UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

IMPLEMENTAÇÃO DE UM *CRAWLER* INCREMENTAL DISTRIBUÍDO: UM SISTEMA DE BUSCA NA *WEB*

TIAGO ROBERTO FISCHER

BLUMENAU
2006

2006/1-39

TIAGO ROBERTO FISCHER

IMPLEMENTAÇÃO DE UM *CRAWLER* INCREMENTAL DISTRIBUÍDO: UM SISTEMA DE BUSCA NA *WEB*

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Alexander Roberto Valdameri - Orientador

BLUMENAU
2006

2006/1-39

IMPLEMENTAÇÃO DE UM *CRAWLER* INCREMENTAL DISTRIBUÍDO: UM SISTEMA DE BUSCA NA WEB

Por

TIAGO ROBERTO FISCHER

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Alexander Roberto Valdameri – FURB
Membro:	Prof. Nome do professor, Titulação – FURB
Membro:	Prof. Nome do professor, Titulação – FURB

Blumenau, dia de mês de ano

Dedico este trabalho à toda a minha família, que sempre esteve presente e me apoiou em todos os momentos de minha vida. E a todos os amigos, especialmente aqueles que me ajudaram direta ou indiretamente na realização deste.

A complete life may be one ending in so full identification with the non-self that there is no self to die.

Bernard Berenson

RESUMO

Atualmente, a internet, principalmente a web, se tornou bastante popular. Como conseqüência deste fato, a demanda por sistemas de busca cresceu substancialmente e continua chamando a atenção dos usuários. Levando em consideração a importância que esta ferramenta tem no atual estágio de desenvolvimento do mundo cibernético, este trabalho tem como objetivo explorar todas as facetas que compõem um sistema de busca de páginas na web. Para isso, foi criado e implementado um crawler incremental distribuído para busca de páginas brasileiras (domínio .br) utilizando o MYSQL. O crawler tem a função de buscar páginas, organizá-las e indexá-las para retornar uma consulta em um tempo satisfatório.

Palavras-chave: *Crawler*. Banco de dados distribuídos. Sistemas de busca. Sistemas distribuídos.

ABSTRACT

Nowadays, the Internet, mainly the Web, has become popular. As a consequence, the demand for search systems has grown substantially and keeps calling attention from users. Taking in account the importance that this tool has in the actual development stage in the cybernetic world, this work has as a purpose, to explore all facets which takes part a search system on Web pages. To achieve this, a distributed incremental crawler has been created and implemented for searching on Brazilian Web pages (.br domain) using MySQL. The crawler's function is to search pages, organize and index them to return a query in a satisfactory time.

Key-words: Crawler. Distributed database. Search systems. Web search. Distributed systems.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

O uso da internet como meio de disseminação da informação nos últimos tempos se tornou muito difundido. Atualmente é comum ouvir o comentário: “procure na internet, através de servidor de busca... alguma coisa relacionada irá encontrar”. Isto é o reflexo da evolução dos sistemas de comunicação, armazenamento e recuperação da informação.

A rápida evolução e o crescimento da internet têm dificultado cada vez mais para que os mecanismos de busca retornem o resultado esperado pelo usuário. Por esse motivo, os sistemas evoluem para cada vez mais lidar com tal crescimento. São exemplos de sistemas de busca: Yahoo! (YAHOO!, 2006), Altavista (ALTAVISTA, 2006), Google (GOOGLE, 2006), entre muitos outros.

O coração de qualquer sistema de busca é o *crawler*. O *crawler* é um robô que busca, armazena e mantém atualizadas páginas na internet, as quais são indexadas em tempo real ou posteriormente. O termo *web search* se refere à interface que aparece para o usuário. Tal interface provê a comunicação para efetuar a busca de páginas que é feita através de requisições ao *crawler*. Um *crawler* não trabalha de forma isolada. Ele é um processo automatizado que se comunica juntamente com outros *crawlers* existentes no sistema, sendo executado em paralelo, armazenando e processando em vários computadores para poder trabalhar com o grande número de informações que a internet dispõe (ARASU, 2001, p. 1-4).

Tendo em vista o crescimento da importância dos sistemas de busca na *web*, desenvolve-se neste trabalho a implementação de um *crawler* incremental distribuído que busque e armazene páginas brasileiras da *web* (com domínio .br) de forma indexada, disponibilizando comunicação entre todos os *crawlers* e a criação de uma interface (*web search*) que efetue a busca de informações entre os *crawlers* de forma transparente.

1.1 OBJETIVOS DO TRABALHO

O objetivo principal deste trabalho é implementar um *crawler* incremental distribuído para busca de páginas brasileiras (.br) na *web*.

Os objetivos específicos do trabalho são:

a) possibilitar a busca de páginas na *web*;

b) possibilitar o armazenamento das páginas localmente de forma indexada;

c) prover comunicação entre todos os *crawlers*;

d) manter as páginas coletadas atualizadas (*up-to-date*);

e) disponibilizar uma interface para busca básica de conteúdos nos *crawlers* de forma transparente (*web search*).

1.2 ESTRUTURA DO TRABALHO

O trabalho está organizado em quatro capítulos. O primeiro capítulo apresentou a introdução e os objetivos do trabalho.

No segundo capítulo, inicialmente, é fornecida uma breve explanação sobre o sistema de busca, seus componentes e funcionalidades essenciais, bem como sobre sistemas de armazenamento e recuperação e sistemas distribuídos.

O terceiro capítulo apresenta a especificação do sistema, o seu desenvolvimento e implementação dos diferentes módulos.

No quarto capítulo são apresentadas as conclusões do trabalho, limitações e possíveis extensões para o mesmo.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a fundamentação bibliográfica envolvendo assuntos como sistemas de busca, sistemas de armazenamento e recuperação e sistemas distribuídos.

2.1 SISTEMAS DE BUSCA

O sistema de busca é comumente conhecido como *web search*, pois é a interface que aparece para o usuário efetuar a sua busca, mas existem outros elementos envolvidos neste processo, sendo um deles o *crawler*.

A Figura 1 ilustra os elementos básicos de um sistema de busca (ARASU, 2001, p. 3) os quais são descritos a seguir:

a) *crawler*: responsável por buscar, armazenar e manter atualizadas as páginas *web*;

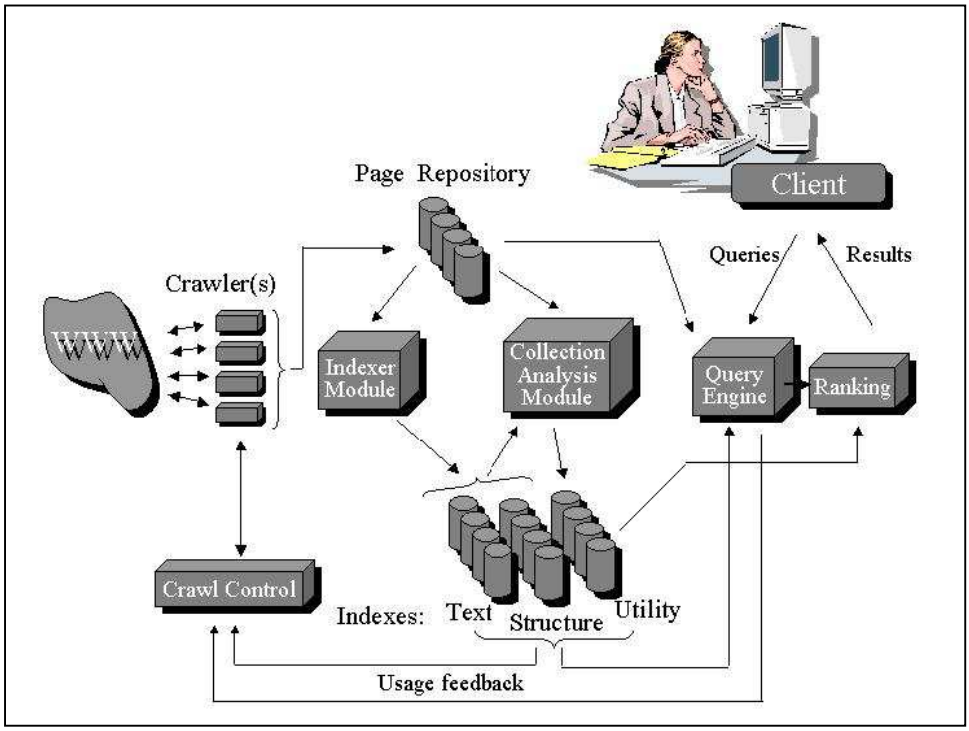
b) *url server*: responsável por organizar e distribuir para os vários *crawlers* as *Universal Resource Locator* (URL);

c) *indexer*: indexa as páginas para facilitar a sua busca;

d) *query engine*: processa a requisição do usuário;

e) *ranking*: define qual página irá aparecer em primeiro nos resultados;

f) *web search*: interface para o usuário.



Fonte: Arasu (2001, p. 2).

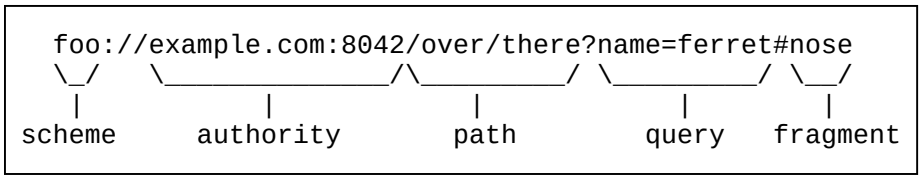
Figura 1 – Arquitetura de um sistema de busca

2.1.1 URI/URL

A URL descreve o caminho para um objeto único, que pode ser um documento, uma imagem, uma página *HyperText Markup Language* (HTML), entre outros. A URL é o material de trabalho fundamental do *crawler*. Dela irá retirar as informações necessárias para percorrer a *web* atrás de novos *links*.

Segundo a RCF3986 (2005) o termo correto a utilizar é *Uniform Resource Identifier* (URI), pois esta especificação pode demonstrar tanto *Locators* (Localizadores) e *Names* (Nomes). É uma definição mais genérica. Neste trabalho será utilizado apenas os *Locators*.

Conforme pode ser observado no Quadro 1, os componentes de uma URI são:



Fonte: RFC3986 (2005).

Quadro 1 – Componentes da URI

a) *Scheme* (Esquema): toda URI inicia-se com um nome de esquema que se refere à especificação do recurso. É formado de uma seqüência de caracteres, iniciados por uma letra e seguidos por qualquer combinação de letras, dígitos ou os caracteres (“+”,“.”,“-”). Exemplo: *HyperText Transfer Protocol* (HTTP);

b) *Authority* (Autoridade): fornece os meios para distinguir o nome e o servidor de onde se encontra o recurso. Podem ser usadas informações de usuários, *host* e porta. Exemplo: “usuario:senha@site.com.br:80”;

c) *Path* (Caminho): corresponde ao caminho para um dado recurso (onde ele pode ser encontrado), geralmente organizado de forma hierárquica. Exemplo: “/arquivo/teste.php”;

d) *Query* (Chamada): contém informações de forma não hierárquica, que servem para localizar o recurso dentro do *path*. Geralmente é utilizado para informações de postagem de dados. Exemplo: “?chave=valor”;

e) *Fragment* (Fragmento): é utilizado para identificar algum recurso dentro do próprio recurso. Geralmente é usado no HTML para identificar uma posição dentro da página (direcionar para uma parte da página). Exemplo: “pagina.html#parte2”.

Alguns exemplos de URI são: ftp://ftp.is.co.za/rfc/rfc1808.txt, telnet://192.0.2.16:80/, http://www.ietf.org/rfc/rfc2396.txt e mailto:John.Doe@example.com.

2.1.2 CRAWLER

Cho e Garcia-Molina (2000, p. 1, tradução nossa) definem que “um *crawler* é um programa que coleta automaticamente páginas da *web* para criar um índice local e/ou uma coleção local de páginas *web*”. Ou seja, é um programa que busca páginas da *web* e pode armazenar os dados de duas formas: o documento original de forma compactada e o mesmo de forma indexada. Todo o trabalho de mantê-las atualizadas também é uma função do *crawler*.

Tradicionalmente, um *crawler* visita a *web* até atingir um número máximo de páginas e pára. Quando for necessário renovar a sua coleção local, a mesma é apagada e substituída por uma completamente nova utilizando o mesmo processo. Esse tipo de *crawler* é chamado de *crawler* periódico. Já o *crawler* incremental ao atingir o seu limite de páginas começa a atualizar as mesmas, substituindo as páginas “menos importantes” (atualizadas com menor freqüência) pelas “mais importantes” (atualizadas com maior freqüência) (CHO; GARCIA-MOLINA, 2000, p. 1).

A eficiência do *crawler* incremental está na maneira como ele consegue estimar o quanto cada página é freqüentemente modificada ou não. Assim, ele acessa menos vezes as páginas pouco modificadas (menos importantes).

2.1.2.1 METÓDOS DE BUSCA E METRICAS

Cho (2001, p. 2, tradução nossa) comenta sobre quais páginas podem ser indexadas por um *crawler*: “Na maioria dos casos, o *crawler* não indexa todas as páginas da *web*. Até mesmo os maiores sistemas de busca atualmente indexam apenas uma pequena parte da *web*.”.

Dada essa afirmação, é fácil notar a importância de saber quais páginas indexar e quais não, mesmo em um estágio inicial do processo de coleta do *crawler*.

Métodos e métricas são utilizados para suprir esta necessidade, sendo os mais conhecidos:

a) *First-In-First-Out* (FIFO): produz uma coleta mais abrangente, visando um grande número de sites. De fácil implementação;
b) *Last-In-First-Out* (LIFO): produz uma coleta mais focada, mais páginas por site;
c) *Backlink count* (Conectividade): produz uma coleta por popularidade, ou seja, quanto mais uma página tiver um *link* apontando para ela, mais ela terá chances de ser *indexada* primeiro.

Métodos e métricas podem ser mesclados entre si para obtenção de um melhor resultado. É comum também utilizar de fatores como a extensão do domínio (.com, .br, entre outros) e o idioma da página como modificadores relevantes nas métricas.

2.1.2.2 *CRAWLER* INCREMENTAL X PERIÓDICO

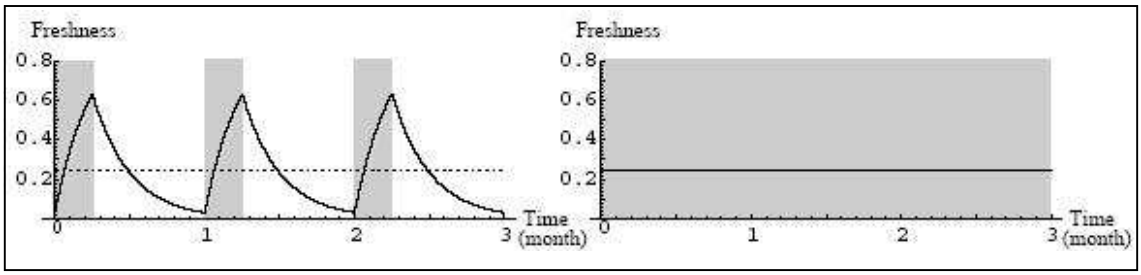
Um *crawler* precisa manter sua coleção de páginas sempre atualizadas. Tentar ter um recorte da Internet que seja o mais parecido com o seu real estado. Para isso o *crawler* necessita atualizar a sua coleção.

Basicamente existem duas maneiras de manter uma coleção de páginas atualizadas:

a) *Crawler* periódico – atualiza toda a coleção a cada x dias. De simples implementação;
b) *Crawler* incremental – atualiza constantemente as páginas da coleção, geralmente utilizando métricas de seleção.

Na Figura 2 é mostrado uma relação de atividade do *crawler versus* o tempo em que as

páginas permanecem atualizadas. O eixo horizontal representa o tempo e a parte cinza é o período em que o *crawler* está operante (coletando páginas). Já o eixo vertical representa o *freshness* (a quão atualizada está a coleção).



Fonte: CHO (2000, p. 11).

Figura 2 – Atualização das páginas. Periódico x Incremental

Pode-se notar que no primeiro caso (periódico), as páginas têm um ponto mais alto de *freshness* somente quando são atualizadas e que vão decaindo com o tempo. Já no segundo caso (incremental) ele tende a manter um padrão.

2.1.2.3 ATUALIZAÇÃO DAS PÁGINAS (MÉTRICAS DE SELEÇÃO)

Segundo Cho, Garcia-molina (2001, p. 8), "a *web* demora cerca de 50 dias para ter 50% das suas páginas modificadas ou substituídas por novas". A *web* é dinâmica, está em constante atualização e expansão, e por tal motivo as páginas precisam ser atualizadas. Um fator importante é tentar determinar o tempo de atualização das páginas, assim reduzindo o número de visitas ao site somente para o necessário.

Existem basicamente dois tipos de estratégias gerais para a atualização de páginas (CHO; GARCIA-MOLINA, 2001, p. 10):

a) política uniforme: todas as páginas são visitas com uma mesma freqüência. Fácil implementação, mais gera muitas "visitas desnecessárias";
b) política proporcional: define uma função de freqüência para cada página com base em seu histórico de alterações. Por exemplo: se uma página P1 é visitada uma vez

por dia durante um mês, e é detectado que houve 10 alterações. A sua função de freqüência é de 3 dias.

2.1.3 PROTOCOLO DE EXCLUSÃO

O protocolo de exclusão é um método que permite aos administradores de sites indicarem para os *crawlers* de busca não indexarem partes do seu site. (THE WEB ROBOTS PAGE, 2005).

Por padrão, quando um *crawler* tenta acessar um site (por exemplo, http://www.furb.br) antes ele irá verificar um arquivo chamado “robots.txt” na raiz do domínio (http://www.furb.br/robots.txt).

No exemplo do Quadro 2 não é permitida a indexação de qualquer parte do site (“/”) para qualquer *crawler* (“*user-agent*”).

```
User-agent: *
Disallow: /
```

Quadro 2 – Exemplo de um arquivo robots.txt

Utilizar esse protocolo de exclusão em um *crawler* em produção é uma questão de ética, pelo fato do *crawler* poder naturalmente acessar qualquer página na *web*. É dar a opção do administrador de um site escolher o que ele não quer que aparece em um sistema de busca.

2.2 SISTEMAS DE ARMAZENAMENTO E RECUPERÇÃO

Um bom sistema de armazenamento é essencial para qualquer aplicação que dispõe de uma grande quantidade de dados. Características como persistência nos dados, segurança no seu acesso e fornecimento de recursos para a recuperação de dados são essenciais. Tais

características são fundamentos básicos de qualquer Sistema Gerenciador de Banco de Dados (SGBD).

"Um dos muitos aspectos relevantes que distinguem os sistemas de bancos de dados de outros sistemas é a habilidade de um SGBD para lidar de forma eficiente com quantidades muito grandes de dados." (GARCIA-MOLINA; ULLMAN; WISDOM, 2001, p. 22).

Existem vários fatores envolvidos que se deve ter em mente quando se pretende armazenar e recuperar informações. A maioria dos SGBDs atuais o fazem de forma satisfatória. Por tal motivo, na maioria das vezes se faz desnecessário implementar separadamente suas funcionalidades. Como exemplo é possível citar: a própria *Structured Query Language* (*SQL*), tipos de registros (*string*, *int*, *BLOB*) e técnicas de indexação (arquivos seqüências, índices secundários, árvores B e tabelas *hash*).

A seguir será comentando sobre o SGDB MYSQL e como funciona o seu índice de busca de texto completo (*full-text search*).

2.3 MYSQL

Atualmente existem vários SGBD no mercado, sejam eles proprietários ou *open-source*. Dentre muitos, o MySQL tem se destacado e se disseminado entre a comunidade de desenvolvedores, principalmente para aplicações em ambiente *web.*

2.3.1 FULL-TEXT SEARCH

Entre muitas funcionalidades que o MySQL apresenta, destacam-se os recursos para buscas de conteúdo, conhecido como *full-text search.*

O *full-text search* é um mecanismo que fornece métodos de busca em texto, em tabelas

MyISAM (acesso seqüencial) e pode ser definido em tipos de colunas CHAR, VARCHAR e TEXT. Para esse recurso, existem três tipos de busca (MYSQL REFERENCE MANUAL, 2005):

a) buscas lógicas: é uma busca por palavras, onde pode ser utilizado o “e” e o “não” lógicos, ou seja, escolher quais palavras devem estar presentes e quais não;
b) busca por linguagem natural: é uma busca por uma frase como na língua natural, não existe operadores. Palavras que estiverem presentes em mais de 50% do texto serão descartadas automaticamente;
c) expansão de busca: é utilizada igualmente como na busca por linguagem natural, com a diferença de que ela é executada duas vezes. A primeira busca é feita normalmente. Já a segunda busca é realizada com o item mais relevante da primeira (a palavra que mais aparece), obtendo como resultado a junção das duas. Esta busca é chamada também de busca por conhecimento. Por exemplo, se for efetuada a busca por “database” e a palavra mais relevante dessa busca for “MySQL”, a segunda busca não necessariamente irá retornar valores que contenham a palavra “database”, mas que, por força da relevância está associada à ela.

No Quadro 3 pode ser observado um exemplo de expansão de busca *full-text search*. Na primeira parte foi efetuada uma busca pela palavra “*database*” dentro de duas colunas da tabela (*title*, *body*), e mostrado seu resultado. Na segunda, foi efetuada a mesma busca, só que agora com a expansão. Nesse caso, pode-se notar que o terceiro item retornado da busca não contém a palavra “*database”*.

```
mysql> SELECT * FROM articles
    -> WHERE MATCH (title,body) AGAINST ('database');
+----+-------------------+------------------------------------------+
| id | title             | body                                     |
+----+-------------------+------------------------------------------+
|  5 | MySQL vs. YourSQL | In the following database comparison ... |
|  1 | MySQL Tutorial    | DBMS stands for DataBase ...             |
+----+-------------------+------------------------------------------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM articles
    -> WHERE MATCH (title,body)
    -> AGAINST ('database' WITH QUERY EXPANSION);
+----+-------------------+------------------------------------------+
| id | title             | body                                     |
+----+-------------------+------------------------------------------+
|  1 | MySQL Tutorial    | DBMS stands for DataBase ...             |
|  5 | MySQL vs. YourSQL | In the following database comparison ... |
|  3 | Optimizing MySQL  | In this tutorial we will show ...        |
+----+-------------------+------------------------------------------+
3 rows in set (0.00 sec)
```

Fonte: MYSQL REFERENCE MANUAL (2006).

Quadro 3 – Exemplo de expansão de busca no MYSQL

2.3.2 ÍNDICE INVERTIDO

O recurso *full-text search* do MYSQL utiliza o modelo de indexação de índice invertido (*full inverted index*). Um índice invertido é uma estrutura indexada onde é guardado um mapa de todas as palavras e sua localização no texto e no documento. É uma das mais importantes estruturas usadas nos sistema de busca (WIKIPEDIA, 2006).

Existem dois tipos de índices invertidos: o *inverted file index* que guarda apenas o documento em que determinada palavra aparece e o *full inverted index,* onde é guardada também a posição em que ele aparece no texto. O primeiro tipo é mais rápido e utiliza menos espaço, mais ao mesmo tempo não permite buscas de frases, enquanto o segundo sim.

Exemplo da utilização do *full inverted index*: dado dois documentos d0 e d1 onde d0 = “gosto de maça” e d1 = “gosto de manga”, o seu modelo indexado resultante pode ser visto no Quadro 4.

“gosto” => {d0,1} {d1,1}
“de” => {d0,2} {d1,2}
“maça” => {d0,3}
“manga” => {d1,3}

Quadro 4 – modelo indexado

Em uma pesquisa de uma frase “gosto de”, será feita uma intersecção entre o conjunto “gosto” e “de” (gosto ∩ de) utilizando o atributo documento (d0 e d1). Nos resultados desta operação são verificados se a condição da posição do texto é verdadeira (“gosto” tem que estar uma posição antes de “de”, ou vice-versa), para saber se é uma frase.

2.4 SISTEMAS DISTRIBUIDOS

“Um sistema distribuído é a coleção de vários computadores independentes que aparecem para o usuário como um único sistema” (TANENBAUM; STEEN, 2002, p. 2, tradução nossa). Para estes autores, a principal característica de um sistema distribuído é a transparência.

A característica de sistemas distribuídos que se aplica diretamente ao *crawler* é a escalabilidade. Por causa da grande quantidade de informações que serão tratadas, o armazenamento e o processamento das páginas estarão distribuídos entre vários computadores. Caso o número de computadores seja insuficiente para atender a demanda, um novo computador (com um novo *crawler*) será adicionado ao sistema.

2.4.1 ESCALABILIDADE

Para Coulouris, Dollimore e Kindberg (2001, p. 19-20) um sistema pode ser

considerado escalável se permanecer efetivo mesmo quando tiver um aumento significativo do seu número de usuários ou de recursos.

Para um sistema conseguir este patamar, existem quatro desafios:

a) controle dos recursos físicos: enquanto um sistema cresce, deve ser possível aumentar o desempenho do sistema adicionando novos recursos físicos. Por exemplo: se um sistema suporta 10 usuário com 1 servidor, com 2 servidores deverá suportar 20, e assim sucessivamente;
b) controle de perda de desempenho: todo o sistema, mesmo quando passível de adição de recursos físicos, terá uma perda de desempenho. Um sistema não deve ter uma perda de desempenho muito grande;
c) prevenir que os recursos de software se esgotem: todas as formas de dados, ou estruturas do sistema devem ser pensadas para se adaptarem em um crescimento futuro. Um exemplo de falha neste quesito, foi a definição de 32 bits para o protocolo IP na internet no ano de 1970. No ano 2000 foi necessário criar o ipv6 (128 bits);
d) evitar gargalos: um gargalo é uma parte do sistema que, devido ao seu crescimento, não consegue mais satisfazer o desempenho e o qual também não é possível ser aumentado por adição de recursos físicos. Uma solução para evitar gargalos é fazer sistemas descentralizados.

2.4.2 TRANSPARÊNCIA

Um sistema é definido como transparente quando o usuário não tem conhecimento do sistema que está por trás do recurso que ele está utilizando (COULOURIS; DOLLIMORE; KINDBERG, 2001, p. 19-20).

O elemento do sistema de busca que representa a transparência é o *web search*. Assim temos transparência em:

a) acesso: são acessados vários recursos apenas pela interface de busca;

b) localização: os recursos são acessados sem conhecimento do seu local;

c) concorrência: vários computadores são acessados sem a interferência do usuário;

d) desempenho: o sistema pode ser configurado para um melhor desempenho (adição de computadores) sem o usuário perceber;

e) escalabilidade: é possível aumentar o sistema sem alterar a sua estrutura ou os seus algoritmos.

3 DESENVOLVIMENTO DO SISTEMA

Neste capítulo é descrito o desenvolvimento do sistema de busca. Na seção 3.1 são apresentados os requisitos dos quatro elementos que compõem o sistema: *crawler, sorter/indexer, web* search e o servidor de URL. Na seção 3.2 é dado uma visão geral do sistema. Na seção 3.3 é apresentada a especificação do sistema, e na seção 3.4 é apresentado todo o processo de desenvolvimento do sistema e seus elementos. A operacionalidade do sistema também é apresentada nesta seção.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Nesta seção é apresentada a característica geral do sistema, dividido em quatro principais elementos: *crawler*, *sorter/indexer*, *web search* e servidor de URL.

O *crawler* possui as seguintes características:

a) buscar páginas na *web* e armazená-las (requisito funcional – RF);
b) processar páginas à procura de novas URLs (RF);
c) prover comunicação com todos os *crawlers* em funcionamento (RF);
d) manter todo o banco de dados (coleção global de páginas) de forma eficaz, não tendo páginas duplicadas em diferentes *crawlers* (RF);
e) manter atualizada a sua coleção local de páginas (RF);
f) processar informações pedidas pelo *web search* na sua coleção local de páginas (RF);
g) efetuar o armazenamento no banco de dados relacional MySQL (requisito não-funcional – RNF);
h) ser desenvolvido em Java (RNF).

O *web search* possui as seguintes características:

a) disponibilizar uma interface para a busca de informações (RF);
b) comunicar-se com os *crawlers* para efetuar a busca de forma transparente para o usuário (RF);
c) ser desenvolvido em *Hypertext Preprocessor* (PHP) (RNF).

O *search*/*indexer* possui as seguintes características:

a) manter organizada fisicamente todas as páginas (RF);
b) manter indexadas todas as páginas (RF);
c) ser desenvolvido em PHP (RNF).

O servidor de URL possui as seguintes características:

a) manter uma relação de todas URL (RF);
b) manter o tipo (MIME-TYPE) de cada URL e o *crawler* a qual a URL pertence (RF);
c) fazer a distribuição de *links* (RF).

3.2 VISÃO GERAL

Uma visão geral do sistema pode ser visto na Figura 3.

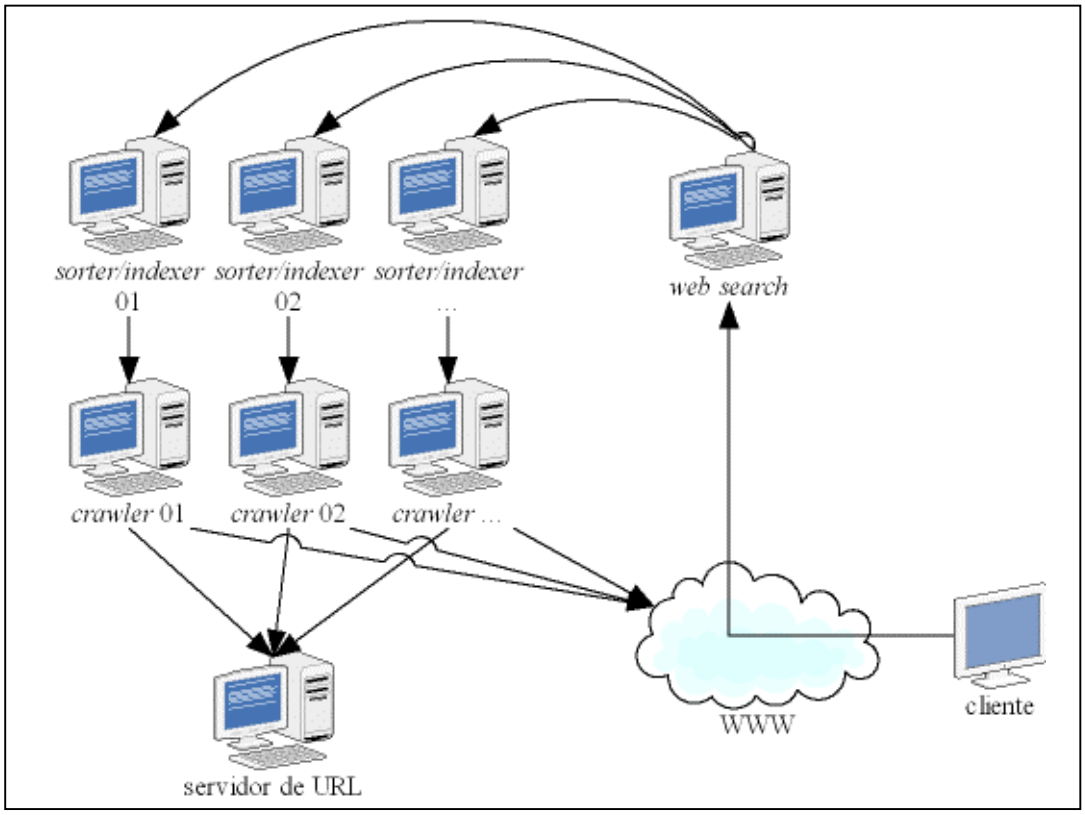


Figura 3 – Visão geral do sistema

Cada *crawler* é responsável por um número de *links* (URLs), do qual será o "dono" e responsável por manter seu conteúdo e seu *ranking* atualizado. Neste caso, o *crawler* requisita um *link* do servidor de URL, e a partir desse momento irá pegar o seu conteúdo e retirar todos os novos *links* encontrados. Para cada *link* encontrado, ele irá verificar se ele já é o "dono" desse *link*, senão irá passar para o servidor de URL. O servidor de URL se torna o "dono" temporário de todos estes novos *links* encontrados até o momento que passar a URL para um *crawler*, seguindo as regras de *ranking*. Cada *link*, tanto se estiver no servidor de URL quanto se estiver no *crawler*, possui a relação de todos os outros *links* que apontam para ele. Esta informação é utilizada para fazer o *ranking*.

O *sorter/indexer* é o conteúdo já organizado, pelo *ranking* de forma decrescente (para aumentar a velocidade da busca) e indexado com o *full-text search*.

Por fim, o cliente quando efetuar uma consulta para o *web search* requisita a busca para cada um dos *sorters/indexers* ativos, junta os dados baseados no *ranking* individual de

cada um e retorna a consulta para o usuário.

Em um ambiente distribuído, o gargalo neste modelo se encontra no servidor de URL, que pode ser corrigido fazendo a sua distribuição em várias máquinas. Neste trabalho, optou-se por trabalhar com apenas um servidor de URL, dado a dificuldade de sua implementação. Em Bharat et al. (1998) pode ser encontrado o desenvolvimento de um servidor de URL eficaz, que foi utilizado para o já conhecido sistema de busca Altavista.

3.3 ESPECIFICAÇÃO

Para a especificação foi utilizado a UML através do diagrama de casos de uso e de classes. Todos os diagramas foram especificados utilizando a ferramenta Enterprise Architect. Na seção 3.2.1 é mostrado o modelo entidade-relacionamento e utilização de índices, construído utilizando a ferramenta DBDesigner 4.

Os diagramas de casos de uso estão divididos em quatro partes e representam sucessivamente o *web search, crawler, search/indexer* e o servidor de URL.

A Figura 4 representa o caso de uso que faz parte do *web search.*

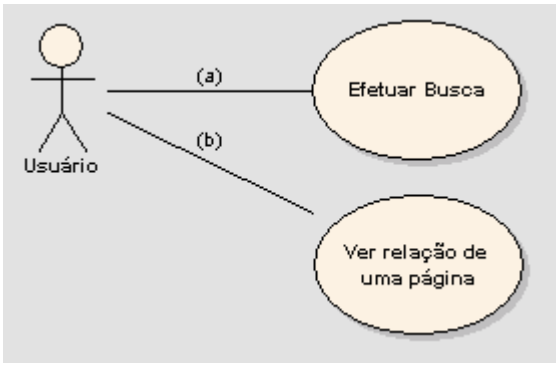


Figura 4 – Diagrama de caso de uso – *web search*

No Quadro 5 é apresentada uma descrição do caso de uso.

Caso de Uso	**Ator**	**Descrição**
Efetuar busca	Usuário	Usuário do sistema efetua uma busca no sistema e

		obtém resultados.
Ver relações de uma página	Usuário	Após efetuar uma busca, para cada item retornado o usuário pode ver estatísticas da página no sistema.

Quadro 5 – Descrição dos casos de uso – *web search*

A Figura 5 representa o caso de uso que faz parte do *crawler*.

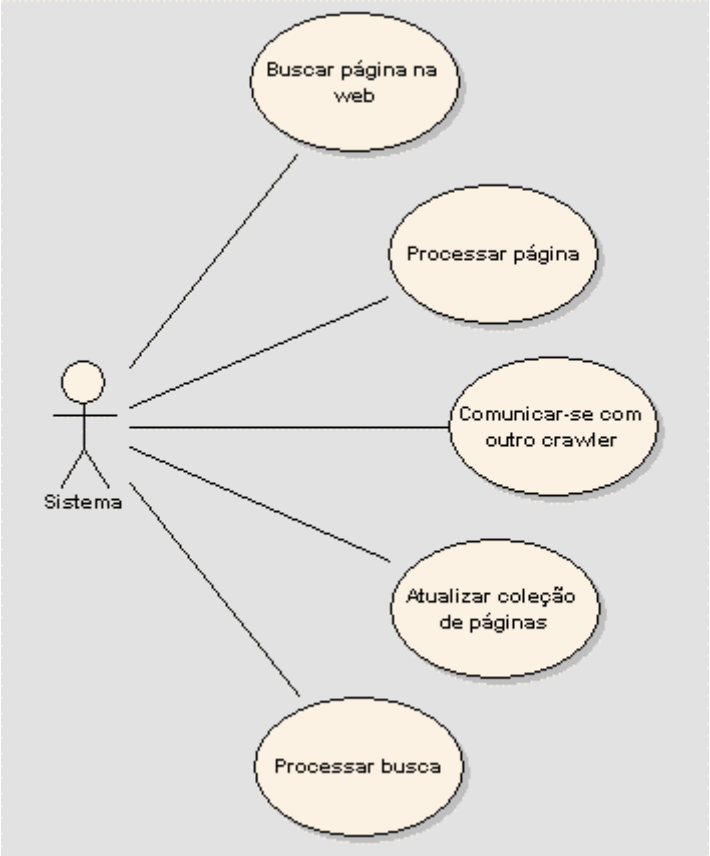


Figura 5 – Diagrama de caso de uso – *crawler*

No Quadro 6 é apresentada uma descrição do caso de uso.

Caso de Uso	**Ator**	**Descrição**
Buscar página na *web*	Sistema	O sistema efetua uma busca de uma página na *web*.
Processar página	Sistema	O sistema busca novas URL e normaliza uma página.
Comunicar-se com outro *crawler*	Sistema	Trocar informações com outro *crawler* dentro da estrutura.
Atualizar coleção de páginas	Sistema	Manter a coleção de páginas atualizadas.
Processar busca	Sistema	O sistema deve fornecer meios necessários para ser

		efetuada uma busca dentro da sua coleção local de páginas.

Quadro 6 – Descrição dos casos de uso – *crawler*

A Figura 6 representa o caso de uso que faz parte do *indexer/sorter*.

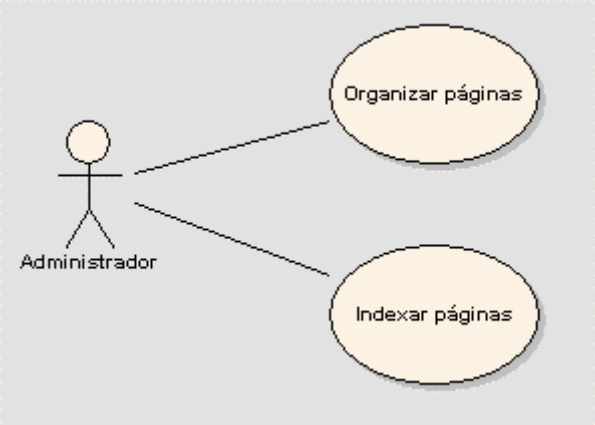


Figura 6 – Diagrama de caso de uso – *indexer/sorter*

No Quadro 7 é apresentada uma descrição do caso de uso.

Caso de Uso	**Ator**	**Descrição**
Organizar páginas	Administrador	Fornece ao administrador a capacidade de organizar as páginas fisicamente.
Indexar páginas	Administrador	Fornece ao administrador a capacidade de indexar a coleção de páginas.

Quadro 7 – Descrição dos casos de uso – *indexer/sorter*

A Figura 7 representa o caso de uso que faz parte do servidor de URL.

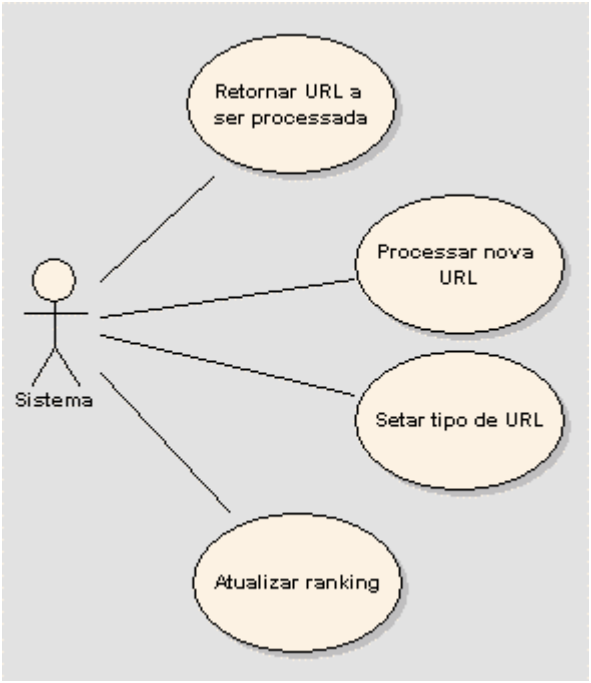


Figura 7 – Diagrama de caso de uso – servidor de URL

No Quadro 8 é apresentada uma descrição do caso de uso.

Caso de Uso	**Ator**	**Descrição**
Retornar URL a ser processada	Sistema	Retorna a próxima URL a ser processada com base no *ranking*.
Processar nova URL	Sistema	Receber uma URL e adicionar ela na base de dados.
Setar tipo de URL	Sistema	Modificar o tipo de URL.
Atualizar ranking	Sistema	Atualizar a relação de um URL com outras URLs.

Quadro 8 – Descrição dos casos de uso – servidor de URL

A Figura 8 representa o diagrama de classes do sistema.

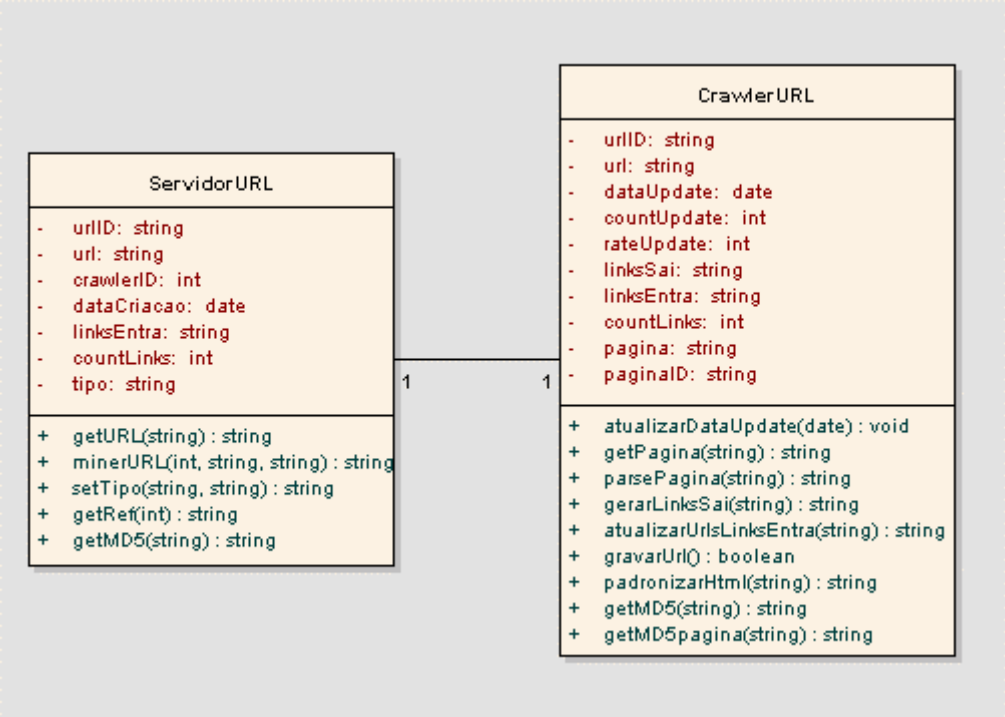


Figura 8 – Diagrama de classes.

A classe ServidorURL é sempre acessada pelos diversos *crawlers* (classe CrawlerURL) e possui uma cardinalidade 1 para 1, ou seja, apenas um *crawler* é “dono” de 1 URL do servidor de URL.

3.3.1 MODELO ENTIDADE-RELACIONAMENTO E ÍNDICES

Na Figura 9 é apresentado o modelo entidade-relacionamento do sistema.

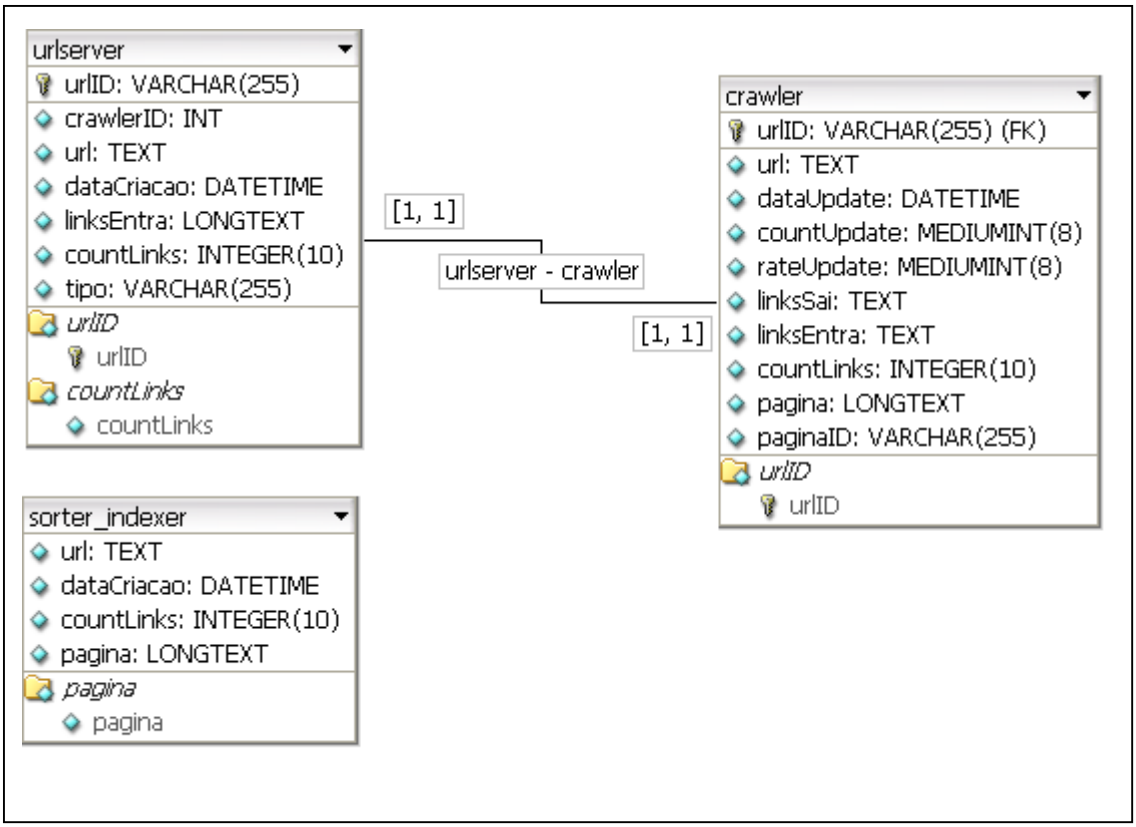


Figura 9 – Modelo entidade-relacionamento.

A tabela “urlserver” representa cada URL que existe no sistema, sendo que está associada a apenas um *crawler*, ou seja, nenhum *crawler* irá ter uma URL repetida. A tabela “sorter_indexer” representa o conteúdo organizado e pronto para a busca, o qual estará organizada fisicamente e será feita efetivamente a busca.

Em um banco de dados com um grande número de registros, a diferença entre uma consulta demorar poucos milisegundos ou até vários minutos está na colocação correta de índices.

No Quadro 9 são listados todos os campos nos quais foram utilizados índices. Também são apresentados uma explicação, o tipo de índice utilizado e a sentença SQL correspondente.

Campo	Tipo de Índice	Explicação	SQL
Urlserver: urlID	UNIQUE	Identificação única da URL. Utilizada na cláusula WHERE.	SELECT crawlerID, linksEntra, countLinks FROM urlserver WHERE

			urlID = ‘xxx’;
Urlserver: countLinks	INDEX	Quantidade de *links* que apontam para essa página. Mesmo tendo uma granularidade menor (vários registros iguais), se tornou necessário por ser utilizada em uma cláusula ORDER BY.	SELECT url, urlID, linksEntra FROM urlserver WHERE crawlerID = 0 ORDER BY countLinks DESC LIMIT 0,1;
Crawler: urlID	UNIQUE	Identificação única da URL. Utilizada na cláusula WHERE.	SELECT linksEntra, countLinks FROM crawler WHERE urlID = 'xxx’;
Sorter_index: pagina	Full-text search	Conteúdo da página devidamente normalizado e padronizado. Utilizado para consultas *full-text search*.	SELECT url, dataUpdate, countLinks, MATCH (pagina) AGAINST ('$query' IN BOOLEAN MODE) AS score FROM `sorter_index` WHERE MATCH (pagina) AGAINST ('$query' IN BOOLEAN MODE) LIMIT $pagina,10;

Quadro 9 – Descrição dos índices utilizados nas tabelas

3.4 IMPLEMENTAÇÃO

Primeiramente é apresentada uma visão geral de todo o sistema com uma breve explicação de cada uma de suas partes. Logo em seguida é explicado e definido como será a utilização dos identificadores únicos no sistema. Depois é explicado e definido o protocolo de comunicação criado para atender a comunicação entre os elementos do sistema em um ambiente distribuído.

Por fim, é apresentado um aprofundamento nos quatro principais componentes do sistema de busca (servidor de URL, *crawler*, *sorter/indexer* e *web search*) e demonstrada a operacionalidade da implementação com um estudo de caso.

3.4.1 IDENTIFICAÇÃO ÚNICA

As diferentes partes do sistema se comunicam entre si e trabalham com dois itens: a URL e o conteúdo de páginas *web*. Para tal, é necessário que exista uma identificação única, padronizada e de tamanho fixo.

A identificação única da URL é utilizada para:

a) consulta: para cada nova URL encontrada, é gerada sua identificação, que será utilizada para futuras consultas;

b) *links*: são registrados todos os *links* para os quais uma página aponta (saída) e todos os links que apontam para a página (entrada).

A identificação única da página *web* é utilizada para:

a) conteúdo: para saber se houve alterações em uma página desde a sua última visita.

Para ambos (URLs e páginas) são efetuados dois processos: o de normalização e o de assinatura computacional, conforme descrito abaixo:

a) URL:

- verificar se está no padrão RFC3986 (2005),
- adicionar a porta para não haver redundâncias (Exemplo: http://www.furb.br e http://www.furb.br:80 apontam para a mesma página),
- converter para letras minúsculas,
- retirar o “/” no final se houver,
- aplicar a assinatura computacional.

b) Página *web*:

- retirar todos os elementos HTML,
- retirar os caracteres especiais,

- substituir todos os caracteres acentuados por não acentuados,
- converter para letras minúsculas,
- aplicar a assinatura computacional.

O algoritmo de assinatura computacional utilizado foi o *Message-Digest algorithm 5* (MD5), com uma assinatura de 128 bits. A probabilidade de acontecer uma colisão (mesma assinatura para dados diferentes) em uma coleção de 10 bilhões de dados diferentes é de 10^{-18} (HIRA et al., 2000, p. 5).

3.4.2 PROTOCOLO DE COMUNICAÇÃO

Pelo fato do sistema ter sido criado de forma distribuída, faz-se necessária a existência de uma forma para que os elementos do sistema (servidor de URL e *crawlers*) se comunicarem.

A forma escolhida foi a implementação de *sockets* em um modelo cliente-servidor. No papel de servidor ficará, como o próprio nome já diz, o servidor de URL. Os *crawlers* serão os clientes. No caso de uma requisição do *web search* (consulta do usuário), os *sorters/indexers* terão o papel de servidores, e o *web search* será cliente.

Abaixo é definido e explicado o funcionamento das quatro mensagens que foram criadas para suprir as necessidades de comunicação.

a) Mensagem 01, pegar URL para ser processada:

Recebe:

01 <crawlerID> (Pegar URL para ser processada)

Retorno:

01 <url> <linksEntra*> (URL a ser processada)

02 (Sem URLs para processar)

03 (Erro ao pegar URL)

Exemplo:

Um *crawler* envia para o servidor a sua requisição junto com seu identificador (número do *crawler*) e obtém como resposta uma URL para ser processada seguida da identificação única dos *links* que apontam para ela, conforme demonstrado na Figura 10.

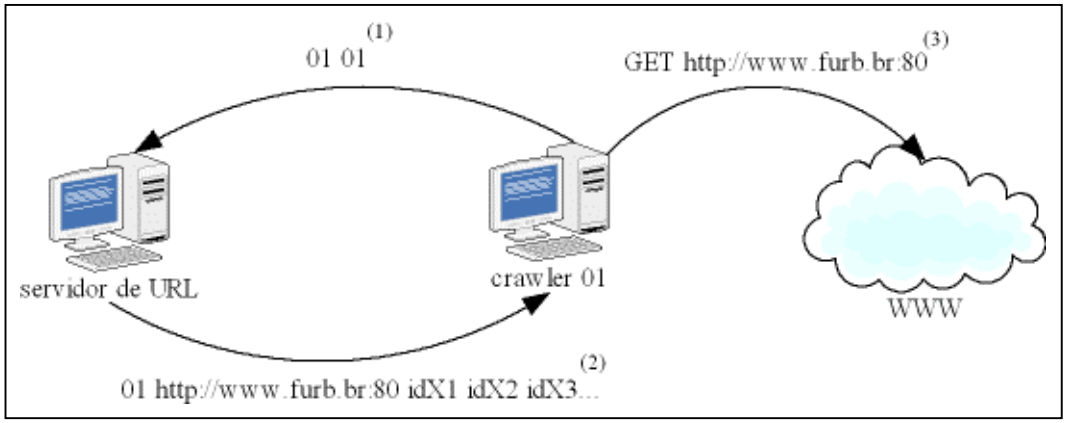


Figura 10 – Envio da mensagem 01

b) Mensagem 02, recebe novas URLs dos *crawlers*:

Recebe:

02 <crawlerID> <urlOrigem> <urlDestino*> (Lista de URL novas)

Retorno:

01 (True)

02 (False)

Exemplo:

Após processar uma URL, extrair seus *links* e verificar se já é o "dono" deles, o restante é enviado para o servidor de URL. Esta mensagem serve para este propósito, conforme demonstrado na Figura 11.

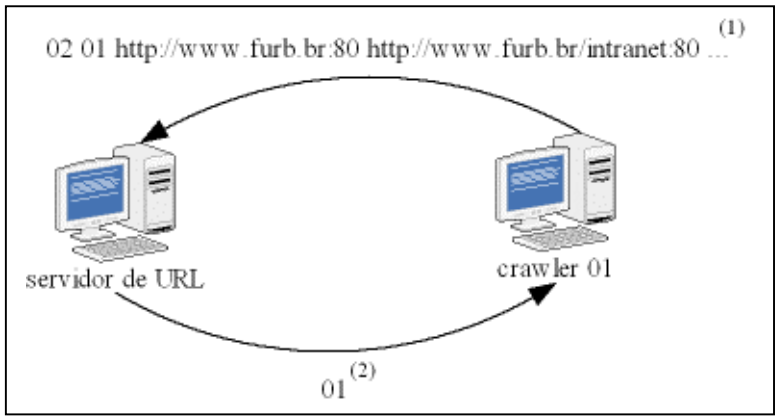


Figura 11 – Envio da mensagem 02

c) Mensagem 03, setar o tipo da URL:

Recebe:

03 <crawlerID> <urlID> <tipo> (Setar um tipo para URL)

Retorno:

01 (True)

02 (False)

Exemplo:

Após pegar uma URL para ser processada (mensagem 01), ao acessar o *link* o *crawler* poderá encontrar um tipo de página diferente de “text/html” ou até mesmo não encontrar a página (“erro/404”). Quando isso acontece, o *crawler* despreza esta página e se comunica com o servidor de URL utilizando esta mensagem. Na Figura 12 é demonstrada a utilização desta mensagem caso o crawler encontre no *link* um arquivo no formato .PDF.

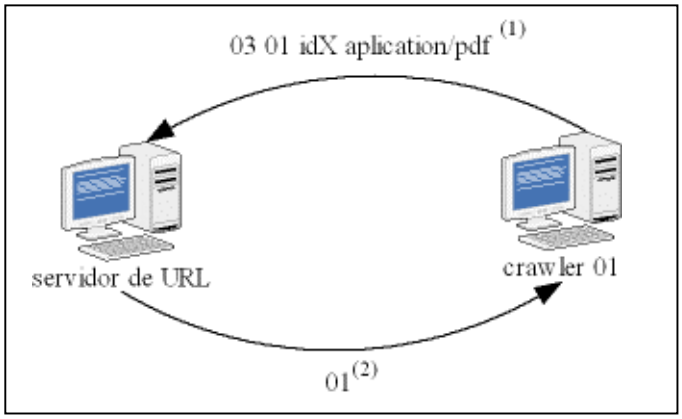


Figura 12 – Envio da mensagem 03

d) Mensagem 04, atualizar referências para o *ranking*:

Recebe:

04 <crawlerID> (Pedir atualização)

Retorno:

01 <urlID> <urlIDs*> (True, lista de atualização)

02 (Erro)

Exemplo:

Quando um *crawler* não é o "dono" de um *link* encontrado, ele passa o *link* para o servidor de URL. Pode acontecer de este *link* já possuir um dono, ou seja, não foi o *crawler* que o achou. Por esse motivo, o servidor de URL reconhece isso e deixa o *link* guardado para passar para o seu "dono". Essa mensagem serve para o *crawler* fazer a requisição, buscando saber se existem ou não *links* para ele, conforme demonstrado na Figura 13.

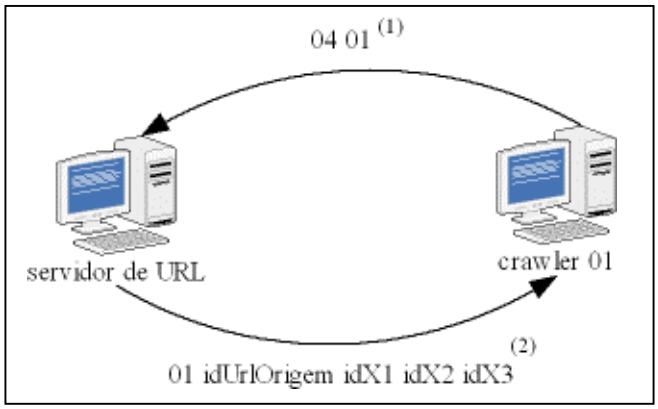


Figura 13 – Envio da mensagem 04

3.4.3 SERVIDOR DE URL

O servidor de URL é responsável por manter uma lista de todos os *links* já visitados ou descobertos pelos *crawlers*. Também é responsável pela distribuição de URLs, baseado no

ranking de conectividade (*backlink metric*). Para poder ter as informações de conectividade, é mantido em sua base a associação entre todos os *links* (quem aponta para quem).

Para manter a relação entre os *links*, são utilizados dois campos na sua base de dados: o "linksEntra" e o "countLinks". O "linksEntra" contém o identificador único de todos os *links* que apontam para o *link* correspondente separados por um espaço, e o "countLinks" é um atalho para saber a soma destes *links*.

No Quadro 10 pode ser observado o momento que o servidor de URL entra na rotina principal e fica aguardando conexões dos *crawlers*. Pode-se notar que é comum para todos uma instância da classe "ServidorURL", que é enviada para cada nova *thread* iniciada.

```
try {
    // Cria o Socket
    ServerSocket listenSocket = new ServerSocket(confPorta);
    ServidorURL urlserver = new ServidorURL();
    while(true) {
        Socket clientSocket = listenSocket.accept();
        ServidorTCP c = new ServidorTCP(clientSocket, urlserver);
    }
} catch (java.io.IOException e) {
    System.out.println("Listen socket: "+e.getMessage());
}
```

Quadro 10 – Criação do servidor de URL

A objetivo da classe "ServidorURL" ser comum para todos é a implementação do método *syncronized* getUrl, que distribui uma URL, conforme pode ser visto no Quadro 11. Isso assegura que nenhum *crawler* será "dono" de uma mesma URL, porque um método *syncronized* garante que não será executado mais de uma vez ao mesmo tempo.

```
/*
 * Retorna uma URL não processada para o crawler
 *
 * Recebido:
 *  01 <crawlerID>
 * Resposta:
 *  1 <url> <linksEntra> // URL a ser processada
 *  2                    // Sem URLs para processar
 *  3                    // Erro ao pegar URL
 */
public synchronized String getURL(int crawlerID) {
    String G_url = "";
    String G_urlID = "";
    String G_linksEntra = "";
    String query = "SELECT url, urlID, linksEntra FROM urlserver WHERE crawlerID = 0";
    query = query + " ORDER BY countLinks DESC LIMIT 0,1;";
    ResultSet rs = mi.executeQuery(query);
    ...
```

Quadro 11 – Método getURL

No Quadro 12 pode ser observado o código do tratamento de uma mensagem no momento que é recebida de um *crawler*.

```
/* MINER URL
 * Processa URLs recebidas
 *
 * Recebido:
 *  02 <crawlerID> <urlOrigem> <urlDestino*>
 * Resposta:
 *  1 // OK
 *  2 // Erro
 */
if (splited[0].equalsIgnoreCase("02")) {
    String crawlerID = splited[1];
    String urlOrigemID = splited[2];
    String urlDestino = "";
    for (int count = 3; count < splited.length; count++) {
        if (splited.length-1 == count) {
            urlDestino += splited[count];
        } else {
            urlDestino += splited[count] + " ";
        }
    }
    String result = this.urlserver.minerURL(Integer.parseInt(crawlerID), urlOrigemID, urlDestino);
    out.writeUTF(result);
    //urlserver.closeConnection();
    crawlerID = null;
    urlOrigemID = null;
    urlDestino = null;
    result = null;
}
/* /MINER URL */
```

Quadro 12 – Recebimento de mensagem

No Quadro 13 pode ser visto a função que gera a identificação única de uma URL.

```
/*
 * Retorna o MD5 do host de uma URL.
 */
public String getMD5host(URL url)
    try {
        MessageDigest md = MessageDigest.getInstance("MD5");
        md.reset();
        byte[] defaultBytes = url.getHost().getBytes();
        md.update(defaultBytes);
        byte md5[] = md.digest();
        StringBuffer hexString = new StringBuffer();
        for (int i=0;i<md5.length;i++) {
            hexString.append(Integer.toHexString(0xFF & md5[i]));
        }
        return hexString.toString();
    } catch (java.security.NoSuchAlgorithmException e) {
        return "0";
    }
}
```

Quadro 13 – Geração do identificador único

O servidor de URL foi desenvolvido em Java e utilizado MySQL como banco de dados.

3.4.4 CRAWLER

O *crawler* tem a função de buscar páginas, extrair seus *links*, guardar seu conteúdo e manter uma relação de conectividade em concordância com o servidor de URL.

Para se tornar efetivo e ser capaz de efetuar esse processo em várias páginas ao mesmo tempo, o *crawler* foi divido em várias *thread*, em processos iguais executados em paralelo.

Na Quadro 14 é mostrado o código inicial do *crawler* com as suas configurações iniciais.

```
String urlServerHost = "localhost";       // Host do servidor de URL
int urlServerPort = 1234;                 // Porta do servidor de URL
String mysqlServerHost = "localhost";     // Host do banco
String mysqlServerDatabase = "crawler";   // Base do banco
int crawlerID = 1;                        // ID unico do crawler
int maxURLs = 2000000;                    // Numero maximo de URLs no Crawler
CrawlerAPI api = new CrawlerAPI(mysqlServerHost, mysqlServerDatabase);
```

Quadro 14 – Configuração do *crawler*

No Quadro 15 é apresentado a execução das diversas *thread* do *crawler*.

```
// instancias do crawler
CrawlerCliente crawler01 = new CrawlerCliente(urlServerHost, urlServerPort, crawlerID, maxURLs, api);
crawler01.start();
//...
CrawlerCliente crawler10 = new CrawlerCliente(urlServerHost, urlServerPort, crawlerID, maxURLs, api);
crawler10.start();
```

Quadro 15 – Inicio das execução das *threads*

No Quadro 16 é mostrado o inicio da função que acessa a página, verifica seu cabeçalho e pega seu conteúdo.

```
public String getPagina() {
    try {
            this.pagina = "";
            URLConnection uc = this.url.openConnection();
            uc.setReadTimeout(15000);
            uc.setConnectTimeout(15000);
            uc.connect();
            if (uc.getHeaderFields().toString().matches(".*404 Not Found.*")) {
                return "erro/404";
            }
            try {
                if (!uc.getContentType().split(";")[0].equalsIgnoreCase("text/html")
                    && !uc.getContentType().split(";")[0].equalsIgnoreCase("text/plain")) {
                    return uc.getContentType().split(";")[0];
                }
            } catch (java.lang.NullPointerException e) {
                return "erro/conexao";
            }
        try {
            BufferedReader in = new BufferedReader(
                            new InputStreamReader(
                            uc.getInputStream()));

            String inputLine;
            StringBuffer tempPagina = new StringBuffer();
            while ((inputLine = in.readLine()) != null) {
                tempPagina.append(inputLine);
            }
            this.pagina = tempPagina.toString();
```

Quadro 16 – Função pegar página

No Quadro 17 é mostrado a seqüência que cada *crawler* acessa e busca as informações para pegar uma URL.

```
try {
    // Criar URL
    url = new CrawlerURL(this.api);
    api.toLog(docID+": URL ("+splited[1]+")...");
    url.url = new URL(splited[1]);  // url
    url.gerarUrlID();               // urlID
    url.atualizarDataUpdate();      // dataUpdate
    url.countUpdate = 0;            // countUpdate
    url.rateUpdate = 0;             // rateUpdate
    url.linksEntra = tempEntra;     // linksEntra
    url.countLinks = tempEntra.split(" ").length;   // countLinks
    // pagina (busca a pagina)
    api.toLog(docID+": getPagina...");
    String tempTipo = url.getPagina();
    // Pega o MD5 da página
    url.paginaID = url.getMD5Pagina(url.pagina);
    if (tempTipo.equalsIgnoreCase("text/html") || tempTipo.equalsIgnoreCase("text/plain")) {
        // retira todos os links da pagina
        api.toLog(docID+": parsePagina...");
        String tempParser = url.parsePagina();
        api.toLog(docID+": parsePagina...OK");
```

Quadro 17 – Processos de um *crawler*

O *crawler* foi desenvolvido em Java e utilizado MySQL como banco de dados.

3.4.4.1 BUSCA E PROCESSAMENTO DE PÁGINAS

A busca e o processamento das páginas acontecem de forma totalmente automatizada. Para que o processo se inicie, é necessário algumas URLs iniciais, também chamadas de *seeds URLs* (URL sementes). Neste trabalho foi utilizada apenas uma única URL semente, a página http://www.inf.furb.br. A partir dos *links* encontrados nesta página e nas páginas subseqüentes, foi possível encontrar todas as páginas que compõem a coleção de páginas do sistema de busca.

Toda página requisitada do servidor de URL passa pelos seguintes passos no *crawler*:

a) normalização e verificação da RFC3986 (2005): é utilizada a classe Java URL, a qual retorna a exceção “MalformedURLException” em caso de falha;

b) geração do identificador único da URL: assinatura computacional MD5 do *link*, utilizando a classe Java MessageDigest;

c) baixar o cabeçalho do *link* e verificar se é “text/html”: somente é capturado os primeiro bytes da uma página (o cabeçalho), e é analisado o campo *Content Type*;

d) capturar o conteúdo da página: após todas as verificações, a página é efetivamente capturada;

e) geração do identificador único do conteúdo da página: assinatura computacional MD5 do conteúdo da página, utilizando a classe Java MessageDigest;

f) buscar *links* dentro do conteúdo: para busca de novos *links* é utilizada a expressão regular: "href=\"[^\"]+\"";

g) atualizar páginas que são apontadas: atualiza referências entre os *links* para formação do *ranking* de conectividade;

h) retirar elementos HTML, caracteres especiais e converter para minúsculo todo o texto: deixa o conteúdo da página devidamente formatado para ser indexado. Para retirar os elementos HTML, é utilizada a biblioteca HTMLParser, disponível em http://htmlparser.sourceforge.net/;

i) gravar no banco de dados: todos os dados são gravados na base de dados do *crawler*.

3.4.4.2 ATUALIZAÇÃO DAS PÁGINAS

Por se tratar de um *crawler* incremental, é essencial que exista alguma métrica de atualização das páginas.

A política de atualização adotada foi a proporcional. Para tanto, é criado um identificador de freqüência de acesso para cada página, utilizando o campo rateUpdate. Esta freqüência indicará se uma página será mais ou menos atualizada.

Após o *crawler* atingir o seu número máximo de páginas, o mesmo irá começar a

atualizar a sua coleção local, com base nessa freqüência. Inicialmente, todas terão a mesma freqüência e serão atualizadas igualmente. De tempos em tempos (período definido em 24h), a freqüência é atualizada utilizando dois campos da base de dados: dataUpdate e countUpdate.

O campo dataUpdate corresponde à data e ao horário em que a página foi atualizada a última vez. Já o countUpdate, representa a quantidade de vezes que essa página foi atualizada com modificações do seu contéudo. Para saber se a página foi modificada ou não, utiliza-se a assinatura computacional da página (campo paginaID).

O rateUpdate é gerado com base na seguinte fórmula que pode ser vista no Quadro 18.

rateUpdate = countUpdate [mínimo 1 e máximo de 10] / (dataUpdate – dataAtual) [mínimo 1 e máximo 10, em dias]

Quadro 18 – Fórmula do rateUpdate

A variável dataAtual é a data e o horário em que a página está sendo atualizada. Exemplo:

O rateUpdate de uma página que já teve duas atualizações com modificações do seu conteúdo (countUpdate), sendo que a diferença entre a data da sua última atualização (dataUpdate) e a data atual (dataAtual) é de três dias, pode ser visto no Quadro 19.

rateUpdate = 2 / 3 rateUpdate = 0.6

Quadro 19 – Exemplo da fórmula

O rateUpdate pode variar entre 0.1 e 10. Essa possível variação quer dizer que uma página com rateUpdate de 2.0 será atualizada duas vezes mais que um página com rateUpdate de 1.0, e assim consecutivamente.

3.4.5 SORTER/INDEXER

O *sorter/indexer* é o elemento do sistema no qual o *web search* irá diretamente fazer a

busca por um consulta de um usuário. Ele é responsável por manter o índice *full-text search* das páginas e também por manter seus dados ordenados fisicamente pelo *ranking* de forma decrescente. Esta ordenação física acontece para que não seja necessário utilizar a cláusula "ORDER BY", o que fornece um ganho enorme de desempenho.

O *sorter/indexer f*oi escrito em PHP, utiliza MySQL como banco de dados e é executado a cada 24 horas. No Quadro 20 é mostrada a parte do seu código onde acontece a atualização da base de dados.

```
$time_start = getmicrotime();
$SQL = "SELECT urlID, url, dataUpdate, countLinks, pagina FROM crawler ORDER by countLinks DESC;";
$result = mysql_query($SQL, $conn);
$time_end = getmicrotime();
$time = $time_end - $time_start;
echo "Tempo SELECT: $time segundos\n";
$time_start = getmicrotime();
while ($row = mysql_fetch_array($result)) {
    $urlID = $row["urlID"];
    $url = $row["url"];
    $dataUpdate = $row["dataUpdate"];
    $countLinks = $row["countLinks"];
    $pagina = $row["pagina"];
    $sql = "INSERT INTO sorter_indexer (urlID, url, dataUpdate, countLinks, pagina)"
    $sql .= " VALUES ('$urlID', '$url', '$dataUpdate', $countLinks, '$pagina');";
    mysql_query($sql, $conn);
}
$time_end = getmicrotime();
$time = $time_end - $time_start;
echo "Tempo INSERT: $time segundos\n";
```

Quadro 20 – Trecho de código do *sorter/indexer*

Na Figura 14 pode ser observada uma parte de uma consulta (SELECT) da lista já organizada fisicamente.

url	dataUpdate	countLinks	pagina
http://www.websites.com.br:80	2006-06-05 08:04:34	2045	web sites factory the internet development compan...
http://www.terra.com.br:80	2006-06-04 02:14:49	1517	terra qual e a sua brasil buscador chat salas na...
http://www.phpbb.com.br:80	2006-06-04 02:13:40	1403	phpbb brasil o grupo baggio esta convidando pessoa...
http://www.blogger.com.br:80	2006-06-04 02:22:48	1393	blogger brasil os ultimos 10 blogs atualizados 2 ...

Figura 14 – Lista de dados

A atualização do *sorter/indexer* segue os seguintes passos:

a) busca dos dados ordenados pelo “countLinks” (*ranking*) do *crawler*;

b) inclusão dos dados;

c) geração do índice *full-text search*.

A geração dos índices *full-text search* (indexação e organização) é um processo demorado. Para torná-lo viável em um ambiente de produção, é necessário aplicar estratégias diferenciadas. Um exemplo é a utilização de várias máquinas. Outra estratégia é rodá-lo em paralelo sem afetar o ambiente de consulta. Como exemplo, pode ser apresentado o sistema google, que utiliza o algoritmo de ordenação *PageRank* e roda completamente em paralelo utilizando quatro máquinas. Todo o processo da sua coleção de páginas estimada em 12 milhões demora cerca de 24 horas (BRIN; PAGE, 1998, p. 14).

Neste sistema, o *sorter/indexer* é executando na mesma máquina que está rodando o seu respectivo *crawler*.

3.4.6 WEB SEARCH

A Figura do *web search* é responsável por fornecer os meios necessários para o usuário efetuar a sua busca. Ele fornece a transparência necessária ao sistema, onde faz toda a comunicação com o *sorter/indexer*.

Na Figura 15 pode ser observado a sua tela inicial.

Figura 15 – *Web search*

O sistema de busca faz suas requisições diretamente para o MYSQL do *sorter/indexer*.

No Quadro 21 é pode ser observado um trecho do código da página de busca.

```
if ($query) {
    $time_start = getmicrotime();

    $SQL = "SELECT urlID, url, dataUpdate, countLinks, MATCH (pagina) AGAINST ('$query' IN
    $result = mysql_query($SQL, $conn);

    $time_end = getmicrotime();
    $time = $time_end - $time_start;
    ?>
    <table width=100% border=0 cellpadding=0 cellspacing=0 bgcolor=#e5ecf9><tr><td bgcolor

    <table border=0 cellspacing=0 cellpadding=0 width=100%>
    <?
    $ver = 0;
    while ($row = mysql_fetch_array($result)) {
        $ver++;
        echo "<tr>";
        echo "<td>";
        $urlID = $row["urlID"];
        $url = $row["url"];
        $dataupdate = $row["dataUpdate"];
        $countlinks = $row["countLinks"];
        $score = $row["score"];
```

Quadro 21 – Trecho de código da página de busca

Ele pode ser acessado em: http://buscaweb.no-ip.org:8000/.

3.4.7 OPERACIONALIDADE DA IMPLEMENTAÇÃO

Neste tópico serão relatados todos os hardwares utilizados e o tempo gasto para a realização de cada processo. Por fim, demonstrados exemplos de uma busca final.

No Quadro 22 é listado todo o equipamento utilizado, juntamente com informações sobre processador e memória.

Modulo do sistema	**Modelo do computador**	**Processador**	**Memória**
Servidor de URL	Pentium IV	3Ghz	1GB RAM
Crawler 01	Pentium IV	3Ghz	1GB RAM
Crawler 02	AMD Duron	800Mhz	512MB RAM

Quadro 22 – Máquinas utilizadas

Utilizando uma conexão de 512kbps foi possível indexar em média três páginas por segundo. Este limite se dá por causa da conexão e não por deficiência de processador ou memória. Durante todo o processo o uso de recursos da CPU ficou entre 10% e 50%.

Os *crawlers* encontraram e enviaram para o servidor de URL um total de 1.812.143 *links* diferentes. De todos eles, foram acessados 936.956 *links*, mas somente os tipos “text/html” foram devidamente capturados por completo, padronizados e indexados. De todos os *links* acessados, foram encontrados 130 tipos diferentes (MIME-TYPES) de conteúdo. No Quadro 23 pode ser encontrada a lista com os dez tipos que mais aparecem, juntamente com a quantidade de itens encontrados.

Tipo	**Quantidade**
text/html	609.212
erro/404	42.608
text/css	21.227
application/pdf	18.473
image/jpeg	10.597
application/msword	5.196
application/x-zip-compressed	3.146

application/zip	2.295
application/octet-stream	2.245
image/gif	1.578

Quadro 23 – Tipo x quantidade

Para esta coleção de 609.212 páginas (text/html), os tempos de execução do processo de organização e indexação do *sorter/indexer* foram:

a) tempo do SELECT (recuperar dados ordenados do *crawler*): 943,89 segundos;

b) tempo do INSERT (inserir na base do *sorter/indexer*): 2.200,21 segundos;

c) tempo da geração do índice *full-text search*: 3.189,88 segundos.

Todo o espaço em disco utilizado é apresentando no Quadro 24.

Recurso	**Tamanho**
crawlers dados	3.002 MB
crawlers índices	29.269 KB
servidor URL dados	302.217 KB
servidor URL índices	106.612 KB
sorter/indexer dados	2.053 MB
sorter/indexer índices	1.377 MB

Quadro 24 – Espaço utilizado

Só após todos esses processos é possível efetuar a busca. No Quadro 25 são apresentados os tipos de buscas disponíveis no *web search*.

Exemplo de busca	**Resultado**
sistemas de busca	busca por qualquer palavra
+sistemas +de +busca	busca por todas as palavras
+sistemas -busca	busca por uma palavra sim e outra não
"sistemas de busca"	busca por frases

Quadro 25 – Tipos de busca disponíveis

No Quadro 26 são demonstradas os quatro tipos de buscas, bem como o tempo de execução para cada uma delas. Cada busca é executada duas vezes seguidas, buscando assim testar a eficiência do *cache* no MYSQL.

busca	**tempo de retorno (segundos)**	**tempo de retorno da segunda busca (segundos)**
busca por qualquer palavra: **café carro bolo**	0.1969	0.0020
busca por qualquer palavra:	0.1900	0.0019

cachorro gato		
busca por todas as palavras: **+informatica +banco +dados**	0.2037	0.0021
busca por todas as palavras: **+aparelho +auditivo**	0.3235	0.0019
busca por uma palavra sim e outra não: **+navegador –firefox**	0.1052	0.0019
busca por uma palavra sim e outra não: **+celular -compra**	0.2185	0.0019
busca por frase: **“sistema de busca”**	0.1076	0.0021
busca por frase: **“quero café”**	17.86	0.0012

Quadro 26 – Exemplos de buscas

Na Figura 16 é apresentada a tela de um busca.

Figura 16 – Tela de uma busca

Para cada *link* do resultado são mostrados o seu ranking e data da última atualização. Existe também um *link* relacionado a sua conectividade, mostrando todos os *links* para os quais ele aponta (saída) e os que apontam para ele (entrada), conforme apresentado na Figura 17.

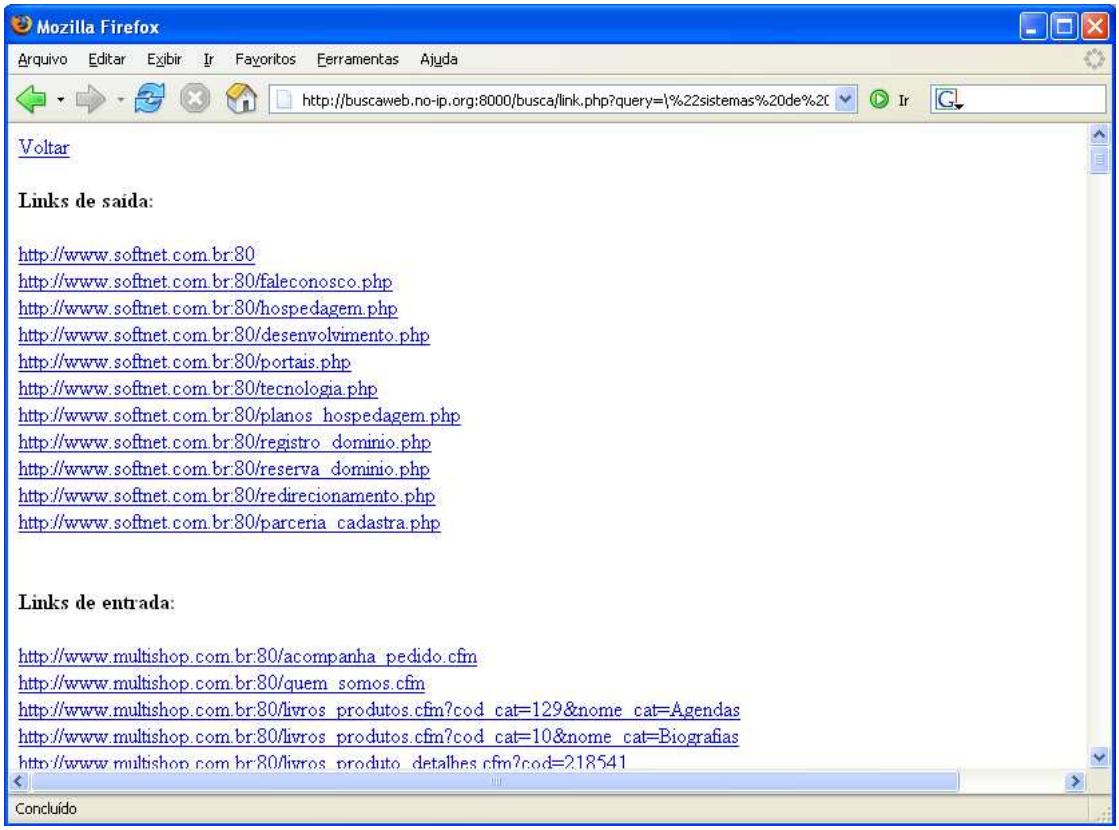


Figura 17 – Informações sobre um *link*

3.5 RESULTADOS E DISCUSSÃO

Existem algumas limitações no sistema de busca desenvolvido. Entre elas estão o servidor de URL como ponto de gargalo, a ausência de um algoritmo melhor de ranqueamento e o *sorter/indexer* que não é executado em paralelo, resultando em uma perda de desempenho quando esta sendo executado. Outro ponto a ser destacado é a ausência de semântica, esse tipo de sistemas de busca baseado unicamente em informações léxicas e sintáticas é uma realidade atual, mais existe uma forte tendência para que informações semânticas sejam incorporadas a sua busca, assim, retornando uma busca mais perto do desejado.

A maioria dos problemas está relacionada à otimização. Eles podem ser resolvidos com a utilização de novas técnicas para suprir partes do sistema que tem perda de

performance ou não estão respondendo corretamente. Como exemplo pode ser citado o *ranking*, que funciona perfeitamente, mas poderia ser aprimorado com a utilização de filtros e técnicas mais avançadas.

Também é importante comentar sobre a busca por frases. Foi detectado que uma busca *full-text search* tem um menor desempenho para buscas por frases. Pode ser observada uma busca que demorou 17 segundos, para a frase “quero café”.

Esse tempo se dá por dois motivos: pelo fato de uma busca de frases precisar acessar maiores informações (a posição das palavras) e principalmente por causa de o resultado realmente existir mas ser encontrado apenas em uma profundidade maior do índice. Esta segunda situação pode acontecer para qualquer tipo de busca.

O fato de uma busca demorar mais quando necessita acessar uma profundidade maior do índice é um fato curioso que acontece neste sistema de busca, e pode ser encontrado também em um sistema de busca já consagrado no mercado, o Google. No sistema de busca Google, quando uma busca é efetuada, mesmo que ele retorne bilhões de resultados, você só poderá ver e ter acesso à no máximo mil resultados, um número realmente pequeno comparado a quantidade de resultados disponíveis. Também, quanto mais perto estiver desse valor máximo de resultados, maior será o tempo gasto para a busca.

Por exemplo, ao efetuar uma busca no Google pela palavra “informática”, os primeiros resultados (1 à 10), demoram 0,06 segundos para aparecerem. Já ao requisitar a última página possível neste caso (811 até 820) dos 232 milhões encontrados, a mesma busca demora 0,74 segundos.

Esta estratégia de limitar o número de resultados é utilizada para evitar que a busca precise percorrer todo o índice, assim causando uma perda significativa no desempenho da busca.

O *cache* do MYSQL se demonstrou em todos os casos eficaz, e uma dificuldade

encontrada no desenvolvimento do trabalho foi a falta de bibliografia sobre esse assunto na língua portuguesa.

CONCLUSÕES

O objetivo inicial de implementar um sistema de busca utilizando o MYSQL como banco de dados foi alcançado. O *crawler* e todos os seus elementos necessários, juntamente com a interface para a pesquisa, foram construídos e foi possível executar a busca de uma grande quantidade de páginas com um tempo de resposta bastante aceitável.

Foram analisadas e indexadas centenas de milhares de páginas brasileiras (domínio .br), bem como o trabalho de mantê-las atualizadas de forma incremental. Foi possível também manter uma relação completa entre todos os *links*, para utilização do *ranking*.

A utilização do MYSQL se tornou bastante satisfatória, atendendo as expectativas de velocidade e fornecendo as ferramentas necessárias para criação de um sistema de busca. O que tornou a implementação mais rápida, pelo fato de se poupar o trabalho de programação de itens essências para o armazenamento e a busca, como o armazenamento seqüencial de dados e o índice invertido.

A criação deste sistema foi um desafio bastante gratificante, pelo fato de possibilitar um maior entendimento do funcionamento de pesquisas em grandes bases de dados e de suas peculiaridades. Foi também possível adquirir vários novos conhecimentos sobre o funcionamento da web na internet, suas linguagens e forma de organização.

A principal contribuição deste trabalho foi mostrar o funcionamento completo de um sistema de busca, do início ao fim, fornecendo detalhes dos seus elementos e das ferramentas necessárias para se tornar viável tal projeto.

Este trabalho desenvolvido é uma introdução e ao mesmo tempo um exemplo completo das ferramentas necessárias para se construir um sistema de busca na web.

4 EXTENSÕES

Como possíveis extensões para o trabalho, destacam-se as seguintes melhorias do sistema de busca:

a) distribuir o servidor de URL, tornando-o escalável;
b) aprimorar o *ranking*, possibilitando um retorno mais próximo do desejo do usuário (por exemplo: utilizando *tags* HTML, distanciamento entre as palavras de uma busca e semântica);
c) possibilitar que o *sorter/indexer* seja executado de maneira paralela;
d) separar partes do HTML para terem mais importância em uma busca (exemplo: título da página).

REFERÊNCIAS BIBLIOGRÁFICAS

ALTAVISTA: web search. [S.l.], 2006. Disponível em: <http://www.altavista.com>. Acesso em: 05 jun. 2006.

ARASU, Arvind et al. Searching the web. **ACM Transactions on Internet Technology**, [S.l.], v. 1, n. 1, p. 2-43, Aug. 2001. Disponível em: <http://portal.acm.org/toc.cfm?id=383034&coll=GUIDE&dl=GUIDE&type=issue&idx=J780&part=periodical&WantType=periodical&title=ACM%20Transactions%20on%20Internet%20Technology%20%28TOIT%29&CFID=44436838&CFTOKEN=65054141>. Acesso em: 02 maio 2005.

BHARAT, Krishna et al. The connectivity server: fast access to linkage information on the web. In: INTERNATIONAL WORLD WIDE WEB CONFERENCE, 7., 1998, Brisbane, Australia. **Proceedings…** Amsterdam: Elsevier Science, 1998. p. 1-7. Disponível em: <http://www.cindoc.csic.es/cybermetrics/pdf/11.pdf>. Acesso em: 02 maio 2006.

BRIN, Sergey; PAGE, Lawrence. The anatomy of a large-scale hypertextual web search engine. In: INTERNATIONAL WORLD WIDE WEB CONFERENCE, 7., 1998, Brisbane, Australia. **Proceedings…** Amsterdam: Elsevier Science, 1998. p. 6-12. Disponível em: <http://www-db.stanford.edu/pub/papers/google.pdf>. Acesso em: 03 maio 2005.

CHO, Junghoo. **Crawling the web**: discover and maintenance of large-scale web-data. 2001. 172 f. Thesis (Ph.D. in Computer Science) – Department of Computer Science, Stanford University, California. Disponível em: <http://oak.cs.ucla.edu/~cho/papers/cho-thesis.pdf>. Acesso em: 03 maio 2005.

CHO, Junghoo; GARCIA-MOLINA, Hector. The evolution of the web and implications for an incremental crawler. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES, 21., 2000, Cairo, Egypt. **Proceedings...** San Francisco: Morgan Kaufmann, 2000. p. 15-19. Disponível em: <http://www-db.stanford.edu/~cho/papers/incre.pdf>. Acesso em: 02 maio 2005.

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. **Distributed systems**: concepts and design. England: Harlow, 2001.

ENTERPRISE ARCHITECT: full lifecycle UML mode. [S.l.], 2005. Disponível em: <http://www.sparxsystems.com.au/>. Acesso em: 08 jun. 2005.

GARCIA-MOLINA, Hector; ULLMAN, Jeffrey D.; WIDOM, Jennifer. **Implementação de sistemas de bancos de dados**. Tradução de Vandenberg D. de Souza. Rio de Janeiro: Campus, 2001.

GOOGLE: web search. [S.l.], 2006. Disponível em: <http://www.google.com>. Acesso em: 05 jun. 2006.

HIRA, Jun et al. Webbase: a repository of web pages. In: INTERNATIONAL WORLD WIDE WEB CONFERENCE, 9., 2000, Amsterdam, Nederlanden. **Proceedings…** Amsterdam: Elsevier Science, 2000. p. 2-7. Disponível em: <http://www.almaden.ibm.com/cs/people/rsriram/pubs/archiver.pdf>. Acesso em: 05 jun. 2006.

MYSQL REFERENCE MANUAL: full-text search functions. [S.l.], [2005?]. Disponível em: <http://dev.mysql.com/doc/mysql/en/fulltext-search.html>. Acesso em: 09 maio 2005.

MYSQL REFERENCE MANUAL: full-text searches with query expansion. [S.l.], [2006?]. Disponível em: <http://dev.mysql.com/doc/refman/5.0/en/fulltext-query-expansion.html>. Acesso em: 11 maio 2006.

RFC3986: A Uniform Resource Identifier (URI). [S.l.], 2005. Disponível em: <http://www.gbiv.com/protocols/uri/rfc/rfc3986.html>. Acesso em: 08 maio 2006.

TANENBAUM, Andrew S.; STEEN, Maarten. **Distributed systems**: principles and paradigms. New Jersey: Upper Saddle River, 2002.

THE WEB ROBOTS PAGE: the robots web page. [S.l.], [2006?]. Disponível em: <http://www.robotstxt.org/wc/robots.html>. Acesso em: 05 jun. 2006

YAHOO!: web search. [S.l.], 2006. Disponível em: <http://www.yahoo.com>. Acesso em: 05 jun. 2006.

WIKIPEDIA: Inverted Index. [S.l.], 2006. Disponível em: <http://en.wikipedia.org/wiki/Inverted_index>. Acesso em: 09 maio 2006.