

Googol: Motor de pesquisa de páginas Web

Sistemas Distribuídos 2023/24 — Meta 1 — 5 de abril de 2024, 20h00

Resumo

Este projeto tem como objetivo criar um motor de pesquisa de páginas Web. Pretende-se que tenha um subconjunto de funcionalidades semelhantes aos serviços Google.com, Bing.com e GMX.com, incluindo indexação automática (*Web crawler*) e busca (*search engine*). O sistema deverá ter diversas informações relevantes sobre cada página, tais como o URL, o título, uma citação de texto e outras que considere importantes. Ao realizar uma busca, obtém-se a lista de páginas que contenham as palavras pesquisadas, ordenada por relevância. Os utilizadores podem sugerir URLs para serem indexados pelo sistema. Partindo desses URLs, o sistema deve indexar recursivamente/iterativamente todos as ligações encontradas em cada página.

1 Objetivos do projecto

No final do projeto cada estudante deverá ter:

- programado um sistema pesquisa de páginas Web com arquitetura cliente-servidor;
- usado sockets Multicast para comunicação fiável entre servidores;
- criado uma camada de acesso aos dados usando Java Remote Method Invocation;
- aplicado processamento paralelo de dados para melhorar o desempenho;
- garantido a disponibilidade da aplicação através de redundância.

2 Visão geral

Um motor de pesquisa sobre a Web permite aos utilizadores realizarem pesquisas por páginas Web, obtendo informação organizada sobre as páginas que contenham os termos pesquisados. Este projeto tem como objetivo criar um sistema que faça indexação automática de páginas Web e que permita aos utilizadores realizarem pesquisas sobre o índice construído.

O acesso e a leitura das páginas Web será realizada através da biblioteca open source jsoup que permite obter e extrair informação de documentos HTML. Para realizar pesquisas, o sistema deve armazenar informação na forma de um *inverted index*¹. Um

¹ Coulouris, G. et al. *Distributed systems: concepts and design*, Ch. 21, Google Case Study, 2005.

índice invertido identifica, para cada palavra, as páginas em que ocorre. É por isso semelhante ao índice remissivo de um livro. Na linguagem Java, uma forma simples de organizar esta informação é utilizar a estrutura

```
HashMap<String, HashSet<String>> index;
```

para associar a cada palavra uma lista de URLs. Assim, por exemplo, ao termo “universidade” associamos os URLs “https://en.wikipedia.org/wiki/University_of_Coimbra”, “http://www.uc.pt”, etc. É possível (e recomendável) fazer-se uso de estruturas de dados mais completas, a desenvolver no âmbito do projeto. É comum, por exemplo, guardar-se também as hiper-ligações que apontam *para* cada página. Outro exemplo é a adoção de *document ids* para evitar repetir muitas vezes cada URL completo.

O sistema a desenvolver deverá ter toda a informação relevante sobre as páginas Web. Deve incluir informações tais como: URL, título da página, citação de texto e palavras que sejam encontradas no documento HTML. Estas são as informações apresentadas pelos motores de busca típicos.

Ao realizar uma pesquisa, o utilizador insere um conjunto de palavras (termos de pesquisa) e obtém a listagem de páginas que contenham ocorrências dessas palavras. A listagem deve ser ordenada pelo número de ligações *de* outras páginas *para* cada página apresentada. Os resultados devem ser agrupados de 10 em 10 e deve ser possível avançar nos mesmos (tal como um motor de busca habitual).

3 Funcionalidades a desenvolver

O sistema tem utilizadores que acedem às funcionalidades usando a aplicação Googol a desenvolver. Deverá ser possível realizar as seguintes operações:

1. **Indexar novo URL.** Um utilizador deve poder introduzir manualmente um URL para ser indexado. Esse URL será então visitado por um Downloader (*Web crawler*) e ficará associado, no índice invertido, às palavras que forem encontradas no texto.
2. **Indexar recursivamente todos os URLs encontrados.** O indexador automático deve visitar os URLs que forem encontrados em páginas previamente visitadas. Sugere-se a utilização de uma fila de URLs para este efeito (processo que na verdade é, em si, iterativo).
3. **Pesquisar páginas que contenham um conjunto de termos.** Qualquer utilizador deve poder realizar uma pesquisa. Para tal, o motor de busca consulta o índice invertido e apresenta a lista de páginas que contenham *todos* os termos da pesquisa. Para cada resultado da pesquisa, deve mostrar o título da página, o URL completo e uma citação curta composta por texto da página. Os resultados da pesquisa devem ser agrupados de 10 em 10.
4. **Resultados de pesquisa ordenados por importância.** Os resultados de uma pesquisa (funcionalidade anterior) devem ser apresentados por ordem de relevância. Para simplificar, considera-se que uma página é mais relevante se tiver mais ligações *de* outras páginas. Assim, o indexador automático deve manter, para cada URL, a lista de outros URLs que fazem ligação para ele.

5. **Consultar lista de páginas com ligação para uma página específica.** É possível saber, para cada página, todas as ligações conhecidas que apontem para essa página. Esta funcionalidade pode estar associada à funcionalidade de pesquisa (por exemplo, uma opção associada a cada resultado).
6. **Página de administração atualizada em tempo real.** Todos os utilizadores têm acesso a uma opção de consulta de informações gerais sobre o sistema. Esta informação será atualizada apenas quando houver alterações (sem refrescamento periódico). Pretende-se saber o estado do sistema, designadamente as 10 pesquisas mais comuns, a lista de Barrels ativos, e o tempo médio de resposta a pesquisas medido pela Gateway discriminado por Barrel (em décimas de segundo).

4 Arquitetura

A Figura 1 mostra a arquitetura global do projeto. A aplicação **Googol** consiste em quatro programas: **Downloaders**, **Index Storage Barrels**, **RMI Gateway** e **RMI Client**. Cada grupo deverá desenvolver estes programas, sendo que os **Downloaders** trabalham em **paralelo** (para aumentar o desempenho) e os **Barrels** **também** (para aumentar a tolerância a falhas e o desempenho).

Toda a **informação** está **armazenada** nos **Storage Barrels** (em ficheiro de texto, ficheiro de objetos, base de dados, O/R mapping, etc.). Estes servidores são réplicas que contêm exatamente a mesma informação, servindo para manter o sistema a funcionar desde que uma réplica esteja ativa (podendo as outras avariar ou serem desligadas). Os **Storage Barrels** **recebem informação processada pelos Downloaders através de Multicast** (one-to-many) usando um protocolo a construir pelos estudantes, que deve contemplar dois Barrels redundantes.

Cada URL é indexado apenas por um **Downloader** que irá fazer multicast dos resultados para os **Storage Barrels**. Assim, os Downloaders executam em paralelo para melhoria de desempenho. Fazem uso de uma **fila de URLs** para escalonarem as futuras visitas a páginas, sendo que esta fila pode ser inteiramente desenhada pelos alunos sem quaisquer restrições.

Os clientes, por sua vez, comunicam apenas com a RMI Gateway, que serve assim de porta de entrada no sistema. Esta Gateway escolhe aleatoriamente (ou round-robin, ou por desempenho) um Storage Barrel ativo para responder a cada pesquisa. A comunicação entre Gateway e Storage Barrels é feita via RMI – a pesquisa é assim feita através de um programa distribuído. Uma solução é a Gateway receber uma referência remota para os Barrels para fazer pedidos por *callback* RMI. Assim, deverão ser criados os seguintes programas:

- **Downloader** – Componentes que obtêm as páginas Web executando em paralelo, as analisam (usando o jsoup) e atualizam o índice através de Multicast.

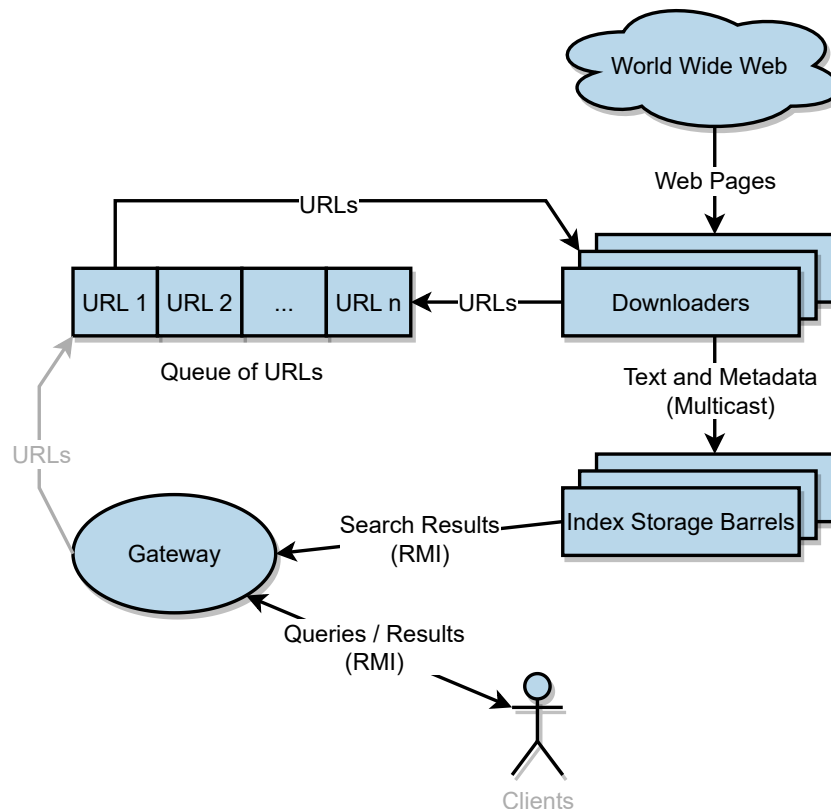


Fig. 1: Arquitetura do sistema.

- **Index Storage Barrel** – É o servidor central (replicado) que armazena todos os dados da aplicação, recebendo os elementos do índice (palavras e URLs) através de Multicast, enviados pelos Downloaders. Para tal, deverão aplicar um protocolo de multicast fiável, uma vez que os storage barrels devem ter informação idêntica ainda que possam existir avarias de omissão (poderá fixar-se o número de réplicas a duas).
- **RMI Gateway** – Este é o componente visível pelos clientes. Comunica com os Storage Barrels usando RMI (note que uma opção é usar RMI callbacks para cada Barrel). Esta RMI Gateway não tem de armazenar quaisquer dados, dependendo inteiramente dos Storage Barrels para satisfazer os pedidos dos clientes. Uma possibilidade será escolher aleatoriamente um Storage Barrel para cada pesquisa, ou balancear a carga de forma mais sofisticada.
- **RMI Client** – É o cliente RMI usado pelos utilizadores para aceder às funcionalidades do Googol. Pretende-se que este cliente tenha uma UI bastante simples e que se limite a invocar os métodos remotos no servidor RMI.
- **URL Queue** – Deverá existir um ou mais componentes capazes de guardar os URLs encontrados pelos Downloaders na forma de uma fila. Esta fila pode ser um programa separado dos restantes, embora tal não seja obrigatório. Alternativamente,

pode ser incorporado na Gateway ou inclusivamente nos Barrels.

5 Protocolo UDP Multicast

O protocolo de comunicação Multicast deverá ser especificado em detalhe. Segue-se uma recomendação da base de um protocolo a ser completado pelos estudantes. A estrutura principal deste protocolo é a mensagem que consiste num conjunto não ordenado de pares chave-valor, semelhante a um HashMap em Java ou a um dicionário em Python, finalizados com uma mudança de linha. Um exemplo seria:

```
chave1 | valor1; chave2 | valor dois
```

Como tal, não é permitido ter o carácter “pipe” (|) nem o carácter ponto e vírgula (;) nem o carácter mudar de linha (\n) no nome da chave nem no valor.

Todas as mensagens têm um campo obrigatório chamado “type”. Este valor permite distinguir o tipo de operação que o cliente pretende fazer no servidor, ou o tipo de resposta que o servidor está a dar. Um exemplo mais realista é portanto:

```
type | login; username | tintin; password | unicorn
```

E a resposta respetiva, sendo que o campo msg é opcional, mas aceite.

```
type | status; logged | on; msg | Welcome to the app
```

Finalmente, para representar listas de elementos é usado o tamanho e o contador de elementos. O tamanho é descrito num campo com o sufixo *x_count*, onde *x* é o campo com a lista, e cada campo do elemento tem o formato *x_i_campo*, onde *i* é o índice da lista, a começar em 0. Um exemplo encontra-se de seguida:

```
type | url_list; item_count | 2; item_0_name | www.uc.pt;  
    item_1_name | www.dei.uc.pt
```

Este protocolo deverá ser completado e especificado pelos alunos no decorrer do projeto. Para facilitar o desenvolvimento, podem usar o código multicast da ficha prática laboratorial, que permite escrever e ler strings.

Quando um Downloader processa um URL, começa por extrair as palavras e adiciona esse mesmo URL ao conjunto de URLs de cada palavra encontrada. Seguidamente, extrai os URLs encontrados na página (*links*) e coloca-os numa fila (ou outra estrutura equivalente) para serem visitados posteriormente. Cada Downloader deve encaminhar alguns URLs para outros Downloaders, por forma a distribuir trabalho e melhorar o desempenho, embora esta distribuição fique ao critério de cada grupo.

6 Requisitos não-funcionais

A aplicação deverá lidar corretamente com quaisquer exceções que estejam previstas. Por forma a garantir que o Googol está sempre disponível para os utilizadores, deverá usar uma solução de failover para garantir que a aplicação continua a funcionar ainda que um servidor qualquer possa avariar.

6.1 Tratamento de exceções e failover

Como o hardware pode falhar, é necessário que os utilizadores não notem nenhuma falha de serviço. Como tal, no caso de um Storage Barrel falhar, é preciso garantir que as introduções de URLs para indexação não se percam (realizando *retries* sempre que necessário). Caso um Barrel avarie a meio de uma pesquisa (retornando *RemoteException*) a Gateway deverá recuperar fazendo o mesmo pedido a outro Barrel.

Também do lado dos clientes é possível que a ligação se perca a meio ou que a Gateway avarie e recupere. É necessário garantir que nenhuma falha do lado do cliente deixe qualquer operação a meio. Em caso de avaria devem reiniciar e fazer *retry*.

6.2 Relatório

Devem reservar tempo para a escrita do relatório no final do projeto, tendo em conta os passos anteriores. Devem escrever o relatório de modo a que um novo colega que se junte ao grupo possa perceber a solução criada, as decisões técnicas e possa adicionar novos componentes ou modificar os que existem. O relatório pode ser inteiramente escrito em Javadoc no código-fonte apresentado pelos estudantes. Deve incluir:

- Arquitetura de software detalhadamente descrita. Deverá ser focada a estrutura de threads e sockets usadas, bem como a organização do código.
- Detalhes sobre o funcionamento da componente Multicast (Downloaders e Barrels). Deve especificar o protocolo usado para comunicação multicast.
- Detalhes sobre o funcionamento da componente RMI. Deverá explicar detalhadamente o funcionamento dos métodos remotos disponibilizados e eventuais callbacks usados, bem como a solução usada para failover.
- Distribuição de tarefas pelos elementos do grupo.
- Descrição dos testes realizados (tabela com descrição e pass/fail de cada teste).

6.3 Distribuição de tarefas

De modo a que a avaliação seja justa num trabalho de grupo, é fundamental uma divisão justa do trabalho. Dado que a nota resultante da defesa será individual, são propostas as duas possíveis divisões de trabalho:

- Elemento 1 será responsável pelos Downloaders e pela componente multicast dos Barrels, e o elemento 2 pela Gateway e pela componente RMI dos Barrels.
- Cada um dos elementos ficará com igual número de funcionalidades a desenvolver, trabalhando um pouco em cada um dos programas.

Finalmente, poderão ser aceites outras distribuições, desde que previamente acordadas com os docentes.

7 Planos futuros para o projeto

Na segunda meta do projeto irão expandir a presente solução, adicionando uma interface Web usando HTML/Spring e irão integrar a aplicação com uma API REST de um serviço externo. Nessa fase, o servidor Web irá usar a API do servidor RMI aqui criado.

8 O analisador de HTML *jsoup*

Deverão usar a biblioteca *jsoup* para extrair o texto e as ligações presentes no HTML. A biblioteca consiste num ficheiro JAR que pode ser obtido em <https://jsoup.org/> juntamente com toda a documentação. Segue-se um curto exemplo que estabelece uma ligação para obter o HTML, lista algumas palavras e apresenta as ligações que encontrar:

```
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;
import java.io.IOException;
import java.util.StringTokenizer;

public class Mini {
    public static void main(String args[]) {
        String url = args[0];
        try {
            Document doc = Jsoup.connect(url).get();
            StringTokenizer tokens = new StringTokenizer(doc.text());
            int countTokens = 0;
            while (tokens.hasMoreElements() && countTokens++ < 100)
                System.out.println(tokens.nextToken().toLowerCase());
            Elements links = doc.select("a[href]");
            for (Element link : links)
                System.out.println(link.text() + "\n" + link.attr("abs:href") + "\n");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

9 Entrega do projeto

O projeto deverá ser entregue num arquivo ZIP. Esse arquivo deverá conter um ficheiro README.md ou .txt com toda a informação necessária para instalar e executar o projeto sem a presença dos alunos. Projetos sem informações suficientes, que não compilem ou não executem corretamente **não serão avaliados**. O ficheiro ZIP deverá ter também **uma pasta com o código fonte completo do projeto**. A ausência deste elemento levará à anulação do projeto.

O ficheiro ZIP deverá conter um Javadoc/HTML com o relatório. O relatório deve seguir a estrutura fornecida, dado que a avaliação irá incidir sobre cada um dos pontos.

O projeto será entregue na plataforma Infoestudante até ao dia **5 de abril de 2024, 20h00**, via <https://infoestudante.uc.pt>