, porém um nó pode ser dono de diversas regiões intercaladas no anel, através de diversos servidores virtuais. A vantagem de dividir a carga entre servidores virtuais é tornar flexível a mudança de cargas entre nós da *DHT*, em suas unidades de servidores virtuais. Isto é, a unidade básica de movimentação de carga é um servidor virtual.

A movimentação de servidores virtuais pode ser vista como uma operação de saída seguida pela operação de entrada em uma *DHT*, ambas suportadas. No caso em que cada nó tivesse somente um servidor virtual, a transferência de carga somente poderia ser realizada entre vizinhos. Apesar da divisão de carga entre servidores virtuais aumentar o tamanho do caminho da cobertura da rede, Rao, Lakshminarayanan *et al.* (2004) acreditam que a flexibilidade de mover a carga de um nó qualquer para um outro nó qualquer, é crucial para qualquer esquema de balanceamento de carga em *DHT*.

Todos os esquemas apresentados por Rao, Lakshminarayanan *et al.* (2004) tentam balancear a carga através da transferência de servidores virtuais entre nós pesados e nós leves. Em um esquema simples, a decisão de transferência envolve somente dois nós, enquanto em um esquema mais complexo, a decisão de transferência envolve um conjunto de nós pesados e nós leves. Um nó é considerado pesado se sua carga é maior que a carga limite estimada para cada nó do sistema, e considerado leve no caso contrário. O foco de todos os algoritmos de balanceamento de carga desenvolvidos por Rao, Lakshminarayanan *et al.* (2004) é minimizar a quantidade de nós pesados do sistema, movendo a carga destes nós para nós leves.

Uma das operações fundamentais para executar o balanceamento de carga é a transferência de um servidor virtual de um nó pesado para um nó leve. Dado um nó *h* considerado pesado e um nó *l* considerado leve, Rao, Lakshminarayanan *et al.* (2004) definem que o melhor servidor virtual *v* para ser transferido deve satisfazer as seguintes condições:

- Transferir *v* de *h* para *l* não tornará *l* pesado.
- *v* é o servidor virtual mais leve que tornará *h* pesado.
- Caso não exista servidor virtual *v* que torne *h* leve, transferir o servidor virtual *v* mais pesado de *h* para *l*.

O objetivo é transferir a menor quantidade possível de carga, de forma a tornar *h* leve e permanecer *l* leve. Se isso não for possível o esquema transferirá o servidor virtual mais pesado de *h* que não altere a condição de *l* permanecer leve. No caso de um servidor virtual considerado pesado não puder ser transferido inteiramente para um nó leve, então uma alternativa é dividir em pequenos servidores virtuais e transferir um pequeno servidor virtual para um nó leve.

Rao, Lakshminarayanan *et al.* (2004) apresentam o esquema Um-para-Um como o primeiro esquema de balanceamento de carga. Este é baseado em um mecanismo de *rendezvous* um-para-um, onde dois nós são escolhidos aleatoriamente e a transferência de um servidor virtual ocorre caso um dos nós seja um nó pesado e o outro um nó leve. Três vantagens são apontadas na escolha desse esquema:

1. Nós pesados são eximidos da carga de investigarem outros nós para realizar a troca de carga, isso é tarefa para os nós leves;
2. Quando a carga do sistema está muito alta e a maioria dos nós são pesados, não existe perigo de sobrecarga da rede;
3. A carga de um nó esta relacionada com o tamanho do espaço pertencente a este nó.

Outro esquema apresentado por Rao, Lakshminarayanan *et al.* (2004) é o esquema Um-para-Muitos. Diferente do primeiro, este esquema admite um nó pesado considerar mais de um nó leve para distribuir sua carga em um mesmo tempo. É utilizada uma solução baseada em diretórios os quais armazenam a carga de informação de um conjunto de nós leves.

O ultimo esquema apresentado por Rao, Lakshminarayanan *et al.* (2004) é o esquema Muitos-para-Muitos. Esse é uma extensão do primeiro e segundo esquemas.

Enquanto o primeiro esquema combina um nó pesado para um nó leve, e o segundo esquema combina um nó pesado para diversos nós leves. O Muito-para-Muitos combina muitos nós pesados para muitos nós leves. Para permitir muitos nós pesados interagirem com muitos nós leves, é utilizado um conceito de *pool* de servidores virtuais, o qual se torna um passo intermediário na locomoção de cargas entre nós pesados e nós leves. O *pool* é somente um local usado para computar a alocação final. Três fases são executadas nesse esquema. Na primeira fase, cada nó pesado *i* transfere seus servidores virtuais de forma gulosa para o *pool* até que o nó *i* se torne leve.

No fim dessa fase todos os nós serão leves, porém os servidores virtuais existentes no *pool* deverão ser transferidos para todos os nós que podem acomodá-los. A segunda fase auxilia a transferência de todos os servidores virtuais do *pool* para nós leves sem criar nenhum nó pesado. Essa fase é executada em estágios, em cada um é escolhido o servidor virtual *v* mais pesado no *pool* e então transferido para um nó leve *k*, a partir do uso de heurísticas. Essa fase continua até que o *pool* se torne vazio ou não existam mais servidores virtuais que possam ser transferidos. Com isso esse processo termina com todos os nós leves e não existindo mais servidores virtuais no *pool*, ou será executada a próxima fase (desalojamento).

A fase de desalojamento troca o servidor virtual *v* mais pesado do *pool* por um outro servidor virtual *v'* de um nó leve *i*, tal que *i* permaneça sendo leve. Caso ainda reste algum servidor virtual no *pool* após o processo nessa fase, tenta-se executar a troca analisando todos os nós leves, esses servidores virtuais serão enviados para os nós de origem.

3.3.2 – Utilizando proximidade para balanceamento de cargas em sistemas *peer-to-peer*

Entre os esquemas de balanceamento de carga apresentados por Rao, Lakshminarayanan *et al.* (2004) somente foram consideradas transferências entre nós pesados e nós leves, sem considerar qualquer tipo de relacionamento entre os nós. Com isso, questões como largura de banda e latência de rede passam a ser ignoradas, podendo incidir em sobrecarga na transferência.

Zhu & Hu (2004) apresentam um esquema de balanceamento que não é restrito a tipos particulares de recursos como armazenamento, largura de banda e CPU. A abordagem desse esquema é a utilização da ciência de localização no balanceamento de carga através do conceito de servidores virtuais, de forma a garantir

não somente a distribuição de carga entre os nós proporcionalmente as suas capacidades, mas também minimizar o custo de balanceamento de carga transferindo servidores virtuais entre nós pesados e nós leves em um contexto de proximidade. Esse esquema é constituído de quatro fases:

1. Agregação de informação no balanceamento de carga: Agregar informação sobre carga e capacidade em todo o sistema.
2. Classificação de nós: Classificar nós como leves ou pesados de acordo suas cargas.
3. Atribuição de servidores virtuais: Determinar a atribuição de servidores virtuais de nós pesados para nós leves de forma que nós pesados se tornem leves.
4. Transferência de servidores virtuais: Transferir servidores virtuais de nós pesados para nós leves.

A idéia de utilizar ciência de proximidade no mecanismo de balanceamento de carga proporciona a transferência de servidores virtuais entre nós pesados e leves, os quais estejam fisicamente próximos, reduzindo o custo do balanceamento de carga e permitindo mais eficiência e agilidade. A Figura 12 mostra uma abordagem ignorando a idéia de proximidade e uma outra introduzindo a idéia de proximidade.

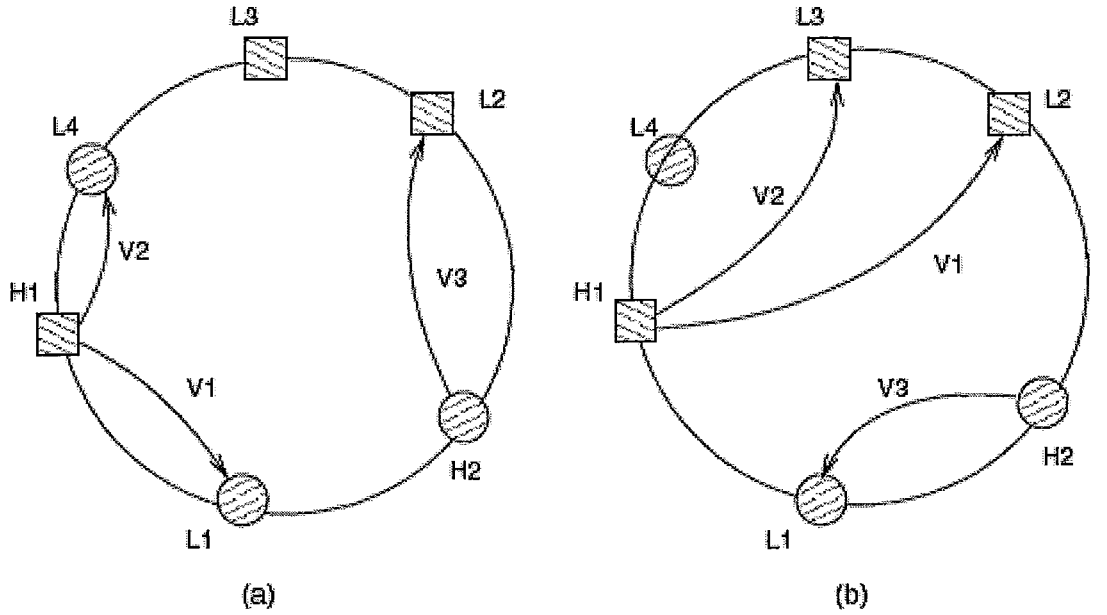


Figura 12 - Abordagem de balanceamento de carga com e sem ciência de proximidade.

Conforme mostrado na Figura 12, as letras "H" e "L" representam nós pesados e leves respectivamente. A letra "v" denota servidores virtuais prontos para serem movidos fazendo nós pesados tornarem leves. Nós com "retângulos" estão

fisicamente próximos uns dos outros. Nós com "retângulos" e nós com "círculos" estão fisicamente distantes uns dos outros.

As informações sobre proximidade são geradas a partir de *landmark clustering* (Ratnasamy, Handley et al., 2002), baseando-se na intuição de que nós fisicamente próximos uns dos outros, provavelmente terão distâncias similares a um pequeno conjunto de nós marcos (*landmark*).

Para um nó *A* de uma *DHT*, supostamente com a distância mensurada para um conjunto *m* de nós marcos (exemplo m = 15) sendo < *d1, d2,...,dn*>, o qual é chamado de vetor de marcos. Se o nó *A* é mapeado para um ponto em um espaço cartesiano *m*-dimensional através das coordenadas do seu vetor de marcos, então esse espaço cartesiano é chamado de espaço do marco. Como resultado, dois nós *A* e *B* da *DHT* fisicamente próximos terão supostamente vetores similares de marcos e estarão próximos um do outro no espaço do marco. É importante ter em mente que um número suficiente de nós marcos precisam ser utilizados, de forma a minimizar a probabilidade de falso *clustering* onde nós que estão distantes fisicamente têm vetores similares de marcos (Zhu & Hu, 2004).

A proximidade entre nós pesados e nós leves pode ser determinada através de seus vetores de marcos ou de suas proximidades no espaço do marco. A idéia que se subsidia a abordagem da ciência de proximidade em balanceamento de carga é a de usar a informação de proximidade para mapear nós pesados e leves que estejam fisicamente próximos dentro do espaço do marco e próximos um do outro no espaço de identificação da *DHT*. Segundo Zhu & Hu (2004) isso contribui para não existir a necessidade de alterar a estrutura de cobertura da *DHT*, tal que nós próximos fisicamente são também próximos um dos outros no espaço de identificação.

3.4 – Conclusão

Após estudo sobre a existência de *crawlers* colaborativos e distribuídos, foi identificada a possibilidade de realizar uma proposta sobre *crawlers* colaborativos formados por agentes de software organizados em uma plataforma *peer-to-peer*, fazendo uso de diferentes estratégias para localizar os servidores hospedeiros de páginas Web mais próximos aos *crawlers* de forma a tornar o processo de captura mais eficiente.

Capítulo 4 – Proposta de *crawler* colaborativo

Nesse capítulo é descrito o *crawler* colaborativo proposto, sua visão funcional, arquitetura baseada em multiagente e a organização das principais funcionalidades para a construção do *crawler* colaborativo em uma plataforma *peer-to-peer*.

4.1 – Visão abstrata do *crawler* colaborativo

Como apresentado no capítulo "2 – Revisão Bibliográfica", um *crawler* é responsável por capturar páginas da Web e armazená-las em um repositório, sendo normalmente o apoio a um mecanismo de indexação. O objetivo do *crawler* colaborativo é coletar o maior número de páginas possível tanto quanto seja permitido pelos recursos computacionais utilizados, incluindo os de rede. O desafio é alcançar esta meta sem violar qualquer regra, seja formal ou informal da conduta de *crawlers* que tenham sido desenvolvidos para a Web ao longo dos tempos.

Diferentes recursos computacionais são utilizados durante o processo de capturar páginas da Web, pois o *crawler* necessita realizar diversas atividades, tais como:

- Escolher uma URL para ser visitada, podendo essa escolha seguir diversas estratégias de navegação;
- Visitar o servidor hospedeiro da página Web respeitando critérios de política de visita;
- Baixar a página da Web e armazená-la em um sistema de arquivos;
- Extrair os *hyperlinks* existentes na página capturada para uma visita posterior;
- Extrair o conteúdo existente na página capturada para servir de informação à determinadas estratégias de navegação adotadas pelo *crawler*, como por exemplo no caso de *crawlers* focados;
- Extrair os metadados existentes na página capturada para servir de informação à determinadas estratégias de navegação adotadas pelo *crawler*, como por exemplo no caso de *crawlers* focados.

Cada uma dessas atividades deve ser otimizada o máximo possível durante sua execução, seja através da aplicação de algoritmos específicos, desacoplamento de

tarefas ou escalonamento de recursos. Devido a enorme quantidade de páginas existentes na Web, os *crawlers* sempre manusearão uma grande porção de dados, sendo imprescindível a otimização de suas atividades. A Figura 13 apresenta as atividades conceituais realizadas por um *crawler*.

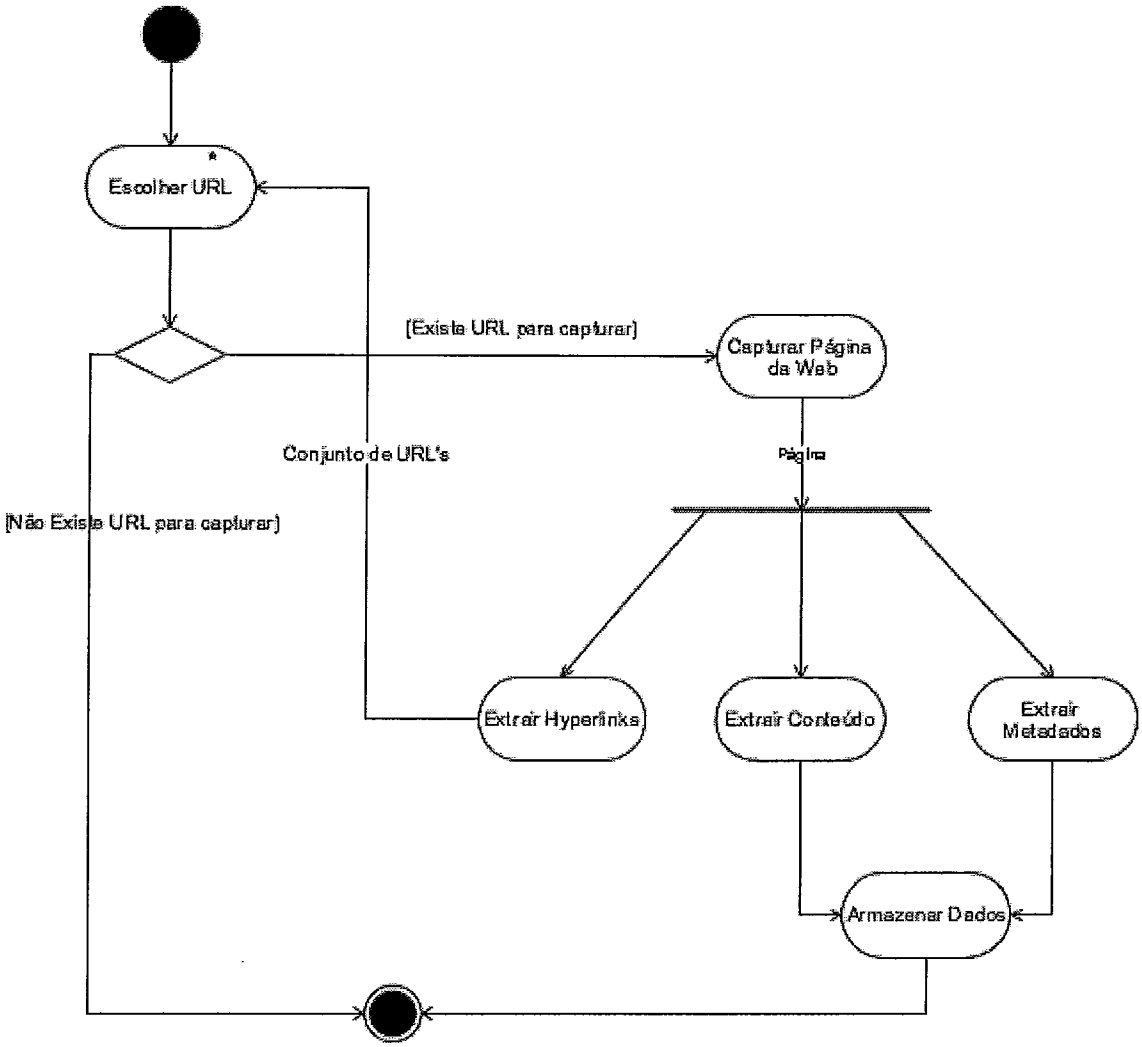


Figura 13 – Fluxo conceitual operações seguidas pelo *crawler*.

Conforme mencionado na seção "2.4.1- Visão Conceitual da Arquitetura de um *Crawler*", as atividades do *crawler* mantêm duas estruturas de dados para gerenciar as URL's: "URL's para serem visitadas" e "URL's visitadas". A lista "URL's visitadas" mantém uma lista de URL's já capturadas, sendo necessária para evitar que o *crawler* capture uma mesma página diversas vezes, porém possibilitando a revisita quanto necessário. A lista "URL's para serem visitadas" é uma fila que contém as URL's a serem capturadas em um momento futuro. O *crawler* trabalha através de uma iteratividade, através do qual obtém as URL's a serem visitadas, capturando-as da Web, armazenando as páginas em um repositório, e colocando as

URL's a serem visitadas em uma fila. Conceitualmente o *crawler* extrai todos os *hyperlinks* de cada página, converte os *hyperlinks* relativos em *hyperlinks* absolutos e verifica se algumas dessas URL's já tenham sido capturadas. Se for o caso, estas são descartadas.

Um *crawler* distribuído paraleliza suas atividades e tem seu desempenho na captura de página da Web superior em relação a um *crawler* monolítico, conforme apresentado por Loo, Krishnamurthy *et al.* (2004). Cada tarefa do *crawler* deve ser executada de uma maneira distribuída, de forma que não exista um coordenador central. Uma estrutura totalmente distribuída permite obter escalabilidade e torna o sistema tolerante a falhas. A distribuição de URL's entre os *crawlers* distribuídos é um importante problema relacionado a eficiência no processo de captura.

4.1.1 – Requisitos para um *crawler* distribuído em *peer-to-peer*

Alguns requisitos precisam ser atendidos para a construção de um bom *crawler* colaborativo e distribuído. Esses requisitos envolvem o planejamento e arquitetura do *crawler*, tanto sua forma quanto suas ações, são gerenciadas. São eles:

4.1.1.1 – Flexibilidade

O *crawler* precisa estar apto a funcionar em diferentes cenários de acordo com as estratégias de navegação adotadas.

4.1.1.2 – Baixo Custo e Alta Desempenho

É necessário sempre procurar escalonar o sistema o máximo possível, para lidar com centenas de páginas por segundo e consumir o mínimo necessário de recursos computacionais. A eficiência em acesso à disco é crucial para manter uma alta velocidade no processo de extração de *hyperlinks*. A estrutura de dados que mantém as URL's a serem visitadas também precisa estar organizada de forma que mantenha a agilidade ao ser acessada e não ultrapasse o limite da memória principal.

4.1.1.3 – Robustez

Existem muitos aspectos neste requisito. Inicialmente, o *crawler* interagirá com milhões de servidores, ele terá então que lidar com armadilhas e ter cautela, mesmo que seja necessário ignorar páginas ou servidores por completo. Em seguida, um *crawler* geralmente permanece executando durante semanas ou até meses, com

isso necessita estar hábil para tolerar falhas e interrupções de rede perdendo o mínimo de dados possível. Portanto, o estado do *crawler* necessita ser mantido em disco, persistindo as listas de URL's serem capturadas, assim como as informações extraídas.

A localização dos agentes formadores do *crawler* também não deve ser um problema, e mesmo que isso implique diretamente em questões de latência, a plataforma *peer-to-peer* deve otimizar ao máximo a comunicação.

4.1.1.4 – Etiqueta e Controle de Velocidade

É extremamente importante seguir o padrão convencionado para o arquivo robots.txt, para assim, fornecer uma URL a um *crawler* e supervisionar a captura. Adicionalmente, o tempo de acesso necessita ser controlado de diferentes formas, evitando que um único servidor seja acessado diversas vezes, sem respeitar um intervalo de tempo razoável e intercalar o processo de visita entre diferentes servidores.

4.1.1.5 – Gerência e Configuração

Os *crawlers* necessitam ser facilmente ajustados e configurados, podendo o administrador dos *crawlers* controlar o tempo de intervalo de visitas aos servidores, conforme descrito na seção "4.3.4 – Política de Visita". Adicionar novos domínios manualmente para que sejam visitados pelos *crawlers* ou adicionar domínios a uma lista negra, também são ajustes importantes a serem realizados. Isso é necessário, pois existem domínios que podem não estar sendo referenciados por outros domínios ou existir domínios maliciosos que aplicam armadilhas aos *crawlers*, como descrito na seção "4.3.5 – Evitar Armadilhas".

4.1.2 – *Crawlers* colaborativos através de um mecanismo de distribuição e balanceamento de carga

Os *crawlers* distribuídos podem encontrar seus limites, principalmente no que diz respeito ao espaço de atuação nos servidores hospedeiros de páginas Web. Visto que os *crawlers* podem estar distribuídos em diversas máquinas diferentes para evitar um congestionamento na utilização de recursos, é comum existir uma interseção nos servidores e URL's acessadas pelos *crawlers* distribuídos, fazendo com que diferentes *crawlers* atuem sobre um mesmo servidor, exercendo uma sobrecarga nos recursos da

máquina hospedeira das URL's, e capturando páginas da Web repetidamente. A carga do processo de captura é dividida pelos diversos *crawlers* distribuídos a fim de executar um balanceamento de carga. A Figura 14 apresenta um exemplo de dois diferentes *crawlers* hospedados em diferentes servidores e acessando simultaneamente um mesmo *site* a procura de páginas Web.

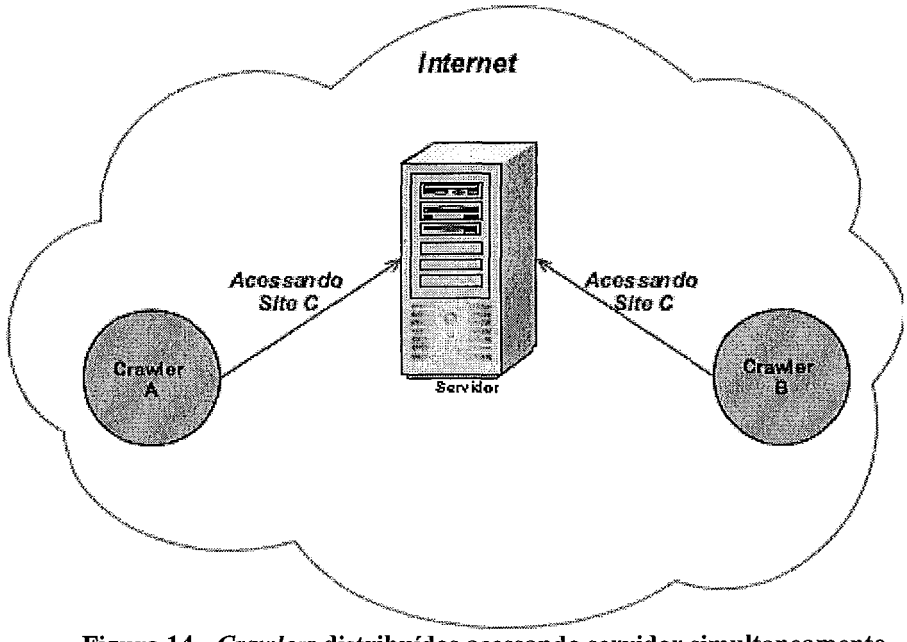


Figura 14 - *Crawlers* distribuídos acessando servidor simultaneamente

Um dos principais requisitos para um *crawler* distribuído e colaborativo é a existência de um mecanismo de distribuição e balanceamento de carga entre os *crawlers* que atuam de forma paralela e complementar.

O espaço de atuação dos *crawlers* na Web precisa ser particionado e distribuído entre eles. Para evitar os acessos simultâneos aos servidores hospedeiros, é interessante existir um mecanismo de distribuição e balanceamento de carga através das URL's a serem visitadas pelos *crawlers* distribuídos. Dessa forma a Web é particionada tornando o convívio dos *crawlers* harmonioso, colaborativo e produtivo em relação ao processo de captura.

O mecanismo de distribuição e balanceamento agrega ao *crawler* distribuído a funcionalidade de direcionar as URL's a serem capturadas aos *crawlers* responsáveis pelas mesmas, setorizando a atuação do *crawler* e buscando balancear a carga entre eles. Diferentes estratégias para organização dos *crawlers* e particionamento da Web podem ser utilizadas no mecanismo de distribuição e balanceamento de carga, essas estratégias podem influenciar tanto na organização dos *crawlers* como na distribuição das URL's.

4.1.2.1 – Migração de clientes

O mecanismo de distribuição e balanceamento de carga tem os *crawlers* distribuídos como seus clientes, este mecanismo fornece serviços complementares às suas tarefas. Alguns clientes vinculados aos mecanismos de distribuição podem não estar atendendo a algumas regras ou estratégias definidas para o mecanismo, como por exemplo: um determinado *crawler* tem pouco recurso de processamento não suportando processar uma grande quantidade de *sites* que estão sob sua responsabilidade. Desta forma, fazendo com que o mecanismo de distribuição o mova para uma outra partição que recebe menos sites para processar.

A ação de migrar os clientes promove uma otimização no processo de captura das páginas, atribuindo ao mecanismo de distribuição o papel de distribuir as partições da Web da melhor forma possível entre os *crawlers*, inclusive tornando dinâmica sua reestruturação ou coordenação.

4.1.3 – *Crawler* colaborativo através de agentes

Cada uma das principais atividades que constituem um *crawler* distribuído pode ser organizada em agentes atômicos. Os agentes atuam coordenando suas atividades através da troca de mensagens permitindo que as atividades sejam executadas paralelamente, buscando uma maior eficiência na execução. A utilização de uma plataforma *peer-to-peer* se faz necessária para prover os serviços e funcionalidades requeridas pelos agentes. Uma visão conceitual dos agentes e suas atividades apoiadas por recursos computacionais e uma plataforma *peer-to-peer* é apresentada na Figura 15:

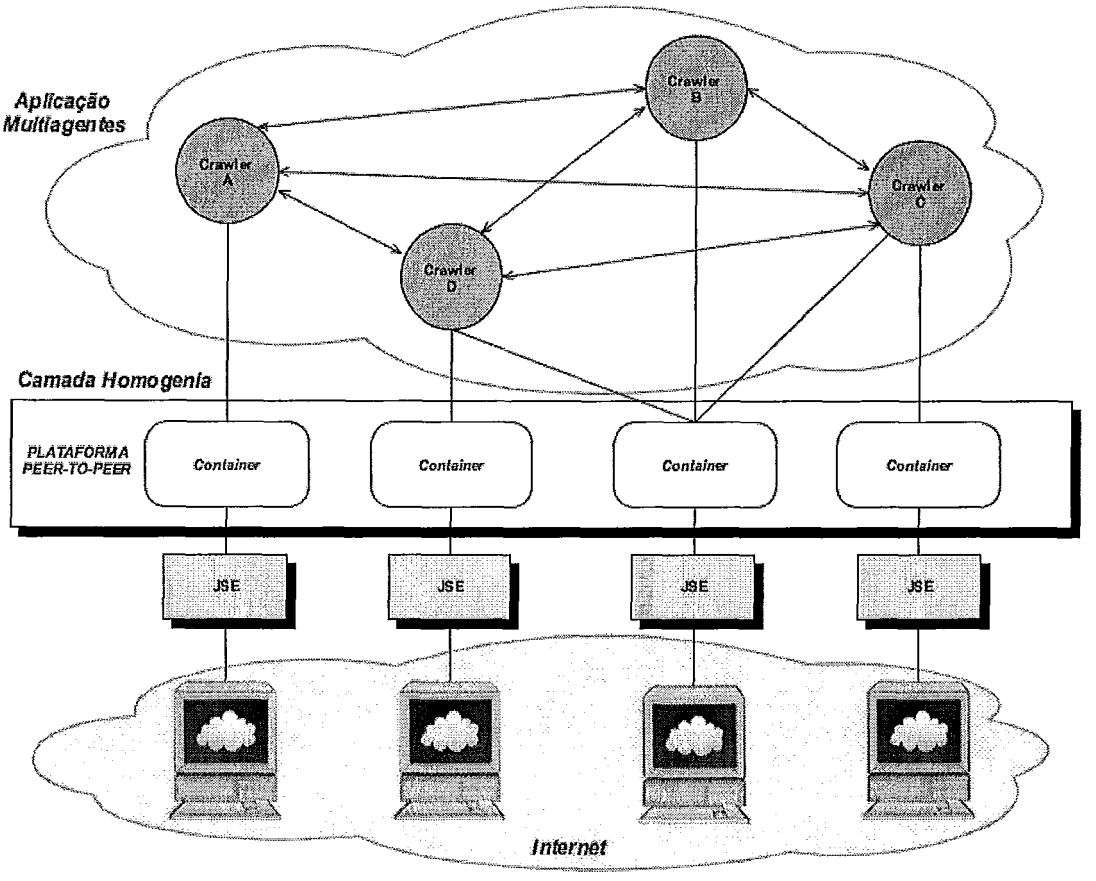


Figura 15 - Visão conceitual do *crawler* distribuído através de agentes

4.1.4 – *Crawler* colaborativo e seus agentes

Um conjunto de agentes que atuam de forma complementar na tarefa de capturar e processar as páginas da Web forma o *crawler* colaborativo. Esses agentes foram elaborados de forma a otimizar e tornar atômicas e colaborativas as ações do *crawler*. Cada agente é responsável por uma tarefa fundamental para o funcionamento do *crawler*. Esse conjunto de agentes colaborativos constitui uma arquitetura dinâmica e de baixo acoplamento, apoiada por uma plataforma *peer-to-peer*.

O *crawler* colaborativo é uma aplicação multiagente organizada através de um grupo de agentes atômicos e autônomos que atuam em conjunto. Esse agentes podem estar agrupados em um ou mais recipientes de acordo com estratégias adotadas pelo *crawler*, ou otimizações necessárias para aumentar a qualidade no processo de captura de páginas da Web. A Figura 16 mostra os agentes que formam o *crawler* colaborativos desenvolvido nesse trabalho.

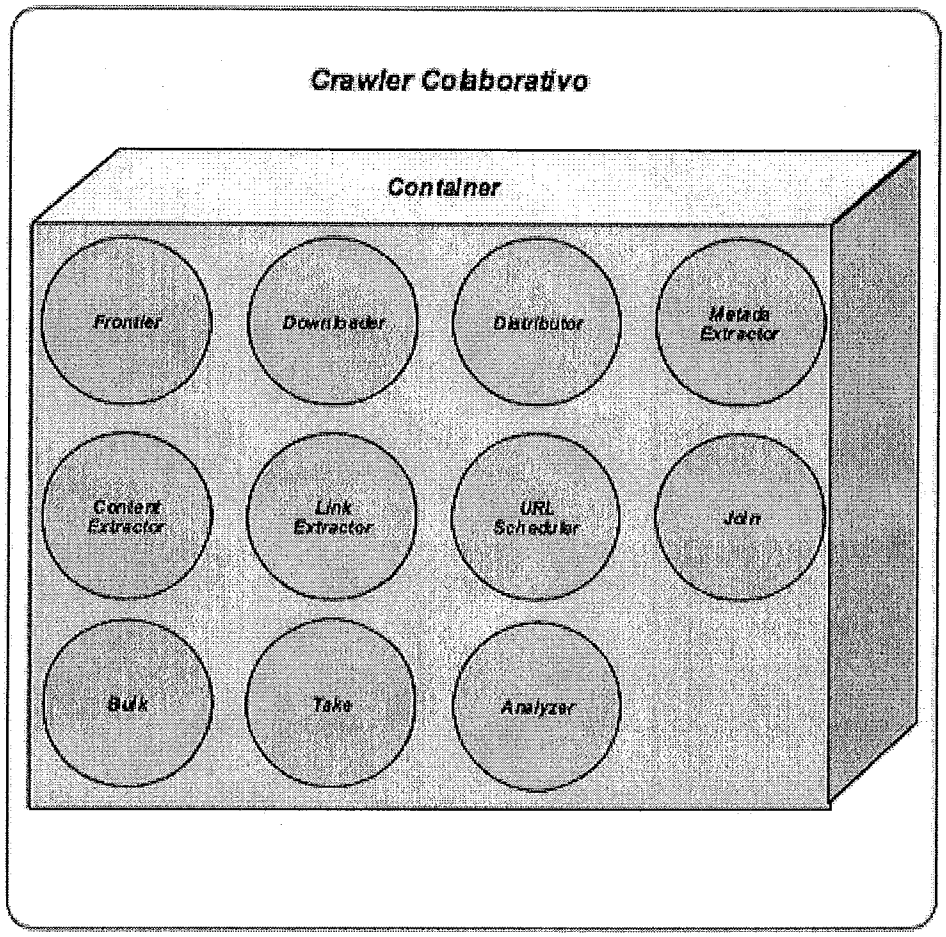


Figura 16 - Grupo de agentes formadores do *crawler* colaborativo

4.1.4.1 – Frontier

O agente *Frontier* é o responsável pela estratégia de escolha da próxima URL a ser acessada. Essa escolha pode ser realizada através de qualquer tipo de estratégia, como por exemplo, uma estratégia *breadth-first*, *depth-first*, *crawler* focados, entre outras. O *Frontier* persegue a próxima URL a ser coletada e identifica quais já foram acessadas, evitando capturar URL's já visitadas e prevenindo o reagendamento de URL's já agendadas para serem capturadas. Esse agente utiliza um mecanismo de filtragem capaz de eliminar as URL's duplicadas e segue políticas de visitas respeitando regras de acesso aos servidores que hospedam páginas Web.

O próprio *Frontier* controla o processo de revisitar as URL's, como por exemplo, através de uma regra de *round-robin*, onde cada URL já capturada é revisitada a cada intervalo de tempo, seguindo a ordem de captura na qual ela foi coletada.

4.1.4.2 – Downloader

O agente *Downloader* é responsável por acessar as páginas da Web através de uma URL e armazená-las em arquivos localizados em um diretório previamente configurado e acessível via *NFS* (Network File System). O agente *Downloader* gerencia essa atividade e para evitar um gargalo na captura das páginas, toda URL recebida é despachada para um componente utilizado por esse agente, que através de um processamento assíncrono é capaz de baixar diversas páginas paralelamente. O número máximo de processos que podem executar paralelamente baixando páginas é configurável de forma a se evitar um grande consumo de recurso de rede. Para capturar os recursos da Web o *Downloader* utiliza o protocolo *HTTP* (*HyperText Transfer Protocol*), seguindo um cenário de requisição/resposta e usa o protocolo *TCP* (*Transmission Control Protocol*) para transferir dados. Também é utilizado um mecanismo para resolução de *DNS(Domain Name System)*, traduzindo os nomes dos domínios para o endereço *IP*.

4.1.4.3 – LinkExtractor

O agente *LinkExtractor* tem a tarefa de extrair os *hyperlinks* existentes na página Web coletada pelo *crawler*, e armazenada em um sistema de arquivos. Após o processo de extração, um conjunto de *hyperlinks* é obtidos da página Web anteriormente capturada, e este conjunto segue para um componente de filtragem, onde podem ser definidas regras de validação para os *hyperlinks* capturados, por exemplo, domínios não interessantes para captura ou tipos de arquivos. As regras podem ser escritas através da utilização de expressão regular (Stubblebine, 2006).

O *LinkExtractor* faz uso de componentes customizados e capazes de funcionar paralelamente para o processamento dos arquivos, não gerando um gargalo para essa atividade e mantendo intactos os arquivos após utiliza-los. Caso as páginas contenham *hyperlinks* relativos, o *LinkExtractor* é capaz de identificar e retornar o conjunto de *hyperlinks* normalizados.

4.1.4.4 – ContentExtractor

O agente *ContentExtractor* é responsável por extrair o conteúdo das páginas capturadas e armazenadas em disco, sendo acessadas via *NFS* (*Network File System*) e mantendo uma cópia armazenada em disco. Parecido com o agente *Downloader*

esse agente utiliza um componente capaz de executar a extração de diversos arquivos assincronamente, tornando-se performático e evitando um gargalo no processamento.

4.1.4.5 – MetadataExtractor

O agente *MetadaExtractor* tem a tarefa de extrair diferentes tipos de metadados relacionadas às páginas Web capturadas. Metadados existentes em páginas Web podem ser utilizados para guiar os *crawlers* durante o processo de captura, influenciando em decisões, tais como: qual caminho seguir, agendamento de URL's, prioridades de URL's a serem capturadas, contextos de sites, entre outros. O agente *MetadataExtractor* extrai informações das páginas e todas estas são armazenadas em disco e podem ser acessadas via *NFS* (*Network File System*) por diferentes componentes, ou mesmo solicitadas ao agente *MetadataExtractor* para uma determinada URL.

O agente *MetadataExtractor* faz uso de interfaces bem definidas que permitem a implementação de diversos mecanismos de extração de metadados, seguindo a implementação do padrão de projeto *Cadeia de Responsabilidade* apresentado por Gamma, Helm *et al.* (1996).

4.1.4.6 – URLScheduler

O agente *URLScheduler* notifica os agentes *LinkExtractor*, *ContentExtractor* e *MetadaExtractor* da existência de páginas Web capturadas, para executar os processamentos relacionados às suas devidas atividades. O agente *URLScheduler* exerce o papel de notificá-los, seguindo uma estratégia *FIFO* (*First In First Out*) em sua organização, ou seja, a ordem em que as páginas são capturadas é a ordem em que são notificadas para os agentes *LinkExtractor*, *ContentExtractor* e *MetadaExtractor*.

4.1.4.7 – Bulk

O agente *Bulk* é responsável por notificar o agente responsável pelo mecanismo de distribuição da existência de um conjunto de URL's a ser distribuído. No caso do *crawler* colaborativo toda distribuição de URL's será realizada através de um mecanismo de distribuição, conforme descrito na seção "5.3 – Implementação do Mecanismo de Distribuição e Balanceamento de Carga entre os *Crawlers* Colaborativos".

4.1.4.7.1 – Take

O agente *Take* é responsável pelo recebimento de todas as URL's para o *crawler* colaborativo através do mecanismo de distribuição, para que estas sejam capturadas pelo *crawler* em um momento posterior.

4.1.4.8 – Analyzer

O agente *Analyzer* é responsável por analisar um conjunto de regras previamente implementadas e adicionadas a esse agente, notificando ao mecanismo de distribuição, para que não mais envie determinadas URL's para o agente *Take*, URL's essas que por algum motivo pertencem ao intervalo de valores abrangido pelo mecanismo de distribuição. Com isso o *crawler* colaborativo pode tomar a decisão de não mais aceitar o recebimento de determinadas URL's com base na análise de regras que visem buscar a otimização e melhoria na execução das tarefas realizadas pelos agentes.

4.1.4.9 – Join

O agente *Join* é o responsável no *crawler* colaborativo por notificar sua existência ao mecanismo de distribuição e balanceamento de carga, enviando um pedido de junção e tornando-se cliente.

4.1.4.10 – Distributor

O agente *Distributor* é responsável pelo mecanismo de distribuição, apoiando o *crawler* colaborativo na distribuição das URL e proporcionando um balanceamento de carga entre os *crawlers*. Qualquer que seja a estratégia de distribuição implementada e oferecida pelo agente *Distributor*, ela será totalmente transparente e desacoplada das atividades e estratégias de navegação realizadas pelo *crawler*. Mesmo não tendo conhecimento do tipo de estratégia usada pelo *Distributor*, é importante lembrar que o *crawler* poderá sugerir ao *Distributor* o não recebimento de determinadas URL's através do agente *Analyzer*, cabendo ao agente *Distributor* aceitar ou não as informações recebidas.

4.1.5 – Tipos e grupos dos agentes de software

Os agentes foram projetados para atuarem da forma mais atômica possível na realização das atividades do *crawler* colaborativo. Os agentes têm a finalidade de

alcançar um objetivo maior, realizam tarefas específicas, coordenando-as entre si de forma que suas atividades se completem. É possível fazer uma classificação nos agentes de acordo com vários aspectos, tais como: mobilidade, relacionamento inter-agentes e capacidade de raciocínio.

Apesar das plataformas *peer-to-peer* suportarem os agentes distribuídos em diferentes máquinas, alguns agentes colaborativos que atuam em um mesmo "grupo de trabalho", ou seja, suas tarefas são complementares, precisam estar em uma mesma máquina ou rede, pois a sua distribuição impactaria diretamente nas responsabilidades atribuídas para estes.

No *crawler* colaborativo esses agentes são chamados de **comunidade de agentes**. Os agentes *Downloader e Analyzer* estão em comunidade, ou seja, não devem atuar em máquinas ou redes diferentes. Isso acontece porque o agente *Analyzer* incide diretamente na captura de URL's informando ao mecanismo de distribuição utilizado, quais URL's não devem mais ser enviadas ao *crawler*. O agente *Analyzer* faz uso de estratégias que necessitam analisar, por exemplo, o tempo de resposta entre o servidor (onde está localizado o agente *Downloader*) que baixa as páginas da Web e os servidores hospedeiros das páginas. O mesmo ocorre na utilização de estratégia para particionamento geográfico.

Todos os outros agentes que constituem o *crawler* colaborativo, com exceção dos agentes *Downloader* e *Analyzer* podem estar separados em diversos recipientes de uma plataforma *peer-to-peer* sobre diferentes máquinas (vide Figura 17), buscando otimizar suas atividades através da utilização de recursos como processamento e conectividade. A figura a seguir mostra um exemplo da forma como os agentes poderiam estar distribuídos através de recipientes e respeitando a lógica de comunidade requerida pelos agentes *Downloader* e *Analyzer*.

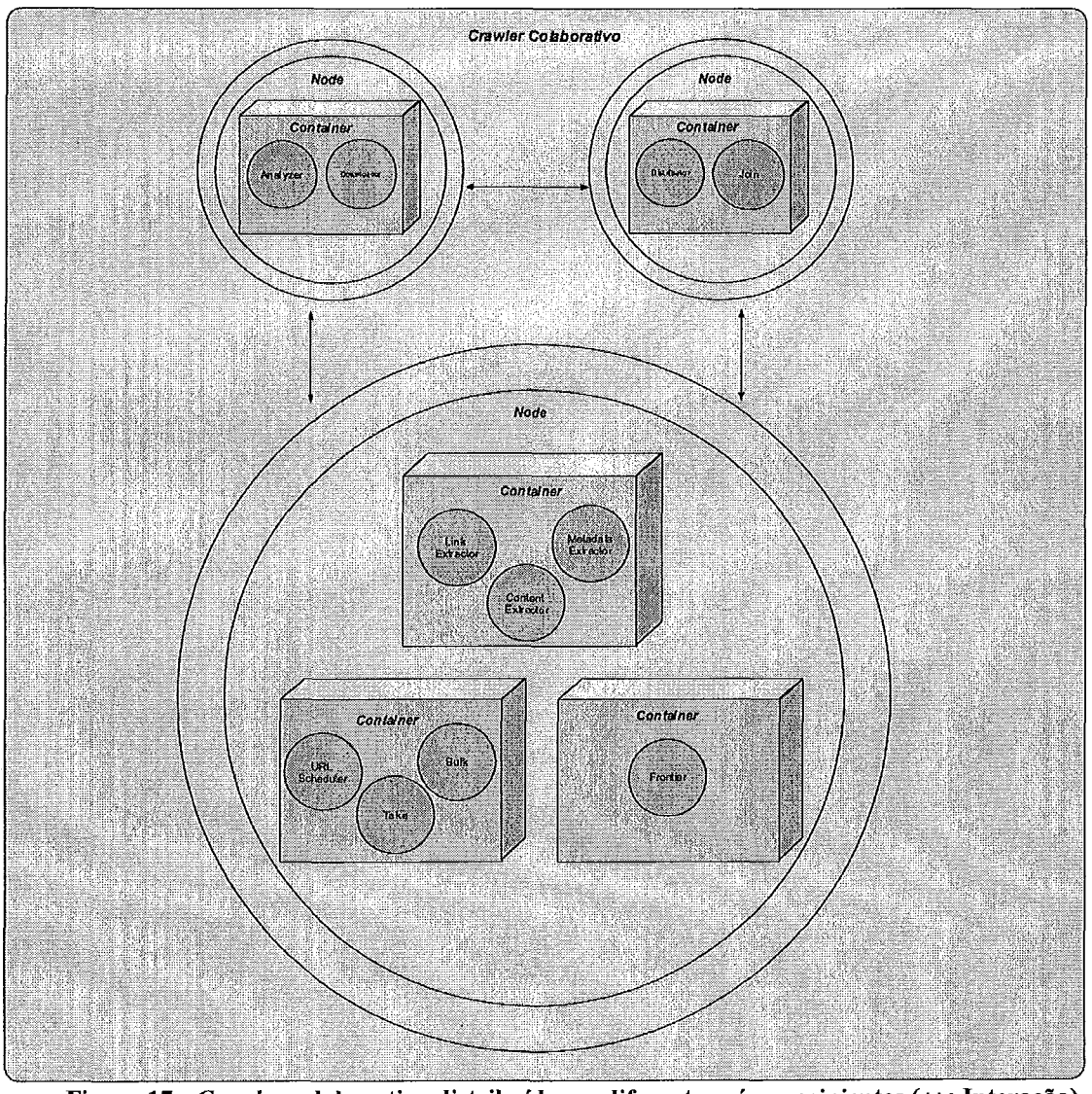


Figura 17 - *Crawler* colaborativo distribuído em diferentes nós e recipientes (↔: Interação)

4.1.6 – Arquitetura conceitual

A arquitetura conceitual mostrada na Figura 18, apresenta uma visão dos agentes e os seus pontos de comunicação. Os pontos de comunicação entre os agentes ocorrem através de espaço compartilhado. A arquitetura definida garante a coordenação entre os agentes para que haja uma coerência nas atividades que estão atuando.

A coordenação está muito relacionada com o compartilhamento de conhecimento entre os agentes, sendo seu principal objetivo, tornar as ações individuais de cada agente coordenadas para se atingir o objetivo final do *crawler* colaborativo. Um motivo para preocupação com a coordenação entre agentes é o fato de que um só agente, dentro do *crawler* colaborativo, não terá informação ou

capacidade suficiente para resolver muitos dos problemas - muitos dos objetivos não podem ser atingidos pelos agentes agindo isoladamente.

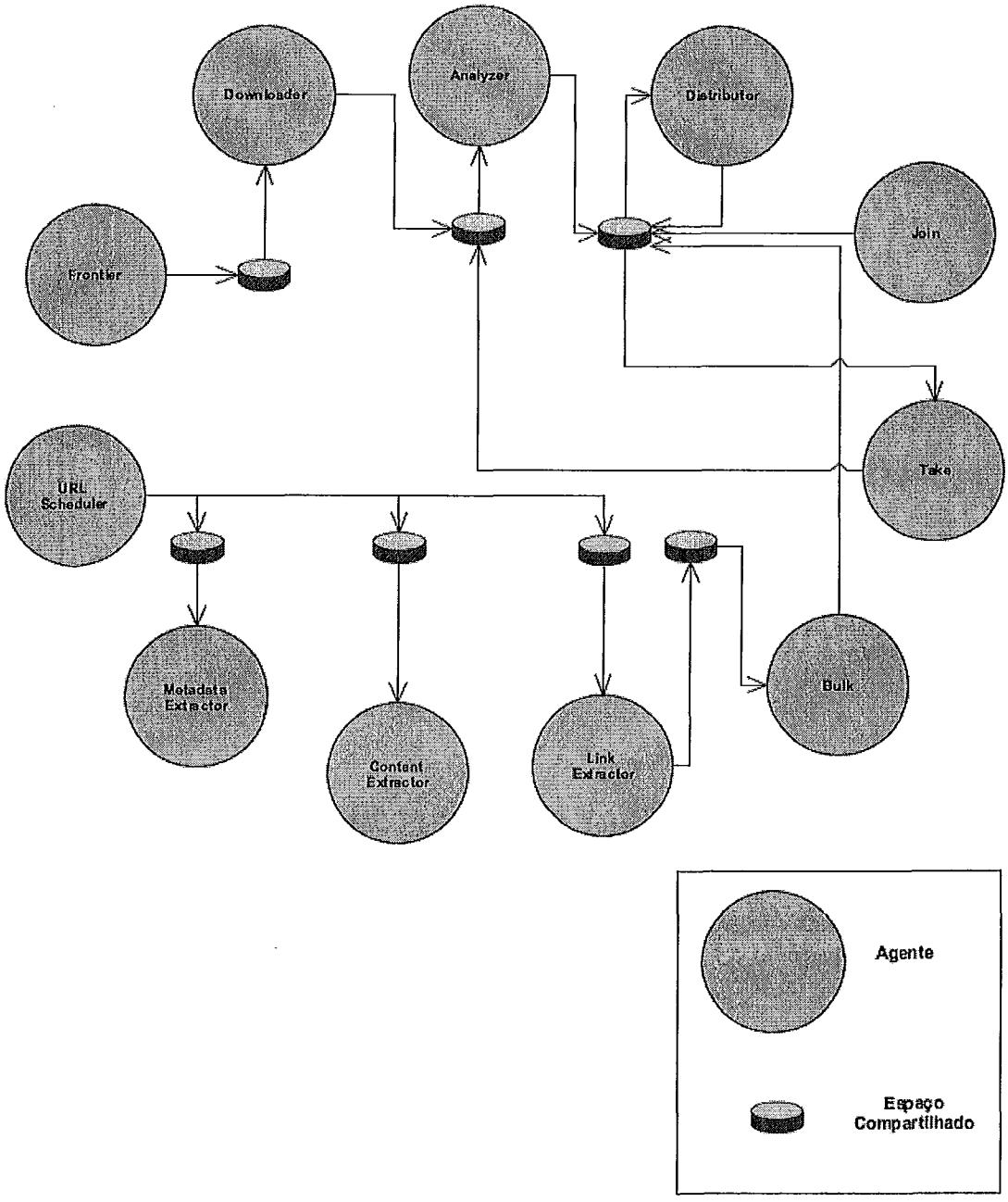


Figura 18 - Arquitetura conceitual do *crawler* colaborativo.

A arquitetura está organizada em agentes atômicos em suas funcionalidades, proporcionando uma eficiente utilização de recursos, podendo esses agentes serem distribuídos através de diferentes computadores evitando uma sobrecarga na leitura e escrita de arquivos ou uso de memória para processamento.

4.1.6.1 – Identificação dos agentes de software

Os agentes são controlados por uma plataforma *peer-to-peer* e associado a eles existe um conjunto de mensagens para que sejam notificadas. Cada agente tem um nome e endereço único. As tabelas a seguir definem os atributos, mensagens e funções que são parte dos agentes formadores do *crawler* colaborativo.

Agente	Frontier
Tipo	Colaborativo
Responsabilidade	Definir a próxima URL a ser acessada
Mensagens de Entrada	
Mensagens de Saída	URLMessage
Agentes Coexistentes	Downloader

Agente	Downloader
Tipo	Colaborativo e Comunidade
Responsabilidade	Baixar páginas da Web
Mensagens de Entrada	URLMessage
Mensagens de Saída	AnalyzeMessage
Agentes Coexistentes	Frontier Analyzer

Agente	URLScheduler
Tipo	Colaborativo
Responsabilidade	Notificar outros agentes da existência de página Web para processamento de conteúdo
Mensagens de Entrada	
Mensagens de Saída	LinkMessage ContentMessage MetadadoMessage
Agentes Coexistentes	LinkExtractor ContentExtractor MetadadoExtractor

Agente	LinkExtractor

Tipo	Colaborativo
Responsabilidade	Extrair os *hyperlinks* existentes nas páginas Web capturadas
Mensagens de Entrada	LinkMessage
Mensagens de Saída	BulkMessage
Agentes Coexistentes	URLScheduler Bulk

Agente	ContentExtractor
Tipo	Colaborativo
Responsabilidade	Extrair o conteúdo textual existente nas páginas Web capturadas
Mensagens de Entrada	ContentMessage
Mensagens de Saída	
Agentes Coexistentes	URLScheduler

Agente	MetadataExtractor
Tipo	Colaborativo
Responsabilidade	Extrair metadados existente nas páginas Web capturadas
Mensagens de Entrada	MetadataMessage
Mensagens de Saída	
Agentes Coexistentes	URLScheduler

Agente	Bulk
Tipo	Colaborativo
Responsabilidade	Enviar conjunto de URL's ao mecanismo responsável pela distribuição e balanceamento de carga entre os *crawlers* colaborativos
Mensagens de Entrada	BulkMessage
Mensagens de Saída	PutMessage
Agentes Coexistentes	LinkExtractor Distributor

Agente	Take
Tipo	Colaborativo
Responsabilidade	Receber conjunto de URL's do mecanismo responsável pela distribuição e balanceamento de carga entre os *crawlers* colaborativos
Mensagens de Entrada	KeyMessage
Mensagens de Saída	
Agentes Coexistentes	Distributor

Agente	Analyzer
Tipo	Colaborativo e Comunidade
Responsabilidade	Notificar o mecanismo responsável pela distribuição e balanceamento de carga entre os *crawlers* colaborativos para não enviar determinadas URL's
Mensagens de Entrada	AnalyzeMessage
Mensagens de Saída	RedirectKeyMessage
Agentes Coexistentes	Downloader Distributor

Agente	Join
Tipo	Colaborativo
Responsabilidade	Envia um pedido de junção ao mecanismo de distribuição e balanceamento, para que o *crawler* passe a ser um cliente do mecanismo
Mensagens de Entrada	
Mensagens de Saída	JoinMessage
Agentes Coexistentes	Distributor

Agente	Distributor
Tipo	Colaborativo
Responsabilidade	Mecanismo responsável pela distribuição e

	balanceamento de carga entre os *crawlers* colaborativos
Mensagens de Entrada	JoinMessage PutMessage RedirectKeyMessage
Mensagens de Saída	KeyMessage
Agentes Coexistentes	Join Bulk Take

4.1.6.2 – Identificação das mensagens

Cada um dos agentes identificados anteriormente está associado a um conjunto de mensagens para notificação. O recebimento e controle dessas mensagens guia e coordena as atividades exercidas pelos agentes formadores do *crawler* colaborativo. As tabelas a seguir definem as mensagens utilizadas pelos agentes.

Mensagem	URLMessage
Objetivo	Informar as novas páginas que serão capturadas
Atributos	URL
Agentes Utilizadores	Frontier Downloader

Mensagem	AnalyzeMessage
Objetivo	Informar os domínios que serão analisados através de métricas definidas no agente *Analyzer*
Atributos	Domínio
Agentes Utilizadores	Downloader Analyzer

Mensagem	LinkMessage
Objetivo	Informar uma nova página capturada para ter seu conjunto de *hyperlinks* extraído
Atributos	Localização da página no sistema de arquivos
Agentes Utilizadores	URLScheduler

	LinkExtractor

Mensagem	ContentMessage
Objetivo	Informar uma nova página capturada para ter seu conteúdo extraído
Atributos	Localização da página no sistema de arquivos
Agentes Utilizadores	URLScheduler ContentExtractor

Mensagem	MetadadoMessage
Objetivo	Informar uma nova página capturada para ter seu conjunto de *hyperlinks* extraído
Atributos	Localização da página no sistema de arquivos
Agentes Utilizadores	URLScheduler MetadadoExtractor

Mensagem	BulkMessage
Objetivo	Receber e organizar o conjunto de URL's extraídas de uma página capturada
Atributos	Conjunto de URL's extraídas de uma página capturada
Agentes Utilizadores	Bulk LinkExtractor

Mensagem	PutMessage
Objetivo	Enviar ao mecanismo de distribuição URL extraída de uma página capturada
Atributos	URL
Agentes Utilizadores	Bulk Distributor

Mensagem	KeyMessage
Objetivo	Receber do mecanismo de distribuição URL

Atributos	URL
Agentes Utilizadores	Take Distributor

Mensagem	JoinMessage
Objetivo	Informar ao mecanismo de distribuição o nome e endereço do agente que irá fazer parte
Atributos	Nome e endereço do agente
Agentes Utilizadores	Join Distributor

Mensagem	RedirectMessage
Objetivo	Informar ao mecanismo de distribuição que um determinado *crawler* não mais se responsabilizará por uma URL
Atributos	URL
Agentes Utilizadores	Analyzer Distributor

Mensagem	IdleMessage
Objetivo	Informar ao mecanismo de distribuição que um determinado *crawler* está ocioso, podendo receber URL's extras para diminuir a carga em outros *crawlers*
Atributos	Identificação do Agente
Agentes Utilizadores	Analyzer Distributor

Mensagem	OverflowMessage
Objetivo	Informar ao mecanismo de distribuição que um determinado *crawler* esta sobrecarregado, precisando enviar URL's extras a um outro *crawler* para diminuir a sua carga

Atributos	Identificação do Agente
Agentes Utilizadores	Analyzer Distributor

Mensagem	StateMessage
Objetivo	Informar ao agente *Analyzer* sobre o estado do *crawler*
Atributos	Estado do *crawler*
Agentes Utilizadores	Analyzer Frontier

4.2 – Projeto do *crawler* colaborativo

A arquitetura de multiagente adotada no *crawler* colaborativo permite a implementação de diversas estratégias pelos agentes sem que aja impacto na organização do *crawler* ou necessidade de reestruturá-lo para atender diferentes estratégias.

4.2.1 – Arquitetura baseada em multiagente

Segundo Weiss (2000) os agentes dentro de um sistema multiagente podem ser heterogêneos, homogêneos, colaborativos, competitivos, entre outros.

A comunicação entre os agentes é fundamental para permitir a colaboração e cooperação entre entidades independentes. Em sistemas multiagentes, é necessário que a comunicação seja disciplinada e organizada para que os objetivos sejam alcançados eficientemente, necessitando assim um mecanismo que possa ser entendido pelos outros agentes presentes no ambiente. Essa comunicação tem como principal objetivo compartilhar o conhecimento entre os agentes e a coordenação de atividades entre os mesmos. Ela deve permitir que agentes troquem informações entre si e coordenem suas próprias atividades, resultando sempre em um sistema coerente.

Existem diversas maneiras para agentes trocarem informações uns com os outros em sistemas multiagentes. Segundo Deepika & Albert (1998) agentes podem trocar mensagens diretamente, podem comunicar-se através de um agente "facilitador", podem também utilizar uma comunicação por difusão de mensagens

(*broadcast*) ou utilizar o modelo de comunicação através de quadro-negro (*blackboard*).

Na comunicação direta, ou via troca de mensagens direta, cada agente se comunica diretamente com qualquer outro agente sem intermediário. Nesse tipo de comunicação faz-se necessário que cada agente envolvido tenha conhecimento da existência dos outros agentes e da forma de como endereçar mensagens para estes. A principal vantagem deste tipo de comunicação entre agentes é o fato de não existir um agente coordenador da comunicação. Isso porque, agentes coordenadores podem levar a um "gargalo" ou até ao bloqueio do sistema caso exista, por exemplo: um grande número de troca de mensagens. As desvantagens são: o custo da comunicação, que se torna grande, principalmente quando há um grande número de agentes no sistema, e a própria implementação, que se torna muito complexa em comparação às outras formas de comunicação.

Segundo Deepika & Albert (1998), na comunicação assistida os agentes utilizam algum sistema ou agente especial para coordenar suas atividades. Uma estrutura hierárquica de agentes é definida e a troca de mensagens ocorre através de agentes especiais, designados "facilitadores" ou mediadores. Essa é uma alternativa à comunicação direta bastante comum, pois diminui muito o custo e a complexidade necessária aos agentes individuais na realização da comunicação. Geralmente é utilizada quando o número de agentes dentro do sistema é muito grande.

A comunicação por difusão de mensagens geralmente é utilizada em situações onde a mensagem deve ser enviada para todos os agentes do ambiente ou quando o agente remetente, não conhece o agente destinatário ou seu endereço, fazendo com que todos os agentes recebam a mensagem enviada.

O paradigma de espaço compartilhado e distribuído provê através de um sistema compartilhado e distribuído, uma visão única do espaço, o qual proporciona sincronização e comunicação entre os agentes através de operações realizadas no espaço compartilhado (*blackboard*). A comunicação entre os agentes ocorre através da inserção|remoção de registros no espaço compartilhado. Três principais operações podem ser executadas: *out()* para exportar um registro para o espaço compartilhado, *in()* para importar e remover um registro do espaço compartilhado e *read()* para ler (sem remover) um registro de um espaço compartilhado (Eugster, Felber et al., 2003).

Segundo Eugster, Felber *et al.* (2003) esse modelo provê o desacoplamento de tempo e espaço, no qual o produtor e consumidor de um registro permanecerão

anônimos um em relação ao outro. O criador de um registro não necessita conhecer sobre o futuro uso do registro ou seu destino. Ao invés de explicitamente enviar um pacote para um destino, cada pacote está associado a um identificador, este é então usado pelo receptor para obter o destino do pacote. Esse nível de indireção desacopla a ação de envio da ação de recebimento, o que possibilita o uso de diferentes estratégias para o compartilhamento de recursos, visando otimizar a captura de páginas da Web.

4.3 – Estratégias de balanceamento de carga

Os *crawlers* colaborativos seguem a utilização de um mecanismo para distribuição de URL's e balanceamento de carga. Os agentes do *crawler* colaborativo responsáveis por distribuir, receber e avaliar as URL's, realizam a comunicação com mecanismo de distribuição e balanceamento de carga.

Utilizar um *framework peer-to-peer* na arquitetura do *crawler* colaborativo, pode proporcionar uma independência do mecanismo de balanceamento de carga entre os *crawlers*, onde isoladamente do processo de captura das páginas Web, o agente *Bulk* envia mensagens, contendo as URL's que serão entregues ao mecanismo de distribuição e balanceamento de carga. Paralelamente está agindo em relação ao mecanismo de balanceamento de carga, o agente *Take*, recebendo URL's que pertençam ao cliente. O diagrama de pacotes apresentado na Figura 26 da seção "5.5 – Pacotes e Classes do Crawler Colaborativo" deixa claro o isolamento e baixo acoplamento entre o *crawler* colaborativo e o mecanismo de distribuição e balanceamento de carga.

O mecanismo de distribuição e balanceamento de carga mantém o foco somente na distribuição de URL's observando intervalos de valores cobertos pelos nós, dessa forma o balanceamento de carga ocorre sem levar em consideração nenhuma outra análise além do intervalo de *hash*, abrangido pelos nós.

Utilizar somente o balanceamento de carga através do intervalo de *hash*, pode não ser eficiente, como por exemplo, no caso dos *crawlers* colaborativos. É fato que a incidência de *crawlers* serem responsáveis pela captura de sites em determinados servidores, os quais não tenham uma boa rota de tráfego com os computadores hospedeiros dos *crawlers*, gera perda de efetividade no processo de captura de páginas da Web. Qualquer que seja o tipo de estratégia utilizada para atribuir melhorias ao mecanismo de balanceamento, não impacta na arquitetura do *crawler*,

pois o papel de aplicar estratégias adicionais ao mecanismo de balanceamento é do agente *Analyzer*. Esse agente analisa as URL's recebidas com base em um conjunto de regras, para notificar ao mecanismo de distribuição, quais URL's não devem mais ser entregues para esse cliente e consequentemente direcionar a outro nó através de um redirecionamento controlado pelo nó, fazendo uso do agente *Getter* do mecanismo de distribuição e balanceamento de carga, pois reconstruir este mecanismo pode ser um processo "caro" e deve ser evitado.

A existência de agentes adicionais ao agente *Analyzer* apoiando o mecanismo de balanceamento é extremamente possível e transparente para o *crawler* colaborativo, permitindo que agentes temporários embutidos de regras para análise de métricas, juntem-se aos agentes do *crawler* colaborativo, ajam por um determinado tempo ou mesmo permanentemente e saiam. Esses agentes podem complementar a atividade do agente *Analyzer,* aplicando métricas sobre as URL's, arquivos obtidos ou mesmo os servidores hospedeiros das páginas Web.

Arquitetura do *crawler* colaborativo, permite aos agentes fazerem dinamicamente parte do processo de análise de métricas para apoiar o mecanismo de balanceamento de carga. Bastando os agentes enviarem mensagens ao agente *Distributor,* orquestrador do esquema de redirecionamento do mecanismo de distribuição e balanceamento de carga.

4.3.1 – Identificação de rotas

Quando comparado o tempo de captura de uma página da Web, é importante considerar o impacto do tráfego e a rota de transmissão. Por exemplo, no caso de uma solicitação do jantar por telefone, se as estradas estão congestionadas, possivelmente a entrega demorará mais que o normal. Com isso, é possível optar por um outro local mais rápido, usando uma estrada maior e com menor tráfego. Por fim, todas as rotas entregarão os dados, entretanto, a rota mais direta será a rota mais rápida.

Um fator que afeta significativamente o tempo de captura de uma página da Web é, por exemplo, se o computador está conectado a Internet através de um provedor que oferece uma boa rota, assim, os dados requisitados serão obtidos rapidamente.

Muitos provedores de acesso à Internet são melhores no momento de selecionar a rota com menos congestionamento e mais direta entre dois computadores, e com isso tornam-se melhores ao prover dados para seus usuários. Uma ferramenta

utilizada para realizar essa medida de eficiência, que é conhecida como roteamento, é chamada de *traceroute*. A utilização do *traceroute* é capaz de ajudar na escolha de rotas e determinar o tempo de *download* de uma página da Web, identificando o caminho desde sua origem até o destino. Porém medidas de segurança tomadas por administradores de rede, podem impedir a detecção de algumas rotas, não garantindo a identificação da rota no momento em que for realizado o *download.*

Para o processo de captura realizado pelos diversos *crawlers* que atuam de forma colaborativa, foi utilizada a estratégia de atribuir a responsabilidade de captura de uma determinada página Web a um *crawler* que esteja melhor posicionado em relação ao servidor hospedeiro da página Web. Para o trabalho proposto, o conceito de "melhor posicionado" se aplica ao agente que conseguirá realizar a captura de forma mais eficiente em relação aos outros agentes, isso envolve a obtenção do tempo de resposta para baixar a página da Web.

4.3.2 – Estratégia de particionamento geográfico

Uma das estratégias capazes de auxiliar o mecanismo de balanceamento de carga, aplicando melhorias para o *crawler* colaborativo, é a estratégia de particionamento geográfico. Segundo Exposto, Macedo *et al.* (2005) essa estratégia tem como objetivo setorizar as áreas de atuação do *crawler* colaborativo, permitindo ao *crawler* somente capturar páginas Web hospedadas em servidores que estejam em uma mesma região geográfica dos servidores hospedeiros dos *crawlers*, buscando uma melhor conectividade entre os servidores para aumentar o desempenho em termos de captura de páginas da Web.

Essa estratégia segue uma estrutura de relacionamento baseada em proximidade e vizinhança. De acordo com Siamwalla, Sharma *et al.* (1998) a descoberta de rotas, enquadra-se no problema, mais geral, da descoberta de topologias de rede para o qual existem muitas abordagens, nenhuma das quais até ao momento, conseguiu capturar totalmente a topologia real da Internet.

A funcionalidade que permite o registro da localização geográfica das máquinas da Internet ocorre através do serviço de diretório chamado *whois* (Harrenstien, Stahl et al., 1985) mantido, entre outras, por entidades gestoras dos endereços IP, como por exemplo, as citadas em (LACNIC) e (ARIN).

O agente *Analyzer* é o responsável pela execução dessa estratégia no *crawler* colaborativo através de uma de suas implementações. Com base no uso do serviço de

whois/RIR's (*Regional Internet Registry*), esse agente identifica a partir do endereço IP do servidor hospedeiro da página Web, qual a sua posição geográfica em relação ao servidor hospedeiro do *crawler*. A posição geográfica é identificada tomando-se como base o país, ou seja, os *crawlers* somente acessam os servidores que estejam fisicamente localizados no mesmo país do servidor em que estejam hospedados. Em caso contrário, o agente notifica o agente *Distributor* sobre o redirecionamento de chaves geradas a partir do endereço *IP* analisado. O serviço de *whois/RIR's* torna-se viável nesse trabalho pelo uso de arquivos texto disponíveis para utilização gratuita, contendo toda a base de informações sobre as faixas de endereços *IP* utilizados no mundo e setorizados por região.

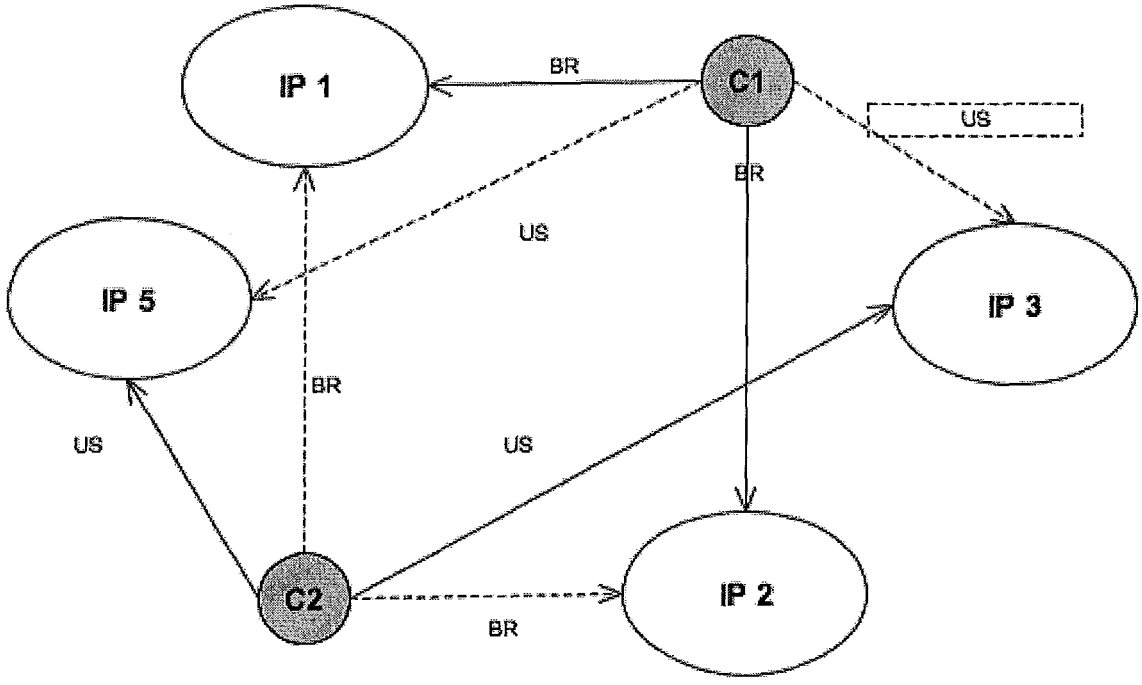


Figura 19 – Particionamento geográfico

A Figura 19 mostra o mecanismo de distribuição e balanceamento de carga definindo que o *crawler* C1 é responsável pelos IP's 1 e 3, e o *crawler* C2 responsável pelos IP's 5 e 2. Após atuação do agente *Analyzer* houve uma redistribuição das responsabilidades dos *crawlers* baseada em estratégia de particionamento geográfico.

4.3.3 – Estratégia de tempo de resposta

Segundo Loo, Krishnamurthy *et al.* (2004) as estratégias de particionamento baseadas na utilização da função *hash* aplicada ao endereço *IP* do servidor hospedeiro

de páginas Web, são considerados bons esquemas de particionamento, pois permitem aos *crawlers* extraírem todas as URL's pertencentes a um mesmo site e portanto, reduzindo a comunicação inter-partição. Este esquema embora robusto, permite um escalonamento leve e um mecanismo de roteamento de URL descentralizado, mas não tem ciência da rede e infra-estrutura de conexão, portanto, reduz as chances de otimizar o processo de captura.

A estratégia de particionamento geográfico apresentada na seção "4.3.2 – Estratégia de Particionamento Geográfico", mostra a vinculação de um mecanismo de distribuição e balanceamento de carga baseado em função de *hash* à um controle geográfico dos endereços *IP* com base no serviço *whois/RIR's*. Esse controle traz ciência de recursos de rede e infra-estrutura para o mecanismo de distribuição e balanceamento de rede, porém não existe a garantia de que o *crawler* mais indicado a capturar as páginas Web, seja o que estiver na mesma região geográfica do servidor hospedeiro das páginas Web, apontado pelo serviço *whois/RIR's*. Isso abriu precedente para a aplicação de uma estratégia baseada em tempo de resposta.

Para a estratégia de tempo de resposta, é utilizado um modelo simplificado com base na relevância dos dados coletados no momento do processo de captura. Como não existem informações de roteamento de rede ou topologia da Web disponíveis através das URL's, um mecanismo de *hash* é usado para definir o particionamento da Web.

Para otimizar o tempo de baixar páginas da Web no processo de captura de páginas, o objetivo é tentar aproximar ao máximo o agente *Downloader* dos servidores que armazenam as páginas a serem baixadas. Diminuindo a distância de comunicação entre o *crawler* colaborativo (agente *Downloader*) e os servidores Web. Isso pode ser realizado particionando o espaço da Web e atribuindo determinados espaços aos agentes *Downloader* dos *crawlers* colaborativos, aplicando estratégias para identificar o *Downloader* responsável por cada espaço identificado.

A estratégia de tempo de resposta (*RTT*) é utilizada pelo agente *Analyzer* para complementar o mecanismo de balanceamento de carga. Através da implementação do padrão de projeto *Cadeia de Responsabilidade* apresentado por Gamma, Helm *et al.* (1996), o agente *Analyzer* pode executar uma ou várias regras atuantes de forma complementar.

No processo de captura das páginas Web, o agente *Downloader* do *crawler* colaborativo atua como uma aplicação cliente, realizando visitas aos servidores e

capturando as páginas desejadas na Web. Durante esse procedimento é utilizado como métrica principal, o tempo de resposta, o qual significa o tempo utilizado para a captura de uma URL, e isto implica em: o tempo que o agente *Downloader* abre uma conexão com o servidor Web; envia uma requisição; o servidor Web aceita a requisição; o arquivo é baixado. Essa definição serve somente para mensurar a porção de tempo do lado cliente (*crawler*). Diferente do que foi apresentado por Olshefski, Nieh *et al.* (2002); Olshefski, Nieh *et al.* (2004) e Olshefski & Nieh (2006), a ação de identificar o *CPRT* (*Client Perceived Response Time*) é executada pela aplicação cliente e não pelo servidor, como realizado por Mukhtar (2003).

A partir da visão do agente *Analyzer*, o foco é determinar o tempo necessário para realizar a captura da página Web com todos os seus objetos relacionados (imagens, CSS, entre outros). Essa definição inclui latências, tais como: a latência de rede, a latência do servidor Web, recuperação dos objetos associados à página Web e a latência da máquina local onde está o agente. O tempo de resposta utilizado para mensurar e gerenciar as atividades dos agentes é o tempo de resposta percebido pelo agente *Analyzer*, auxiliando os agentes relacionados ao mecanismo de balanceamento a identificarem a rota que melhor proverá a URL a ser baixada.

No caso do *crawler* colaborativo a análise da métrica de tempo de resposta é feita pelo agente *Analyzer*. Para cada servidor visitado pelo agente *Downloader*, o agente *Analyzer* mede o tempo de resposta através da ação de baixar uma página da Web qualquer, existente em um dos sites armazenados pelo servidor. Caso o tempo de resposta esteja abaixo do configurado para o mínimo aceitável pelo agente *Downloader*, o agente *Analyzer* notifica o agente *Distributor* para que o cliente não mais receba chaves geradas para o *IP* do servidor analisado. O tempo de resposta é calculado em *Kilobytes*/segundo, medido com base no tempo total do processo de baixar dividido pelo tamanho da página Web capturada.

O agente *Analyzer* passa a agir como o responsável em realocar as URL's entre as partições já definidas pelo mecanismo de distribuição e balanceamento de carga.

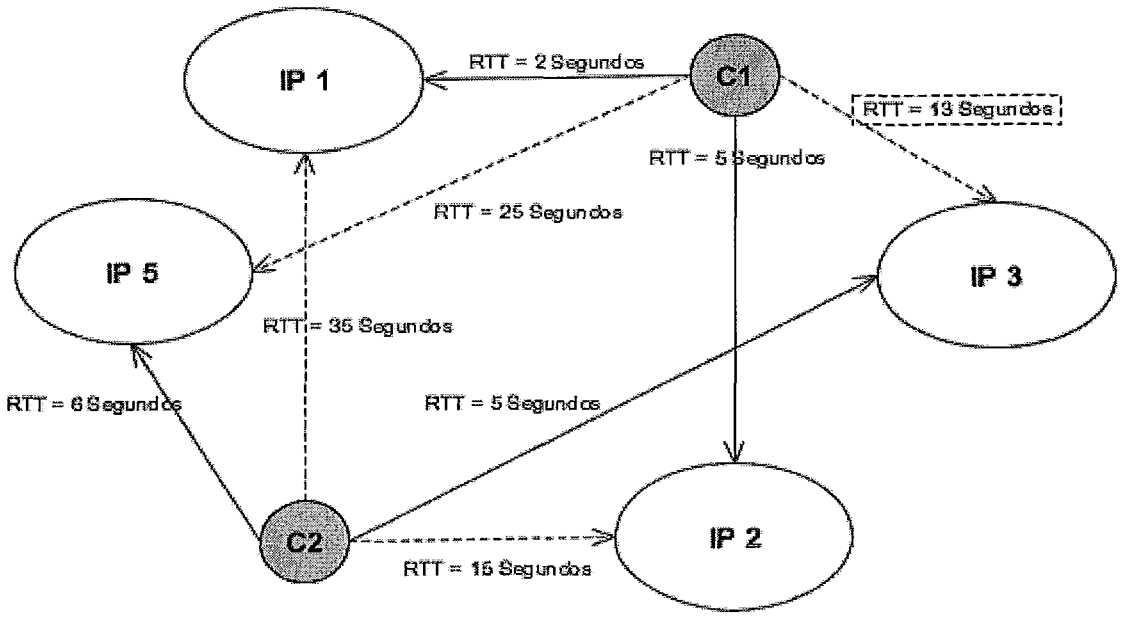


Figura 20 – Tempo de resposta

A Figura 20 mostra o mecanismo de distribuição e balanceamento de carga definindo que o *crawler* C1 é responsável pelos IP's 1 e 3, e o *crawler* C2 responsável pelos IP's 5 e 2. Após atuação do agente *Analyzer* houve uma redistribuição das responsabilidades dos *crawlers*, baseada em estratégia de tempo de resposta.

Uma vez que os servidores Web estejam vinculados aos *crawlers*, através da atuação do agente *Analyzer*, não significa que estes servidores não serão mais visitados por este e distribuídos novamente. O papel do agente *Analyzer* é exatamente agregar dinamismo e flexibilidade ao mecanismo de distribuição e balanceamento de carga, permitindo aos *crawlers* existentes e aos novos *crawlers*, receberem constantemente a responsabilidade de atuar em diferentes servidores Web otimizando o tempo de captura de páginas na Web.

4.3.4 – Política de visita

As atividades dos *crawlers* abrangem uma porção significante do tráfego da Web. Atualmente os *crawler* são bem vindos nos sites, pois atuam na sua grande maioria vinculados as máquinas de busca, onde essas ajudam exponencialmente os sites a conseguirem novos visitantes. Entretanto, alguns sites desejam limitar a quantidade ou escopo de acessos não realizados por seres humanos, pois muitos

destes sites pagam seus custos baseados em recursos como por exemplo: consumo de rede ou processamento de CPU.

Os *crawlers* seguem convenções desenvolvidas na Web, as quais governam suas ações. Essas regras não são aplicadas por nenhuma autoridade, mas *crawlers* que agem corretamente devem seguí-las. As principais regras são: a consulta ao arquivo robots.txt (Koster, 1996) existente em cada site é o princípio de "cortesia", onde os administradores de sites esperam que os *crawlers* não requisitem páginas em uma rápida sucessão, pois as ações dos *crawlers* iriam impactar as visitas dos usuários nos sites em termos de desempenho. É esperado que os *crawlers* efetuem uma pausa entre cada requisição sucessiva para um mesmo site. Infelizmente não existe uma regra determinada para o tempo dessa pausa, esta pode variar conforme a largura de banda de cada site. Heydon & Najork (1999) definem um intervalo de tempo de 60 segundos, independente do tipo de site. Najork & Wiener (2001) definem o tempo de intervalo igual ao tempo gasto na captura da última página do mesmo servidor Web, Cho & Garcia-Molina (2003) definem 10 segundos de intervalo independente do tipo de site e Cho, Garcia-Molina *et al.* (2004) definem um intervalo de 5 segundos para grandes sites comerciais e 20 segundos para pequenos sites privados.

O agente *Frontier* adiciona à estratégia de navegação à utilização do princípio de "cortesia", seguindo o intervalo de 60 segundos apresentado por Heydon & Najork (1999). Além de manter uma política que não prejudica a velocidade de acesso pelos usuários do site, isso evita que os que mantém uma administração rigorosa, possam bloquear o acesso dos *crawlers*, proibindo qualquer requisição que tenha como origem o endereço *IP* do servidor hospedeiro do *crawler*. O principio de "cortesia" é aplicado pelo agente *Frontier* utilizando uma lista auxiliar a fila de URL's a serem capturadas. Nesta lista são armazenados todos os endereços IP dos servidores visitados até o momento, e o horário a partir do qual poderão ocorrer novos acessos ao servidor. Sempre que uma nova URL é capturada, o horário de permissão para o servidor que hospeda a URL é atualizado, e caso ainda não exista um registro desse servidor na lista, automaticamente é adicionado um novo registro já com o próximo horário de permissão. Toda URL que for bloqueada no momento de captura por não ter ainda permissão em relação ao principio de "cortesia", é devolvida a fila de URL's a serem capturadas e uma nova URL que se encontra em posição posterior, é obtida para o processo de captura, porém, sendo também analisada pelo princípio de "cortesia".

Para a regra de utilização do arquivo robots.txt, cada site deve ter em seu diretório raiz um arquivo com o nome robots.txt, o qual especifica o que cada *crawler* pode pesquisar. Esse arquivo lista os prefixos das URL's onde os *crawlers* não são bem vindos. Os *crawlers* "bem" desenvolvidos devem honrar a aplicação dessas restrições. Porém, quando sites estão utilizando domínios *virtuais*, a utilização do arquivo robots.txt pode se tornar um problema.

O mecanismo de domínios *virtuais* permite um único servidor físico hospedar diferentes sites. O conjunto de páginas web para exemplo.org.br e exemplo.com.br devem ambos residirem no mesmo servidor com um único endereço *IP*. Para um ser humano usar os endereços exemplo.com.br e exemplo.org.br, o procedimento é diferente. O servidor Web receberá a requisição e a gerenciará de acordo. Para *crawlers* entretanto, a situação é complicada pelo fato de que diversos "apelidos" são também frequentemente registrados para um único site. Os endereços http://exemplo.com.br e http://www.exemplo.com.br, por exemplo, devem fazer referência a um único site, não a dois sites como no exemplo anterior. O *crawler* proposto nesse trabalho identifica quando dois nomes mapeiam um único site, evitando a captura de páginas repetidas, enquanto que no mesmo período de tempo trata os nomes de vários sites virtuais de forma distinta, mesmo sabendo que eles fazem referência a um mesmo endereço *IP*.

Crawlers não sofisticados não fazem distinção entre casos de diversos nomes e cegamente unem as diversas URL's associadas a um mesmo endereço *IP*.

O *crawler* proposto nesta dissertação resolve esse problema através do agente *Downloader*, associando cada domínio à existência de um arquivo robots.txt, e os compara antes de concluir que dois domínios referem-se a um único site. Essa análise de "apelido" é extremamente custosa se executada todas as vezes em que uma página é capturada. Porém, sempre que for identificado um novo domínio, é obtida a lista de domínios que resolvem o mesmo endereço *IP*, e uma análise comparativa entre todos os arquivos robots.txt é realizada. Sendo identificados arquivos robots.txt iguais, o *crawler* analisa um conjunto aleatório de páginas pertencentes aos domínios que tiveram arquivos robots.txt iguais, caso sejam identificadas igualdades entre as páginas, isso é analisado através do valor *hash* gerado do arquivo, o *crawler* passa a identificar que os domínios proprietários desses arquivos fazem referência a um mesmo site.

4.3.5 – Evitar armadilhas

Uma armadilha para *crawler* é uma URL ou conjunto de URL's que atrapalham um *crawler* na captura de páginas indefinidamente. Algumas armadilhas não são intencionais. Por exemplo, um *hyperlink* simbólico dentro de um sistema de arquivos pode criar um ciclo. Outras armadilhas são introduzidas intencionalmente. Por exemplo, pessoas escrevem armadilhas usando programas que dinamicamente geram um conjunto infinito de páginas HTML, sem nenhum valor, ou seja, sem conteúdo algum, ou sempre com o mesmo conteúdo.

Um outro exemplo seria uma página dinâmica que implementa um calendário, onde um usuário sempre clica na opção próximo mês e em algum momento não será mais interessante que esta opção seja constantemente visitada, pois uma possível data será encontrada satisfazendo a pesquisa do usuário. Uma análise humana pode racionalmente identificar que não é interessante pesquisar o calendário 50 anos a frente, entretanto, um *crawler* não. Isso é um dos muitos exemplos de armadilhas encontradas que envolvem ciclos e/ou duplicidades de informação, os quais podem ser identificados em um momento posterior a captura, mas o interessante é evitar a captura deles.

Em geral, técnicas automáticas para evitar armadilhas não são aplicadas, não tendo sido observado nenhum caso documentado a respeito, entretanto, sites contendo armadilhas para *crawlers* são facilmente percebidos, a partir da grande quantidade de documentos descobertos.

O *crawler* descrito nesta dissertação também utiliza a propriedade de tempo limite, com o foco em evitar armadilhas. Conforme mencionado anteriormente, evidências de armadilhas são páginas com tamanho de *Gigabytes* ou mesmo páginas infinitas onde programas maliciosos fazem com que o *crawler* fique agindo em ciclo. O *crawler* usa tempo limite o qual refere-se ao período em que uma página está sendo capturada. Se a página enquanto capturada excede ao tempo limite, o seu processo de captura é abortado e uma nova URL tem o processo de captura iniciado pelo *crawler*.

Capítulo 5 – Construção do *crawler* colaborativo

Nesse capítulo é descrita a implementação do *crawler* colaborativo, os componentes desenvolvidos, a implementação de estratégias para orientar a captura de páginas da Web e as decisões de planejamento tomadas para a construção do *crawler* colaborativo em uma plataforma *peer-to-peer*.

São descritos detalhes e decisões adotadas na implementação do projeto com foco no processo de captura de página e colaboração entre os *crawlers* distribuídos. Este capítulo mostra as principais características das classes envolvidas e seus relacionamentos com outros componentes do projeto. Cada seção descreve qual o ponto da arquitetura abordado e os detalhes de programação utilizados para completá-lo.

5.1 – Plataforma *peer-to-peer* para os agentes

Para suportar os diversos agentes citados anteriormente, os quais formam os *crawlers* colaborativos, foi utilizado o COPPEER (Miranda, Xexéo et al., 2006), que consiste em um projeto de pesquisa em andamento no Laboratório de Banco de Dados da Universidade Federal do Rio de Janeiro COPPE/UFRJ. A primeira versão do COPPPER foi elaborada como um *framework peer-to-peer* para suportar arquiteturas de computação em grade (*grid*). Atualmente o projeto COPPEER 2.0 tem como foco a implementação de um *framework* para desenvolvimento e execução de aplicações colaborativas em *peer-to-peer*. A arquitetura global do COPPEER é composta de quatro camadas(Miranda, Xexéo et al., 2006):

- Agência: executa ações genéricas de *peer-to-peer* para as outras camadas;
- Negociação: abrange a localização de aplicações e gerenciamento de sessão;
- Integração: oferece paradigmas computacionais de alto nível para componentes de aplicações;
- Aplicação: gerencia as questões relacionadas a colaboração.

Quando o COPPEER é inicializado em um *peer*, uma agência é criada para gerenciar diferentes ambientes, células e agentes em nome de aplicações.

Um ambiente é um conjunto de células interconectadas. Uma agência pode atuar em diversos ambientes simultaneamente, entretanto, pode gerenciar somente uma célula por ambiente. Portanto, uma célula é identificada de forma não ambígua por sua agência e seu ambiente.

Uma célula é um espaço compartilhado que oferece a seus clientes uma interface contendo operações para leitura e escrita de entradas, assinatura para notificações de escritas contendo dados específicos e construir ou finalizar conexões para outras células conhecidas.

Um cliente de uma célula pode ser um agente ou qualquer aplicação. Um mecanismo de utilização é empregado para controlar por quanto tempo uma célula estará apta a cuidar da requisição de um cliente. Quando um cliente deseja armazenar uma entrada, este especifica um determinado tempo de utilização. Então, a célula utiliza um determinado controlador para gerenciar este tempo.

Uma entrada é um objeto de dados o qual pode ser armazenado em uma célula, continuamente propagar seus dados para as células de vizinhos e mudar seu próprio estado interno. Desenvolvedores de aplicações devem criar entradas com regras específicas de propagação através da implementação de um conjunto de métodos de ciclo de propagação.

Um agente é um pedaço de software associado a um ambiente o qual acessa células e as move através de agências para executar ações distribuídas. Agentes podem criar novos agentes e deslocar-se entre agências.

O *crawler* desenvolvido e descrito nesta dissertação seguiu a orientação de desenvolvimento baseado em multiagente, sendo formado por um conjunto de agentes atômicos em suas funções e construídos sobre o *framework* COPPEER. A utilização deste *framework* adicionada ao planejamento de agentes atômicos e autônomos, permitiu ao *crawler* colaborativo uma arquitetura de agentes fracamente acoplados. Podendo ser substituídos a qualquer momento por outros agentes que executem as mesmas tarefas, porém, seguindo estratégias diferentes, como por exemplo, o caso do agente *Frontier* e dos agentes *Bulk* e *Take* que processam a distribuição de URL's entre os *crawlers* colaborativos.

O mecanismo de troca de mensagens entre os agentes, implementado pelo COPPEER, permite um baixo acoplamento entre os agentes, bastando que um agente conheça o tipo de entrada que precisa ser lida e ser notificado quando ocorre a escrita em uma célula. Seguindo essa abordagem, todas as estratégias descritas nesse capítulo

se tornam independentes do *crawler* colaborativo, tornando a arquitetura deste totalmente independente e não orientada às estratégias utilizadas.

O desenvolvimento do *crawler* colaborativo cooperou com o projeto COPPEER sendo a primeira aplicação a executar na Internet, ajustando, validando e testando o mecanismo de comunicação utilizado pelo COPPEER para a troca de mensagens entre os agentes. Também auxiliou nos testes e ajustes do mecanismo de controle dos agentes, sendo a primeira aplicação a utilizar um considerável número de agentes trabalhando de forma atômica.

5.2 – A comunicação entre os agentes formadores do *crawler* colaborativo

Cada um dos agentes formadores do *crawler* colaborativo comunica-se com um ou mais agentes através do envio e recebimento de mensagens, conforme ilustrado na Figura 21. Estas constituem a interface de comunicação entre os agentes. Cada agente está associado a uma célula que funciona como uma "caixa de entrada", onde mensagens são inseridas. Quando uma mensagem é colocada na "caixa de entrada" o agente é notificado. Entretanto, será de responsabilidade do agente decidir se e quando ele lerá a mensagem e como reagir. Esse esquema proporciona baixo acoplamento e flexibilidade a todos os agentes construídos sobre o *framework* COPPEER, bastando que este conheça as mensagens de entrada e saída de cada um dos agentes, para que novos agentes possam agir competitivamente ou agir colaborativamente na realização de tarefas.

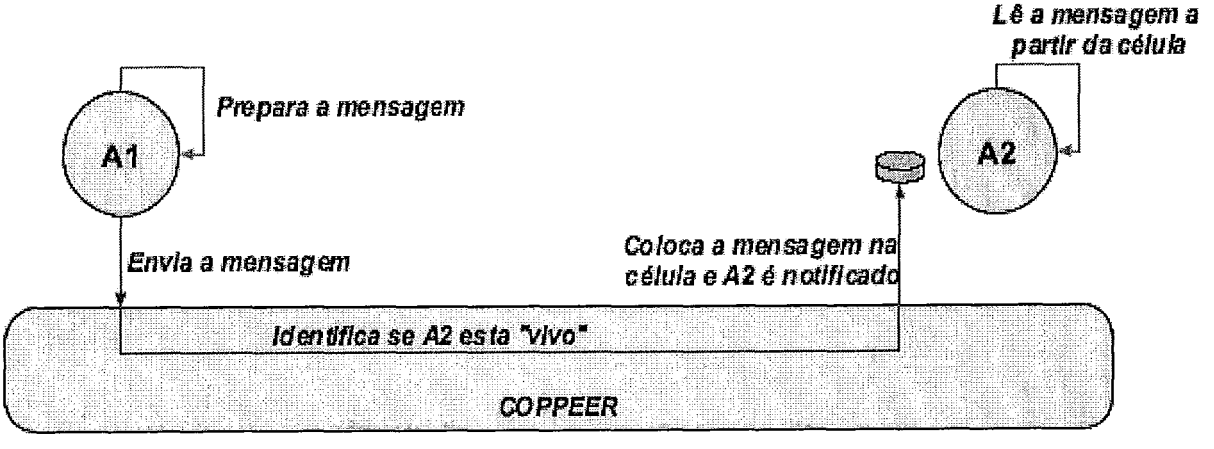


Figura 21 - Modelo de comunicação entre os agentes formadores do *crawler* colaborativo.

5.2.1 – Mecanismo para distribuição e balanceamento de carga entre os *crawlers* colaborativos

Uma das estratégias utilizadas para a distribuição e balanceamento de carga entre *crawlers* distribuídos é a aplicação de uma *DHT*. Segundo Yatin, Sriram *et al.* (2005) uma *DHT* fornece propriedades úteis de balanceamento de carga, roteamento baseado em conteúdo e custo logarítmico para localização de nós em uma estrutura de rede organizada. Isso permite facilmente despachar requisições entre os *crawlers* e organizar a distribuição de URL's entre eles.

A localização de dados em um sistema formado por diversos nós deve ser rápida; nós devem estar aptos a entrar e sair frequentemente do sistema sem afetar sua robustez e eficiência; e a carga deve ser balanceada entre os diversos nós disponíveis. Para atender a alguns desses pontos, foi adotada a utilização de *DHT* na estratégia de distribuição e balanceamento de carga, visto a transparência no balanceamento de carga caracterizado pelas URL's a serem capturadas, a estrutura de distribuição de dados e a localização de nós em tempo logarítmico. A *DHT* utilizada foi construída com base no trabalho realizado por Stoica, Morris *et al.* (2003).

Cada nó da *DHT* tem um identificador único gerado através de uma função *SHA-1(Secure Hash Algorithm)*, obtido executando-se a função sobre o endereço *IP* e o número da porta em que os agentes estejam sendo executados. A *DHT* controla esses identificadores ocupando um espaço circular identificado. Chaves são também mapeadas dentro desse espaço, através da execução da função *SHA-1* sobre o endereço IP do servidor que mantém o domínio da URL a ser capturada pelo *crawler*, onde é gerado um identificador para a chave. A *DHT* define o nó responsável pela chave como sendo o sucessor do identificador da chave. O sucessor de um identificador *j* é o nó com o menor identificador maior do que *j* dentro do anel. Para implementar esse processo a *DHT* mantém em cada nó uma tabela auxiliadora com informações sobre $O(\log N)$ outros nós, onde N é o número total de nós. O mecanismo de pesquisa da *DHT* envia mensagens para $O(\log N)$ nós consultarem suas tabelas.

Portanto a *DHT* pode localizar dados eficientemente mesmo com um grande número de nós. A *DHT* suporta somente uma operação: dada uma chave, ela determinará o nó responsável pela chave. A *DHT* não necessariamente armazena as

chaves e valores, mas provê uma camada que permite seus clientes armazenarem as chaves e valores; o *crawler* é um tipo de cliente que utiliza essa camada da *DHT*.

5.2.1.1 – O processo de localização de nós na *DHT*

Os nós da *DHT* usam duas estruturas de dados para executar a pesquisa: um sucessor e uma tabela auxiliadora. O sucessor é requerido para localizar os nós com exatidão, logo, a *DHT* é cautelosa em manter sua precisão. A tabela auxiliadora acelera a pesquisa, mas não garante precisão, pois a *DHT* é menos agressiva em manter esta tabela, assim, a mesma é sempre atualizada durante o procedimento de estabilização.

Cada nó mantém uma lista de identificadores e endereços de seus sucessores imediatos no anel que forma a *DHT*. O fato de que todo nó conhecer seu próprio sucessor, significa que o nó pode sempre processar a pesquisa corretamente, se a chave desejada estiver entre o nó e seu sucessor; de outra forma, a pesquisa pode ser transferida para o sucessor, que move a pesquisa exatamente mais próxima ao seu destino.

Um novo nó *n* aprende sobre seus sucessores quando ele se une a *DHT*, solicitando a um nó existente que execute uma pesquisa pelo sucessor de *n*. *n* então pede ao seu sucessor pela lista de sucessores que constituirão a tabela auxiliadora. A principal complexidade envolvida com a lista de sucessores está em notificar um nó existente, quando um novo nó passa a ser seu sucessor. O procedimento de estabilização resolve este problema, preservando a conectividade com os sucessores.

Caso o mecanismo de pesquisa seja executado somente com a lista de sucessores, ele irá requerer uma média de *N*/2 trocas de mensagens, onde *N* é o número de nós. Para reduzir o número de mensagens requeridas para *O*(log N), cada nó mantém uma tabela auxiliadora com *m* entradas. A i^{th} entrada na tabela auxiliadora no nó *n* contém a identificação do primeiro nó que sucede *n* pelo menos 2^{i-1} no identificador do círculo. Portanto, todo nó conhece a identificação de nós em um intervalo de potência de 2 no círculo de identificadores a partir de sua posição. Um novo nó inicia sua tabela auxiliadora através da pesquisa de nós existentes.

5.2.1.2 – Ociosidade e sobrecarga nos *crawlers* colaborativos

Dependendo do mecanismo de distribuição e balanceamento de carga implementado para apoiar os *crawlers* distribuídos e colaborativos, poderá ocorrer da

distribuição de carga não ficar balanceada entre os nós, mesmo que temporariamente, como é o caso de uma *DHT*. Também é necessário lidar com situações onde alguns *crawlers* podem ser constituídos de recursos abundantes, tornando suas atividades de rápida execução ou mesmo desprovidos de recursos, tornando suas atividades menos performáticas em relação à de outros *crawlers*.

A ociosidade e sobrecarga dos *crawlers* passam a ser distribuídas entre os *crawlers* que atuam em conjunto, independente do mecanismo de distribuição e balanceamento de carga utilizado. São regras adicionais implementadas no agente *Analyzer*, que somente serão executadas caso o agente perceba a ociosidade ou sobrecarga em capturar páginas. A partir do momento em que é percebida a ociosidade ou sobrecarga, o agente *Analyzer* envia uma mensagem para o agente *Distributor*, informando que o *crawler* está ocioso e poderá receber URL's extras para capturar. Quando um *crawler* está sobrecarregado, ele envia uma mensagem ao mecanismo de distribuição, para que seja localizado um *crawler* que esteja ocioso e possa receber um conjunto de URL's para serem capturadas. A atuação sobre estas URL's por parte do *crawler* que as receberá é normal, como se pertencessem ao intervalo determinado pelo mecanismo de distribuição.

Esse envio de mensagem ao mecanismo de distribuição é uma notificação ao agente *Distributor* informando que o *crawler* precisa despachar parte de suas URL's para outros *crawlers*, a fim de desafogar suas atividades e aproveitar a ociosidade de outros *crawlers*.

O controle entre a ociosidade e a sobrecarga para o recebimento e envio de URL's é feito através do estado em que o *crawler* se encontra. A seguir são apresentados estes estados:

- Estado Ocioso: recebe URL's enviadas pelo mecanismo de distribuição e que sejam pertencentes ao intervalo alocado como de responsabilidade do *crawler* e URL's enviadas por *crawlers* que estejam sobrecarregados;
- Estado Normal: recebe somente URL's enviadas pelo mecanismo de distribuição e que sejam pertencentes ao intervalo alocado como de responsabilidade do *crawler*;
- Estado Sobrecarga: recebe somente URL's enviadas pelo mecanismo de distribuição e que sejam pertencentes ao intervalo alocado como de

responsabilidade do *crawler*. Porém esse estado informa ao mecanismo de distribuição que o *crawler* tem a necessidade de despachar URL's caso ele localize um *crawler* ocioso.

O estado de um *crawler* colaborativo é determinado e notificado ao agente *Analyzer* através do agente *Frontier*. O agente *Analyzer* informa ao mecanismo de distribuição e balanceamento de carga, como estes devem atuar perante o *crawler*, enviando somente as URL's pertencentes ao intervalo alocado para o *crawler*, ou enviando também URL's extras vindas de outros *crawlers* sobrecarregados. Os estados "ocioso", "normal" ou "sobrecarga" são definidos a partir da quantidade de URL's existentes na base dos *crawlers* a serem capturadas. Os valores que definem esses estados são:

- Quantidade mínima de URL's que indicam ociosidade: Quando o agente *Frontier* identificar que existe um número inferior a uma quantidade mínima aceitável de URL's na fila de URL's a serem capturadas, este notifica o agente *Analyzer* sobre o estado "ocioso";
- Quantidade de URL's que indicam o estado normal: Quando o agente *Frontier* identificar que existe um número entre uma quantidade mínima e máxima de URL's na fila de URL's a serem capturadas, ele notifica o agente *Analyzer* sobre o estado "normal";
- Quantidade de URL's que indicam o estado sobrecarga: Quando o agente *Frontier* identificar que existe um número superior a uma determinada quantidade de URL's na fila de URL's a serem capturadas, ele notifica o agente *Analyzer* sobre o estado "sobrecarga".

Associado ao estado de "sobrecarga" existe a quantidade de URL's a serem despachadas para um *crawler* ocioso.

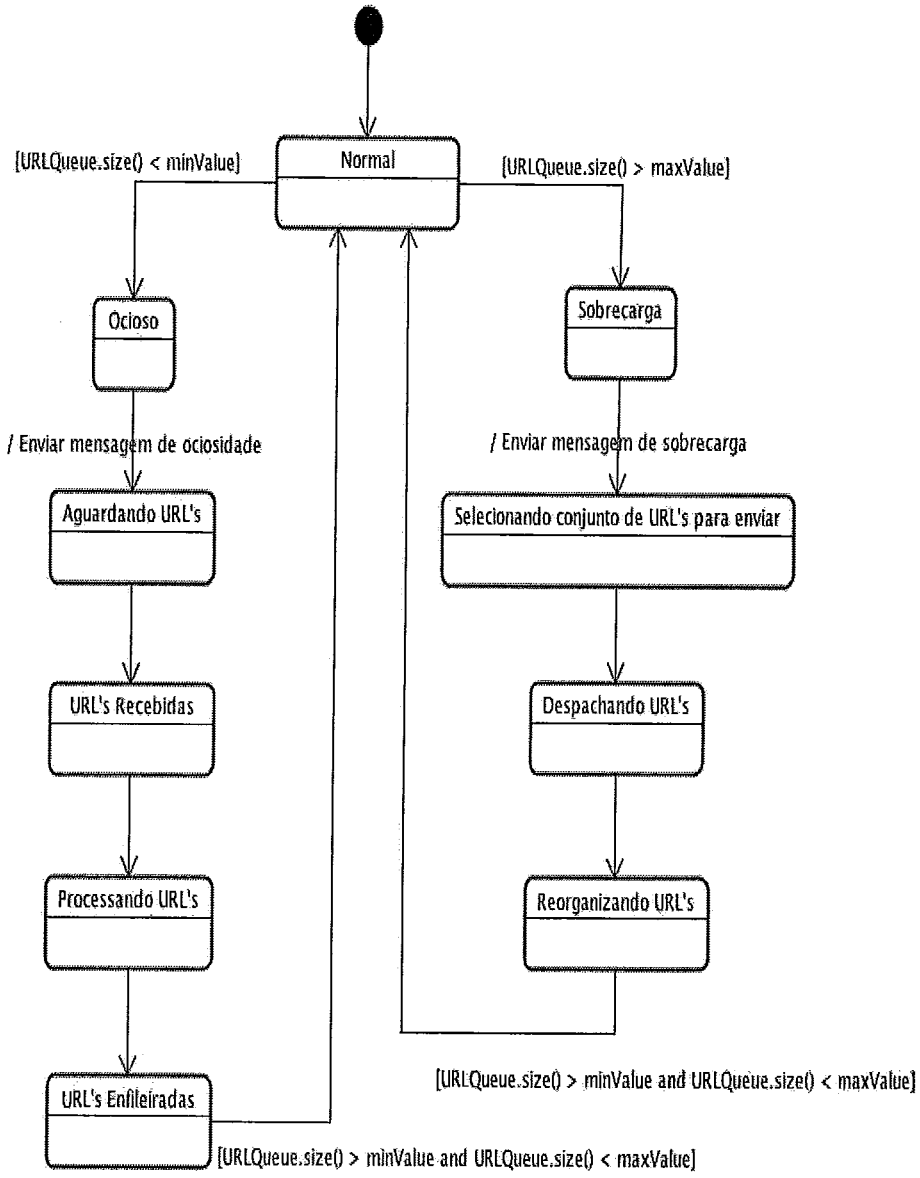


Figura 22 - Diagrama de Estado do *crawler* colaborativo em relação a ociosidade e sobrecarga (UML 1.5)

O diagrama de estado apresentado na Figura 22, é uma visão simplificada dos estados que o *crawler* atravessa durante o processo de captura de páginas na Web.

5.3 – Implementação do mecanismo para distribuição e balanceamento de carga entre os *crawlers* colaborativos

O mecanismo de distribuição e balanceamento de carga utilizado foi baseado no trabalho apresentado por Stoica, Morris et *al.* (2003), dando origem a uma *DHT*, implementada através dos agentes *Distributor* e *Synchronizer*. Esses executam sob o

COPPEER, o qual oferece uma arquitetura *peer-to-peer* simétrica e descentralizada, sendo capaz de lidar com um grande volume de nós participantes na *DHT*.

O agente *Distributor* controla as informações de um nó da *DHT*, como o sucessor, tabela auxiliadora, chaves recebidas e chaves enviadas. Todo o mecanismo de pesquisa da *DHT* é gerenciado pelo agente *Distributor* que envolve o envio e recebimento de mensagens para localização e distribuição das chaves. As chaves são as URL's a serem capturadas pelos *crawlers* colaborativos e são geradas a partir do endereço IP do servidor hospedeiro da página Web, dessa forma, a agente *Distributor* exerce o balanceamento de carga através da setorização de servidores Web entre os *crawler* colaborativos. Os *crawler* colaborativos são vistos como clientes da *DHT*, onde essa é uma camada responsável por localizar os nós que cuidarão de cada uma das URL's. Os agentes *Take* e *Bulk* do *crawler* colaborativo agem diretamente com o agente *Distributor* da *DHT*.

Toda a comunicação entre esses agentes é realizada através de troca de mensagens do *framework* COPPEER, o que torna o cliente independente do mecanismo de distribuição e balanceamento de carga utilizado. O agente *Bulk* é o responsável por enviar o conjunto de *hyperlinks* extraídos de cada página Web capturada ao agente *Distributor*, e o agente *Take* receberá cada URL pertencente ao cliente, do mecanismo de distribuição através do agente *Distributor*.

5.3.1 – O relacionamento entre um nó da *DHT* e seus clientes

A *DHT* implementada não está acoplada a um único tipo de cliente, com isso, qualquer agente pode fazer uso desse mecanismo adicionado ao *framework* COPPEER, de forma transparente, não sendo necessário que a *DHT* conheça o cliente. A utilização do mecanismo de distribuição e balanceamento de carga é iniciada através de uma mensagem de pedido de junção feito pelo agente *Join* do *crawler* colaborativo cliente, diretamente ao agente *Distributor* controlador de um nó da *DHT*, e a partir disso o cliente é adicionado ao nó da *DHT*. Toda chave enviada ao nó e que pertença ao intervalo desse nó, será despachada diretamente para o cliente.

Com a utilização do *framework* COPPEER um cliente não precisa necessariamente estar no mesmo servidor do nó da *DHT*, pois a comunicação entre o cliente e nó da *DHT* é realizada através dos agentes da *DHT* e do cliente. No momento em que um nó passa a fazer parte da *DHT* não sendo necessário ter um cliente vinculado a ele. Isso porque cada nó contém um agente chamado

Synchronizer, responsável pela sincronização de chaves com seu cliente. Dessa forma, um nó poderá receber e despachar chaves naturalmente na *DHT*, armazenando em uma fila todas as chaves recebidas que pertençam ao seu intervalo.

No momento em que exista o pedido de junção de um cliente para esse nó, todas as chaves armazenadas no nó são automaticamente despachadas para o cliente que no caso do *crawler* colaborativo, as recebe através do agente *Take*, respeitando um limite máximo de chaves enviadas em cada transação. Todas as chaves recebidas pelo nó anteriormente à junção do cliente, são persistidas em disco de forma a otimizar a alocação de memória por parte do nó da *DHT*.

Um nó da *DHT* não está vinculado a um único cliente, podendo lidar com um conjunto de clientes distribuídos através de diversos servidores. No caso de um único cliente, toda chave pertencente ao nó será automaticamente despachada para seu cliente, porém, no caso de vários clientes, podem ser utilizadas regras de distribuição entre estes, facilmente implementadas através de interfaces definidas na *DHT*. Um exemplo de regra de distribuição implementada e utilizada foi a técnica *round-robin*, onde é realizado um balanceamento das chaves recebidas entre os clientes de um nó, de forma que cada cliente receba alternativamente uma chave.

A *DHT* implementada como uma estratégia para o mecanismo oferecido pelo agente *Distributor* foi adicionada de um esquema de otimização, que visa melhorar os recursos utilizados pelos clientes, de forma transparente e independente.

Os nós da *DHT* provêem a funcionalidade de migrar seus clientes através do agente *Wanderer*, despachando para seu sucessor um determinado cliente que não esteja atendendo as necessidades do nó da *DHT*, como por exemplo, a capacidade de processamento ou armazenagem. Para o *crawler* colaborativo foi implementada uma regra que determina que a partir do momento no qual o agente *Downloader* estiver sendo executado, e um determinado número de domínios abrangidos pelo nó da *DHT* e visitados pelo agente *Downloader* estiver com o tempo de resposta abaixo de um determinado valor, uma notificação de mudança de nó é enviada pelo agente *Distributor*. Isso é possível porque o agente *Distributor* contabiliza todas as mensagens de solicitação de não recebimento de URL's enviadas pelo agente *Analyzer*.

Com a notificação de mudança, o nó envia o cliente ao seu sucessor, porém caso o sucessor rejeite o recebimento do cliente, o nó automaticamente busca o próximo nó a enviar o cliente em sua tabela auxiliadora. Essa ação ocorre até que o

cliente seja aceito por um nó da *DHT* ou não existam mais nós na tabela auxiliadora para enviar o cliente, fazendo com que o cliente permaneça no nó original. Por se basear no mecanismo de troca de mensagens do COPPEER, a arquitetura do *crawler* colaborativo torna-se totalmente independente, não importando a qual nó da *DHT*, ou seja, agente *Distributor*, ele esteja vinculado, nem se foi aplicada uma estratégia de migração de cliente, pois suas tarefas independem de qualquer orientação dessa estratégia.

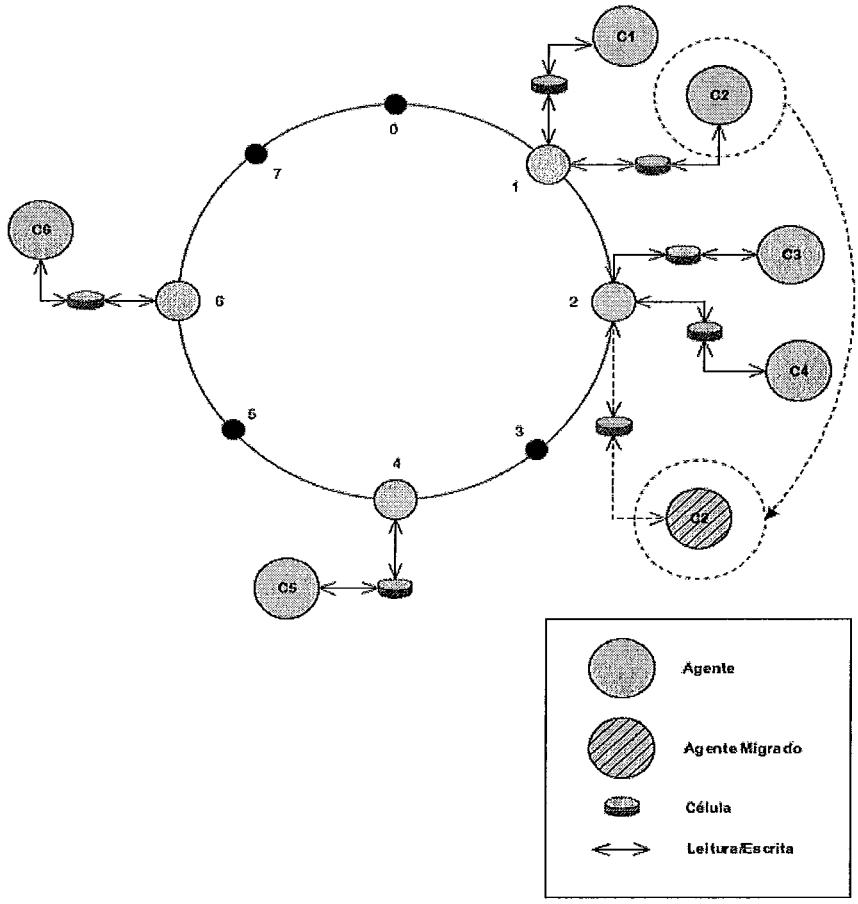


Figura 23 – Formação da *DHT* com seus clientes

A Figura 23 mostra uma *DHT* formada com seus clientes, ilustrando a trajetória de um cliente entre dois nós distintos. Os círculos tracejados representam as células do COPPEER, mecanismo de leitura e escrita de mensagens entre os agentes.

5.3.2 – Operações dinâmicas e falhas de nós da *DHT*

Na prática, uma *DHT* precisa lidar com a junção de nós no sistema, com nós que falham ou nós que deixam a *DHT* voluntariamente. A *DHT* implementada para esse trabalho e utilizada como um componente do *framework* COPPEER, lida apenas

com a junção de nós no sistema, entretanto, visualiza-se como um importante trabalho futuro, o tratamento de falhas e saídas voluntárias dos nós, como por exemplo, as técnicas apresentadas por Reza, Esther *et al.* (2007). Com isso nenhum tipo de replicação das chaves e valores é mantida, sendo assim, se um nó sai voluntariamente ou involuntariamente da formação da *DHT*, todas as chaves e valores armazenados nesse nó serão perdidas e ocorrerá a "quebra" da *DHT*.

A implementação atual garante que cada nó mantenha seu sucessor atualizado. Isso é possível através da utilização de um protocolo básico de estabilização, desenvolvido com base no protocolo de estabilização apresentado por Stoica, Morris et *al.* (2003). Esse procedimento de estabilização garante adicionar nós a *DHT* visando preservar a acessibilidade dos nós existentes, notificando estes da existência de um novo nó na *DHT*.

Caso a junção de um nó que tenha ocorrido em determinada região da *DHT*, ocorrer antes da estabilização ter sido finalizada, uma das seguintes ações poderá ocorrer: a mais comum é que todas as entradas da tabela auxiliadora envolvidas no processo de localização, estejam razoavelmente atualizadas, permitindo ser localizado o sucessor correto em $O(\log N)$ passos; o segundo caso ocorre quando o sucessor está sendo apontado corretamente, entretanto, a tabela auxiliadora está incorreta. Isso produzirá uma localização correta, porém mais lenta; no último caso, os nós existentes na região "afetada" estão com os sucessores incorretos, ou chaves não tenham sido migradas para os novos nós que se juntaram a *DHT*, ocasionando falha no mecanismo de pesquisa. Os agentes da *DHT* perceberão que o dado desejado não foi localizado, a pesquisa então é executada novamente.

O processo de junção ocorre quando um nó *n* inicia, enviando um pedido de junção ao agente *Distributor*, esse agente representa um nó *n'* existente na *DHT* e conhecido pelo nó *n*. O pedido de junção solicita ao nó *n'* para localizar o sucessor imediato de *n*.

Cada nó executa periodicamente o agente *Stabilizer*, tornando possível conhecer os novos nós participantes da *DHT*. Quando um nó *n* executa o agente *Stabilizer*, este pede ao seu sucessor pelo sucessor do predecessor *p*, e decide se *p* deve ser sucessor de *n*. Isso no caso de *p* ter se juntado recentemente ao sistema. O agente *Stabilizer* também notifica o sucessor de *n* da existência de *n*, dando ao sucessor a chance de mudar seu predecessor para *n*. O sucessor fará isso somente se ele não conhecer a existência de um nó mais próximo do que *n*. Sempre que for

identificada uma mudança de sucessor, a tabela auxiliadora é atualizada. A Figura 24 exemplifica o procedimento de junção de um novo nó a *DHT*.

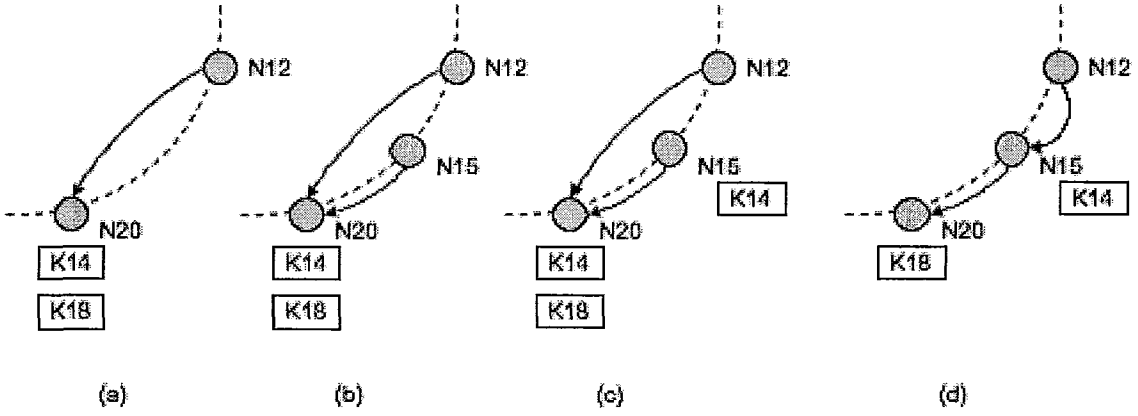


Figura 24 - Exemplo ilustrando a operação de junção.

Na Figura 24, o nó 15 junta ao sistema entre os nós 12 e 20. O arco representa o relacionamento de sucessor. a Estado inicial: nó 12 aponta para o nó 20; b nó 15 localiza seu sucessor 20 e aponta para ele; c nó 15 copia todas as chaves existentes em 20, que pertençam ao intervalo entre 15 e 20; d executa o procedimento de estabilização para atualizar o sucessor do nó 12 para o nó 15.

5.3.3 – Organização dos *crawlers*

O desenvolvimento dos *crawlers* sobre a *DHT* foi uma tarefa simples, assim, o mecanismo de balanceamento funciona através de mensagens de envio e recebimento de chaves por parte do *crawlers* colaborativos, implementados a partir de interfaces bem definidas pelo mecanismo de distribuição e balanceamento de carga e sendo apoiados pelo uso do COPPEER. Todo conjunto de URL's extraído de uma página são enviados a *DHT* como um conjunto de chaves e a partir disso são distribuídos para os nós responsáveis, sendo recebidos pelos seus clientes *crawlers*.

Os *crawlers* estão organizados de forma a manter um balanceamento de carga entre eles, particionando a Web através do uso da *DHT* descrita anteriormente.

Os *crawlers* iniciam o processo de captura de páginas com base em um conjunto de URL's sementes previamente cadastrados, onde cada novo conjunto de URL's obtido através do processo de extração de *hyperlinks* é enviado ao agente *Bulk* do *crawler* colaborativo, que notifica o agente *Distributor* da existência de um novo conjunto de URL's a ser distribuído entre os clientes *crawlers* dos nós. Toda nova URL antes de ser capturada pelo *crawler*, primeiro é enviada a *DHT*, onde através do mecanismo de pesquisa logarítmico, localiza o nó responsável por essas URL's,

enviando-as para este. As URL's serão enviadas ao agente *Take* do *crawler*, sendo persistidas em uma base de dados, para em um momento posterior ser acessada pelo agente *Frontier*, seguindo a estratégia *breadth-first*. A utilização do agente *Frontier* implica diretamente na utilização de uma estratégia de navegação do *crawler*. Nesse trabalho foi adotada a estratégia *breadth-first*, entretanto, poderia ter sido adotada qualquer outra estratégia, tais como: *depth-first* ou *crawlers* focados. É importante perceber que qualquer que seja a estratégia de navegação adotada, esta não interferirá ou orientará a arquitetura do *crawler* colaborativo, pois esta é uma tarefa atômica de um agente autônomo suportado pelo COPPEER.

O agente *Distributor* através de um componente de resolução de *DNS* obtém o endereço *IP* de cada URL recebida, gerando a partir desse endereço o valor de *hash* a ser utilizado para despachar a URL a um dos nós que constituem a *DHT*. A utilização da estratégia de geração de *hash* a partir do endereço IP de uma URL garante que não existirão diferentes *crawlers* cuidando de um mesmo servidor. Isso proporciona facilidade para aplicabilidade de diferentes mecanismos para aperfeiçoamento do esquema de balanceamento de carga.

5.4 – Desafios encontrados

O *crawler* desenvolvido tem como base uma arquitetura *peer-to-peer*, distribuída através de diversos agentes os quais atuam explorando a Web através de informações contidas em páginas HTML. Entre os desafios encontrados, estão: o desconhecimento das rotas entre os agentes e os servidores Web, de forma a dificultar a rápida identificação do *crawler* com a melhor conectividade a um determinado servidor Web; o gerenciamento de memória para a fila de URL's a serem capturadas; a visita aos servidores Web sem violar qualquer regra formal ou informal da conduta de um *crawler*; entre outros.

5.4.1 – Gerenciamento da fila de URL's

A política de agendamento de visitas as páginas a serem capturadas utilizada no desenvolvimento do *crawler* colaborativo foi *breadth-first* (Najork & Wiener, 2001). Este produz uma coleção com alto grau de qualidade, através da localização de "boas" páginas em um menor tempo. Para o desenvolvimento do *crawler* foi implementada uma estratégia para gerenciamento de fila de URL's pendentes para

visita, de forma a reduzir seu tamanho máximo em até 50%, enquanto preservando a abrangência e qualidade das páginas visitadas.

O objetivo de aplicar esta política para um *crawler* distribuído em uma rede *peer-to-peer*, é exatamente por este operar em um ambiente restrito e não ter a ciência sobre em quais máquinas os agentes estarão funcionando, sendo assim, tentar economizar recursos de memória especialmente onde a fila é armazenada, sem afetar a qualidade das páginas capturadas.

A questão de economizar recursos de memória para a fila que armazena URL's a serem capturadas é extremamente relevante para *crawlers* que não operam sobre todo o grafo da Web. Um exemplo disto, são informações que devem ser capturadas por um grupo de agentes, de forma a agregar pesquisas onde cada grupo de agentes é responsável por um conjunto de páginas da Web (Castillo, 2006).

A estratégia utilizada foi apresentada por Castillo (2006), a qual combina as boas propriedades da *breadth-first* em termos de qualidade e política de visita ao servidores Web, enquanto usando o tamanho de fila compatível ao da *depth-first.* Essa estratégia é chamada de estratégia *Sydney*, a qual pode reduzir o tamanho da fila enquanto no mesmo tempo preserva a cobertura, qualidade de páginas capturadas e a política de visita em relação ao servidores Web. O esquema básico é formado por duas filas, referenciadas como fila primária denotada por P e fila secundária denotada por S. Também é mantida uma fila de páginas visitadas, denotada por V e inicialmente vazia.

O algoritmo é iniciado através da inserção de um conjunto de sementes na fila primaria P e com a fila secundária S vazia. Enquanto P não é esvaziada, URL's são extraídas e capturadas uma por vez. v será uma página extraída de P e N(v) será o conjunto de novas páginas apontadas por v, significando os *hyperlinks*, não são considerados os *hyperlinks* que apontam para páginas já existentes em P, S ou V. É selecionado randomicamente pelo algoritmo um subconjunto de t URL's a partir de $N(v)$, onde t é um parâmetro do algoritmo. Esse subconjunto é denotado R. Agora se $N(v) - R = 0$, significa que todos os *hyperlinks* de saída de v foram visitados, então a URL v é descartada. Por outro lado, se ainda há *hyperlinks* de saída que não foram visitados, v será inserido na fila secundária, para ter seus "vizinhos" explorados mais adiante.

Quando P torna-se vazia, o conteúdo de S é esvaziado em P. A visita ás páginas termina quando ambas as filas tornam-se vazias. O pseudocódigo do algoritmo é mostrado a seguir.

$P \leftarrow$ URLs iniciais fila primária

$S \leftarrow \varnothing$ fila secundária

$V \leftarrow$ páginas visitadas

Enquanto $P = \varnothing$ faça:

Obter uma página v de P e a capture

$V \leftarrow V \cup \{v\}$ marque como visitada

$N+v \leftarrow$ *v's hyperlinks* de saída que apontam para novas páginas "novas" significa que não esta em P, S ou V

Se $|N+v| \leq t$ então

$R \leftarrow N+v$

Senão

$R \leftarrow$ uma amostragem randômica de t nós de $N+v$

$S \leftarrow S \cup \{v\}$

Final se

$P \leftarrow P \cup R$

Se $P = \varnothing$ então

$P \leftarrow S$

$S \leftarrow \varnothing$

Final se

Final enquanto

Figura 25 – Pseudocódigo do algoritmo de redução de fila apresentado por Castillo (2006)

Para Castillo (2006) existe uma explicação satisfatória do motivo pelo qual a estratégia *Sidney* é eficiente em termos de memória. Tendo como preocupação o melhor valor para o parâmetro t, naturalmente atribuindo $t=\infty$ implica diretamente em uma estratégia puramente *breadth-first* com a fila primária extremamente grande. Por outro lado, atribuindo valores pequenos demais para t, implica na fila secundária extremamente grande. O problema com a fila secundária é a necessidade de visitar as páginas novamente para extrair os *hyperlinks* de saída. Essa revisita implica

diretamente em custos de conectividade de rede, mas felizmente, somente poucas páginas precisam ser revisitadas mais de uma vez, como por exemplo páginas que são muito visitadas. Existe um ponto interessante entre o número máximo de *hyperlinks* de saída por página e o tamanho máximo da fila, onde por exemplo, o tamanho de *t* varia de 8 a 16, o tamanho da fila varia de 50% a 64% da fila da *breadth-first*, mas somente 1% das páginas são revisitada.

Segundo Castillo (2006), em seus estudos foram aplicadas métricas de qualidade que comprovaram que a estratégia de gerenciamento de fila de URL's Sidney é superior se comparada a estratégia *breadth-first* pura. A conclusão para se usar a estratégia *Sidney* é robustez, reduzindo o tamanho da fila sem reduzir a qualidade das páginas obtidas e proporcionando melhorias no processo de captura.

5.4.2 – Controle da política de visita entre os *crawlers* colaborativos

Cada um dos *crawlers* colaborativos distribuídos sobre a arquitetura *peer-to-peer* utilizando o *framework* COPPEER, é responsável pela sua própria política de visita junto aos servidores hospedeiros das páginas Web, sendo papel do agente *Frontier* realizar essa tarefa.

A utilização do mecanismo de balanceamento através da distribuição de URL's, tornou os *crawlers* colaborativos desenvolvidos sobre o COPPEER livres da questão de interseção no tempo de visita entre servidores, pois nenhum dos *crawlers* colaborativos distribuídos age sobre um mesmo servidor. Isso é possível pois a distribuição de URL's acontece com base na geração do valor de *hash* do endereço *IP* do servidor. Não houve a necessidade de implementar tratamentos específicos para garantir que nenhum dos *crawlers* colaborativos quebrasse o princípio de "cortesia", definido segundo Heydon & Najork (1999) em políticas de visita, onde é necessário considerar uma pausa entre os acessos que ocorram a um mesmo servidor.

5.4.3 – Proteger os *crawler* colaborativos de armadilhas

É fato que os *crawlers* colaborativos encontrarão diversas armadilhas construídas intencionalmente por administradores de sites ou servidores Web. Essas armadilhas buscam impossibilitar o *crawler* de exercer o processo de captura, desviando-os para *hyperlinks* dinâmicos que não contenham informações ou simplesmente forçando o *crawler* a baixar *hyperlinks* que contenham grandes

arquivos, podendo "paralisar" a atuação do *crawler*. Para lidar com algumas dessas armadilhas o *crawler* colaborativo fornece mecanismos que podem ser configurados através de intervenção humana, como o mecanismo de filtragem e controle de tempo limite para baixar páginas da Web.

O mecanismo de filtragem de URL's suportado pelo agente *LinkExtractor* provê uma forma configurável para controlar o conjunto de URL's que serão capturadas. Antes de adicionar uma URL a fila com as URL's a serem capturadas, o agente *LinkExtractor* consulta o filtro de URL's. O componente de filtro tem um método que indica se o agente deve ou não considerar a URL. O componente de filtro pode ser estendido para uma coleção de diferentes tipos de filtro, podendo diversificar e facilitar o processo de filtragem com o uso de expressões regulares, como por exemplo através de domínio, prefixo, protocolo, conjunção, disjunção ou negação de outros filtros. Componentes de terceiros também podem fornecer seus próprios filtros, os quais são iniciados dinamicamente no *crawler*. Uma ação humana poderá manualmente excluir o site da lista do *crawler*, através da customização do filtro de URL's.

O controle de tempo limite utilizado para baixar páginas da Web é de 60 segundos, conforme apresentado por Heydon & Najork (1999). Esse mecanismo é gerenciado pelo agente *Downloader*, evitando por exemplo, que o *crawler* fique "preso" em uma página com tamanho de *Gigabytes*. O agente *Downloader* controla o tempo de baixar páginas da Web a partir do momento em que a conexão com o servidor Web é estabelecida, e se após isso o tempo limite for ultrapassado, um processo paralelo encerra a conexão com o servidor e uma nova URL é enviada ao agente *Downloader* através do agente *Frontier*.

5.5 – Pacotes e classes do *crawler* colaborativo

O *crawler* colaborativo está organizado e implementado através de diversos pacotes e classes, os quais serão apresentados a seguir.

5.5.1 – Diagrama de pacotes

O diagrama a seguir mostra as dependências existentes entre os pacotes que contem as classes desenvolvidas para o *crawler*, os componentes e projetos utilizados para suportar as atividades e estratégias mencionadas anteriormente.

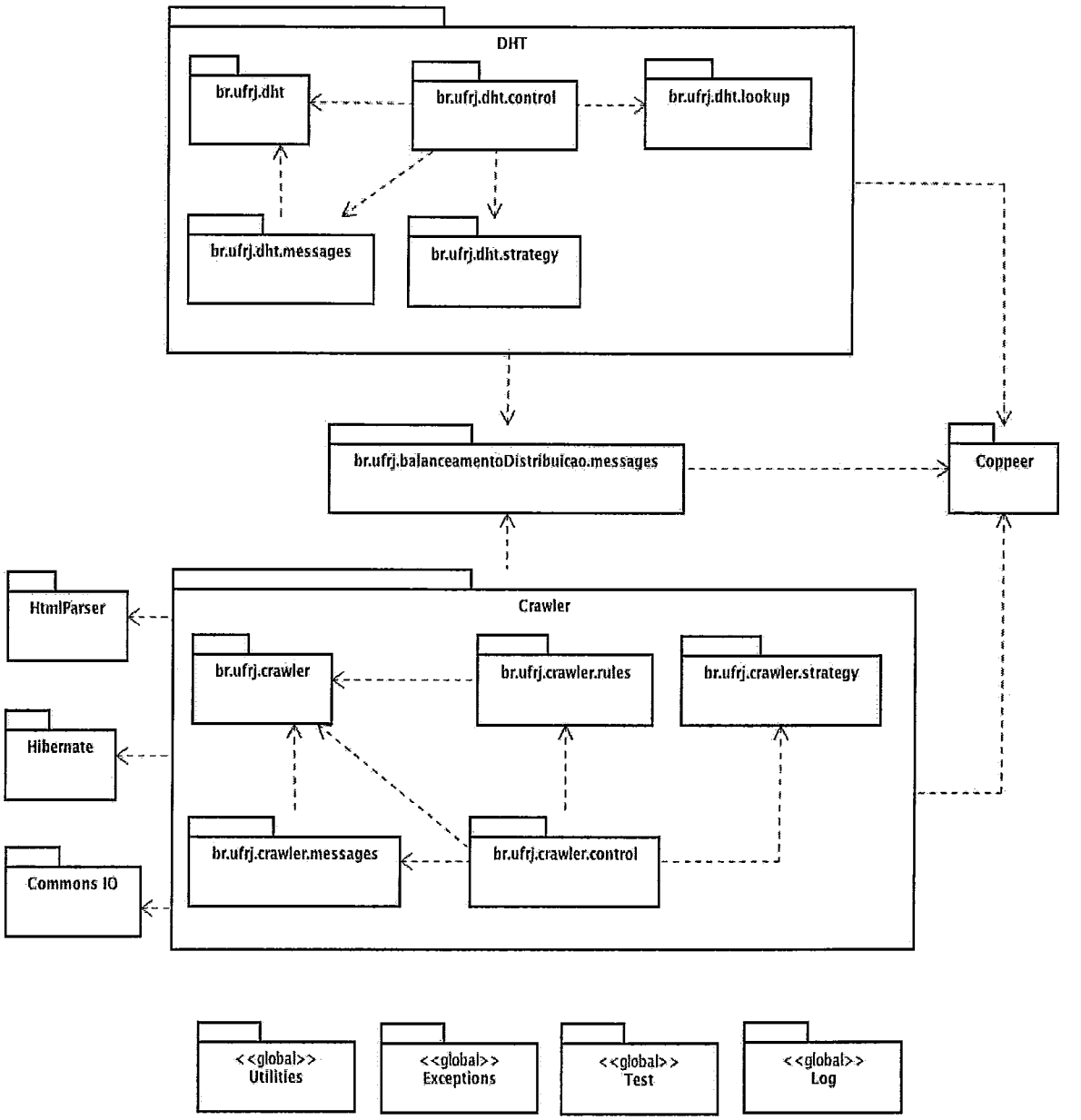


Figura 26 - Diagrama de pacotes do *crawler* colaborativo

A Figura 26 apresenta um diagrama simplificado de pacotes, destacando os pacotes e componentes abordados nesta dissertação.

O diagrama de pacotes implementados para o *crawler* colaborativo, apresentado anteriormente, faz uso de alguns pacotes essenciais para seu funcionamento. Para o *crawler* colaborativo alcançar seus objetivos, as classes desenvolvidas nesse trabalho fazem uso dos pacotes:

- *Hibernate* da JBoss Group, que permite a manipulação de esquemas de SGBD como objetos Java (JBoss Group, 2007);

- *Commons IO* da Apache Foundation, auxiliando a manipulação de arquivos (APACHE Group, 2007);
- *HtmlParser* hospedado no SourceForge, para manipulação dos arquivos HTML (HtmlParser, 2007).

5.5.2 – Diagrama de classes

Conforme apresentado no diagrama da Figura 26 e no diagrama na Figura 27, o *crawler* colaborativo faz uso de alguns pacotes essenciais para seu funcionamento.

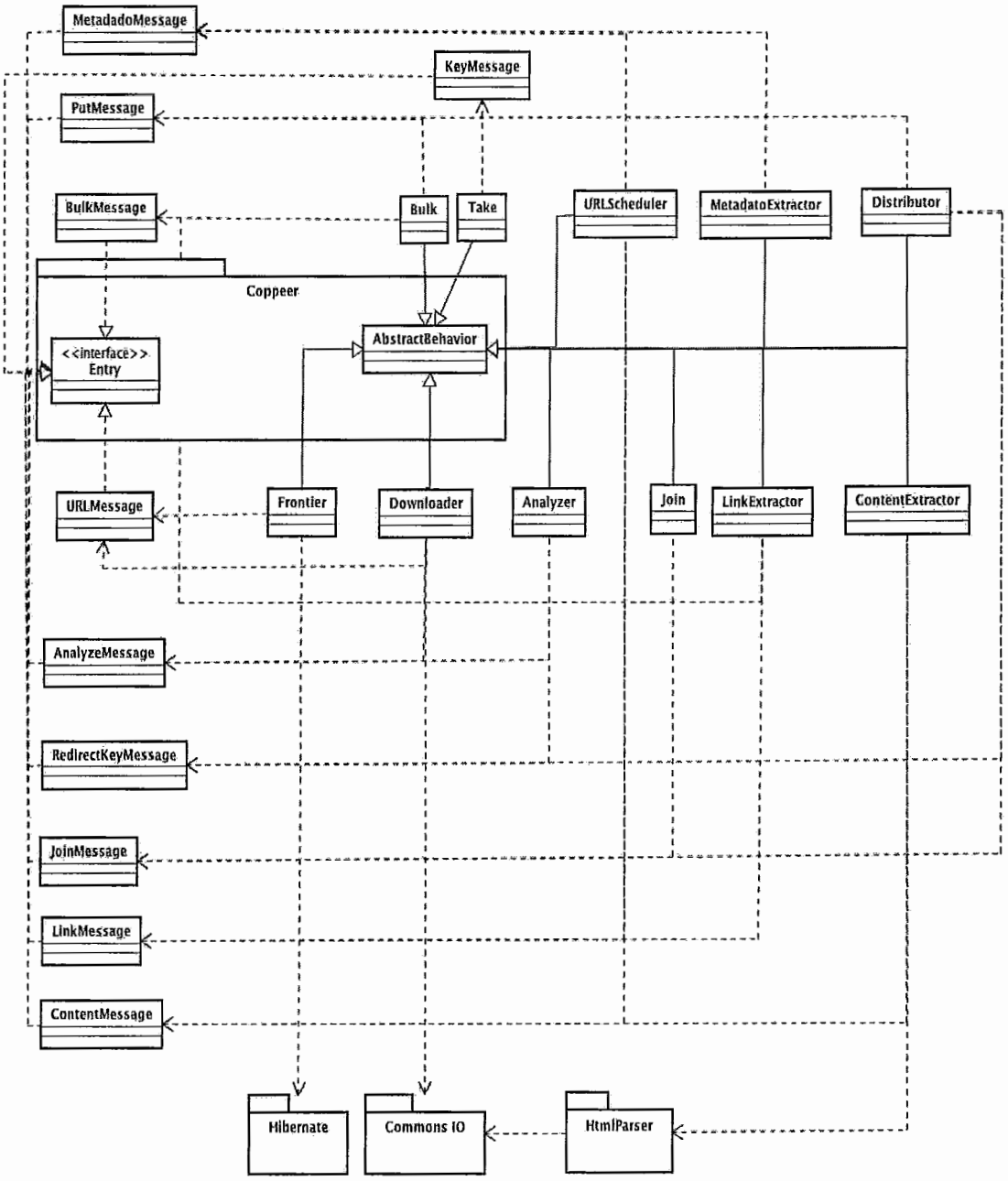


Figura 27 - Diagrama de classes dos agentes do *crawler* colaborativo (em UML 1.5)

A Figura 27 apresenta um diagrama de classes simplificado dos agentes que compõe o *crawler* colaborativo, apresentando as classes implementadas nesta dissertação e os pacotes externos utilizados.

5.6 – Ferramenta complementar

Para apoiar o desenvolvimento dos *crawlers* colaborativos sob a *DHT*, foi implementada uma ferramenta que auxilia a visualizar a formação da *DHT* em tempo real. Essa formação possibilita identificar a posição de cada cliente (*crawler*) da *DHT* no anel, mostrando seus sucessores e predecessores, conforme mostrado na Figura 28.

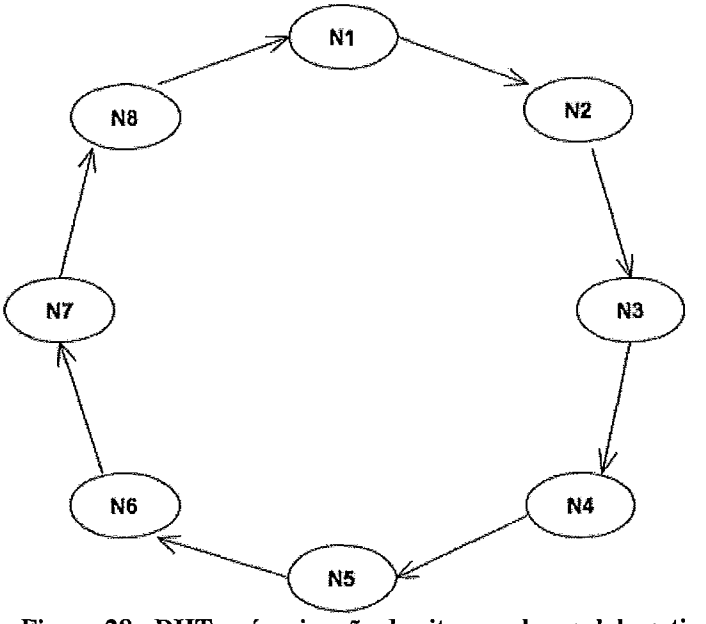


Figura 28 - DHT após a junção de oito crawlers colaborativos

A identificação dos clientes da *DHT* e da posição que estão ocupando é feita pelo agente *DHTViewer* implementado para essa ferramenta. Esse agente envia mensagens para o nó de *bootstrap* da *DHT* perguntando pelo seu sucessor, e assim consecutivamente até chegar ao último nó da *DHT*. Após percorrer toda a *DHT* e identificar todos os nós que estão na sua formação, o agente *DHTViewer* gera um gráfico da formação da *DHT*, permitindo visualizar sua estrutura e nós participantes.

5.7 – Características importantes dos *crawlers* colaborativos

A seguir serão apresentadas as características mais importantes dos *crawlers* colaborativos:

- *Crawler* colaborativo baseado em uma arquitetura multiagente utilizando como base o *framework* COPPEER;

- O *crawler* foi totalmente implementado em Java, sendo multiplataforma;
- Utiliza a estratégia *breadth-first* para navegabilidade do *crawler*;
- Detecta sites lentos e reagenda a visita;
- Totalmente configurável por meio de arquivos XML;
- Segue o protocolo de exclusão de robôs (Koster, 1996), utilizando tempo extra para requisição nos pequenos sites;
- Faz uso de um mecanismo de balanceamento de carga entre os *crawlers* e os servidores Web, através de identificação de rotas e a distribuição dos *crawlers* em uma *DHT*;
- Faz uso de um mecanismo de balanceamento de carga para o redirecionamento de chaves entre os *crawlers*, de forma que a redistribuição ocorra entre os *crawlers* conhecidos na *DHT* e que tenham posições próximas na rede.

5.8 – Considerações sobre a implementação

O desenvolvimento dos *crawlers* colaborativos está organizado em uma arquitetura multiagente baseada em um outro trabalho acadêmico, o *framework* COPPEER. A implementação apresentada neste capítulo pode ser utilizada como referência por outros trabalhos que venham a suplantar as funcionalidades originais apresentadas na arquitetura. Diversos agentes, mecanismos e estratégias implementadas nesse projeto contribuem diretamente para o projeto COPPEER (Miranda, Xexéo et al., 2006), como a *DHT* implementada e utilizada.

O uso de software livre no desenvolvimento deste trabalho encoraja a livre distribuição do protótipo implementado, de forma que a comunidade acadêmica possa colaborar com o seu desenvolvimento, bem como voluntários que desejem aprimorar os recursos e funcionalidades aqui expostas.

Capítulo 6 – Estudo observatório

A partir do momento em que foram estabelecidas as estratégias utilizadas na construção dos *crawlers* colaborativos, foi decidido realizar um estudo para observar a utilização das estratégias propostas. Este estudo teve o foco na aplicação das estratégias sob o mecanismo de distribuição e balanceamento de carga utilizado pelos *crawler* colaborativos em uma arquitetura *peer-to-peer*. Tais estratégias foram aplicadas através dos agentes dos *crawlers* colaborativos para apoiar o mecanismo de distribuição e balanceamento de carga: *DHT, DHT* apoiada em tempo de resposta para baixar as páginas da Web e *DHT* apoiada em particionamento geográfico.

A realização desse estudo foi dividida em quatro fases: a definição, o planejamento, a execução e a análise do estudo.

Segundo Barros *et al.* (2002) a definição do estudo consiste em resumir seus objetivos, seu foco de qualidade e os objetos que serão analisados. O planejamento envolve a descrição do perfil dos participantes, dos instrumentos, do processo de execução e uma avaliação crítica dos problemas que podem ser encontrados ao longo desta execução. A execução consiste na realização do estudo pelos participantes, utilizando os instrumentos e o processo definidos no planejamento. A análise consiste na organização dos resultados gerados pelos participantes durante a execução e a realização de inferências sobre estes resultados.

6.1 – O Estudo observatório

O estudo realizado teve como base um modelo proposto por Barbara *et al.* (2006).

É sabido sobre a questão de fluxo de dados na Internet, e da existência de gargalos no tráfego de dados que variam de acordo com o horário e região, conforme pode ser verificado em (ITR, 2007). Na tentativa de evitar desvantagens nas amostragens geradas durante o estudo observatório, foi adotado um mecanismo de execução aleatório.

6.2 – Definição do estudo observatório

Objeto de Estudo: a utilização de estratégias no mecanismo de distribuição e balanceamento de carga utilizado pelos *crawlers* colaborativos para melhora no processo de captura de páginas da Web.

Objetivo: identificar a viabilidade de utilização da estratégia de tempo de resposta associada à *DHT* no mecanismo de distribuição e balanceamento de carga utilizado pelos *crawlers* colaborativos.

Foco de Qualidade: os ganhos obtidos pela utilização da estratégia de tempo de resposta, medidos através da abrangência da área de cobertura atingida pelos *crawlers* colaborativos e tempo de realização dessa abrangência, e as dificuldades encontradas em relação a utilização de recursos necessários.

Perspectiva: o estudo foi desenvolvido sob a ótica de um pesquisador, avaliando a viabilidade de utilização da estratégia mencionada anteriormente.

Contexto: o processo de captura de páginas na Web a partir de *crawlers* distribuídos e colaborativos, com comportamento determinado pelas estratégias implementadas e descritas nesse trabalho. O estudo foi conduzido no formato de múltiplos testes sobre os *crawlers*.

6.3 – Planejamento do estudo observatório

Contexto Local: este estudo teve como intenção avaliar a viabilidade da utilização das estratégias de tempo de resposta e particionamento geográfico. Os *crawlers* colaborativos foram executados a partir dessas estratégias, cujo comportamento foi determinado por uma escolha aleatória a partir de intervalos de tempo pré-definidos.

Participantes: Os participantes do estudo foram os *crawlers* colaborativos. O estudo foi executado em um ambiente formado por três servidores, em diferentes localidades, como Estados Unidos, Alemanha e Brasil. Este estudo não pôde ser mais extenso devido as dificuldades de recursos necessários para sua execução. Entretanto os três servidores utilizados geraram resultados significativos ao estudo.

Critérios: o foco de qualidade do estudo exige critérios que avaliem os ganhos proporcionados pela utilização das estratégias de tempo de resposta e particionamento geográfico e as dificuldades encontradas na utilização de recursos finitos como processamento de CPU e rede. Os ganhos obtidos pela utilização das

estratégias foram avaliados quantitativamente, através da abrangência da área de cobertura atingida pelos *crawlers* colaborativos e tempo de realização dessa abrangência. Esses critérios foram selecionados a partir da necessidade de critérios quantitativos para comparação do desempenho dos *crawler* colaborativos atuando nas diferentes estratégias.

Hipótese Nula: no estudo atual a hipótese nula determina que a utilização de um mecanismo de distribuição e balanceamento de carga (exemplo: *DHT*) com a aplicabilidade da estratégia de tempo de resposta, não produz benefícios suficiente para *crawler* distribuídos e colaborativos.

H_0: μ execução sem estratégia de tempo de resposta $\approx$ μ execução com estratégia de tempo de resposta

Hipótese Alternativa: O estudo observatório tem como objetivo provar a hipótese alternativa, contestando a hipótese nula. Nesse estudo observatório a hipótese alternativa determina que os *crawlers* do estudo utilizando o mecanismo de distribuição e balanceamento de carga (*DHT*), com a estratégia de tempo de resposta, têm resultados superiores aos *crawlers* que utilizaram apenas mecanismo de distribuição e balanceamento de carga (*DHT*), mecanismo de distribuição e balanceamento de carga (*DHT*) com particionamento geográfico ou *crawlers* que atuam sem algum tipo de mecanismo de distribuição de carga e isolados. De acordo com os critérios selecionados, esta hipótese se traduz em maior abrangência de cobertura e menor tempo para atuação no processo de captura das páginas Web.

H_1: μ execução sem estratégia de tempo de resposta $<<$ μ execução com estratégia de tempo de resposta

Análise Quantitativa: tem o objetivo de avaliar a área de abrangência de cobertura por parte dos *crawlers* colaborativos. A análise quantitativa será realizada através das estratégias apresentadas anteriormente. Esta avaliação teve a intenção de verificar se as estratégias influenciaram os resultados do estudo.

Capacidade Aleatória: foi exercida na escolha das estratégias e momento de execução. Idealmente os *crawlers* que participaram do estudo tiveram as estratégias selecionadas aleatoriamente.

Balanceamento: durante a realização do estudo foi limitado a distribuir um número similar de *crawler* na utilização das estratégias propostas.

Mecanismo de Análise: esse estudo compara os resultados obtidos pelos *crawlers* nas diferentes estratégias analisadas.

Validade Interna do Estudo: a validade interna de um estudo é definida como a capacidade de um novo estudo repetir o comportamento do estudo atual com os mesmo participantes e objetos com que foi realizado. A validade interna do estudo é dependente do número de participantes executando o estudo (Barros *et al.*, 2002). Esse estudo foi realizado com três *crawlers*. Certamente um número maior de *crawlers* melhoraria a validade interna do estudo. Outro ponto que pode influenciar o resultado do estudo são os recursos computacionais utilizados.

Validade de Conclusão do Estudo: a validade de conclusão do estudo mede a relação entre os tratamentos e os resultados, determinando a capacidade do estudo em gerar alguma conclusão (Barros *et al.*, 2002).

6.4 – Execução do Estudo observatório

Os participantes deste estudo observatório foram os *crawlers* colaborativos conforme apresentado na seção "6.3 – Planejamento do Estudo Observatório", parágrafo *Participantes*.

Até o momento em que esta dissertação esteve no ponto de realização do estudo, o simulador para execução do *framework* COPPEER ainda encontrava-se em desenvolvimento, não permitindo utiliza-lo como instrumento para emular a execução dos *crawlers* colaborativos. Dessa forma, para a execução do estudo observatório foram selecionados três servidores distintos localizados em países que pertencem a diferentes continentes, permitindo que fossem realizados os testes para os *crawlers* colaborativos geograficamente distribuídos.

O procedimento de participação dos *crawler* colaborativos no estudo foi dividido em quatro diferentes grupos: "Isolado", "DHT", "GEO", "RTT". O primeiro atuou independente de qualquer mecanismo de distribuição e balanceamento de carga, o segundo utilizou o mecanismo de distribuição e balanceamento de carga sem a adição de qualquer estratégia, o terceiro foi baseado no mecanismo de distribuição e balanceamento de carga com particionamento geográfico (vide seção "4.3.2 – Estratégia de Particionamento Geográfico"), e o quarto, baseado no mecanismo de distribuição e balanceamento de carga com tempo de resposta (vide seção "4.3.3 – Estratégia de Tempo de Resposta"). Ambos foram iniciados com o mesmo conjunto de "sementes" para os *crawlers*. Este conjunto foi formado pelas seguintes URL's: dir.yahoo.com, yahoo.com, dmoz.org, cnn.com e globo.com.

Foram executadas trinta rodadas com cada grupo, onde para cada grupo foram selecionados aleatoriamente 4 estratégias. Embora as rodadas tenham sido relativamente curtas, este tempo ocupou cerca de quarenta minutos de cada um dos grupos por rodada.

A Tabela 1 apresenta informações simplificadas sobre os servidores que hospedam os *crawlers* do estudo observatório. Para cada uma das estratégias analisadas foi utilizado um determinado repositório.

Tabela 1 – Informações sobre os ambientes que hospedam os *crawlers* participantes no estudo observatório

ID	País	Largura de Banda	Capacidade de Armazenamento	*JVM Heap Size*
1	Estados Unidos	1 MBit/s	30 GB	512 MB
2	Alemanha	1 MBit/s	30 GB	512 MB
3	Brasil	1 MBit/s	30 GB	512 MB

6.5 – Análise dos resultados do estudo observatório

A análise do estudo observatório foi separada em análise de domínios identificados, análise de páginas capturadas, média de *Kilobytes*/páginas e tempo médio de captura por página, apresentando assim a análise quantitativa do estudo. Os dados que fundamentam o trabalho realizado resultam das rotas obtidas pelos *crawlers* durante o processo de captura, a partir de um conjunto de "sementes" previamente selecionadas.

A Tabela 2 apresenta as estatísticas descritivas sobre os resultados do estudo.

Tabela 2 – Estatísticas descritivas sobre o resultado

Comparação	Estratégia	Média	Mediana	Desvio padrão	*Shapiro-Wilk*	
					w	p
Total páginas	Isolado	81,33	73,50	36,09	0,94	0,11
	DHT	91,13	87,00	42,88	0,928	0,0507
	GEO	71,20	54,50	39,55	0,930	0,057
	RTT	121,83	112,50	43,96	0,96	0,40
Total	Isolado	8,33	2	13,08	0,67	0,00

domínios	DHT	10,36	4	15,97	0,68	0,00
	GEO	2,9	2	4,92	0,55	0,00
	RTT	15,53	10	13,84	0,88	0,003
Total *Kilobytes*	Isolado	2319,32	1920,13	225,32	0,91	0,01
	DHT	2769,68	2185,29	318,67	0,86	0,001
	GEO	1201,24	1221,65	100,10	0,97	0,64
	RTT	3187,38	2817,93	238,56	0,89	0,006
Média tamanho páginas	Isolado	27,96	26,87	5,19	0,97	0,73
	DHT	29,83	28,81	9,43	0,95	0,27
	GEO	17,72	16,58	6,18	0,90	0,01
	RTT	26,35	26,38	5,19	0,94	0,13
Tempo médio captura	Isolado	1,95	1,41	1,27	0,790	0,000
	DHT	2,50	1,72	2,02	0,715	0,000
	GEO	1,90	1,83	1,07	0,799	0,000
	RTT	1,49	1,20	1,03	0,814	0,000

Nas análises a seguir, o teste T afirmou que a estratégia de tempo de resposta foi superior as outras estratégias comparadas. Para o teste T foi usado o nível de significância de 95% e o número de rodadas igual a 30, dessa forma, todos os valores apresentados (P-valor) nas tabelas abaixo que estiverem em negrito afirmam a negação da hipótese nula (H_0).

Os valores em negrito apresentados na Tabela 3 mostram que a estratégia de tempo de resposta com relação ao total de páginas capturadas, foi superior a todas as estratégias comparadas.

Tabela 3 – Comparação por total de páginas (Nível de significância de 95%)

Total Páginas	DHT	GEO	RTT
Isolado	0,319	0,362	**0,00025**
DHT		0,073	**0,0085**
GEO			**0,00002**

Na Tabela 4 é possível observar a superioridade da estratégia de tempo de resposta em relação à estratégia sem mecanismo de distribuição e balanceamento de carga.

Tabela 4 – Comparação por total de domínios (Nível de significância de 95%)

Total Domínios	DHT	GEO	RTT
Isolado	0,591	0,037	**0,042**
DHT		0,017	0,185
GEO			**0,00002**

Conforme a Tabela 5, a hipótese nula pode ser negada para o total de *Kilobytes* capturados.

Tabela 5 – Comparação por total de *Kilobytes* (Nível de significância de 95%)

Total *Kilobytes*	DHT	GEO	RTT
Isolado	0.255	0.000	**0.01**
DHT		0.000	0.304
GEO			**0.000**

Para a comparação por média do tamanho de páginas em *Kilobytes*, o objetivo da estratégia por tempo de resposta é afirmar a hipótese nula (vide Tabela 6), isto significa que a estratégia de tempo de resposta foi superior a estratégia sem mecanismo de distribuição e balanceamento de carga, em relação à descoberta de novas páginas.

Tabela 6 – Comparação por média do tamanho de páginas em *Kilobytes* (Nível de significância de 95%)

Média Tamanho Páginas	DHT	GEO	RTT
Isolado	0,346	0,000	0,234
DHT		0,000	0,082
GEO			0,000

A Tabela 7 apresenta o tempo médio de captura por página sendo analisado ao nível de significância de 90%.

Tabela 7 – Comparação por tempo médio de captura por página (Nível de significância de 90%)

Tempo Médio	DHT	GEO	RTT
Isolado	0,896	0,440	**0,067**
DHT		0,077	**0,008**
GEO			**0,068**

Com base nas estatísticas descritivas do estudo (vide Tabela 2), é possível observar que as análises não seguiram uma distribuição normal, com exceção da análise de total de páginas. O teste de normalidade *Shapiro-Wilk* comprova a não distribuição normal a um nível de significância de 95%. Com o objetivo de aferir as conclusões do teste T, os dados previamente analisados para análise de domínios identificados, total de *Kilobytes* capturadas, média de *Kilobytes*/páginas e tempo médio de captura por página foram submetidos a um teste estatístico não-paramétrico: o teste de Mann-Whitney (Wonnacott, 1990). Este teste, que se baseia na ordenação e no ranqueamento dos dados analisados, chegou às mesmas conclusões do teste T (apresentando P-valor) conforme as tabelas abaixo.

Tabela 8 – Comparação por total de domínios utilizando Mann-Whitney (Nível de significância de 95%)

Total Domínios	DHT	GEO	RTT
Isolado	0,742	0,129	**0,007**
DHT		0,240	**0,025**
GEO			**0,00002**

Tabela 9 – Comparação por total de *Kilobytes* capturado utilizando Mann-Whitney (Nível de significância de 95%)

Total *Kilobytes*	DHT	GEO	RTT
Isolado	0,442	0,000	**0,003**
DHT		0,000	**0,032**
GEO			**0,000**

Tabela 10 – Comparação por média do tamanho de páginas em *Kilobytes* utilizando Mann-Whitney (Nível de significância de 95%)

Média Tamanho Páginas	DHT	GEO	RTT
Isolado	0,442	0,000	0,220
DHT		0,000	0,124
GEO			0,000

Tabela 11 – Comparação por tempo médio de captura por página utilizando Mann-Whitney (Nível de significância de 90%)

Tempo Médio	DHT	GEO	RTT
Isolado	0,132	0,745	**0,088**
DHT		0,240	**0,003**
GEO			**0,033**

O sucesso dos testes estatísticos realizados leva a concluir que a estratégia de tempo de resposta é viável e auxilia os *crawlers* durante o processo de captura de páginas na Web.

Abaixo são apresentados gráficos com o intuito de mostrar a eficiência da estratégia de tempo de resposta com a evolução do tempo, para total de páginas, domínios, *Kilobytes* capturados e tempo médio de captura.

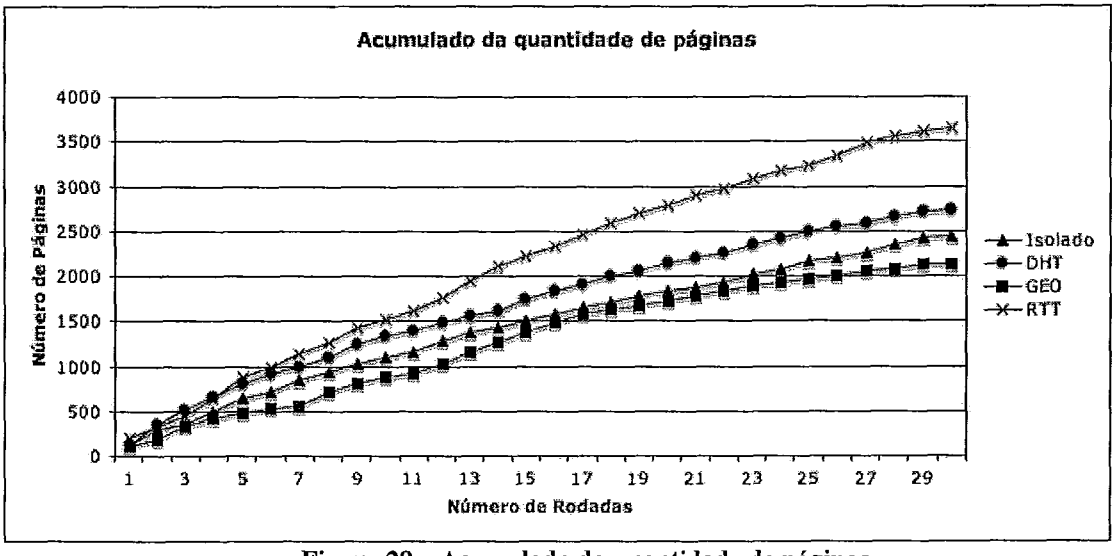


Figura 29 – Acumulado da quantidade de páginas

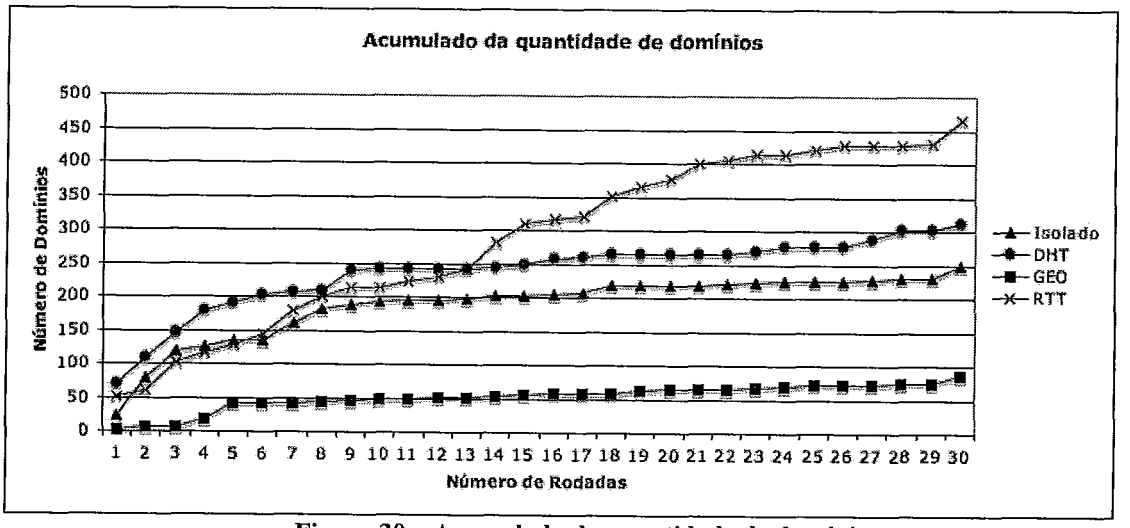


Figura 30 – Acumulado da quantidade de domínios

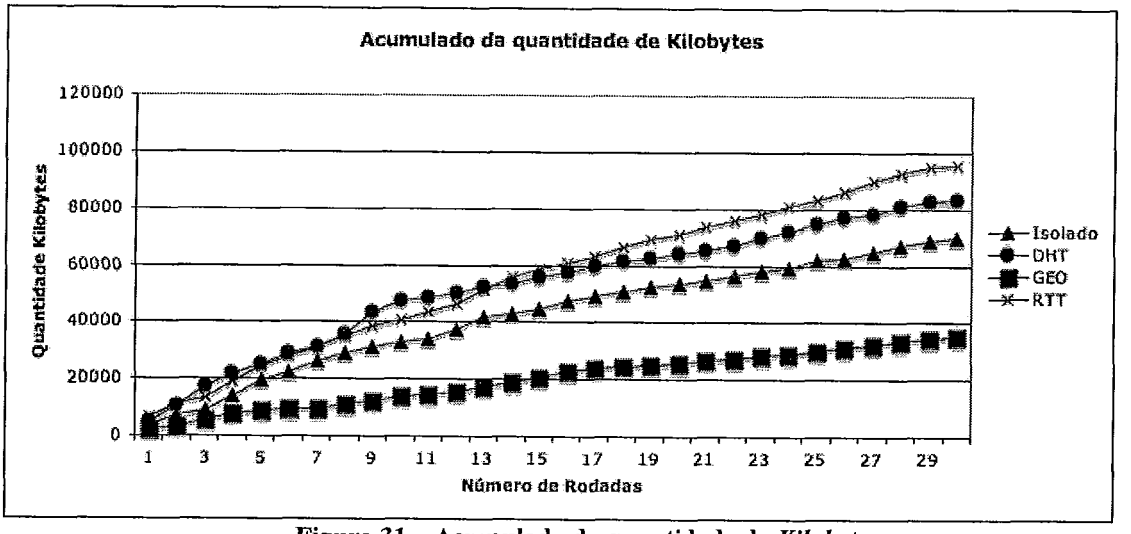


Figura 31 – Acumulado da quantidade de *Kilobytes*

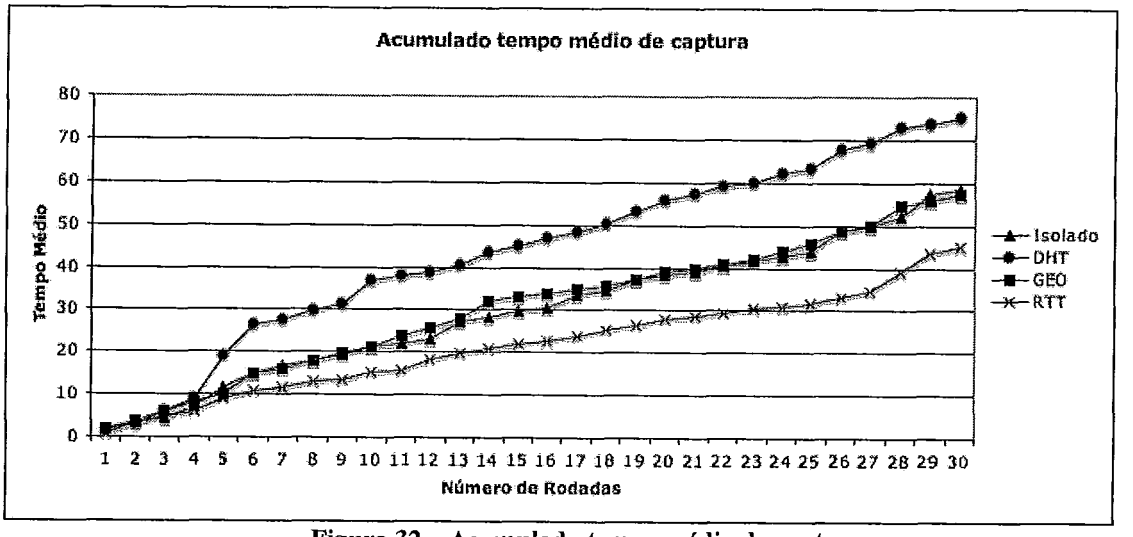


Figura 32 – Acumulado tempo médio de captura

6.6 – Análises finais

Embora esse estudo observatório tenha utilizado um número pequeno de servidores hospedeiros, ele permitiu a análise do comportamento dos *crawlers* distribuídos atuando em diferentes redes e hospedados em diferentes regiões geográficas.

Durante este estudo foi observado que a estratégia de tempo de resposta obteve os melhores rendimentos ao longo do processo de captura de páginas na Web.

Capítulo 7 – Conclusão

O objetivo do sistema multiagente aqui proposto, foi mostrar que *crawlers* distribuídos formados por agentes de software suportados pelo COPPEER, atuando colaborativamente e guiados pela estratégia RTT foram superiores aos mesmos *crawlers* seguindo as estratégias Isolado, DHT e Geográfico. Essa superioridade mostrou-se possível por meio da escolha mais adequada de servidores hospedeiros de páginas Web a serem seguidos pelos *crawlers* distribuídos, servidores estes, identificados através do uso da estratégia de tempo de resposta.

Cho & Garcia-Molina (2002) e Shkapenyuk & Suel (2002) apresentaram características importantes para o bom funcionamento de *crawlers* na Web. Essas características foram usadas como base para esta dissertação. Najork & Wiener (2001), Chakrabarti *et al.* (1999) e Baeza-Yates, Castillo *et al.* (2005) apresentaram diferentes estratégias de navegação para *crawlers*. A arquitetura desenvolvida e proposta nesta dissertação esta preparada para utilizar qualquer das estratégias apresentadas, porém a estratégia de navegação implementada e utilizada foi a estratégia *breadth-first* apresentada por Najork & Wiener (2001).

A estratégia de tempo de resposta implementada nesta dissertação teve como origem à aplicabilidade do conceito de *CDN*(*Content Delivery Network*) apresentado por Hofmann (2005), o qual visa fornecer conteúdo de modo transparente e mais rápido, identificando servidores que estejam próximos ou na mesma rede que seus clientes, adicionado ao conceito de *CPRT* (*Client Perceived Response Time*) apresentado por Mukhtar (2003). Com isso é possível setorizar a Web em particionamentos otimizados para cada um do *crawlers* em execução, auxiliando na tarefa de abranger uma maior gama de páginas e domínios existentes na Web.

Através da adição de "aprendizados", os *crawlers* puderam utilizar os recursos de rede de forma mais eficiente, buscando filiação com os servidores hospedeiros de páginas Web. A construção do *crawler* baseada em agentes permitiu que suas principais atividades fossem distribuídas e executadas paralelamente de maneira facilitada, tornando-se autônomas e propiciando uma arquitetura de baixo acoplamento, ficando a cargo do *framework* COPPEER gerenciar os agentes.

A utilização de regras formais e informais para políticas de visitas foram implementadas e utilizadas nos *crawlers* distribuídos, respeitando os servidores hospedeiros de páginas Web. Particionar a Web entre os *crawlers* permitiu uma gerência mais eficiente e transparente dessas políticas.

O mecanismo de distribuição e balanceamento de carga implementado e utilizado nesta dissertação teve como inspiração o trabalho proposto por Stoica, Morris *et al.* (2003), mostrando-se eficaz e simples de ser utilizado por outros sistemas multiagentes. Isto foi possível pela estrutura de mensagens providas pelo COPPEER.

O estudo observatório realizado neste trabalho levou à conclusão de que a aplicabilidade da estratégia de tempo de resposta adicionada ao mecanismo de distribuição e balanceamento de carga (vide seção "4.3.3 – Estratégia de Tempo de Resposta"), trará ganhos significativos ao processo de captura de páginas na Web. Os dados analisados mostraram que com o aumento do tempo de execução os *crawlers* tendem a "seguir" sempre os servidores mais próximos, onde o conceito de proximidade aqui é o tempo de resposta, e isto permitiu uma maior captura de páginas na Web em relação as outras estratégias analisadas.

Este trabalho contribuiu diretamente para o *framework* COPPEER com:

- A construção de um mecanismo de distribuição e balanceamento de carga (*DHT*);
- A validação de suas funcionalidade e serviços fornecidos;
- Além de se tornar a primeira aplicação a executar na rede mundial utilizando-o.

Ao longo do estudo observatório diversos dados como páginas, domínios e metadados oriundos do cabeçalho HTTP e HTML foram colhidos, os quais serviram como base para os trabalhos realizados por Barros *et al.* (2008), onde este pode analisar a qualidade de diversas páginas capturadas sobre um mesmo tópico. Isso foi possível através de um conjunto de sementes previamente selecionado e informado aos *crawlers* no inicio do processo de captura.

7.1 – Trabalhos futuros

A proposta desta dissertação é apenas uma instância do que pode ser realizado com o potencial das idéias propostas sobre estratégias para *crawlers* distribuídos. É

fato o crescimento constante da Web e a necessidade da captura de novas informações e a atualização das informações já capturadas. Trabalhos futuros podem ser realizados na direção de buscar desenvolver agentes de software que visem otimizar cada vez mais o uso dos recursos computacionais empregados no processo de captura de páginas da Web.

Referências Bibliográficas

Altigran, S. d. S., A. V. Eveline, et al. (1999). CoBWeb A Crawler for the Brazilian Web. Proceedings of the String Processing and Information Retrieval Symposium \& International Workshop on Groupware, IEEE Computer Society.

Altingovde, I. S. and O. Ulusoy (2004). "Exploiting Interclass Rules for Focused Crawling." IEEE Intelligent Systems **19**(6): 66 - 73.

APACHE Group. (2007). "Commons IO." from http://commons.apache.org/io/.

ARIN. from http://www.arin.net/whois/.

Baeza-Yates, R. and C. Castillo (2002). "WIRE, an open source Web information retrieval environment." Workshop on Open Source Web Information Retrieval (OSWIR). Compiegne, France: 27-30.

Baeza-Yates, R., C. Castillo, et al. (2005). "Crawling a country: better strategies than breadth-first for web page ordering." Special interest tracks and posters of the 14th international conference on World Wide Web. Chiba, Japan, ACM Press.

Baeza-Yates, R. and B. Ribeiro-Neto (1999). Modern Information Retrieval, ACM Press / Addison-Wesley.

Balakrishnan, H., M. F. Kaashoek, et al. (2003). "Looking Up Data in P2P Systems." Communications of the ACM **46**(2): 43-48.

Barbara, K., A.-K. Hiyam, et al. (2006). Evaluating guidelines for empirical software engineering studies. Proceedings of the 2006 ACM/IEEE international symposium on International symposium on empirical software engineering. Rio de Janeiro, Brazil, ACM Press.

Barros, M. d. O., C. M. L. Werner, et al. (2002). "Um Estudo Experimental sobre a Utilização de Modelagem e Simulação no Apoio à Gerência de Projetos de Software." XVI Simpósio Brasileiro de Engenharia de Software - Gramado, RS, Brasil: 191-206.

Barros, R., G. Xexéo, et al. (2008). A Web Metadata Based-Model for Information Quality Prediction; A Chapter of the Handbook of Research on Web Information Systems Quality;, University of Castilla-La Mancha - Spain.

Boldi, P., B. Codenotti, et al. (2002). "UbiCrawler: a scalable fully distributed web *crawler*." In Proc. AusWeb02. The Eighth Australian World Wide Web Conference.

Bra, P. D., G.-J. Houben, et al. (1994). "Information Retrieval in Distributed Hypertexts."

Brin, S. and L. Page (1998). "The Anatomy of a Large-Scale Hypertextual Web Search Engine."

Chakrabarti, S., M. v. d. Berg, et al. (1999). "Focused Crawling: A new approach to topic-specific web resource discovery." Computer Networks: The International Journal of Computer and Telecommunications Networking **31**(11 - 16): 1623 - 1640.

Chakrabarti, S., B. Dom, et al. (1998). "Enhanced hypertext categorization using *hyperlinks*." ACM SIGMOD Record , Proceedings of the 1998 ACM SIGMOD international conference on Management of data SIGMOD **27**(2): 307 - 318.

Cho, J. (2001). "Crawling the Web: Discovery and Maintenance of a Large-Scale Web Data."

Cho, J. and H. Garcia-Molina (2000). "The evolution of the web and implications for an incremental *crawler*." In Proceedings of 26th International Conference on Very Large Databases (VLDB).

Cho, J. and H. Garcia-Molina (2002). "Parallel Crawlers." In Proceedings of the 11th World Wide Web conference (WWW11).

Cho, J. and H. Garcia-Molina (2003). "Effective page refresh policies for web *crawlers*." ACM Transactions on Database Systems **28**(4).

Cho, J., H. Garcia-Molina, et al. (2004). "Stanford WebBase Components and Applications." ACM Trans. Inter. Tech. 6(2): 153-186.

Cho, J. and A. Ntoulas (2002). "Effective change detection using sampling." In Proceedings of 28th International Conference on Very Large Databases (VLDB).

Cho, J., N. Shivakumar, et al. (2000). "Finding replicated Web collections." Proceedings of the 2000 ACM SIGMOD international conference on Management of data: 355 - 366.

Chung, C. and C. Clarke (2002). "Topic-oriented collaborative *crawling*." CIKM: 34 - 42.

Deepika, C. and D. B. Albert (1998). JAFMAS: a multiagent application development system. Proceedings of the second international conference on Autonomous agents. Minneapolis, Minnesota, United States, ACM Press.

Dikaiakos, M. and D. Zeinalipour-Yazti (2001). "WebRACE: A Distributed WWW Retrieval, Annotation, and Caching Engine." In PADDA01: International Workshop on Performance-oriented Application Development for Distributed Architectures , Munich, Germany, April 2001.

DMOZ. (1998). "Open Directory Project." from http://www.dmoz.org/.

Edwards, J., K. McCurley, et al. (2001). "An Adaptive Model for Optimizing Performance of an Incremental Web Crawler." In Proceedings of the Tenth Conference on World Wide Web: 106 - 113.

Eichmann, D. (1994). "The RBSE spider: balancing effective search against Web load." In Proceedings of the First World Wide Web Conference, Geneva, Switzerland.

Eiron, N., K. S. McCurley, et al. (2004). "Ranking the Web Frontier." Proc. 13th Int'l Conf. World Wide Web (WWW 13).

Eugster, P. T., P. A. Felber, et al. (2003). "The many faces of publish/subscribe." ACM Comput. Surv. **35**(2): 114-131.

Exposto, J., J. Macedo, et al. (2005). Geographical partition for distributed web *crawling*. Proceedings of the 2005 workshop on Geographic information retrieval. Bremen, Germany, ACM Press.

FERREIRA, A., 1999, "Dicionário Aurélio eletrônico. V. 2.0". Rio de Janeiro: Nova Fronteira.

Gamma, E., R. Helm, et al. (1996). Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley.

Gao, W., H. C. Lee, et al. (2006). "Geographically focused collaborative *crawling*." Proceedings of the 15th international conference on World Wide Web: 287 - 296.

Harrenstien, K., M. K. Stahl, et al. (1985). NICNAME/WHOIS, RFC Editor.

Henzinger, M. R., A. Heydon, et al. (1999). "Measuring index quality using random walks on the Web." Proceeding of the eighth international conference on World Wide Web: 1291 - 1303.

Henzinger, M. R., A. Heydon, et al. (2000). "On near-uniform URL sampling." computer and telecommunications networking: 295 - 308

Heydon, A. and M. Najork (1999). "Mercator: A scalable, extensible web *crawler*." World Wide Web Conference **2**(4): 219 - 229.

Hofmann, M. and L. R. Beaumont (2005). Content Networking: Architecture, Protocols, and Practice., Morgan Kaufmann.

HtmlParser. (2007). "HtmlParser Project." from http://htmlparser.sourceforge.net/.

ISC.(2007)."Internet Systems Consortium." from http://www.isc.org/index.pl?/ops/ds

ITR. (2007). "Internet Traffic Report." from http://www.internettrafficreport.com.

JBoss Group. (2007). "Hibernate." from http:/www.hibernate.org/.

Karger, D., E. Lehman, et al. (1997). "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web." ACM Symposium on Theory of Computing: 654-663.

Koster, M. (1995). "Robots in the web: threat or treat?" ConneXions **9**(4).

Koster, M. (1996). "A standard for robot exclusion."

LACNIC. from http://www.lacnic.net/cgi-bin/lacnic/whois.

Lawrence, P., B. Sergey, et al. (1999). "The PageRank Citation Ranking: Bringing Order to the Web."

Lawrence, S. and C. L. Giles (1998). "Searching the World Wide Web." Science **280**(5360): 98.

Lawrence, S. and C. L. Giles (2000). "Accessibility of information on the web." intelligence **11**(1): 32 - 39.

Loo, B. T., S. Krishnamurthy, et al. (2004). "Distributed Web Crawling over DHTs." Technical Report No. UCB/CSD-04-1305.

Lyman, P. and H. R. Varian. (2003). "How Much Information." from http://www.sims.berkeley.edu/how-much-info-2003.

Maymounkov, P. and D. Mazieres (2002). "Kademlia: A peer-to-peer information system based on the XOR metric." Proceedings of IPTPS02, Cambridge, USA, March 2002.

McBryan, O. A. (1994). "GENVL and WWWW: Tools for Taming the Web." Proceedings of the first International World Wide Web Conference.

Miranda, M., G. Xexéo, et al. (2006). "Building Tools for Emergent Design with COPPEER." Computer Supported Cooperative Work in Design, 2006. CSCWD '06. 10th International Conference: 1-6.

Najork, M. and J. L. Wiener (2001). "Breadth-first *crawling* yields high-quality pages." Proceedings of the 10th international conference on World Wide Web: 114 - 118.

O'Neill, E. T., B. F. Lavoie, et al. (2003). Trends in the Evolution of the Public Web: 1998 - 2002. D-Lib Magazine.

Olshefski, D. P. and J. Nieh (2006). Understanding the management of client perceived response time. Proceedings of the joint international conference on Measurement and modeling of computer systems. Saint Malo, France, ACM Press.

Olshefski, D. P., J. Nieh, et al. (2002). Inferring client response time at the web server. Proceedings of the 2002 ACM SIGMETRICS international conference on

Measurement and modeling of computer systems. Marina Del Rey, California, ACM Press.

Olshefski, D. P., J. Nieh, et al. (2004). "ksniffer: Determining the Remote Client Perceived Response Time from Live Packet Streams." Sixth Symposium on Operating Systems Design and Implementation (OSDI 2004).

Olston, C. (2003). "Approximate Replication." PhD thesis, Stanford University.

Olston, C. and J. Widom (2002). "Best-effort cache synchronization with source cooperation." Proceedings of the 2002 ACM SIGMOD international conference on Management of data: 73 - 74.

Pandey, S. and C. Olston (2005). "User-Centric Web Crawling." Proceedings of the 14th international conference on World Wide Web: 401 - 411.

Papapetrou, O. and G. Samaras (2004). "IPMicra: An IP-address based Location Aware Distributed Web Crawler."

Pinkerton, B. (2000). "WebCrawler: Finding What People Want." In Proceedings of the First World Wide Web Conference, Geneva, Switzerland.

Rao, A., K. Lakshminarayanan, et al. (2004). "Load Balancing in Structured P2P Systems."

Ratnasamy, S., P. Francis, et al. (2001). "A scalable content-addressable network." Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications: 161-172.

Ratnasamy, S., M. Handley, et al. (2002). "Topologicallyaware overlay construction and server selection." Proceedings of IEEE INFOCOM'02.

Reza, A., P. Esther, et al. (2007). Data currency in replicated DHTs. Proceedings of the 2007 ACM SIGMOD international conference on Management of data. Beijing, China, ACM Press.

Risvik, K. M., et al. (2002). "Search Engines and Web Dynamics." Computer Networks **39**: 289 – 302.

Rowstron, A. and P. Druschel (2001). "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems." Lecture Notes in Computer Science **2218**: 329.

Shkapenyuk, V. and T. Suel (2002). "Design and Implementation of a High-Performance Distributed Web Crawler." Proceedings of the 18th International Conference on Data Engineering: 357.

Siamwalla, R., R. Sharma, et al. (1998). "Discovering Internet topology."

Singh, A., M. Srivatsa, et al. (2003). "Apoidea: A Decentralized Peer-to-Peer Architecture for Crawling the World Wide Web " Proceedings of the SIGIR 2003 Workshop on Distributed Information Retrieval, Lecture Notes in Computer Science **2924**.

Stoica, I., R. Morris, et al. (2003). "Chord: a scalable peer-to-peer lookup protocol for internet applications." IEEE/ACM Transactions on Networking (TON) **11**(1): 17-32.

Stubblebine, T. (2006). Regular Expression Pocket Reference, O'Reilly.

Weiss, G. (2000). A Modern Approach to Distributed Artificial Intelligence, MIT Press.

Wonnacott, T. H. and R. J. Wonnacott (1990). Introductory Statistics for Business and Economics, Wie Wiley.

Yatin, C., R. Sriram, et al. (2005). A case study in building layered DHT applications. Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications. Philadelphia, Pennsylvania, USA, ACM Press.

Zhu, Y. and Y. Hu (2004). "Towards Efficient Load Balancing in Structured P2P Systems." Proceedings of the 18th international Parallel and Distributed Processing Symposium.