

UNIVERSIDADE FEDERAL DE SANTA CATARINA
TECNOLOGIAS DA INFORMAÇÃO E COMUNICAÇÃO

PEDRO HENRIQUE SOUZA

**UM SISTEMA DE COLETA DE DADOS DE FONTES HETEROGÊNEAS BASEADO EM
COMPUTAÇÃO DISTRIBUÍDA**

Araranguá, 26 de fevereiro de 2013

PEDRO HENRIQUE SOUZA

UM SISTEMA DE COLETA DE DADOS DE FONTES HETEROGÊNEAS BASEADO EM
COMPUTAÇÃO DISTRIBUÍDA

Trabalho de Conclusão de Curso submetido à
Universidade Federal de Santa Catarina como
parte dos requisitos necessários para a
obtenção do Grau de Bacharel em Tecnologias
da Informação e comunicação. Sob a
orientação do Professor Alexandre Leopoldo
Gonçalves.

Araranguá, 2013

Pedro Henrique Souza

**UM SISTEMA DE COLETA DE DADOS DE FONTES HETEROGÊNEAS BASEADO EM
COMPUTAÇÃO GRÁFICA**

Trabalho de Conclusão de Curso submetido à
Universidade Federal de Santa Catarina, como
parte dos requisitos necessários para a
obtenção do Grau de Bacharel em Tecnologias
da Informação e Comunicação.



Professor Alexandre Leopoldo Gonçalves, Dr.
Presidente da Banca - Orientador



Professor Juarez Bento da Silva, Dr.
Membro



Professor Anderson Luiz Fernandes Perez, Dr.
Membro

Araranguá, 26 de fevereiro de 2013

*“Dedico este trabalho especialmente a
minha mãe que me apoiou e a todos que
caminharam junto comigo”*

AGRADECIMENTOS

Agradeço a todos que estiveram comigo nesta caminhada, especialmente:

A Deus pelo dom de vida que me concedeu e por ter iluminado o meu caminho durante todos estes anos.

A minha família que sempre esteve ao meu lado.

Ao orientador Prof. Alexandre Leopoldo Gonçalves que teve papel fundamental na elaboração deste trabalho.

Aos meus colegas e amigos pelo companheirismo e disponibilidade para me auxiliar, incentivar e apoiar em vários momentos.

RESUMO

Atualmente a quantidade de informação cresce de maneira exponencial, seja na Web ou redes internas das organizações. Um dos fatores que justificam esse fato é a participação cada vez mais frequente de usuários comuns não somente no consumo da informação, mas também na produção de conteúdo. Sendo assim, são requeridas maneiras eficientes de se coletar e armazenar grandes volumes de informação. Como tarefa, a coleta de informação constitui-se primeiramente na localização de determinada fonte de informação e posteriormente em sua coleta. De modo geral, as informações estão dispersas em servidores distribuídos geograficamente quando se fala na Web, mas também dentro das organizações espalhadas nos servidores e computadores pessoais. Coletar essa quantidade de informação exige poder de processamento computacional. Visando promover suporte a esta demanda o presente trabalho propõe um sistema em que a tarefa de coleta de dados seja realizada de maneira distribuída. A demonstração de viabilidade é realizada através de um protótipo implementado a partir da proposição de uma visão lógica (funcionamento geral) e física (detalhamento dos componentes tecnológicos). O protótipo desenvolvido contém os serviços de análise da estrutura de determinado recurso da Web e a coleta propriamente dita de maneira distribuída, permitindo deste modo, atender a demanda de um sistema de coleta de informação em grande escala. Visando possibilitar a análise do sistema proposto, são elaborados três cenários para verificar a adaptação do coletor, bem como, a sua capacidade de processamento. A aplicação do protótipo nestes cenários de coleta permitiu demonstrar que este é capaz de obter resultados consistentes e satisfatórios em relação à adaptação e desempenho em diferentes configurações, tanto na fase de análise quanto na fase de coleta.

Palavras-chave: Coleta de Dados; Web; Computação Distribuída, Coletor Distribuído.

ABSTRACT

Currently the amount of information grows exponentially, either on the Web or on organizational networks. One of the factors that justify this fact is the participation increasing of common users not only downloading information, but also in the content production. Therefore area required efficient ways to crawl and store large volumes of information. In this sense, the crawling of information as a task aims first to locate a particular information source and later, in how to collect it. In general, information is spread in geographically distributed servers when taking into account the Web, but also within organizations scattered on servers and personal computers. Collect this amount of information requires a high computational processing power. In order to to promote support to this demand, the present work proposes a system in which the crawling task is performed in a distributed way. The feasibility demonstration is performed using a prototype implemented from the proposition of a logical view (general functioning) and physical (detailing the technological components). The prototype contains services for analyzing the structure of a given web resource and the crawler itself in a distributed way. It allows gathering the demand for a system which collects information on a large scale. Aiming to analyze the proposed system it was elaborated three scenarios to verify the adaptation of the crawler as well as its processing capacity. The application of the prototype in some scenarios was able to demonstrate that the proposed system is able to obtain consistent and satisfactory results taken into account adaptation and performance on different configurations, in both analysis and collect phases.

Keywords: Crawling; Web; Distributed Computing, Distributed Web Crawler.

LISTA DE ILUSTRAÇÕES

Figura 1 - Arquitetura geral para a coleta automática de dados	22
Figura 2 - Modo de coleta <i>firewall</i>	27
Figura 3 - Modo de coleta sobrescrita	27
Figura 4 - Processo de Sincronização.....	30
Figura 5 - Cluster Beowulf típico	41
Figura 6 - Cluster de alta disponibilidade com armazenamento compartilhado	42
Figura 7 - Cluster de alta disponibilidade com armazenamento local.....	42
Figura 8 - Cluster de balanceamento de carga.....	43
Figura 9 - Classificação de Clusters	43
Figura 10 - Visão lógica do sistema de coleta distribuída de dados.....	48
Figura 11 - Visão física do sistema de coleta distribuída de dados.....	50
Figura 12 – Diagrama de sequência do protótipo do coletor.....	52
Figura 13 - Modelo de coleta de documentos PDFs.....	60

LISTA DE GRÁFICOS

Gráfico 1 - Gráfico da quantidade de arquivos por nós ativos no sistema	57
Gráfico 2 - Gráfico entre o tempo de coleta de dados e os nós ativos no sistema.....	59
Gráfico 3 - Gráfico entre o tempo de coleta de dados e os nós ativos no sistema.....	61

LISTA DE TABELAS

Tabela 1 - Taxonomia de Flynn.....	37
Tabela 2 - Exemplo de sementes utilizadas.....	49
Tabela 3 - Resultados da coleta sem restrições	56
Tabela 4 - Resultado da Coleta do Jornal da Ciência.....	58
Tabela 5 - Resultado da coleta de PDFs com servidores web e diferentes números de nós	61

LISTA DE ABREVIATURAS E SIGLAS

API - *Application Programming Interface*

CPU - *Central Processing Unit*

EDW - *Enterprise Data Warehouse*

ETL - *Extraction Transformation Loading*

FTP - *File Transfer Protocol*

HDFS - *Hadoop Distributed File System*

HTML - *Hyper Text Markup Language*

HTTP - *Hyper Text Transfer Protocol*

HTTPS - *Hypertext Transfer Protocol over Secure Socket Layer*

IP - *Internet Protocol*

JCIFS - *Java Common Internet File System*

LAN - *Local Area Network*

MIMD - *Multiple Instruction Multiple Data*

MISD - *Multiple Instruction Single Data*

NNTP - *Network News Transfer Protocol*

PDF - *Portable Document Format*

RAM - *Random Access Memory*

SIMD - *Single Instruction Multiple Data*

SISD - *Single Instruction Single Data*

SMB - *Server Message Blocking*

TCP - *Transmission Control Protocol*

URL - *Uniform Resource Locator*

WAIS - *Wide Area Information Server*

WAN - *Wide Area Network*

SUMÁRIO

1. INTRODUÇÃO.....	15
1.1 <i>PROBLEMATIZAÇÃO</i>	17
1.2 <i>OBJETIVOS</i>	18
1.2.1 Objetivo Geral	18
1.2.2 Objetivos Específicos	18
1.3 <i>METODOLOGIA</i>	18
1.4 <i>ORGANIZAÇÃO DO TEXTO</i>	19
2. COLETA DE DADOS.....	20
2.1 <i>INTRODUÇÃO</i>	20
2.2 <i>ARQUITETURA PARA COLETA DE DADOS</i>	21
2.3 <i>CARACTERÍSTICAS</i>	23
2.3.1 Coleta Focada	23
2.3.2 Coleta Eficiente	24
2.3.3 Coleta Distribuída	24
2.3.3.1 Particionamento.....	25
2.3.3.2 Funções para Particionamento	25
2.3.3.3 Modos de Coleta	26
2.3.3.4 Comunicação entre processos.....	28
2.3.4 Polidez.....	28
2.3.5 Sincronização	29
2.3.6 Desempenho	30
2.3.7 Tolerância à Falha	32
2.3.8 Múltiplos Protocolos e Tipos De Arquivos	32
2.4 <i>PLATAFORMAS E TRABALHOS RELACIONADOS</i>	33
2.4.1 Googlebot	33
2.4.2 Crawler4J	33
2.4.3 Nutch	34
2.4.4 Retriever	35
3. PROCESSAMENTO DISTRIBUÍDO.....	36
3.1 <i>INTRODUÇÃO</i>	36
3.2 <i>CARACTERÍSTICAS DOS SISTEMAS DISTRIBUÍDOS</i>	38

3.3	<i>ARQUITETURAS DISTRIBUÍDAS</i>	39
3.3.1	Cluster	40
3.3.2	Grid.....	44
3.4	<i>FRAMEWORKS</i>	45
3.4.1	GridGain	45
3.4.2	Hadoop.....	46
4.	SISTEMA PROPOSTO	48
4.1	<i>VISÃO LÓGICA</i>	48
4.2	<i>VISÃO FÍSICA</i>	49
4.2.1	Detalhamento do Protótipo.....	51
5.	APRESENTAÇÃO DOS RESULTADOS	54
5.1	<i>INTRODUÇÃO</i>	54
5.2	<i>DESCRIÇÃO DOS CENÁRIOS</i>	54
5.2.1	Coleta Livre de Dados.....	56
5.2.2	Coleta de Notícias	57
5.2.3	Coleta de documentos PDF	59
6.	CONSIDERAÇÕES FINAIS	62

1. INTRODUÇÃO

O aumento crescente da informação, em geral textos, produzida e publicada na web e nas organizações vem promovendo um incremento nas pesquisas de áreas capazes de lidar com a coleta, armazenamento, integração e a disponibilização dessa informação. Segundo autores como Greengrass (2000), Kobayashi e Takeda(2000), Lyman (2000) e Himma (2007), esse aumento ocorre de maneira exponencial, gerando assim desafios, principalmente no que tange a capacidade de coletar e processar informações de maneira eficiente. A pesquisa realizada por (LYMAN, 2000), apontou que a quantidade de conteúdos disponíveis na Internet duplicava anualmente, e estimou em mais de dois bilhões o número páginas disponíveis na Internet no início do ano 2000.

Milhões de usuários publicam e têm acesso à informação livremente, fazendo uso da Internet para diversos objetivos (MODESTO et. al, 2005). Como exemplo, podem-se citar como atividades comuns dos usuários os *uploads* de vídeos ou fotos, atualização de seus *status* nas redes sociais, e o *download* dos mais variados arquivos. Pelo fato de ter um acesso fácil e rápido a todas as informações disponíveis, e todo usuário ser uma fonte de disseminação em potencial, a Internet vem se consolidando como o melhor e o mais eficiente meio de comunicação. Além do conteúdo disponível na Web, as organizações em geral produzem cada vez informações textuais.

Segundo (BOVO, 2011), é possível localizar dentro das organizações, vários outros tipos de informação textual em formato digital, tais como:

- os diversos tipos de relatórios técnicos;
- manuais disponíveis sobre procedimentos ou *softwares*;
- pesquisas respondidas pelos usuários sobre satisfação de um produto ou serviço;
- os registros (arquivos de *log*) de sistema de sistemas de busca corporativa ou mesmo de motores de busca de uso geral (*search engines*), como o Google®;

- *e-mails*, currículos, *e-books*, mensagens de comunicação instantânea.

Tal acúmulo de informação promove desafios para a coleta e disponibilização de modo que está possa ser útil aos usuários que dela necessitam. O processo de coleta consiste basicamente na localização de conteúdo em fontes de informação, por exemplo, sites da Web, com o armazenamento em alguma base de dados para posterior disponibilização. Uma vez que se localize uma fonte de informação (um site qualquer) é necessário, a partir de um ponto de entrada (página principal, por exemplo), extrair os dados e as ligações com outras páginas internas ou externas à fonte de informação em particular que está sendo analisada.

A coleta de dados é tratada na literatura como *web crawler* ou *crawling* consistindo em um processo computacional que automatiza a recuperação de dados, nos mais variados formatos alcançados por uma variedade de protocolos (CHO, GARCIA-MOLINA; PAGE, 1998; GOMES; SILVA, 2008). Os dados em si são representados por arquivos, e os protocolos indicam quais fontes de informação podem ser inspecionadas. Um protocolo comum é o HTTP que suporta grande parte dos arquivos residentes na Web. Outro exemplo é o protocolo SMB que possibilita a coleta de arquivos que sejam compartilhados entre computadores utilizando sistemas operacionais como Linux™ e Windows™ (SANTOS, 2009).

Contudo, para que os coletores atendam a demanda crescente por informações, diversas políticas devem ser seguidas. Isto se justifica uma vez que este processo demanda muita largura de banda, podendo ocasionar lentidão aos usuários que se utilizam da Web ou de redes internas nas organizações, e mesmo sobrecarregar os servidores com um número elevado de requisições. Deste modo, a eficiência é fundamental de modo que não prejudique o funcionamento de outros serviços da *Web* (CASTILLO, 2004).

Como já mencionado, *crawlers* (coletores) se utilizam em geral dos *links* que constam nas páginas Web para acessar e coletar novas páginas. Entretanto, nem todas as informações estão acessíveis diretamente por *links*, precisando desta maneira, de uma resposta a consultas efetuadas a um banco de dados (PANAGIOTIS, 2001). A Web que não pode ser diretamente acessada por um coletor é chamada de *Deep Web*.

Segundo pesquisa realizada por He (2007), a *Deep Web* permanece largamente inexplorada e com pouca cobertura pelos motores de busca, uma vez que seu conteúdo é gerado dinamicamente por *sites*. Esta fraca cobertura dos seus dados por motores de busca sugere que esta estrutura não é devidamente suportada. O autor ainda faz uma comparação

com a superfície da Web que é grande, possui um rápido e diversificado crescimento, enquanto que na *Deep Web*, este crescimento é mais estruturado e sofre uma inerente limitação de rastreamento.

Além da *Web* e *Deep Web* menciona-se ainda o fato de cada vez mais dados estarem sendo gerados pelos usuários. Estes passam de consumidores de informações estáticas conectadas por *hiperlinks*, a produtores de informações que geram impactos, tanto na disseminação da informação quanto nos mais variados modelos de negócio. Esta evolução vem sendo chamada de Web 2.0 (MODESTO et. al, 2005), uma vez que as informações passam a ser geradas a partir de um processo de socialização.

1.1 PROBLEMATIZAÇÃO

A busca e armazenamento de informação são partes importantes dos mecanismos atuais de busca, tais como o Google®, que utiliza o Googlebot® e o Bing®, utilizando o Msnbot®, que trazem resultados sobre as mais variadas informações dispostas na Web.

Nesse sentido, as atividades de coleta na *Web* ou *Deep Web* despendem de um alto nível de processamento. De maneira geral, é necessário utilizar uma estrutura capaz de executar coletas de grandes volumes de informação em um período razoável de tempo, pois os índices que representam as páginas coletadas e que são o suporte para mecanismos de busca tendem a ficar rapidamente desatualizados.

Além disso, deve-se levar em consideração políticas adequadas na execução da tarefa de coleta, evitando, por exemplo, que determinado *site* seja sobrecarregado como um número indevido de requisições do coletor. Políticas inadequadas podem conduzir a paralisação de determinado Servidor *Web*.

Seguindo as políticas de coleta um coletor deve ter algumas características para atender os requisitos da busca. O sistema deve ser flexível, atender aos diferentes tipos de documentos, conseguir navegar pela *Web*, assimilando os diferentes padrões encontrados em suas páginas. Além disso, pode priorizar a coleta baseando-se em termos referentes a determinado assunto, extraíndo assim documentos pertinentes a um objetivo em particular. Outra importante característica é a distribuição da tarefa de coleta em vários computadores, de modo que estes trabalhem como um único sistema.

De modo geral, o processo como um todo que envolve a coleta de dados de várias fontes e com diversos formatos, deve ser polido para evitar sobrecarga de servidores *Web* e

deve possuir um alto desempenho, visto o grande volume de informações disponível atualmente, tanto na Web quanto em organizações. Sendo assim, projetar sistemas coletores que reúnam tais características pode ser visto como um desafio.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

O objetivo desse trabalho é propor e desenvolver um sistema de coleta de dados que seja distribuído e com capacidade de atender variados cenários de uso.

1.2.2 Objetivos Específicos

Visando-se atingir o objetivo principal, alguns objetivos específicos são requeridos, entre eles:

- Pesquisar e analisar diferentes soluções para a coleta de dados objetivando compreender as características necessárias à realização de um coletor de dados;
- Propor uma visão lógica e física de um sistema de coleta que permitam suportar cenários de coleta;
- Desenvolver um protótipo que permita avaliar o sistema de coleta de dados proposto neste trabalho;
- Apresentar cenários de uso em que o trabalho proposto possa ser analisado;
- Realizar uma discussão dos resultados obtidos a partir dos cenários de uso em que o coletor de dados foi aplicado.

1.3 METODOLOGIA

Este trabalho apresenta sistema distribuído de coleta de informações utilizando como base alguns *frameworks* de código aberto. Para que o coletor seja distribuído, deve trabalhar com um conjunto de máquinas e conseguir realizar uma coleta de grande escala, que podem vir a ser necessidades de usuários futuros. As etapas deste trabalho estão assim divididas:

Etapa 1: análise da literatura focando nas seguintes áreas: coleta de dados e computação distribuída;

Etapa 2: analisar soluções disponíveis de código aberto que possam auxiliar no desenvolvimento do sistema proposto neste trabalho;

Etapa 3: proposição de uma visão lógica e física que promova suporte à um sistema de coleta distribuída de informações;

Etapa 4: desenvolvimento de um protótipo de coleta de dados com base na visão lógica e física;

Etapa 5: testes do protótipo considerando alguns cenários de uso visando demonstrar as características implementadas;

Etapa 6: avaliação dos resultados obtidos através da utilização do sistema distribuído de coleta.

1.4 ORGANIZAÇÃO DO TEXTO

O documento está dividido em 6 capítulos. Neste capítulo apresenta-se uma breve contextualização, a problemática e os objetivos, geral e específicos. No segundo capítulo é realizada uma revisão sobre a área de Coleta de Dados apresentando suas principais características.

O terceiro capítulo faz-se uma revisão da literatura relacionada à computação distribuída, expondo conceitos de sistemas com múltiplos processadores, as características dos sistemas distribuídos e uma abordagem sobre a configuração de um *cluster*, modelo utilizado neste trabalho. O quarto capítulo propõe uma visão lógica e física para um coletor de dados. Enquanto a visão lógica apresenta uma introdução que serve de base para o desenvolvimento de um coletor, a visão física detalha os componentes tecnológicos utilizados no desenvolvimento.

O quinto capítulo detalha o protótipo obtido a partir da visão lógica e física. Além disso, apresenta os cenários de utilização do protótipo e discute os resultados obtidos pelo mesmo. O sexto capítulo discute as considerações finais sobre este trabalho, bem como, propõe trabalhos futuros.

2. COLETA DE DADOS

Este capítulo apresenta uma introdução sobre a coleta de dados e uma arquitetura geral que os coletores seguem. Posteriormente, apresenta as características da coleta e alguns coletores utilizados por motores de busca ou disponibilizados em formato de código aberto (*open-source*).

2.1 INTRODUÇÃO

O processo de coleta constitui-se na varredura da Web por meio de uma lista de endereços pré-definidos e no armazenando do conteúdo coletado em uma base de dados para uso futuro. Entre esses usos destaca-se a disponibilização da informação para sistemas de recuperação de informação (KORFHAGE, 1997). Além disso, tal informação pode ser fonte para a área de extração de informação objetivando a identificação de dados relevantes que possam ser utilizados na análise de determinado domínio de problema.

No contexto do presente trabalho, a coleta, também tratada na literatura como *crawling*, é um processo computacional que automaticamente recupera dados alcançados em uma rede, por uma variedade de protocolos. Conceitos semelhantes são apresentados por Cho, Garcia-Molina e Page (1998). Pode-se representar os dados como os arquivos, e os protocolos veem para delimitar as fontes de informações a serem inspecionadas.

De modo geral, a área está diretamente relacionada a coleta de informação autônoma, que se utiliza de robôs, também chamados de *crawlers*, ou seja, programas com objetivo de vasculhar e coletar automaticamente conteúdo na Web visando manter os mecanismos de busca, tais como Google[®] e Bing[®], atualizados. Contudo, em virtude do tamanho da Web e da multiplicidade de formatos do conteúdo disponível, o desenvolvimento de coletores capazes de lidar com ambientes complexos promove desafios. Desse modo, as próximas seções irão analisar os diversos componentes de uma arquitetura de um coletor de dados visando promover o entendimento de todo o processo de uma coleta de dados.

Olhando pelo lado do usuário, foram oferecidos novos meios de acesso à informação na *Web* e compartilhamento de informações. As aplicações *Web* tornaram-se mais fáceis e ágeis e conseguem ser usadas mesmo existindo uma latência na rede. A *Web* mudou, nasceu como um meio e virou uma plataforma, que reúne dados de várias fontes e de dispositivos em tempo real, permitindo que as pessoas contribuam com ideias e conteúdos (WANG, 2009).

A *Web* nasceu no centro europeu de pesquisa nuclear (CERN), na Suíça, em 1989, como um veículo de troca de documento entre uma comunidade de físicos conectados pela internet (COULOURIS; DOLLIMORE; KINDBERG, 2007).

Pode-se dizer que a *Web 2.0* é a segunda geração de serviços na rede, fazendo um trocadilho com o termo utilizado na informática para a versão de um software, buscando ampliar as formas de produção cooperada e compartilhamento de informações online. Um conjunto de novas estratégias mercadológicas para o comércio eletrônico e aos processos de interação social mediado pelo computador. No seu surgimento, a chamada *Web 1.0*, os usuários eram meros espectadores da informação, em que existiam apenas os textos e seus *hiperlinks* (PRIMO, 2006).

Ainda considerando o aprimoramento da *Web* muitos mencionam que o estágio atual seja caracterizado com *Web 3.0*, sendo esta a segunda fase da evolução da *Web*. Neste ponto o processo de criação de informação passa a ter a influência das máquinas tornando-se mais dinâmico, interativo, eficiente e administrável (PATTAL; YUAN; JIANQIU, 2009).

2.2 ARQUITETURA PARA COLETA DE DADOS

A coleta de informação começa com um URL pré-definido (uma semente), procurando por uma proximidade das informações que se deseja coletar, este é o ponto de partida do *crawler*. Inicia varrendo (analisando) a página semente, coletando seus dados e posteriormente seus links, que serão armazenados em uma lista. Após esta primeira coleta ser executada, o coletor parte para os *links* encontrados na página semente, executando os mesmos passos nestas novas páginas.

A Figura 1 **Erro! Fonte de referência não encontrada.** apresenta uma arquitetura genérica segundo Liu (2009) para a coleta de dados automática. O processo começa utilizando uma lista de URLs entendidas como sementes. Com base nessa lista o coletor executa repetidamente os seguintes passos: a) as sementes são adicionadas ao *frontier*, e a partir disso

é selecionado um URL do *frontier* na qual será executada a coleta (*download*) do documento correspondente e, dependendo do formato, extrair quaisquer links contidos neste URL; e b) para cada um dos links extraídos, é verificado se é um URL que ainda não foi visitado, caso ainda não tenha sido, é alocada no *frontier* (HEYDON; NAJORK, 1999).

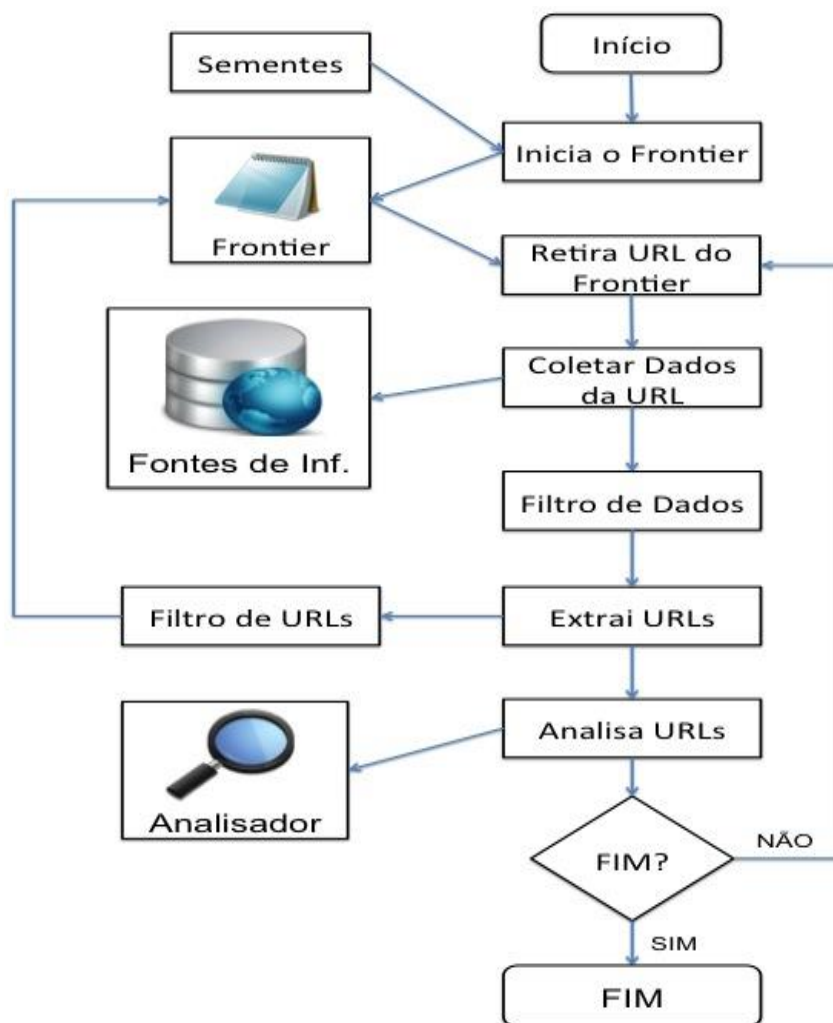


Figura 1 - Arquitetura geral para a coleta automática de dados

Fonte: Adaptada de Liu (2009)

As sementes e os filtros URL já devem ser direcionados para o conteúdo que os usuários desejam (BATSAKIS, 2009). Caso os dados requeridos sejam notícias sobre a economia europeia, as sementes devem ser sites europeus que tratem de economia, e não sites de músicas dos EUA. Diminuindo a gama de informações que serão analisadas e focando apenas nas informações relevantes ao usuário. Por isso, as sementes são importantes, pois a partir delas é que se dará o começo da busca. É de grande ajuda que a semente seja uma página que já está contida no contexto da coleta, para que não se desperdice tempo em páginas que não levaram diretamente as informações que se deseja coletar.

Os filtros são importantes, uma vez que objetivam delimitar um domínio, por exemplo, um site de notícias qualquer. Assim, a coleta estará restrita a determinado site evitando percorrer toda a Web. Pode-se ainda adicionar como filtro diversos tipos de documentos, tais como, documentos, imagens, áudio e mesmo vídeos (SANTOS, 2009).

2.3 CARACTERÍSTICAS

Para que um coletor possa varrer a *Web* de maneira eficiente, coletando os dados relevantes para a sua pesquisa, conseguindo navegar e coletar sem interrupções ou causar problemas nos serviços *Web*, deve-se levar em conta alguns aspectos para a criação desse coletor. Assim se busca a construção de um coletor com boas maneiras, que consiga fazer sua coleta sem prejudicar os serviços que a *Web* oferece ou seus usuários.

Em trabalhos relacionados à coletores, aspectos como uma arquitetura flexível (SANTOS, 2009) e coletor eficiente (CASTILLO, 2004) são mencionados. A seguir, são detalhadas algumas características, entre elas, a coleta focada, a coleta eficiente, a coleta distribuída, a polidez, a sincronização, o desempenho, a tolerância a falhas e múltiplos protocolos e tipos de arquivos.

2.3.1 Coleta Focada

As coletas, dependendo da necessidade, podem ter um tema de interesse, ou seja, um determinado tópico. Como exemplo temos uma explicação de Kumar e Vig (2012) em que coletores focados são projetados para coletar documentos/informações da *Web* que são relevantes para um domínio pré-definido, evitando áreas irrelevantes da web.

Assim como vendedores de carros desejam páginas com preços ou características dos veículos para a venda, investidores da bolsa estão interessados nas informações que dizem respeito as empresas das quais possuem ações. Esse tipo de estratégia é chamada de coleta focada e é direcionada a um determinado tópico/assunto.

Segundo Ahmadi-abkenari (2011), existem dois tipos de crawlers: (a) os focados que limitam a sua busca as páginas relacionadas à uma taxonomia pré-definida, evitando assim regiões da *Web* irrelevantes e tempo de processamento para analisar e excluir dados não pertinentes a pesquisa; e (b) os não focados, os quais buscam pela web inteira para construir uma base de dados com perspectiva de uso geral. Assim, esse tipo de estratégia exige mais

tempo e esforço computacional para criar, manter e conservar uma base de dados em grande escala atualizada.

De modo geral, o processo de coleta focada possui seu desempenho diretamente relacionado às páginas fornecidas como sementes. Normalmente estas páginas são escolhidas como as melhores respostas por um motor de busca, usando o tema da busca. Sementes de boa qualidade podem ser páginas relevantes ao assunto ou páginas apontadas por esta página relevante (BATSAKIS, 2009). Para que a coleta tenha um melhor aproveitamento é sugerido que as sementes já estejam no contexto da busca. Por exemplo, se a busca for por notícias de jornais, seria mais adequado indicar previamente sementes que se relacionem as páginas de grandes jornais.

Segundo Castillo (2004), considerando a maneira como a *Web* vem crescendo o processo de coleta deverá concentrar-se nas páginas mais “valiosas”, pois nenhum coletor conseguirá rastrear a *Web* inteira. Desse modo, a coleta focada passa a ser uma solução possível para encontrar informações pertinentes.

2.3.2 Coleta Eficiente

Apresentado por (CHO; GARCIA-MOLINA; PAGE, 1998), a coleta eficiente tem como base a ordenação de URLs no *frontier*, propondo heurísticas que auxiliam a maneira como um coletor deve selecioná-las para começar e continuar a coleta.

Diferentemente da coleta focada, onde a coleta esta relacionada e se restringe a determinado tópico, a coleta eficiente, na maioria das vezes, recuperará todos os documentos de uma determinada fonte, priorizando a coleta das informações mais relevantes nos momentos iniciais da sua execução. Depois de ter realizado a coleta de determinada página, o coletor poderá revisita-la buscando por alguma atualização considerando como estratégia um tempo decorrido desde a última visita. Caso os documentos sejam significativos e a atualização desses uma necessidade, é essencial que o coletor revise os documentos atualizados e depois o restante.

2.3.3 Coleta Distribuída

Considerando a quantidade de informações disponíveis na *Web* e mesmo em determinadas organizações, um processo de coleta que se utilize de apenas um computador não conseguirá completar a tarefa em um tempo satisfatório. Nesse sentido, a coleta

distribuída torna-se essencial para cumprir a tarefa de maneira adequada. As subseções abaixo promovem uma visão geral das funções necessárias para uma coleta distribuída.

2.3.3.1 Particionamento

Para que isso seja executado de maneira eficiente deve-se particionar a coleta de tal modo que as partes sejam distribuídas entre um determinado número de computadores. Assim, deve-se aplicar as funções de particionamento para conseguir um coletor distribuído. As funções determinam como os URLs devem ser distribuídos entre os computadores que participam de um determinado processo de coleta. Cho e Garcia-Molina (2002) apresentam essas funções em três categorias:

- Independente: os processos são executados, cada um, em determinado computador sem trocas de mensagens com outros processos. Permite que o mesmo URL seja recuperado mais de uma vez por dois ou mais processos;
- Dinâmica: baseada em um coordenador central responsável pela divisão lógica do domínio de coleta associando cada URL a um determinado processo em tempo de execução;
- Estática: o modelo de execução é semelhante à dinâmica, ou seja, distribui URLs para os processos, porém a maneira como a associação ocorre é definida antes do início da coleta.

2.3.3.2 Funções para Particionamento

As funções de particionamento são focadas para seu uso na Web, mas elas podem se adaptar a outras fontes de informação. As funções abordadas neste trabalho são descritas por (CHO; GARCIA-MOLINA, 2002):

- Funções de particionamento baseadas na codificação de URLs: cada URL é enviada através de uma função *hash*, retornando o valor utilizado para identificação do processo responsável pela coleta das informações. Desta maneira, mais de um processo poderá coletar informações das diferentes páginas de um mesmo site;
- Funções de particionamento baseadas na codificação de sites: utilizando a parte do URL que contém o nome do site para a computação do código *hash*. Diferente do modelo da codificação de URL, apenas um processo será responsável pela coleta de todas as páginas referente ao site;

- Divisão hierárquica da Web: procura decompor a Web por domínio, neste caso, todos os URLs pertencentes a um certo domínio (.com, por exemplo) serão coletadas por determinado processo, assim cada URL de outros domínios são responsabilidades de outros processos, atribuindo um domínio a cada processo. Assim pode-se selecionar URLs restritos a um determinado país (utilizando .br, .uk) ou domínios de organizações (.org). Conseqüentemente, divide-se os processos geograficamente, enquanto um processo está no Brasil (domínio .br) outro estará na Inglaterra (domínio .uk), fazendo com que os servidores não sejam sobrecarregados pelo coletor.

Considerando os estudos de Wan e Tong (2008) sobre funções *hash* para a divisão da *Web*, estas devem ser realizadas de maneira eficiente, visando descentralizar o processamento de um coletor. A função precisa explorar a estrutura de *links* entre os documentos de maneira eficiente, conseguindo equilibrar o volume de trabalho entre os processos. Desse modo, existe a necessidade de que os coletores possuam autonomia para efetuarem a melhor escolha, tanto das funções *hash* quanto dos algoritmos para a divisão hierárquica.

2.3.3.3 Modos de Coleta

As informações são particionadas e cada um dos processos é responsável por coletá-las. Porém, devido à natureza da *Web*, em determinado momento um processo pode conter URLs referentes a outros processos. Visando manter o foco de cada processo, o coletor pode operar sob alguns modos específicos, conforme Cho e Garcia-Molina (2002), como:

- *Firewall*: cada processo recupera apenas os URLs que pertençam a sua partição, descartando os outros URLs irrelevantes. Sendo assim, é assegurado que o mesmo URL não seja coletado por múltiplos processos. Contudo, alguns documentos podem não ser recuperados, pois determinados URLs somente serão alcançados através do relacionamento entre as partições. Como exemplificado na Figura 2, caso C1 encontre um URL que não pertença a sua partição, ela é enviada para o processo coordenador que enviada para a partição de C2. Ele é responsável por decidir se uma nova partição deve ser criada ou o URL pode ser alocado em uma partição já existente. Não há necessidade de comunicação entre os processos;

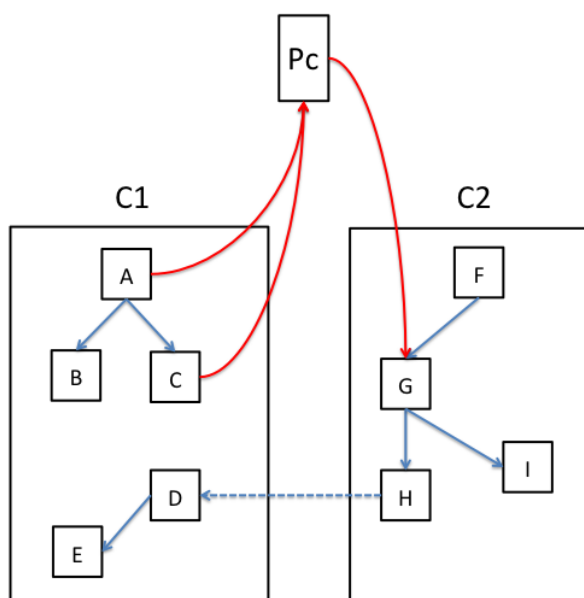


Figura 2 - Modo de coleta *firewall*

Fonte: Adaptada de Cho e Garcia-Molina (2002)

- **Sobrescrita:** Cada processo realiza, inicialmente, a coleta dos URLs pertencentes à sua partição e armazenará para coleta futura URLs que pertençam a uma outra partição. Quando os URLs de C1 acabam, os URLs restantes (pertencentes a outras partições) serão inspecionadas visando encontrar links que remetam novamente para links da partição C1 que não tenham sido referenciadas pelo conjunto inicial de URLs dessa partição, como visto na Figura 3. Essa estratégia se diferencia do modo *firewall*, pois não segue URLs de outras partições. Assim, um URL pode ser recuperado por mais de um processo, gerando duplicidades na coleta. Neste modo, como no anterior, também não existe a necessidade de comunicação entre os processos;

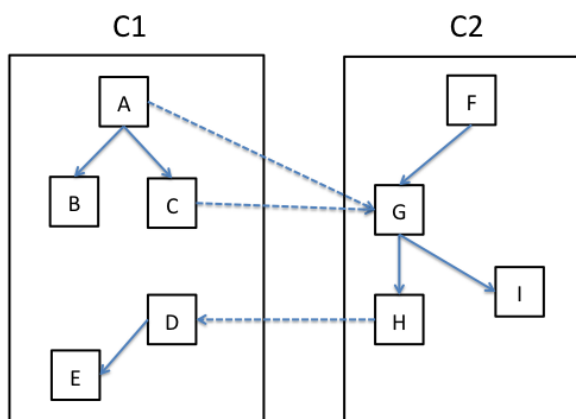


Figura 3 - Modo de coleta sobrescrita

Fonte: Adaptada de Cho e Garcia-Molina (2002)

- Troca: marcado pela permuta constante de URLs entre os processos de coleta, onde um determinado URL referente a uma partição em específica não é coletada por outro processo. De modo geral, se uma partição C1 encontra um URL referente a uma partição C2 ela transfere esta URL para a partição correspondente. Este modo de troca, diferente dos demais citados, impõe uma comunicação excessiva entre os processos que deve ser minimizada obrigando que os processos se conheçam entre si.

2.3.3.4 Comunicação entre processos

Para a minimização da carga imposta pela troca de URLs entre os processos, segundo Cho e Garcia-Molina (2002), podem-se destacar as seguintes comunicações:

- Em lotes: os URLs não são submetidos às outras partições no momento que as encontra, primeiramente são armazenadas em um conjunto delas e enviadas todas de uma única vez. Assim melhorando a comunicação entre os processos, pois todos os URLs são enviados em uma única troca. Caso determinado URL apareça duas ou mais vezes, este será submetido apenas uma vez;
- Replicação: Mantém cópias dos URLs, consideradas mais importantes, em cada um dos processos deixando claro a que processos o URL se referencía. Assim não haverá a necessidade de troca desses URLs uma vez que o processo sabe quais são as suas e quais não são. Os URLs mais importantes podem ser definidos em uma coleta prévia.

2.3.4 Polidez

Coletores necessitam de recursos como banda de internet, memória e ciclos do CPU. Isto é requerido tanto nos computadores e rede de onde parte a coleta, como nos locais de onde a informação será extraída. Uma vez que tais recursos não são abundantes em todos os locais da busca, existe a preocupação durante o desenvolvimento, de que o coletor não venha a consumir esses recursos de maneira demasiada (SANTOS, 2009). Os recursos mais exigidos por um coletor, apontados por (CASTILLO, 2004), são: (a) largura de banda; (b) operações de alto grau de paralelismo; (c) sobrecarga do servidor caso a frequência de acesso a um servidor específico seja muito alta;

Um coletor que sobrecarregue a máquina em que está executando, pode vir a consumir a largura de banda disponível, atrapalhando outros usuários da mesma rede. Pode ainda, sobrecarregar servidores com excessivas requisições.

Os coletores dos grandes buscadores podem explorar a *Web* com conexões exclusivas para seu fim. Contudo os motores de busca departamentais, em sua grande parte, compartilham a largura de banda de rede com outros serviços e usuários. Por esta razão, Diligenti (2004) assume que o controle de banda é um ponto chave do projeto do coletor. Assumindo que essa política deve alcançar dois objetivos. Primeiramente o rastreamento não deve impedir a utilização da rede dos usuários que compartilham desta conexão, o coletor não pode ocupar a rede disponível, assim monopolizando-a. Segundo, deve conseguir explorar todo o recurso ao qual lhe foi disponibilizado.

Uma proposta para defender os servidores é apresentada por Castillo (2004) através da implementação de inibidores de coletores. Esse tipo de estratégia faz com que o servidor restrinja a ação dos coletores à determinados arquivos fazendo com que o servidor consiga servir melhor aos usuários. Citam-se ainda os coletores ditos polidos que levam em conta uma quantidade de tempo entre cada requisição ao servidor.

Os servidores podem ajudar no processo de coleta, disponibilizando um conjunto de metadados. Eles contêm informações sobre quais dados foram atualizados, assim o coletor saberá quais os dados que devem ser coletados, economizando tempo de análise dos dados (BRANDMAN et al., 2000). Com esse recurso a coleta de informação e atualização de dados pode ser feita de maneira mais rápida e eficiente.

Por outro lado, se o recurso de metadados não é utilizado, torna-se necessário que o coletor recupere os arquivos, analisando-os para ver se houve ou não alguma atualização. Isso acarreta em uma transmissão elevada de dados, causando tráfego de informações.

2.3.5 Sincronização

Sincronização diz respeito à atualização dos documentos da base de dados. O coletor deve verificar se os documentos salvos ainda condizem com os dados da fonte de informação (CHO; GARCIA-MOLINA, 2000). A Figura 4 apresenta como ocorre a atualização. Entretanto, não há como saber quando a fonte de informação será atualizada. Por isso as páginas devem ser revisitadas para a verificação de seu conteúdo. Algumas páginas podem ficar meses sem uma alteração, enquanto outras sofrem atualizações periódicas. Páginas de

jornais, *blogs* e *twitter*, são fontes de informações que causam uma grande carga de trabalho aos coletores pelo seu alto grau de atualizações.

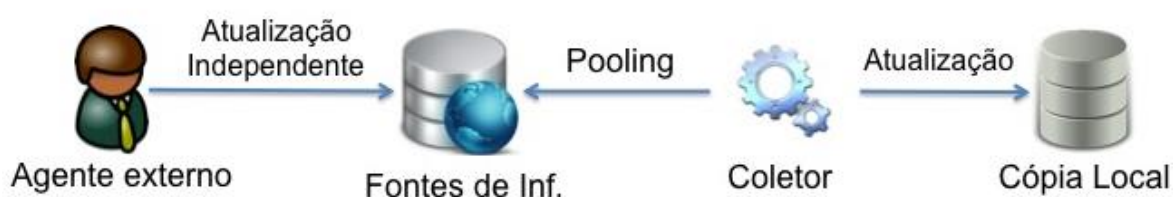


Figura 4 - Processo de Sincronização

Fonte: Adaptada de Cho e Garcia-Molina (2000)

Em relação aos grandes buscadores, eles são fundamentais, para que não fique disposto aos usuários *links* que levam a páginas que podem nem existir mais. Segundo Castillo (2004), algumas pesquisas concentram-se na vida útil de páginas, focando-se na disponibilidade das informações. É importante também concentrar-se na taxa de mudança de páginas, frequência com que ela é atualizada, sabendo a taxa de mudança pode-se produzir uma melhor ordem para visitar a página.

Castillo ainda apresenta algumas características das alterações de páginas *Web*, são métricas relacionadas ao tempo:

- Idade: data da visita – data de modificação;
- Tempo de vida: data de exclusão – data de criação;
- Número de mudanças durante a vida: alterações;
- Intervalo de troca Média: tempo de vida / alterações;

Depois de uma estimativa dos valores acima, são obtidos alguns números, por exemplo:

- Distribuição de intervalos de troca;
- Média de vida útil de páginas.

2.3.6 Desempenho

Os coletores atuam em ambientes compostos por milhões de documentos, geralmente distribuídos geograficamente sendo esta uma característica da *Web*. Consequente, o desempenho de um coletor é parte fundamental desse processo. Qualquer esforço que possa ser minimizado para ganho de tempo ou diminuição da sobrecarga dos recursos, favorece o

desempenho do coletor.

Autores como Santos (2009) e Habib e Abrams (2000), listam alguns aspectos que influenciam no desempenho de um coletor, entre eles:

- Fontes de Latência: existem fontes de latência durante o processo de recuperação de páginas Web, alocando cliente e servidor tanto no mesmo país quanto em países diferentes. Apontando como fontes de latência a resolução de *hosts* em IPs, o tempo para o estabelecimento de uma conexão TCP, depois de estabelecida a conexão entre cliente e servidor, o tempo que o primeiro *byte* leva para ser recuperado e para o *download* completo de um arquivo. Uma solução para amenizar essa latência seria a conexão TCP ser mantida, para que várias requisições sejam atendidas em uma mesma sessão. Lembrando também que o servidor tem grande impacto, podendo atrasar o trabalho do coletor, por estar sobrecarregado de requisições ou não ter capacidade de *hardware* para suportar a demanda de trabalho;
- Múltiplas *threads* e requisições assíncronas: a estratégia baseada em múltiplas *threads* consiste na alocação delas para o tratamento dos URLs a serem coletadas. Dividindo entre ela o trabalho de coleta e processamento dos dados coletados. Já uma única *thread* realizando assincronamente requisições à Web permite que diversos documentos sejam recuperados simultaneamente, criando um cenário onde é possível processar os documentos coletados paralelamente ao *download* deles e assim, como na estratégia empregando *threads*, maximizar os recursos de hardware. Técnicas possíveis com os CPU com múltiplos núcleos, podendo realizar as *threads* de forma simultânea;
- Normalização de URLs: URLs escritas de maneiras diferentes podem levar a mesma página Web. Pode-se pegar como exemplo <http://globo.com> e <http://www.globo.com>, são sintaticamente diferentes, mas levam a um mesmo destino o site da Rede Globo. Logo esses URLs serão tratados como diferentes e assim sendo analisadas novamente pelo buscador. Isso demanda tempo que é fundamental no desempenho. Para apurar esse problema Gomes e Silva (2008) propõem que haja uma normalização dos URLs durante o processo de coleta. Logo após identificar um novo URL para ser coletado, ele passará por um processo com as seguintes atividades: adição de uma barra ao final do URL, remoção de âncoras, inclusão do prefixo 'www' para alguns sites (corresponde a 51% dos URLs duplicadas) e remoção de nomes de arquivos que podem ser omitidos (podendo diminuir o número de coletas duplicadas para 36%).

2.3.7 Tolerância à Falha

Um coletor trabalha com grandes quantidades de dados e precisa assegurar a integridade dos mesmos, não importando os erros que podem ocorrer no processo de coleta. Esse processo pode sofrer uma parada eminente, o coletor deve continuar do ponto de parada, ressaltando a perda de alguns poucos documentos já coletados. Procurar estar em conformidade com o padrão ACID (atomicidade, consistência, isolamento e durabilidade) para o armazenamento dos dados coletados (KRITIKOPOULOS; SIDERI; STROGGILOS, 2004).

Muitos erros podem ser encontrados na Web, como a perda da conexão, mau funcionamento dos servidores, até páginas HTML corrompidas. Dependendo do tamanho das informações a coleta pode variar de tempo, aumentando a possibilidade de falhas durante uma coleta longa. Segundo Hafri e Djeraba (2004), o sistema deve retirar, periodicamente, os dados da memória e salvar em disco, para que uma reinicialização possa ser efetuada.

Assim, é esperado que o coletor tolere falhas esperadas nos seus processos. Como exemplo de uma manutenção no sistema de coleta, ou até mesmo problemas com o *hardware* onde o coletor está operando.

2.3.8 Múltiplos Protocolos e Tipos De Arquivos

Durante uma coleta podem ser encontrados vários tipos de documentos, em uma mesma página Web. O coletor deve conseguir trabalhar com esses diferentes formatos de informações, utilizando ferramentas para cada um dos tipos. Para conseguir extrair a informação que seja relevante a sua busca.

O Mercator (HEYDON; NAJORK, 1999) é um exemplo de coletor que consegue compreender os diversos protocolos e tipos dos documentos. Utilizando módulos feitos especificamente para o entendimento dos variados protocolos como também tratamento dos tipos de documentos. Além de já suportar alguns mais conhecidos como HTML e PDF. Assim pode inspecionar as fontes de informação sem que seja surpreendido ou despreze algum documento que não consiga manipular.

Segundo Santos (2009), a cooperação do coletor com novos protocolos é realizada geralmente pela utilização de bibliotecas de terceiros. Exemplos de tecnologias como o Java concretizam a especificação dos protocolos file, FTP, gopher, HTTP, HTTPS, news, NNTP e WAIS. O autor ainda fortalece que existem muitos projetos, na sua maioria *open source*, que provém o conhecimento necessário para a utilização de protocolos. Exemplos são o JCIFS

para o protocolo SMB, e o JavaSVN para o protocolo utilizado na comunicação com o Subversion, um sistema de controle de versão empregado para o armazenamento e gestão das revisões de documentos.

2.4 PLATAFORMAS E TRABALHOS RELACIONADOS

Atualmente existem em ação alguns coletores de grandes buscadores para coletar e manter atualizações das páginas *Web*. Após a coleta essas páginas serão indexadas para que, quando a busca pelo usuário for realizada a resposta será facilmente encontrada. São encontrados também *crawlers* de código aberto para usuários avançados e também para pesquisas e estudos na área. Os grandes buscadores se preocupam com informação em geral, enquanto coletores mais específicos podem ser construídos para oferecer um serviço de busca mais personalizado aos clientes. Alguns exemplos de *crawlers* que temos hoje são:

2.4.1 Googlebot

Googlebot® é o robô de rastreamento da *Web* do Google® (também chamado de "indexador"). De modo geral, Googlebot® descobre páginas novas e atualizadas para serem incluídas no índice do Google. Usando um grande conjunto de computadores para buscar (ou "rastrear") bilhões de páginas na *Web*. O Googlebot® usa um processo de algoritmos que determinam quais sites devem ser rastreados, com que frequência e quantas páginas devem ser buscadas em cada *site*.

O processo de rastreamento do Googlebot® começa com uma lista de URLs de páginas *Web*, gerada a partir de processos anteriores de rastreamento e aumentada com dados dos *Sitemaps* fornecidos por *webmasters*. Conforme o Googlebot® visita cada um desses sites, ele detecta os links (SRC e HREF) de cada página e os inclui na sua lista de páginas a serem rastreadas. Novos sites, alterações em sites existentes e links inativos serão detectados e usados para atualizar o índice do Google.

Como visto anteriormente o Googlebot® funciona no mesmo princípio que foi apresentado na arquitetura básica de um *crawler*.

2.4.2 Crawler4J

Crawler4j é um *crawler* de código aberto escrito em linguagem Java que fornece uma interface simples para rastrear a *Web*. É possível configurar um *crawler multi-thread*. Projetado de forma muito eficiente tem a capacidade de rastrear domínios de maneira rápida

(por exemplo, tem sido capaz de rastrear 200 páginas do Wikipédia® por segundo). No entanto, esta prática vai contra as políticas de rastreamento, pois impõe enorme carga nos servidores. Desde a versão 1.3, por padrão o coletor espera pelo menos 200 milésimos de segundo entre as solicitações. Este parâmetro pode ser ajustado com o "setPolitenessDelay".

Deve-se implementar uma classe de controlador que especifica as sementes do rastreamento, a pasta em que os dados de rastreamento devem ser armazenados e número de *threads* concorrente.

A partir da versão 2.5, é possível limitar a profundidade de rastreamento. Por exemplo, suponha que tem-se uma página semente "A", que liga a "B", que liga a "C", que liga a "D". Então, tem-se a estrutura de *links* deste modo:

A -> B -> C -> D

Uma vez que, "A" é uma página semente, ele terá uma profundidade de 0(zero). "B" terá profundidade de 1(um) e assim por diante. Pode-se definir um limite para a profundidade de páginas que o coletor irá se aprofundar. Por exemplo, ao se definir este limite a 2, ele não irá rastrear a página "D". Este limite pode ser especificado no arquivo *crawler4j.properties*.

2.4.3 Nutch

Apache Nutch é um rastreador *Web open source* escrito em Java. Decorrente do *Apache Lucene*, ele agora se baseia no *Apache Solr* acrescentando funcionalidades específicas para *Web*, como um rastreador, um banco de dados gráfico e suporte para análise, que são tratados pelo *Apache Tika* para HTML.

Pode ser executado em uma única máquina, mas ganha mais desempenho de execução em um *cluster*. Ao usá-lo, pode-se encontrar *hyperlinks* em páginas *Web* de uma forma automatizada. Também podendo reduzir os lotes de trabalhos de manutenção, por exemplo, verificação de links quebrados, e criação de uma cópia de todas as páginas visitadas.

O Nutch é composto por:

- O banco de dados de rastreamento, ou *crawl*db. Este contém informações sobre cada URL conhecida pelo Nutch, incluindo se ela foi obtida, e, em caso afirmativo, quando;
- A base de dados link, ou *link*db. Esta contém a lista de ligações conhecidas a cada URL, incluindo tanto a fonte URL e o texto âncora do link;
- Um conjunto de segmentos. Cada segmento é um conjunto de URLs que são buscadas como uma unidade. Segmentos são diretórios com os seguintes subdiretórios:

- um *crawl_generate* nomes de um conjunto de URLs a serem buscados;
- um *crawl_fetch* contém o status de buscar cada URL;
- um *conteúdo* contém o conteúdo bruto recuperado de cada URL;
- um *parse_text* contém o texto analisado de cada URL;
- um *parse_data* contém outlinks e metadados analisado de cada URL;
- um *crawl_parse* contém os URLs outlink, usado para atualizar o crawl db.

2.4.4 Retriever

O Retriever (SANTOS, 2009) foi desenvolvido na linguagem de programação Java, para ser um coletor com uma arquitetura flexível para a coleta de dados que possa ser operacionalizada como parte de outros sistemas. Fazendo uso de sua API, que contém as funções e procedimentos necessários para a construção de um *software* de coleta. Isso dependerá dos valores e das operações utilizadas, assim o coletor poderá desempenhar um comportamento diferente. Assim conseguindo construir um coletor que atenda as necessidades de busca do usuário, através de diferentes requisitos na coleta.

O coletor pode ter as políticas de polidez, para que seja feito um bom uso de toda a infraestrutura disponível para a coleta de dados. Disponibiliza a utilização de filtros para que as coletas possam ser focadas em um determinado tema, uma determinada página, um formato de documento específico e definir os URLs sementes para começar a coleta. Consegue compreender os diferentes tipos de documentos que possam vir a aparecer em seu caminho na *web*.

Para este trabalho, o Retriever foi escolhido por ser capaz de ser executado em um *cluster*, para que as máquinas do sistema consigam trabalhar, de forma ordenada, em uma mesma coleta.

3. PROCESSAMENTO DISTRIBUÍDO

Neste capítulo é apresentado o contexto geral sobre processamento distribuído, incluindo uma introdução, as características dos sistemas distribuídos e as arquiteturas distribuídas. Por fim, são apresentados alguns *frameworks* que auxiliam a distribuição de sistemas.

3.1 INTRODUÇÃO

Atualmente, tanto em áreas científicas quanto em ambientes corporativos, existe uma demanda crescente em processamento e armazenamento de dados, o que torna cada vez maior a necessidade de recursos computacionais que possam realizar tarefas de forma rápida e confiável (RESENDE, 2010).

Esta necessidade de aumento por processar vasta quantidade de informação requer que as máquinas possuam um *hardware* com alto poder de processamento. Gordon Moore, o primeiro a expressar uma preocupação sobre a capacidade do *hardware*, afirmou que a cada 18 meses o nível de transistores reunidos em um mesmo circuito integrado dobraria, ou seja, o nível de complexidade aumentaria nos dispositivos, enquanto seu custo não sofreria aumento (MOORE, 1965). Assim esse documento ficou conhecido como a Lei de Moore.

Tal lei que se segue nos tempos de hoje, a estimativa proposta por ele na década de 70, que prevê a duplicação do número de transistores comportados em uma pastilha a cada 18 meses (MOLIK, 2006).

Para Dantas (2005), o desenvolvimento de processadores nos dias de hoje esbarra nos limites da física, da matéria. Isso se reflete na dificuldade da contínua diminuição dos transistores para serem inseridos em um processador, no aquecimento do mesmo em virtude das altas frequências de processamento que podem ser atingidas, nas configurações de barramento e no armazenamento e transmissão de informações na memória. Desse modo,

considerando a tecnologia atual, os componentes devem ter um aumento no seu tamanho físico para que se possa continuar evoluindo nos níveis de processamento.

Porém, é válido ressaltar, que hoje o nível de processamento foi fragmentado devido ao avanço dos componentes e a criação de ambientes distribuídos, assim fornecendo um alto desempenho quando comparados a um sistema centralizado. Segundo Tanenbaum (2007), através de um sistema de computação distribuída é possível utilizar um conjunto de computadores independentes, que na visão do usuário comportam-se como um sistema único e coerente.

Segundo Deitel, Deitel e Choffnes (2005), sistemas de multiprocessamento englobam qualquer sistema que contenha mais de um processador. Os autores ainda afirmam que Flynn desenvolveu um primeiro esquema para classificar computadores com configurações de paralelismo. Como mostra a Tabela 1, a taxonomia consiste em 4 categorias baseadas nos dois tipos de fluxos (fluxo de instrução e fluxo de dados) usados nos processadores. Por fluxo entende-se como uma sequência de *bytes* executada em um processador.

Segundo Dantas (2005) a taxonomia de Flynn, apesar de ser antiga, ainda é a mais utilizada. Na sua descrição a arquitetura que possui múltiplos processadores é classificada como MIMD, em que cada processador executa instruções de forma independente dos demais. Os sistemas com múltiplos processadores possuem duas ou mais CPU's trabalhando em conjunto, que se comunicam através de um meio físico. Esta característica possibilita a execução paralela de instruções, oferecendo um ganho de desempenho quando comparado a o modelo mais tradicional, ou seja, o processamento sequencial (TANENBAUM, 2007).

	<i>Single Instruction</i>	<i>Multiple Instruction</i>
<i>Single Data</i>	SISD	MISD
<i>Multiple Data</i>	SIMD	MIMD

Tabela 1 - Taxonomia de Flynn.

Fonte: adaptado de (DANTAS, 2005)

Pode-se dividir esta arquitetura em três categorias: Multicomputador, Multiprocessador e Sistemas Distribuídos. Um sistema distribuído é semelhante a um multicomputador, contudo, podem ser interligados por uma rede normal, uma conexão *ethernet* por exemplo. Os componentes desta estrutura geralmente são sistemas completos, com todos os periféricos, o trabalho é dividido entre todos, assim os processos são executados de forma distribuída.

Baseado em rede de computadores Coulouris, Dollimore, Kindberg (2007) apresentam alguns exemplos de sistemas distribuído usados diariamente como a internet, as intranets e redes em dispositivos móveis. Citam ainda os processadores multinúcleos que dividem os processos por seus núcleos.

3.2 CARACTERÍSTICAS DOS SISTEMAS DISTRIBUÍDOS

A computação distribuída consiste em um conjunto de máquinas conectadas através de uma rede de comunicação, atuando logicamente como um sistema único. Neste sistema cada máquina (nó) possui seus recursos próprios, como: memória principal e sistema operacional. A comunicação é feita por uma rede através de protocolos específicos (FOSTER; KESSELMAN; TUECKE, 2001). Além disso, o sistema distribuído precisa ser transparente, tolerante a falhas, escalável e conseguir um alto poder de processamento.

De acordo com Coulouris, Dollimore, Kindberg (2007), Deitel, Deitel e Choffnes (2005), Silberschatz, Galvin e Gagne (2010), pode-se citar as seguintes características básicas em relação aos sistemas distribuídos:

- **Confiabilidade:** o objetivo central do desenvolvimento de sistemas distribuídos é conseguir torná-los mais confiáveis tanto quanto os sistemas centralizados. Para alcançar uma disponibilidade plena é necessário aumentar o número de cópias de certas peças-chaves e algumas funções importantes, ou seja, aumentar a redundância. A disponibilização de maiores quantidade de dados poderá implicar em uma inconsistência alta, isto devido ao fato que é difícil controlar todas as cópias. O elevado número de cópias pode trazer mais um problema, a queda de desempenho devido ao número das repetições. Outro aspecto relacionado à confiabilidade é a tolerância a falhas. Onde o sistema não deve parar sua execução, escondendo as falhas ocorridas, ou seja, caso um nó venha a falhar outro nó qualquer podem assumir e continuar a operar;
- **Escalabilidade:** soluções desenvolvidas para sistemas distribuídos de pequeno porte não funcionam da mesma maneira para os sistemas distribuídos de grande porte. Como exemplo, normalmente em sistemas distribuídos de pequeno porte, como uma tabela de e-mails poderia ser armazenado em apenas um servidor. Caso o sistema dependa deste servidor para sua execução, uma vez que todas as aplicações que necessitam utilizar essa tabela só funcionam se o servidor estiver

funcionando. Este tipo de dependência não deve ocorrer em um sistema de grande porte;

- Desempenho: é necessário que a taxa de execução do sistema seja alta. A aplicação executada em um ambiente distribuído não pode ser mais lenta que a executada em um ambiente centralizado. Para conseguir acompanhar o desempenho do sistema, métricas como o tempo de resposta e o número de *Jobs* por hora (*throughput*) são utilizadas, assim, estas variáveis ficam dependentes ao desempenho da rede;
- Transparência: característica muito relevante dos sistemas distribuídos. Para chegar ao nível de transparência têm-se duas necessidades, a primeira a) o usuário interage com uma interface, mas não sabe onde ela está executando ou armazenando seus arquivos e b) o sistema distribuído pode aumentar ou diminuir, sem que a aplicação deixe de executar;
- Comunicação: um sistema distribuído pode proporcionar a execução de funções não levando em consideração a geografia dos usuários. Um grupo de pessoas em regiões, até mesmo países distantes podem usufruir da mesma aplicação.

3.3 ARQUITETURAS DISTRIBUÍDAS

Existem algumas diferenças entre os modelos existentes de arquiteturas distribuídas. Particularidades da maneira como os elementos são compartilhados e como será a comunicação. Deve haver uma maneira de se manter a comunicação entre os nós que compõem o sistema distribuído. Sendo assim, a comunicação é primordial e representa uma característica forte de um sistema distribuído para que este consiga fazer o sistema trabalhar em cooperação e harmonia. Segundo Dantas (2005), as arquiteturas de sistemas distribuídos podem ser:

- Multiprocessador: um sistema computacional fortemente acoplado no qual duas ou mais CPUs compartilham acesso total a mesma memória RAM. Os processos se comunicam através da memória, o que cada processo escreve pode ser lido pelos demais. Devido a memória ser compartilhada técnicas de sincronização como semáforos e monitores tornam-se necessárias. Independente do hardware de conexão, a característica principal é o compartilhamento da memória RAM;
- Multicomputador: um sistema onde cada CPU possui uma memória local. Caracteriza-se por acoplamento fraco, pois a comunicação entre processos é feita

pela troca de mensagens entre os processos em execução. Os nós de uma estrutura de multicomputador geralmente possuem uma CPU, RAM, uma interface de rede e talvez um disco rígido para a paginação. Vale a pena salientar que o meio físico de comunicação entre os nós do multicomputador é de alta velocidade, e geralmente trabalham paralelamente;

- **Sistemas Distribuídos:** um sistema semelhante ao de multicomputadores, porém podem ser interligados pelas redes (Ethernet). Geralmente nesta estrutura, cada nó é um sistema completo, com todos os periféricos, em que os processos são executados de forma distribuída entre os nós;
- **Clusters:** um sistema que agrega computadores, não necessariamente de forma dedicada, para execução de uma aplicação específica. A escalabilidade é um fator diferencial, pois a capacidade computacional cresce a medida que mais recursos estiverem disponíveis. Agregando novas máquinas com configurações de baixo desempenho é possível aumentar o processamento, assim como a relação custo-desempenho;
- **Grids:** um sistema agregando máquinas localizadas em diferentes regiões e até mesmo países. Voltando toda a potencialidade dos recursos e serviços disponíveis na rede para o processamento de tarefas.

A computação distribuída pode ser uma solução para se conseguir um alto desempenho, em que computadores fisicamente separados são utilizados para executar processos que demandem um alto nível de processamento. Desse modo, possibilita-se que computadores, com uma menor capacidade de processamento, alcancem um alto nível adequado de desempenho, podendo assim configurar-se em uma alternativa à supercomputadores trabalhando isoladamente.

3.3.1 Cluster

Um *cluster* é um agregado de computadores criado para conseguir se alcançar aumento de desempenho ou disponibilidade. Pode ser classificado como um sistema composto por N computadores fracamente acoplados com níveis de processamento iguais ou diferentes. Esta estrutura comporta-se como um único sistema, trabalhando de forma transparente ao usuário.

Tanenbaum e Steen (2007), afirmam que o hardware que venha a ser inserido no sistema se equipare aos já existentes, conectados por uma rede local de alta velocidade.

Sobretudo, os nós deverão executar o mesmo sistema operacional. Esta definição pode ser considerada tradicional, visto que um *cluster* pode ter outras características. Outra definição apresentada por Stallings (2010), afirma que um *cluster* consiste de um conjunto de computadores completos, com todos os seus componentes, conectados entre si. Trabalhando juntos como um único recurso computacional, criando a ilusão ao usuário de estar operando uma única máquina.

Deitel, Deitel e Choffnes (2005) relatam alguns exemplos de *clusters*, que podem ser implementados em Sistemas Operacionais como o Linux® e o Windows®. Entre os mais conhecidos está o Beowulf, um *cluster* de alto desempenho construído pela primeira vez em 1994 pela NASA rodando Linux®. A Figura 5 mostra a organização de um *cluster* Beowulf típico que pode conter até 700 nodos sendo interligados por uma rede de alta velocidade. O *cluster* possui um nó mestre, para distribuir a carga de trabalho, controlar o acesso ao *cluster* e os recursos compartilhados. Os nós escravos devem ter o mesmo processador, memória e espaço de disco, de modo que todos consigam concluir o trabalho ao mesmo tempo.

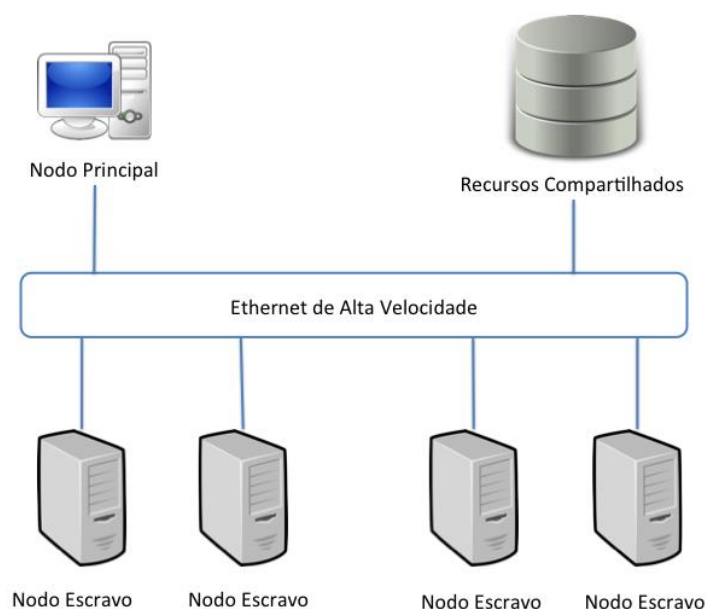


Figura 5 - Cluster Beowulf típico

Fonte: Adaptada de Deitel, Deitel e Choffnes (2005)

Os autores ainda trazem um modelo de *cluster* com o Sistema Windows®, que pode ser usado para construção de *cluster* de alta disponibilidade e também um *cluster* de balanceamento de carga. O *cluster* de alta disponibilidade pode ter até 8 nodos, todos os nodos podem compartilhar de um dispositivo de armazenamento como mostra a Figura 6, ou

também cada nodo pode ter um armazenamento local, para manter uma cópia dos dados gerados por outro(s) nó(s) como mostra a Figura 7.

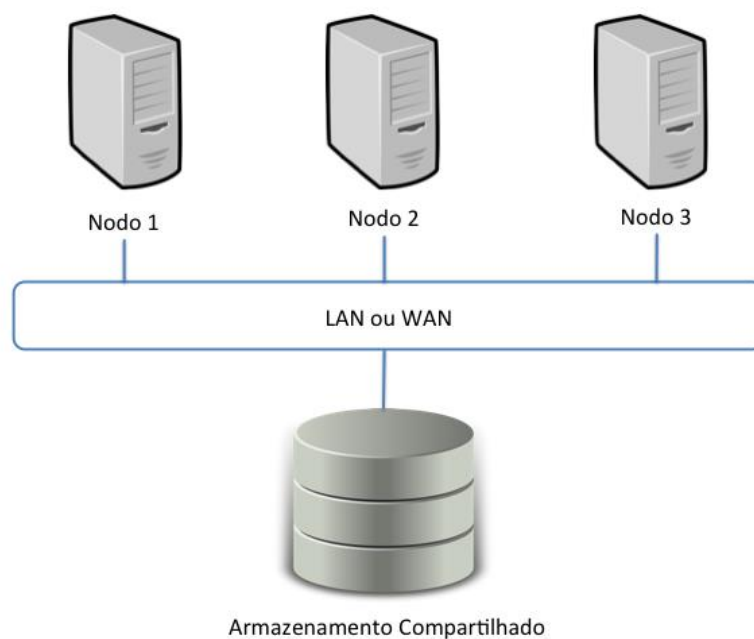


Figura 6 - Cluster de alta disponibilidade com armazenamento compartilhado

Fonte: Adaptada de Deitel, Deitel e Choffnes (2005)

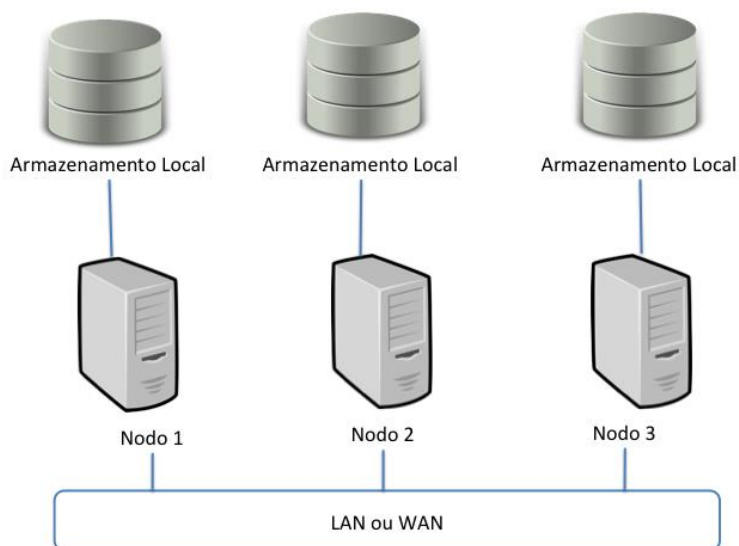


Figura 7 - Cluster de alta disponibilidade com armazenamento local

Fonte: Adaptada de Deitel, Deitel e Choffnes (2005)

O *cluster* de alta disponibilidade pode ser conectado por uma LAN ou WAN. No modo de armazenamento compartilhado, somente o nodo que está em execução terá o controle do dispositivo de armazenamento. Em caso de falha o nodo reserva assume o controle do armazenamento. Contudo, se o armazenamento compartilhado falhar o *cluster*

inteiro deixa de executar. Já no modo de armazenamento local, é possível mais de um nodo estar em execução, caso ocorra uma falha qualquer outro nodo pode assumir sua execução.

Com o *cluster* de balanceamento de carga não requer armazenamento compartilhado, pois cada nodo executa seu serviço independentemente. Caso um nodo venha a falhar, outros nodos são capazes de manipular as requisições dos usuários. Pode-se adicionar ou eliminar nodos de maneira fácil, sem haver complicações com os outros nodos. O número máximo de nodos é de 32, usualmente interconectados por uma conexão de alta velocidade, como exemplificado na Figura 8.

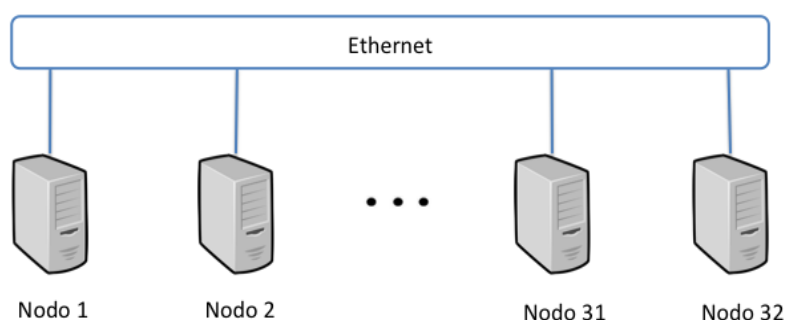


Figura 8 - Cluster de balanceamento de carga

Fonte: Adaptado de Deitel, Deitel e Choffnes (2005)

Existem algumas métricas apontadas por Dantas (2005) para a classificação de um *cluster* conforme apresentado na Figura 9:

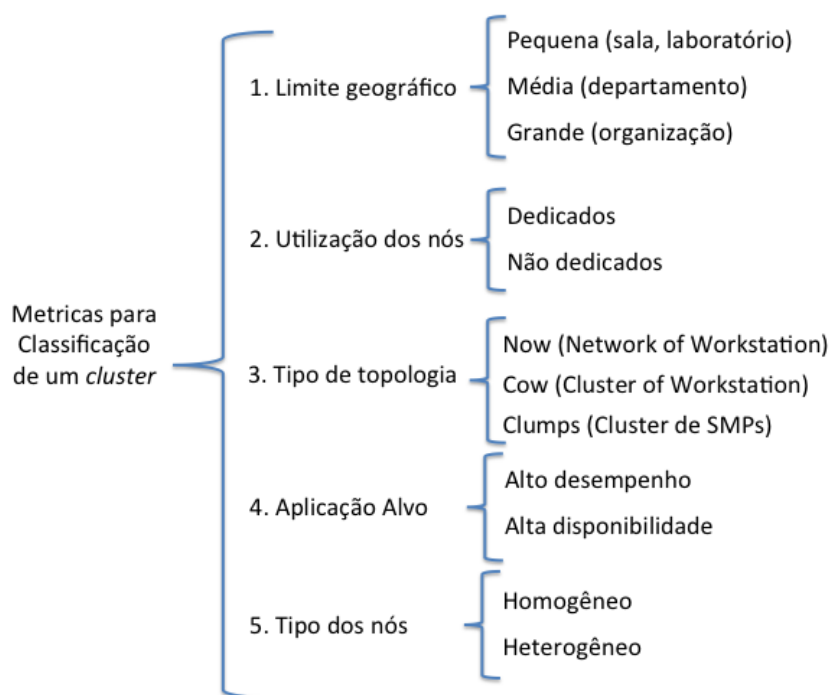


Figura 9 - Classificação de Clusters

Fonte: Adaptada de Dantas (2005)

Pode-se concluir que clusters são compostos por nós, com a mesma organização, conectados entre si através de uma rede. O principal objetivo é o ganho de desempenho/disponibilidade. Esta estrutura se caracteriza pela transparência ao usuário, que usa como se estivesse utilizando um único sistema. Mesmo permitindo a existência de nós heterogêneos, normalmente são utilizados nós homogêneos, com mesmos hardwares e sistemas operacionais.

É válido ressaltar que uma característica forte de um *cluster*, é conseguir um alto desempenho com um baixo custo. Como afirmavam Baker, Buyya e Hyde (1999), um *cluster* de alto desempenho consegue competir com os melhores supercomputadores que as grandes companhias, como IBM ou a SGI, podem oferecer. Uma organização pode montar um *cluster* deste tipo por cerca de \$50000, frente a construção de um supercomputador que pode chegar a cerca de \$200000.

3.3.2 Grid

A computação em grade (*grid*) baseia-se na dispersão das funcionalidades de cada servidor. Onde cada servidor terá uma função em específico. Onde cada máquina irá fornecer um determinado serviço, diferente das outras demais. Pode-se apontar como seu principal objetivo é garantir a disponibilidade e confiabilidade dos serviços prestados. Caso um problema de hardware ou software venha a ocorrer em determinada máquina, o sistema como todo não deixará de funcionar, apenas o serviço disposto pela máquina ficará temporariamente indisponível.

Um *grid* computacional é uma infraestrutura onde se envolve o uso integrado e colaborativo de computadores, redes, bancos de dados e instrumentos científicos que venham à pertencer e serem geridos por várias organizações (BUYYA; VENUGOPAL, 2005). Ainda é possível afirmar que "... cada sistema pode cair sob um domínio administrativo diferente, e podem ser muito diferente no que tange a hardware, software e tecnologia de rede empregada" (TANENBAUM; STEEN, 2007).

O que motiva a utilização de *grids* computacionais é a possibilidade de utilizar recursos de diversos domínios, sendo que estes recursos se encontram ociosos em grande parte do tempo. Os recursos de um *grid* não são dedicados, tornando possível sua utilização mesmo que em determinado momento estejam dedicados a outros fins, como computadores de empresas, universidades, de uso pessoal, entre outros. Estes recursos podem ser sensores, dados, computadores, câmeras, etc.

Outra característica importante que permite um *grid* ser altamente escalável é possibilidade de utilizar recursos heterogêneos, ou seja, um *grid* pode ser composto por hardwares, sistemas operacionais, redes distintas (TANENBAUM; STEEN, 2007). Ainda é válido afirmar que esta estrutura computacional também pode possuir clusters em sua composição.

3.4 FRAMEWORKS

Existem *frameworks* que auxiliam no desenvolvimento de um sistema distribuído principalmente por prover funções como a divisão do trabalho, a recuperação de erros, e a comunicação entre os nós. A seguir são discutidos deles, GridGain® e Hadoop®.

3.4.1 GridGain

O projeto teve início em 2005 e atualmente conta com duas versões: GridGain *Community Edition* que é a versão *open source*, e a versão GridGain *Enterprise Edition*, que é a versão comercial do software. Neste trabalho foi utilizada a versão *Community Edition*.

GridGain é um *framework* de código aberto utilizado na computação em *grid*, construído utilizando a linguagem de programação Java, que permite desenvolvedores desta mesma linguagem, melhorar o desempenho do processamento de aplicações dividindo e paralelizando a carga de trabalho (RESENDE, 2010). Segundo Ivanov (2012), o GridGain é um *middleware* desenvolvido em JAVA que permite o desenvolvimento de aplicações distribuídas de alto desempenho. Um *middleware* permite à um sistema distribuído manter-se uniforme mesmo operando em hardwares e sistemas operacionais distintos (TANENBAUM, 2010).

O GridGain trabalha com o conceito de tarefas (*task*) e problemas (*jobs*), sendo que uma *task* é dividida em diversos pequenos *jobs*. O envio dos *jobs* e o balanceamento de carga fica a cargo do GridGain. Assim, a aplicação apenas deve definir uma *task* e como ela vai ser dividida dando origem aos *jobs*. Após a divisão os *jobs* são enviados aos nós, onde o GridGain também se encontra, que compõem o *grid* para serem executados.

Visando o processamento de um grande montante de dados e análises. Construído de baixo para cima, baseado em uma plataforma de processamento na memória escalando de um servidor para várias máquinas. Utilizando dados antigos para fazer lotes de processamento quando estiver off-line, também, analisando em tempo real e processando os dados no

momento que estiver online.

Normalmente utilizado em aplicações de longo prazo de armazenamento. Eficaz em transmissão e transação de dados, que não podem ser diretamente armazenados e necessitam ser processados em tempo real e também no processamento de dados já coletados.

3.4.2 Hadoop

O Hadoop é um *framework* de código aberto para salvar e executar aplicações distribuídas que processam uma grande quantidade de dados. Algumas distinções do Hadoop para outras aplicações distribuídas:

- Acessível: pode ser executado em grandes conjuntos de máquinas ou em serviços de computação nas nuvens;
- Robusto: por se destinar a executar em uma comunidade de hardware, foi arquitetado com suposições de avarias frequentes de software;
- Escalável: escalado linearmente para lidar com volume de dados maiores, adicionando mais nós ao cluster;
- Simples: permite que usuários escrevam rapidamente códigos paralelos com eficiência.

Projetado para processamento *off-line* e análise de dados em grande escala, não funciona para armazenar e ler aleatoriamente alguns registros. É melhor utilizado para armazenar uma vez e ler várias vezes os dados.

Segundo Xu et al. (2011), alguns dos benefícios do uso do HDFS como um servidor de carga intermediária para armazenar dados a serem carregados para um Teradata® EDW, são:

- Significativamente mais espaço pode ser facilmente adicionado para os dados a serem carregados, simplesmente pela adição de mais nós;
- Uma vez que os dados são escritos para HDFS, mesmo ainda não sendo carregados para Teradata® EDW, não é mais necessário para as fontes de dados mantê-los, conforme recomendado pela maioria das ferramentas de extração, transformação e carga dos dados;
- Programas de MapReduce podem ser usados para transformar e adicionar estruturas para representar dados não estruturados ou semiestruturados;
- Uma vez que um arquivo é distribuído em HDFS, este pode ser carregado mais rapidamente em paralelo com Teradata® EDW.

4. SISTEMA PROPOSTO

Neste capítulo será descrito o sistema proposto neste trabalho, dividindo a apresentação em duas etapas. A primeira etapa refere-se à visão lógica em que se detalha a interação entre os diferentes componentes do sistema. A segunda etapa apresenta a visão física, descrevendo os componentes tecnológicos, bem como, a justificativa de utilização dos mesmos.

4.1 VISÃO LÓGICA

Neste trabalho é proposta a construção de um sistema de coleta (*crawler*) com uma abordagem distribuída, fazendo com que mais de um computador (nó) em um *cluster* realize o trabalho de coletar documentos. Com isto a busca e a obtenção do conteúdo (documentos) tendem a ser mais rápidas, uma vez que cada nó se ocupa de um conjunto de URLs. A Figura 10 abaixo exemplifica o sistema de coleta e os passos que permitem a sua completa execução.

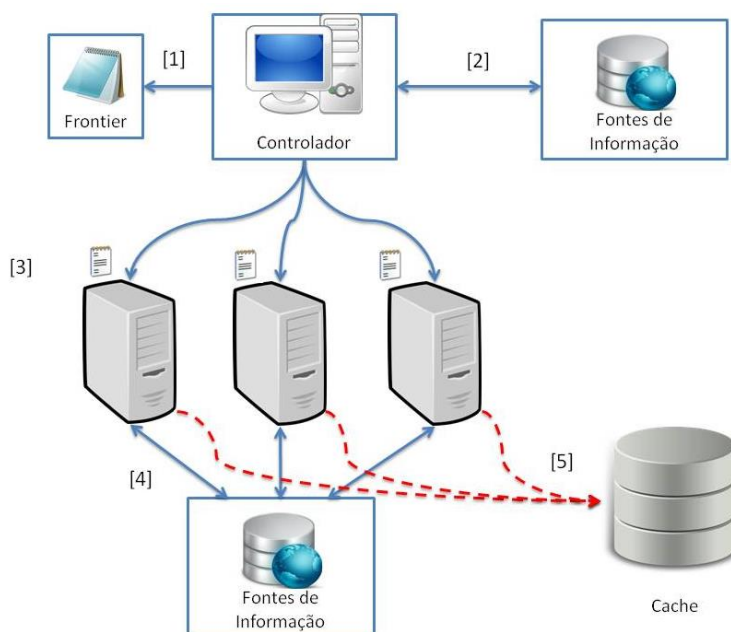


Figura 10 - Visão lógica do sistema de coleta distribuída de dados

De modo geral o sistema possui um controlador que inicia o processo de coleta e é responsável por organizar a distribuição do serviço, que ocorre em algumas etapas:

Obtenção dos URL sementes do *Frontier* [1]: o controlador obtêm os URLs sementes definidas pelo usuário do *Frontier* para começar o serviço de coleta. A Tabela 2 mostra exemplos de alguns URLs utilizados. Elas são acessadas somente pelo controlador nesta 1ª etapa, que possui como objetivo nestas sementes outros URLs visando assim, incrementar o *Frontier*.

URLs sementes
http://pt.wikipedia.org/wiki/Brasil
http://www.jornaldaciencia.org.br/index2.jsp?id=2625
http://www.terra.com.br/portal

Tabela 2 - Exemplo de sementes utilizadas

Análise das sementes (URLs) [2]: enquanto o controlador analisa as sementes, ele preenche uma lista com os novos URLs encontradas nas páginas sementes. Neste ponto, o controlador apenas irá adicionar os novos URLs e não fará nenhum tipo de coleta dos dados em si.

Distribuição da lista de URLs [3]: com o *frontier* carregado com novos URLs, o controlador irá dividi-lo entre os nós participantes do *cluster*. Cada nó será responsável por coletar uma parte do *frontier*.

Coleta dos Dados [4]: cada nó é responsável por iniciar uma conexão via algum protocolo aos URLs delegados a ele. Deste modo, o nó é responsável por coletar o conteúdo disponível em um URL específico, bem como, manter este conteúdo armazenado localmente.

Serviço de Indexação [5]: após completar a coleta das informações de uma determinada lista, os dados devem ser enviados a um serviço de indexação para que o conteúdo coletado possa ser recuperado posteriormente. Este serviço não faz parte do trabalho, apesar ser representado na visão lógica.

4.2 VISÃO FÍSICA

A seguir serão detalhados os componentes tecnológicos e como estes estão interconectados, visando oferecer uma visão física do sistema proposto. Para a viabilização do sistema dois componentes principais foram utilizados. Na base do desenvolvimento foi

utilizada a biblioteca chamada *Retriever* (SANTOS, 2009), por ser de sua fácil utilização e por apresentar uma arquitetura flexível.

Para a realização da coleta distribuída utilizou-se o *framework* GridGain™ *Community Edition* v4.5 (IVANOV, 2012), responsável pela comunicação e divisão da carga de trabalho entre os nós do *cluster*. Este *framework* foi escolhido pela sua facilidade de instalação e execução. A Figura 11 apresenta a visão física do sistema proposto.

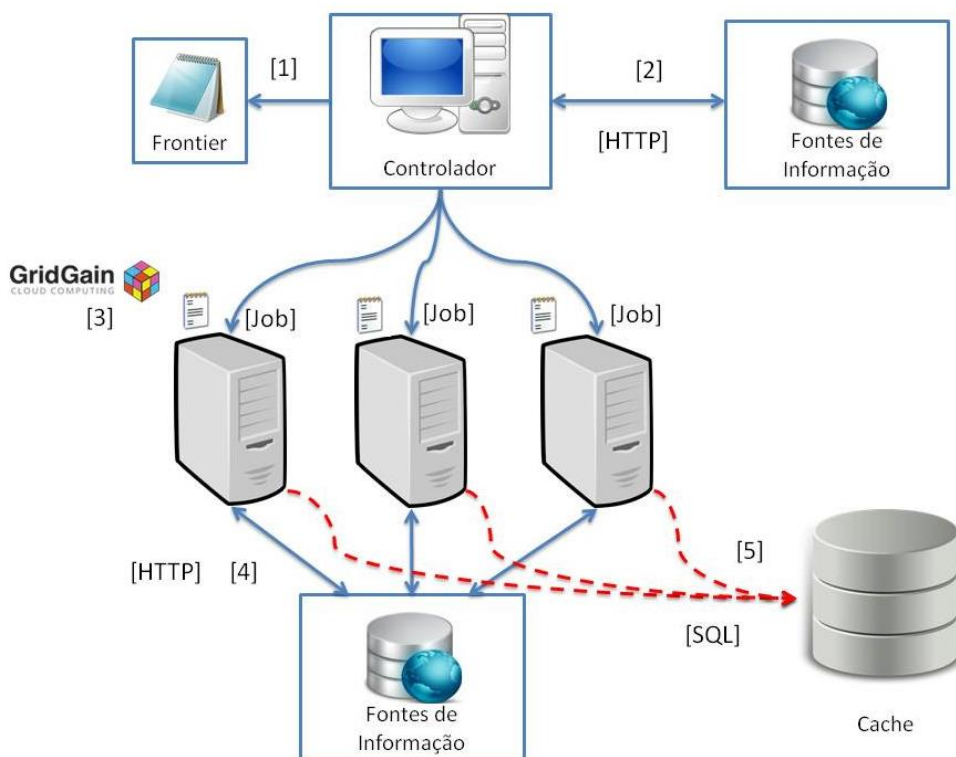


Figura 11 - Visão física do sistema de coleta distribuída de dados

Como mencionado anteriormente o sistema possui um controlador que inicia o processo de coleta e é responsável por organizar a distribuição do serviço, que ocorre em algumas etapas:

Obtenção dos URL sementes do *Frontier* [1]: inicialmente o controlador obtêm as lista de URLs sementes que foram indicadas por determinado usuário e representam a necessidade inicial de coleta. Tradicionalmente, uma semente é representada por um URL que indica um determinado *Site Web*. Contudo, o *Frontier* pode ser carregado com um conjunto fixo de endereços que representam arquivos em determinados formatos, por exemplo, pdf ou doc, que em geral são mais difíceis de serem analisados.

Análise das sementes (URLs) [2]: de posse da lista de URLs vindas do *Frontier*, o controlador estabelece uma conexão HTTP considerando um endereço da lista e analisa o

conteúdo deste visando identificar novos endereços que possam ser adicionados a lista. Vale mencionar que a análise é em geral efetuada em arquivos no formato HTML, pois estes possuem *links* para outros URLs. Estes novos *links*, caso apontem para arquivos HTML, são então enviados ao *Frontier* que representa a base do processo de coleta.

Distribuição da Lista de URLs [3]: com o uso do *framework* GridGain é realizada a divisão da lista de URLs presente no controlador em partes visando distribuir a carga de trabalho do coletor. Cada parte da lista é enviada a um nó sendo tratada como um *job*. O número de *jobs* deve ser definido de modo que exista um balanceamento de carga e que a comunicação entre os nós seja minimizado. Criar *jobs* dividindo a quantidade de URLs em função da quantidade de nós pode não ser adequado, visto que estes talvez não tenham configurações semelhantes. Por outro lado, atribuir cada URL à um *job* irá promover uma comunicação excessiva entre os nós e o controlador..

Coleta dos Dados [4]: cada nó estabelece uma conexão HTTP com os URLs que lhe foram delegados realizando a coleta de todos os dados encontrados nestes URLs. Como mencionado, o conteúdo que será coletado depende do formato do arquivo que o URL se refere. Por exemplo, um arquivo HTML talvez tenha referências a vários outros arquivos, tais como imagens ou vídeos.

Serviço de Indexação [5]: após completar a coleta do conteúdo referente a uma determinada lista os dados podem ser enviados para um serviço de indexação ou um cache, visando assim a integração com outros sistemas. Contudo, a disponibilização desses dados voltados à integração está além do escopo do trabalho.

4.2.1 Detalhamento do Protótipo

Para um melhor entendimento da visão física esta seção detalha o protótipo de coleta distribuída de dados através de um diagrama de sequência (Figura 12).

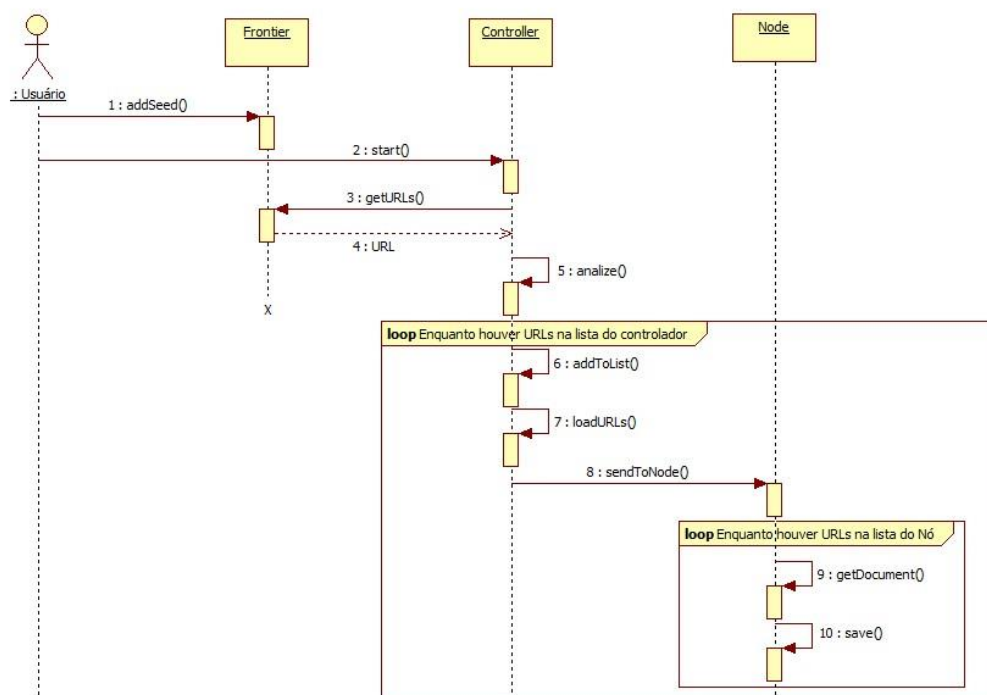


Figura 12 – Diagrama de sequência do protótipo do coletor

O processo inicia com a definição das sementes (URLs) que serão inspecionadas na coleta. Isto é realizado através do método *addSeed()* constante na classe *Frontier*. Por simplificação, as sementes são embutidas diretamente no código em uma classe principal (representada pelo usuário) que cria e preenche o *Frontier*.

Como próximo passo, na classe principal é inicializado o controlador, classe *Control*, que obtêm todas os URLs presentes no *Frontier* através do método *getURLs()*. A partir disso o controlador analisa cada um dos URLs obtidos, ou seja, realiza um *parser* através do método *analize()* objetivando identificar novos URLs presentes no documento web.

Cada URL encontrado nos documentos analisados a partir das sementes são então adicionados a uma lista de novos URLs pelo no próprio controlador através do método *addToList()*. Ao final do processo a lista conterà todos os URLs que necessitam ser coletados pelos nós.

Como a tarefa de coletar um único URL é geral realizada rapidamente, o controlador irá atribuir uma parte da lista a um determinado nó. Para tal, o controlador cria uma tarefa que será depois enviada ao *job* com um conjunto de URLs que são carregadas através do método *loadURLs()*. O protótipo utiliza uma abordagem simplificada para a divisão dos URLs entre os nós simplesmente separando o conjunto pelo número de nós ativos no *cluster* multiplicado

por dois. Então cada parte da lista é enviada para um determinado nó através do método *sendToNode()* da classe *Node*.

De posse da lista o nó inicia o *download* dos documentos utilizando o método *getDocument()*. Após a coleta o nó armazena/salva o documento em um cache através do método *save()*, que neste caso, é representado pelo próprio sistema de arquivos do sistema operacional. Esse processo se repete até que não existam mais URLs na lista para serem coletadas.

5. APRESENTAÇÃO DOS RESULTADOS

Este capítulo visa promover um melhor entendimento das potencialidades do processo de coleta considerando alguns cenários, seja este executado a partir de um único ponto ou através de múltiplos pontos.

5.1 INTRODUÇÃO

Utilizando um coletor com uma arquitetura distribuída, foi possível conseguir uma coleta em tempo hábil e mais abrangente. Considerando que cada nó do sistema é responsável por uma coleta, tem-se então mais de uma coleta ocorrendo ao mesmo tempo, com o mesmo objetivo em comum. Para tal, o protótipo segue a afirmação de Dantas (2005) e Deitel (2005), em que é possível, a partir de computadores de pequeno porte, atingir um alto desempenho computacional.

Desse modo, são propostos cenários visando analisar o desempenho do sistema em suas coletas. Assim é possível analisar melhor se o coletor consegue manter seu desempenho coletando dados em diferentes cenários. As avaliações foram realizadas comparando o número de documentos coletados em um determinado espaço de tempo ou o tempo necessário para coletar um número fixo de documentos, considerando sempre um coletor simples contra um coletor distribuído.

A próxima seção promove maiores detalhes sobre a configuração dos cenários de coleta.

5.2 DESCRIÇÃO DOS CENÁRIOS

Visando promover uma análise do comportamento do coletor, foram propostos cenários que demonstrassem o seu uso. A seleção dos cenários foi realizada visando as

funcionalidades de coleta perante diferentes desafios, tanto restrito a domínios web como documentos específicos. Os cenários escolhidos foram:

- **Coleta Livre de Dados:** O cenário consistiu no uso do coletor para a coleta de informação sobre documentos presentes na Web. A coleta é livre, pois partiu de alguns URLs (sementes) manualmente definidos e seguiu suas ligações sem a utilização de qualquer tipo de filtro, por exemplo, limitando a varredura no domínio da semente;
- **Coleta de Notícias:** O coletor ficou restrito a um site que contém notícias. Sua principal função era localizar e coletar as notícias encontradas no site. O coletor não deveria, em momento algum, procurar notícias externas ao domínio especificado;
- **Coleta de Documentos PDFs:** Foi fornecido um conjunto de URLs que representavam documentos em formato PDF. O coletor deveria identificar e coletar esses documentos.

Ressalta-se que para este trabalho a largura de banda de aproximadamente 40MB foi provida pelo Campus UFSC/Araranguá e que a mesma é de uso comum para todas as atividades do Campus. Nesse sentido, as medições da execução do processo de coleta nos diferentes cenários podem ser impactadas uma vez que, dependendo do horário, a carga da rede pode ser menor ou maior.

Para a realização dos cenários foi utilizado um *cluster* com no máximo de 3 nós com as seguintes configurações:

- **Nó controlador:** Notebook DELL, modelo Vostro 3500. Sistema Windows 7, Processador Intel Core I5 2.67GHz, 6GB de memória RAM;
- **Nó 2:** Desktop HP, modelo Compac 6005C, Sistema Windows 7, processador AMD Phenom II 3,00 GHz, 8GB de memória RAM;
- **Nó 3:** Desktop HP, modelo Compac 6005C, Sistema Windows 7, processador AMD Phenom II 3,00 GHz, 8GB de memória RAM.

Também foi utilizando um Switch DLink DGS-3100-48, com capacidade de 116Gbps, taxa de envio de pacotes de 64 bytes em 86.3Mpps, tamanho da tabela de endereços MAC de 8.000 e buffer do pacote de 1.5MB.

Vale mencionar que os tempos apresentados na discussão dos cenários a seguir representam a média de cinco execuções em momentos diferentes. Isto foi realizado visando

minimizar a influência da disponibilidade de banda da rede em relação a quantidade de nós utilizados em uma determinada medição.

5.2.1 Coleta Livre de Dados

Neste cenário, o coletor foi configurado para caminhar livremente pela Web e armazenar os dados disponíveis pelas páginas analisadas. O processo foi iniciado com um conjunto pré-selecionado de sementes e seguiu as ligações entre páginas HTML.

Uma das sementes utilizadas no processo de coleta foi a página do site Terra® (<http://www.terra.com.br/>). Essa semente foi escolhida, pois contém um número elevado de links que levam a novas páginas HTML, facilitando a dispersão do coletor por diferentes espaços da Web. Outras sementes utilizadas foram a página do Globo.com®¹. Por se tratarem de páginas de notícias, as sementes mostram um amplo conteúdo.

O tempo da coleta foi limitado em 10 minutos, pois o protótipo não foi implementado com as características de um coletor polido. A falta de um gerenciamento eficiente da carga de acesso à determinado site por um coletor por resultar no bloqueio a requisições vindas de IP em específico. A seguir a Tabela 3 apresenta a comparação da utilização do sistema com diferentes quantidades de nodos.

Número de nós	Tempo (minutos)	Quantidade de Arquivos	Tamanho
1	10:00	3525	110 MB
2	10:00	4763	127MB
3	10:00	5186	135 MB

Tabela 3 - Resultados da coleta sem restrições

A execução do cenário demonstra que é possível realizar a coleta de dados a partir de um coletor distribuído. Contudo, a abordagem utilizada neste cenário se mostra pouco útil, uma vez que não limita o formato a ser coletado e utiliza recurso computacional para coletar arquivos que possuem pouca utilidade.

O gráfico abaixo (Gráfico 1) representa a quantidade de arquivos coletados pelas diferentes configurações do sistema neste cenário. Fica evidente que o aumento do número de arquivos coletados é mais expressivo quando o sistema é configurado com 2 nós.

¹ <http://www.globo.com/>

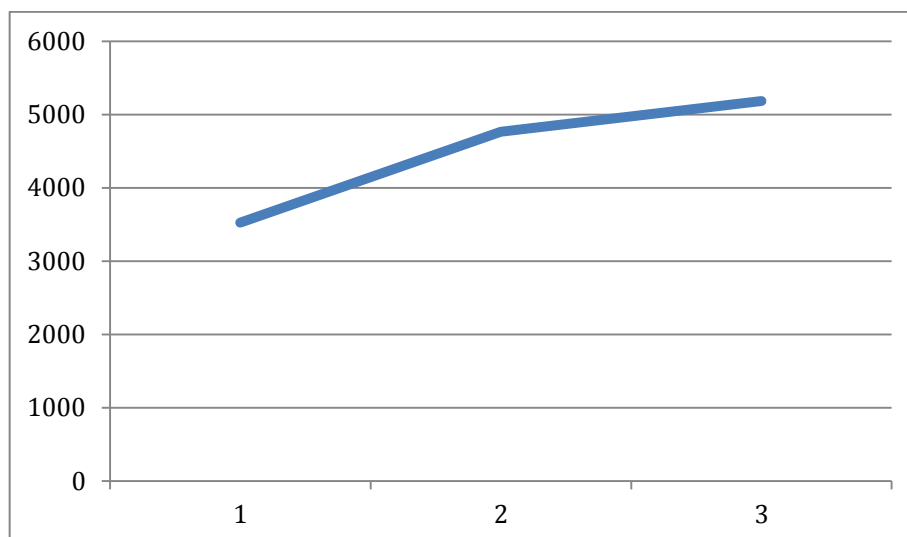


Gráfico 1 - Gráfico da quantidade de arquivos por nós ativos no sistema

A proposta do cenário foi a de coletar livremente documentos da Web dada uma janela de tempo, armazenando dados acerca da execução do processo de coleta. Nesse contexto, a configuração do coletor se mostrou simples. A implementação atual mantém as informações do *frontier* em memória. Contudo, seria mais adequado a construção de um novo *frontier*, que mantenha em disco os URLs já coletados visando otimizar o trabalho em uma nova coleta quando o processo fosse reiniciado.

Com o tempo de coleta fixado percebeu-se, ainda que em diferentes proporções, o aumento no número de arquivos coletados, bem como no tamanho desses arquivos. Isso se deve a alguns fatores, entre eles: (a) os nós não possuem a mesma configuração; (b) o coletor não faz qualquer distinção dos arquivos coletados. Nesse sentido, um nó pode coletar mais páginas com conteúdo relevante enquanto outro pode coletar arquivos de configuração ou mesmo elementos gráficos de uma determinada página; e (c) o conteúdo coletado por um nó pode também ser coletado por outro visto que a implementação atual não possui esse tipo de controle.

5.2.2 Coleta de Notícias

Este cenário foca na coleta restrita, ou seja, cada nó considera somente URLs indicadas em sua lista para realizar a coleta sem analisar os *links* internos de cada página. Para tal, foi utilizado o site de notícias do Jornal da Ciência², que mantém todo o seu acervo de notícias já publicadas. O coletor se limitou à página em que a notícia se encontra, obtendo

² <http://www.jornaldaciencia.org.br>.

somente o texto da notícia sem procurar possíveis URLs que levassem para outras notícias. Assim, para que se consiga coletar as notícias utilizou-se o URL base < <http://www.jornaldaciencia.org.br/Detalhe.jsp?id=N>> onde *N* se refere ao identificador numérico da notícia que atualmente chega a mais de 87000.

Como a coleta foi focada a um determinado URL limitou-se a quantidade de notícias em 10000 visando a comparação com a alternância no número de nós. O número foi limitado visando evitar a sobrecarga do site do Jornal da Ciência uma vez que o coletor não possui a característica de polidez. A Tabela 4 apresenta os resultados.

Número de Nós	Quantidade de Arquivos	Tamanho (MB)	Tempo (minutos)
1	10.000	116	17:02
2	10.000	116	13:23
3	10.000	116	9:13

Tabela 4 - Resultado da Coleta do Jornal da Ciência

Além, da configuração inicial com os 10000 URLs referentes ao site de notícias, também foi configurado um filtro para evitar que o coletor ultrapassasse os limites do jornal da Ciência.

O acréscimo de cada nó promove, na configuração utilizada, um decréscimo linear no tempo de processamento. Basta verificar que a diferença de tempo considerando 2 nós em relação a somente 1 nó é de 3:49 minutos, e considerando 3 nós em relação a 2 nós é de 4:10 minutos. É importante ressaltar que os tempos de coleta foram prejudicados na implementação atual porque o sistema não realiza corretamente o balanceamento de carga, sendo o número total de URLs igualmente distribuído entre os nós sem considerar a disponibilidade para o processamento do mesmo.

O gráfico a seguir (Gráfico 2) demonstra a diminuição do tempo de coleta de dados conforme o sistema tem mais nós ativos. Pode-se verificar a diferença de tempo de aproximadamente 8 minutos entre a configuração baseada em 1 nó e a com 3 nós.

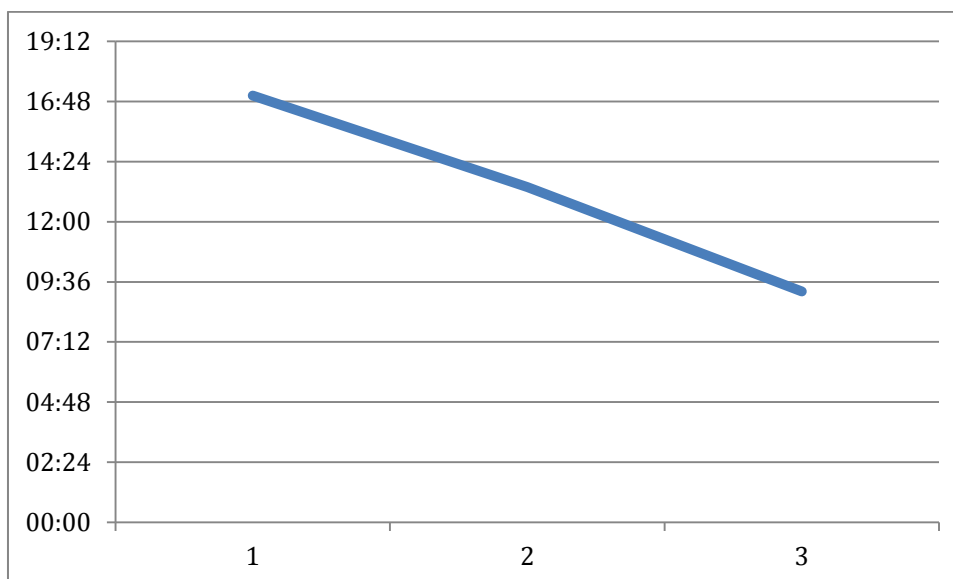


Gráfico 2 - Gráfico entre o tempo de coleta de dados e os nós ativos no sistema

5.2.3 Coleta de documentos PDF

Neste cenário o coletor foi limitado apenas a identificar e coleta de apenas arquivos no formato PDF. A configuração do ambiente difere dos cenários anteriores devido aos problemas de sobrecarga identificados nos cenários anteriores. Além disso, a tentativa de utilização de servidores públicos que continham arquivos PDF se mostrou inadequada, uma vez que os tempos se alteravam muito de uma simulação para outra.

Sendo assim, configurou-se uma estrutura contendo três servidores *web* que continham arquivos PDFs e outros três computadores que formavam o *cluster* em que o coletor foi executado. Tanto os servidores *web* quanto os nós do *cluster* possuíam a mesma configuração, sendo, Desktop HP, modelo Compac 6005C, Sistema Windows 7, processador AMD Phenom II 3,00 GHz com 8GB de memória RAM. Cada servidor *web* continha um total de 500 arquivos em que cada um possuía 5MB, totalizando 1500 arquivos e 7.5GB de dados que deveriam ser coletados.

A Figura 13 exemplifica a solução adotada em que se pode verificar a composição dos servidores *web* e do *cluster*. Como mencionado anteriormente o desempenho do coletor quando executando de maneira distribuída é fortemente impactado em função do *job* que cada nó irá executar. No contexto do trabalho se este *job* for definido como a coleta de um único URL haverá muita comunicação entre os nós e o controlador, reduzindo em muito o desempenho global do sistema.

Neste sentido, foi necessário um balanceamento de carga para definir quantos *pdfs* seriam coletados por cada *job*, sendo definido a seguir:

$$NA / (NN \times NS)$$

onde, *NA* se refere ao número total de arquivo a serem coletados, *NN* o número de nós participantes no *cluster* e *NS* o número de servidores.

Por exemplo, analisando a quantidade de arquivos total (1500) e utilizando 3 servidores web e 3 nós no *cluster* cada *job* foi constituído por 166 arquivos. A diferença de 6 arquivos foi atribuída ao último *job*.

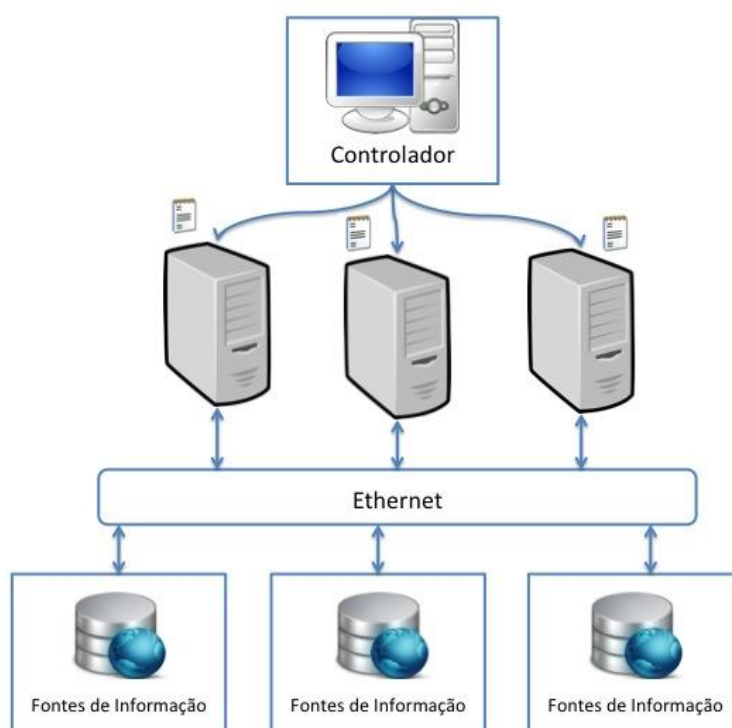


Figura 13 - Modelo de coleta de documentos PDFs

Os resultados da utilização do coletor considerando esta nova configuração são apresentados na Tabela 5. Pode-se verificar que a adição de novos nós contribui na melhoria do desempenho. A redução no tempo com a adição de um nó é de 65%. Isto se justifica devido a capacidade dos servidores web atenderem múltiplas requisições vindas de diferentes nós aumentando assim o paralelismo da solução. Com a adição de um terceiro nó, ainda que reduza o tempo total, promove um decréscimo de 32% em relação a configuração com dois nós.

Número de nós	Quantidade de Arquivos	Tamanho (GB)	Tempo (milisegundos)
1	1500	7,5	285.000
2	1500	7,5	100.000
3	1500	7,5	68.000

Tabela 5 - Resultado da coleta de PDFs com servidores web e diferentes números de nós

Analisando a queda menos acentuada no tempo a medida que se adicionam mais nós, como mostra o gráfico abaixo (Gráfico 3), é de se esperar que exista um ponto de inflexão, em que a solução distribuída irá se tornar pior do que executada por um único computador. Por exemplo, levando-se em conta a equação exposta anteriormente e considerando 20 nós em um *cluster* cada job conteria somente 25 URLs. O limite, em função da quantidade de arquivos e de servidores web neste cenário, seria de 500 nós, em que cada um destes nós coletariam apenas um documento. Isto geraria tráfego desnecessário na rede, ou seja, uma comunicação excessiva entre os nós e o controlador reduzindo o desempenho do coletor.

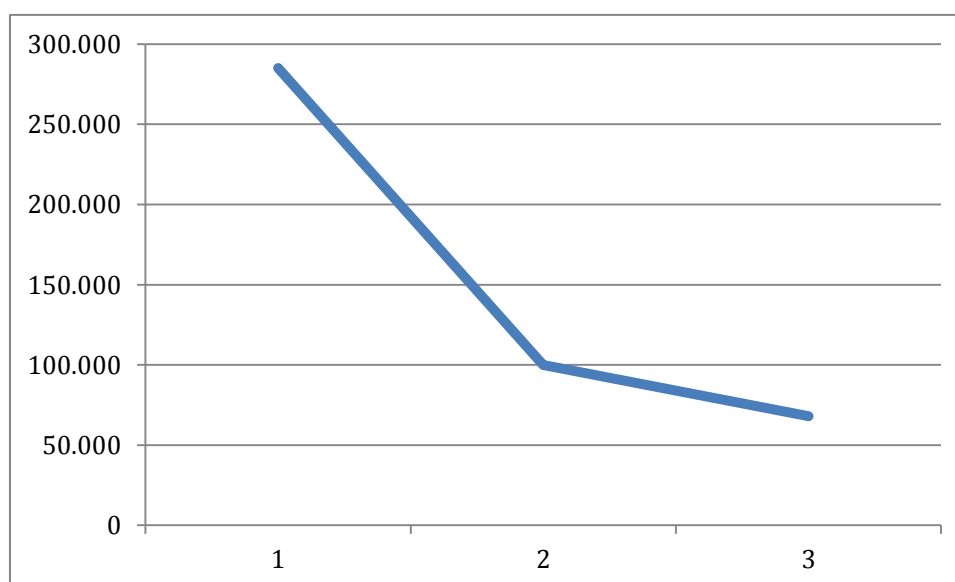


Gráfico 3 - Gráfico entre o tempo de coleta de dados e os nós ativos no sistema

6. CONSIDERAÇÕES FINAIS

O objetivo geral desse trabalho foi o desenvolvimento de um sistema considerando uma arquitetura distribuída para a coleta de dados, capaz de atender diferentes cenários de coleta. Este objetivo foi possível através da pesquisa bibliográfica nas áreas relacionadas ao trabalho, coleta de dados e computação distribuída, bem como, pelo estudo de coletores existentes e *frameworks* que possibilitassem a distribuição do processo de coleta.

Utilizando os conceitos obtidos por meio da revisão bibliográfica, foi proposto um sistema de coleta de informação com uma arquitetura distribuída. Para melhor compreensão do sistema proposto o mesmo foi dividido em duas partes representando a visão lógica e visão física.

A visão lógica promove uma explicação das funcionalidades gerais do sistema envolvendo o próprio processo de coleta de dados, a obtenção de novos URLs, a análise de URLs e a distribuição do processamento. Com a visão física definiu-se o funcionamento dos componentes do sistema, explicando todos os passos, desde a adição de URLs (sementes), divisão destas pelo sistema, passando pela coleta das informações/dados e por fim, preservando-os em meio físico. Tais informações podem posteriormente ser indexadas e disponibilizadas através de um sistema de recuperação de informação.

O protótipo desenvolvido atendeu as expectativas gerando resultados satisfatórios quando comparada a utilização de um único computador frente a utilização de um *cluster*. Os cenários propostos para a utilização do coletor permitiram analisar o comportamento do mesmo considerando diferentes demandas de coleta.

De modo geral, durante a fase de teste do protótipo nos diferentes cenários notou-se que a falta de uma estratégia adequada no balanceamento de carga para a distribuição das tarefas entre os nodos do *cluster* afetou o desempenho final do sistema.

Durante as análises foi possível verificar ainda que existe um número limite de nós a serem utilizados no *cluster*. O incremento de mais nós irá, em um determinado ponto, produzir resultados piores em relação a utilização de um único computador que possua as mesmas configurações dos nós.

No decorrer do desenvolvimento do protótipo, houve a percepção de melhorias que poderiam ocorrer no sistema. Contudo, a desenvolvimento dessas melhorias tornariam o trabalho extenso.

Entre as melhorias pode-se destacar a flexibilização da arquitetura, dividindo o protótipo em módulos de serviços. Cada parte do coletor poderia ser disponibilizado como um serviço web, como por exemplo, o preenchimento da lista de URLs (*frontier*) e a divisão do trabalho de coleta dos URLs.

Outro ponto importante a ser melhorado no protótipo seria o método de divisão da tarefa de coleta entre os nós que participam do processamento. No estudo realizado neste trabalho os cenários tiveram estratégias diferentes. Neste sentido, seria adequado permitir que o método de divisão dos *jobs* fosse definido de maneira dinâmica, podendo ser inclusive disponibilizado na forma de um serviço web.

Considerando a diversidade de assuntos, torna-se relevante que os coletores tenham a capacidade de analisar a semântica do conteúdo, visando deste modo um resultado mais qualificado considerando determinado domínio do interesse. Exemplo disto é o coletor proposto por Wang, Zhu e Li (2013), que consiste em coletar os comentários referentes à determinados produtos no site da Amazon™, e após a coleta, é realizada uma classificação do produto indicando se este teve ou não a aprovação de seus compradores.

Por último, mas sem exaurir as possibilidades, a coleta focada usando ontologias vem se tornando a base de muitos estudos para um aprimoramento dos coletores. Como exemplo, podem-se citar as pesquisas de Zheng, Kang e Kim (2008), Jannach, Shchekotykhin e Friedrich (2009), Yang (2010) e Du e Hai (2013).

REFERÊNCIAS

- AHMADI-ABKENARI, Fatemeh; SELAMAT, Ali. An architecture for a focused trend parallel Web crawler with the application of clickstream analysis. **Information Sciences**, v.184, n. 1. p. 266-281. 2011.
- BAKER, Mark; BUYYA, Rajkumar; HYDE, Dan. Cluster Computing: A High-Performance Contender. **IEEE Computer Society**, p.79-83, jul. 1999.
- BATSAKIS, Sotiris; PETRAKIS, Euripides G.m.; MILIOS, Evangelos. Improving the performance of focused web crawlers. **Data & Knowledge Engineering**, v.68, n.10, p. 1001-1013. 2009.
- BOVO, Alessandro Botelho. **Um modelo de descoberta de conhecimento inerente à evolução temporal dos relacionamentos entre elementos textuais**. 2011. 155 f. Dissertação (Doutorado) - Universidade Federal de Santa Catarina, Florianópolis, 2011.
- BRANDMAN, O. et al. Crawler-Friendly Web Servers. **Acm Sigmetrics Performance Evaluation Review**, New York, v.28, n.2, p. 9-14. 2000.
- BUYYA, Rajkumar; VENUGOPAL, Srikumar. A Gentle Introduction to Grid Computing and Technologies. **Csi Communications**, n. 1, p.9-19, 2005.
- CASTILLO, Carlos. **Effective Web Crawling**. 2004. 179 f. Tese (PH.D) - University Of Chile, Chile, 2004.
- CHO, J.; GARCIA-MOLINA, H. **Parallel crawlers**. Proceedings of the 11th International Conference On World Wide Web, Honolulu, p. 124-135, 2002.
- CHO, J.; GARCIA-MOLINA, H. Synchronizing a database to improve freshness. **Acm Sigmod Record**, v.29, n.2, p. 117-128, 2000.
- CHO, J.; GARCIA-MOLINA, H.; PAGE, L. **Efficient crawling through URL ordering**. Proceedings of the Seventh International World Wide Web Conference, Brisbane, p. 124-135, 1998.
- COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. **Sistemas distribuídos: conceitos e projetos**. 4. ed. Proto Alegre: Bookman, 2007. 792 p.

CRAWLER4J. **Open Source Web Crawler for Java**. Disponível em <<http://code.google.com/p/crawler4j/>> acesso em : 14/06/2012.

DANTAS, Mario A. R. **Computação distribuída de alto desempenho: redes, clusters e grids computacionais**. Rio de Janeiro: Axcel Books, 2005. 278 p.

DEITEL, Paul J.; DEITEL, Harvey M. **Ajax, rich internet applications e desenvolvimento web para programadores**; Tradução Célia Taniwaki e Daniel Viera. São Paulo: Ed Pearson Prentice Hall, 2008.

DEITEL, Paul J.; DEITEL, Harvey M.; CHOFFNES, D. R.. **Sistemas Operacionais**. 3ª edição. São Paulo: Pearson Prentice Hall, 2005.

DILIGENTI, Michelangelo et al. **Design of a Crawler with Bounded Bandwidth**. Proceedings of the 13th international World Wide Web conference, New York, p. 292-293, 2004.

DU, Yajun; HAI, Yufeng. Semantic ranking of web pages based on formal concept analysis. **Journal Of Systems And Software**, Seoul, v. 86, n.1, p. 187-197, 2013.

FOSTER, Ian; KESSELMAN, Carl; TUECKE, Steven. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. **International Journal of High Performance Computing Applications**, p. 200-222. 2001.

GOMES, D.; SILVA, M.J. The Viúva Negra crawler: an experience report. **Software: practice And Experience**, New York, p. 161-188. fev. 2008.

GOOGLEBOT. **Ferramentas do Google para Web Masters**. Disponível em <<http://support.google.com/webmasters/bin/answer.py?hl=pt-BR&answer=182072>>. Acesso em: 11/06/2012.

GREENGRASS, E. **Information Retrieval: A Survey**. 2000. 224 p. Disponível em <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.1855>>. Acesso em: 11/06/2012.

HABIB, Ahsan; ABRAMS, Marc. **Analysis of Sources of Latency in Downloading Web Pages**. Proceedings of the Webnet World Conference on the WWW and Internet 2000, Chesapeake, p.227-232, 2000.

HADOOP. Welcome to Apache™ Hadoop®. Disponível em <<http://hadoop.apache.org/>>. Acesso em: 12/06/2012.

HAFRI, Y.; DJERABA, C. **High performance crawling system**. Proceedings of the 6th Acm Sigm International Workshop On Multimedia Information Retrieval, New York, p. 299-306. 2004.

HE, Bin et al. Accessing the deep web. **Magazine Communications of the ACM**, New York, v. 50, n. 5, p.94-101, 2007.

HEYDON, A.; NAJORK, M. **Mercator**: A scalable, extensible Web crawler. World Wide

Web, Toronto, p. 219-229. dez. 1999.

HIMMA, K. The concept of information overload: A preliminary step in understanding the nature of a harmful information-related condition. **Ethics and Information Technology**, v.9, n.4, p.259-272. 2007.

IVANOV, Nikita. Real Time Big Data Processing with GridGain, 2012. Disponível em: <<http://www.gridgain.com/book/book.html>>. Acesso em: 20 agosto 2012.

JANNACH, Dietmar; SHCHEKOTYKHIN, Kostyantyn; FRIEDRICH, Gerhard. Automated ontology instantiation from tabular web sources — The All Right system. **Web Semantics: Science, Services And Agents On The World Wide Web**, v.7, n.3, p. 136-153, 2009.

KOBAYASHI, M. e K. TAKEDA. Information retrieval on the web. **ACM Computing Surveys**, v.32, n.2, p.144-173, 2000.

KORFHAGE, Robert R. **Information storage and retrieval**. New York: Wiley Computer Publishing, 1997. 368 p.

KRITIKOPOULOS, Apostolos; SIDERI, Martha; STROGGILOS, Kostantinos. **CrawlWave: A Distributed Crawler**. Proceedings of the 3rd Hellenic Conference On Artificial Intelligence, Pythagorion, 2004.

KUMAR, Mukesh; VIG, Renu. Learnable Focused Meta Crawling Through Web. Proceedings of the International Conference on Communication, Computing & Security, Rourkela: Elsevier, v.6, p. 606-611, 2012.

LIU, B. **Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data**. 2. ed. Springer, 2009. 532 p.

LYMAN, P. **How Much Information? USA**: University of California at Berkeley, 2000.

MODESTO, Marco et. al. Um novo Retrato da Web brasileira. **Anais do XXV Congresso da Sociedade Brasileira de Computação**. São Leopoldo, p. 2005-2017, 2005.

MOLIK, E. Establishing Moore's Law. **Annals of the History of Computing**, v. 28, n. 3, p. 62-75, 2006.

MOORE, Gordon E. Cramming more components onto integrated circuit. **Electronics Magazine**. v. 38, n. 38, 1965.

PANAGIOTIS, Ipeirotis, G.; GRAVANO, Luis; SAHAMI, Mehran. Probe, count, and classify: categorizing hidden-web databases. **ACM Sigmod**, Santa Barbara, v. 30, n. 2, p. 1-2, 2001.

PATTAL, Malik Muhammad Imran; YUAN, Li; JIANQIU, Zeng. **Web 3.0: A real personal Web! More opportunities & more threats**. Proceedings of the Third International Conference on Next Generation Mobile Applications, Services and Technologies, v. 3, 2009, p. 125 - 128.

PRIMO, Alex. O aspecto relacional das interações na Web 2.0. E- Compós (Brasília), v. 9, p. 1-21, 2007.

RESENDE, Cristiano Marcelo. **Ambiente Grid utilizando Software Livre**. 2010. 62f. Monografia (Especialização Lato Sensu) - Universidade Federal de Lavras, Lavras,2010.

SANTOS, Lucas Nazário Dos. **Uma arquitetura flexível para a coleta de dados**. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação). São José: Universidade Do Vale Do Itajaí, 2009. 91 p.

SILBERSCHATZ, Abraham.; GALVIN, Peter B.; GAGNE, Greg. **Fundamentos de sistemas operacionais**. 8. ed. Rio de Janeiro: LTC, 2010. xvii,515p.

STALLINGS, William. **Arquitetura e organização de computadores**. 8. ed. SãoPaulo (SP): Pearson, 2010. xiv, 624p.

TANENBAUM, Andrew S. **Sistemas operacionais modernos**. 3. ed. Rio de Janeiro (RJ): Prentice-Hall do Brasil, 2010. xiii, 653p.

TANENBAUM, Andrew S.; STEEN, Maarten van. **Sistemas distribuídos: princípios e paradigma**. 2. ed. São Paulo: Pearson Prentice Hall, 2007. 402 p.

WAN, Yuan; TONG, Hengqing. URL Assignment Algorithm of Crawler in Distributed System Based on Hash. **International Conference On Networking, Sensing And Control, 2008. Icnc 2008. Ieee**, Sanya, p.1632-1635, 2008.

WANG, Dingding; ZHU, Shenghuo; LI, Tao. SumView: A Web-based engine for summarizing product reviews and customer opinions. **Expert Systems With Applications**, Miami, v. 40, n. 1, p. 27-33, 2013.

WANG, Ye Diana; ZAHADAT, Nima. **Teaching Web Development in the Web 2.0 Era**. Proceedings of the 10th ACM Conference on SIG-Information Technology Education, 2009, p. 80-86.

XU, Yu. KOSTAMMA, Pekka. QI, Yan. **A Hadoop Based Distributed Loading Approach to Parallel Data Warehouses**. New York, Proceedings of the ACM SIGMOD International Conference on Management of data, 2011. 9 p.

YANG, Sheng-yuan. OntoCrawler: A focused crawler with ontology-supported website models for information agents. **Expert Systems With Applications**, v. 37, n.7, p. 5381-5389, 2010.

ZHENG, Hai-tao; KANG, Bo-yeong; KIM, Hong-gee. An ontology-based approach to learnable focused crawling. **Information Sciences**, v. 178, n. 23, p. 4512-4522, 2008.