ucBusca: Motor de pesquisa de páginas Web

Sistemas Distribuídos 2019/20 — Meta 1 — 26 de outubro de 2019 (21:59)

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 DuckDuckGo.com, incluindo indexação automática (*Web crawler*) e busca (*search engine*). O sistema deverá ter todas as informações relevantes sobre as páginas, tais como o URL, o título, uma citação de texto e outras que considere importantes. Ao realizar uma busca, um utilizador obtém a lista de páginas que contenham as palavras pesquisadas, com as informações pretendidas, e o número aproximado de resultados da pesquisa. Apenas os administradores podem introduzir URLs específicos para serem indexados pelo sistema. Partindo desses URLs, o sistema deve indexar recursivamente todos as ligações encontradas em cada página indexada.

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 TCP, UDP e Multicast para comunicação entre clientes e servidores;
- seguido um modelo multithread para desenhar os servidores;
- criado uma camada de acesso aos dados usando Java RMI;
- garantido a disponibilidade da aplicação através de redundância com failover.

2 Visão geral

Um motor de pesquisa tal como o Yahoo.com 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 realizar pesquisas sobre o índice resultante.

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

invertido identifica, para cada palavra, as páginas em que ocorre. É por isso semelhante a um 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*. Uma alternativa que permite URLs duplicados é

HashMap< String, ArrayList<String>> index;

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.

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. Deve ser apresentada a contagem total de resultados da pesquisa.

A funcionalidade de pesquisa estará disponível para quaisquer utilizadores mas todas as outras funcionalidades requerem registo e login com password. Um utilizador deve registar-se para usar o sistema. Apenas utilizadores com privilégios de administrador podem acrescentar informações e obter informações sobre o estado do sistema. O primeiro utilizador registado no sistema é automaticamente administrador.

3 Funcionalidades a desenvolver

O sistema tem *utilizadores* que acedem às funcionalidades usando a aplicação ucBusca a desenvolver. Alguns dos utilizadores são também *administradores* que podem acrescentar e consultar informações internas. Deverá ser possível realizar as seguintes operações:

1. **Registar pessoas.** Os utilizadores devem poder registar-se, sendo que o acesso à aplicação deve estar protegido com username e password. Deverá guardar toda a informação pessoal que considere necessária bem como uma password (código de acesso). Para simplificar, considere que o primeiro utilizador a registar-se tem automaticamente privilégios de administrador em todo o sistema.

2. **Indexar novo URL.** Um administrador deve poder introduzir manualmente um URL para ser indexado. Esse URL será então visitado pelo indexador automático (*Web crawler*) e as palavras que forem encontradas no texto serão acrescentadas ao índice invertido.

3. **Indexar recursivamente todos os URLs encontrados.** O indexador automático deve visitar os URLs que forem encontrados em páginas previamente visitadas. Assim, o índice é construído recursivamente. Sugere-se a utilização de uma fila de URLs para este efeito.

4. **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 contêm 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. Deve ser apresentado o número total aproximado de resultados da pesquisa. Esta funcionalidade é a única que deve estar disponível para utilizadores que não tenham login efetuado (utilizadores anónimos).

5. **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.

6. **Consultar lista de páginas com ligação para uma página específica.** Aos utilizadores que tenham registo e login efetuado é possível saber, para cada página, todas as ligações conhecidas que apontem para essa página. Esta funcionalidade pode estar associada à funcionalidade de pesquisa.

7. **Consultar pesquisas realizadas anteriormente.** Os utilizadores que tenham registo e login efetuado podem consultar todas as pesquisas que os próprios tenham feito anteriormente.

8. **Página de administração atualizada em tempo real.** Os administradores têm acesso a uma opção de consulta de informações gerais sobre o sistema. Esta informação será atualizada apenas quando houver atualizações. Pretende-se saber as 10 páginas mais importantes, as 10 pesquisas mais comuns realizadas pelos utilizadores e a lista de servidores multicast ativos (IP e porto).

9. **Dar privilégios de administrador a um utilizador.** Aos utilizadores que tenham privilégios de *administrador* é possível conceder esses mesmos privilégios a outros utilizadores. Para tal, escolhem o utilizador que pretendem promover a administrador e esse utilizador passa a poder alterar informações no sistema.

10. **Notificação imediata de privilégios de administrador.** Quando um utilizador é promovido a editor deve receber imediatamente (em tempo real) essa informação se estiver ligado à aplicação, sem precisar de realizar nenhuma operação.

11. **Entrega posterior de notificações a utilizadores desligados.** Qualquer utilizador que devesse ter recebido uma notificação imediata, mas não se encontrasse ligado à aplicação (offline), recebe a notificação assim que se ligar a próxima vez.

4 Arquitetura

A Figura 1 mostra a arquitetura global do projeto. A aplicação ucBusca consiste em três programas: servidor RMI, servidor Multicast e cliente RMI. Cada grupo deverá programar os *servidores RMI*, que são idênticos embora um seja inicialmente primário e outro

secundário. Cada grupo deverá igualmente programar os *servidores Multicast*, que são idênticos exceto possíveis questões de configuração. Cada grupo deverá programar um *cliente RMI*, à parte, que se liga aos servidores RMI e permite aos utilizadores acederem ao ucBusca.

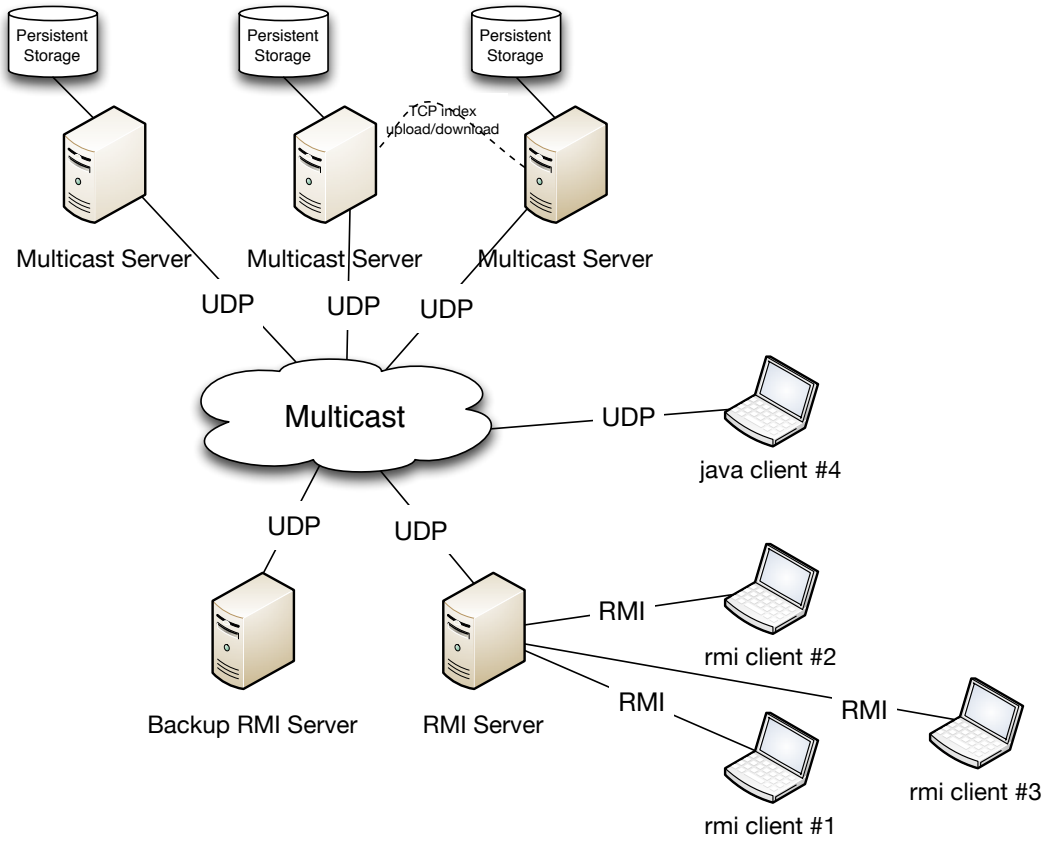

Fig. 1: Arquitetura do sistema.

Toda a informação está armazenada nos servidores Multicast (em ficheiro de texto, ficheiro de objetos, base de dados, O/R mapping, *etc*.). Estes servidores são réplicas que contêm a mesma informação, servindo para manter o sistema a funcionar desde que uma réplica esteja ativa (podendo as outras avariar ou serem desligadas). Os servidores recebem pedidos e enviam respostas através de comunicação Multicast (one-to-many) usando um protocolo a construir pelos estudantes. Pretende-se que cada URL seja indexado apenas por um servidor Multicast e que, perioricamente, esse servidor sincronize a sua parte do índice com os outros servidores Multicast.

Os servidores RMI não guardam nenhuma informação persistente e usam a rede multicast para enviar pedidos aos servidores Multicast. O sistema deverá aceitar qualquer número de clientes RMI, que consistem numa pequena aplicação que permite aceder ao ucBusca. Pretende-se com isto que os clientes RMI sejam muito simples, com uma interface elementar, ainda que na prática se aceite outras soluções. Assim, deverão ser criados os seguintes programas:

- Multicast Server – É o servidor central (replicado) que armazena todos os dados da aplicação, suportando por essa razão todas as operações necessárias através de pedidos recebidos em datagramas multicast.
- RMI Server – Existem dois servidores RMI que não armazenam dados localmente e que disponibilizam, através da sua interface remota, um conjunto de métodos acessíveis a aplicações com Java RMI. Quando um método é invocado, o servidor RMI traduz esse pedido num datagrama UDP enviado por multicast para os servidores Multicast, aguardando depois pela resposta enviada igualmente por multicast.
- RMI Client – É o cliente RMI usado pelos utilizadores para aceder às funcionalidades do ucBusca. Pretende-se que este cliente tenha uma interface bastante simples e que se limite a invocar os métodos remotos no servidor RMI, lidando corretamente com avarias do servidor primário mudando para o servidor secundário.

5 Protocolo UDP Multicast

De forma a permitir a implementação de clientes ucBusca para qualquer plataforma (seja Windows, Linux, macOS, Android, iOS), o protocolo 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 ucBusca
```

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, é fornecido juntamente com o enunciado um cliente de teste em Java que permitirá comunicar diretamente com os servidores Multicast (e que está identificado por "java client #4" no diagrama da arquitetura).

Sendo os servidores Multicast replicados, cada pedido de indexação de um novo URL deve ser executado apenas por um dos servidores. Assim, a construção do índice invertido é feita em paralelo por questões de desempenho. Periodicamente, cada servidor Multicast envia a sua parte do índice para outros servidores Multicast, ficando assim o índice eventualmente sincronizado. Cabe aos servidores RMI agregarem os resultados das pesquisas.

Quando um servidor Multicast processa um URL, começa por extrair as palavras e adiciona esse mesmo URL à lista 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 servidor Multicast deve encaminhar alguns URLs para outros servidores Multicast, por forma a distribuir trabalho e melhorar o desempenho.

6 Requisitos não-funcionais

A aplicação deverá lidar corretamente com quaisquer exceções que estejam previstas. Por forma a garantir que o ucBusca 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

Como o hardware pode falhar, é necessário que os utilizadores não notem nenhuma falha de serviço. Como tal, no caso do servidor RMI falhar, é preciso garantir que as introduções de URLs para indexação não se percam (realizando *retries* sempre que necessário). No caso das avarias temporárias (inferiores a 30 segundos) os clientes não se devem aperceber desta falha.

Também do lado dos clientes é possível que a ligação se perca a meio. É necessário garantir que nenhuma falha do lado do cliente deixe nenhuma operação a meio. Deve bastar reiniciá-los para que se possa depois continuar a usar o ucBusca.

Os servidores Multicast também podem avariar e ser reiniciados. Quando isso acontece, pretende-se que a mesma informação exista noutras réplicas. Para tal, os servidores Multicast devem comunicar entre si para se manterem sincronizados usando TCP. Esta é a única situação em que se deve usar TCP. Uma possível solução consiste em ter um pedido específico, via multicast, para que os servidores ativos se identifiquem (IP e porta TCP). Periodicamente, um processo liga-se a outros e envia-lhes a versão mais recente do índice.

6.2 Failover

De modo a que quando o servidor RMI falhar, o servidor secundário o substitua, é necessário ter alguns cuidados. Em primeiro lugar o servidor secundário deve periodicamente testar o servidor primário através de uma qualquer chamada RMI. Se algumas destas chamadas resultarem em erros (por exemplo cinco chamadas seguidas) o servidor secundário assume que o primário avariou e liga-se para o substituir.

Os clientes RMI também devem detetar quando existe uma falha permanente do servidor RMI e devem tentar ligar-se ao secundário. Dado o uso de persistência nos servidores Multicast, os dados visíveis pelos clientes devem ser exatamente os mesmos. Do lado dos clientes, todo este processo deve ser transparente. Para eles esta falha nunca deverá ter qualquer efeito visível.

Finalmente, se o servidor RMI original recuperar, deverá tomar o papel de secundário e não de primário.

6.3 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 do servidor Multicast. Deve especificar detalhadamente o protocolo usado para comunicação multicast.
- Detalhes sobre o funcionamento do servidor 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).

6.4 Distribuição de tarefas

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

- Elemento 1 será responsável pelo Multicast Server e o elemento 2 pelo RMI Server e pelo RMI Client. Esta divisão assume que o protocolo multicast é especificado inicialmente e que as alterações serão daí em diante mínimas.
- Cada um dos elementos ficará com igual número de funcionalidades a implementar, trabalhando um pouco em cada um dos programas a desenvolver.

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

7 Planos futuros para o projeto

Na segunda meta do projeto (dezembro) irão expandir a presente solução, adicionando uma interface Web usando HTML/JSP/Struts2/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 necessária. 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.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**.

Dentro do ficheiro ZIP deverá também estar um Javadoc/PDF/HTML com o relatório. O relatório deve seguir a estrutura fornecida, dado que a avaliação irá incidir

sobre cada um dos pontos. Também no ficheiro ZIP deverão existir três ficheiros JAR: dataserver.jar (servidor Multicast), server.jar (servidor RMI), e client.jar (cliente RMI).

Finalmente, 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 com o projeto deverá ser entregue na plataforma Inforestudante até ao dia **26 de outubro de 2019 (21:59)**, via `https://inforestudante.uc.pt`