1 2  9 0 FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE D COIMBRA

dei departamento de engenharia informática

Sistemas Distribuídos

GOOGOL: MOTOR DE PESQUISA DE PÁGINAS WEB

2022/2023

Trabalho realizado por:

- **Tomás Bernardo Martins Dias nº 2020215701**
- **Henrique Miguel da Silva Costa nº 2020214120**

Conteúdo

Introdução

Neste trabalho prático foi pedido que fosse desenvolvido um motor de busca de páginas Web, com um conjunto de funcionalidades semelhantes aos serviços google.com, bing.com. Este motor de busca teria de possuir indexação automática (Web Crawler) e busca (Search Engine).

Ao longo deste relatório, vamos explicar detalhadamente a arquitetura de software, o funcionamento do servidor, os testes que realizamos à plataforma e o que implementamos para assegurar fiabilidade e consistência à nossa aplicação.

Arquitetura de software

O software é constituído por 5 componentes: Index StorageBarrel, RMI SearchModel, RMI Client, Downloaders e a URL Queue.

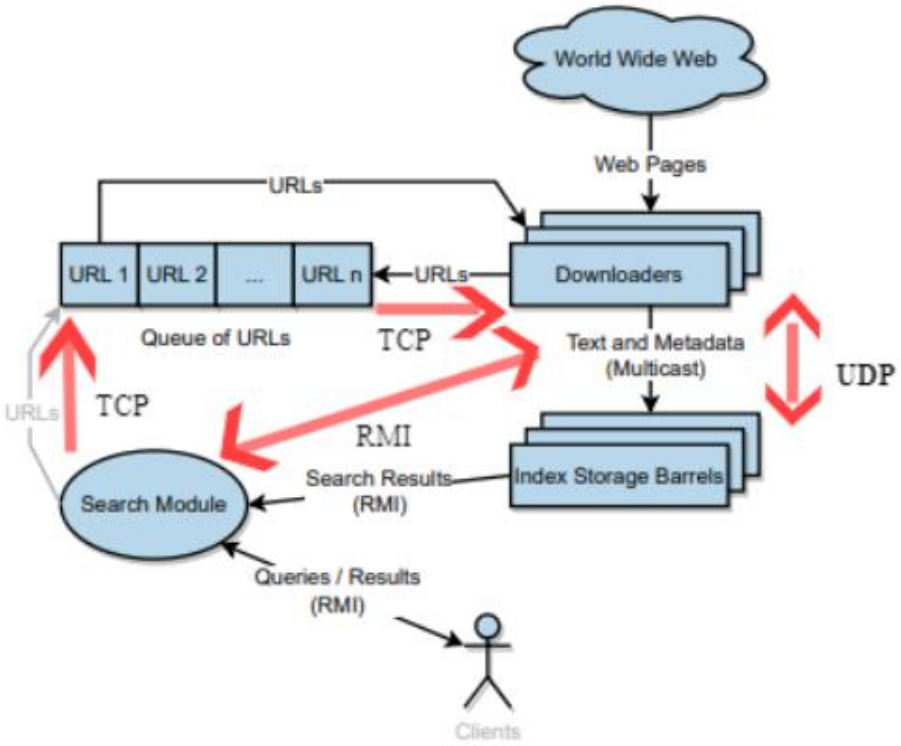


Imagem 1 - Estrutura do Software

Como era referido no enunciado era necessário implementar uma componente capaz de armazenar os URL's encontrados pelos Downloaders e para isso decidimos armazenar estes dados num programa á parte. Na implementação da Queue foi utilizada um thread que se estabelece uma ligação TCP com o Downloader. Para evitar problemas de concorrência foi usada uma estrutura thread-safe para o armazenamento dos dados do tipo 'LinkedBlockingQueue', uma lista ligada thread-safe do tipo FIFO (First-In-First-Out). Para alem de comunicar por TCP com o Downloader a Queue também comunica por TCP com o RMI SearchModel pois era necessário o cliente conseguir indexar Urls na queue.

Os Downloaders também foram implementados usando uma thread, que comunica através de TCP com a Queue, como referido anteriormente, e comunica através de multicast com os Index StorageBarrels para o envio de informação e recebe informação (Acknowledgement) por UDP por parte dos Index StorageBarrels.

Os Index StorageBarrels são o local onde a informação se encontra guardada os dados (Index Invertido). Este programa apresenta um thread responsável por receber a informação enviada pelos Downloaders através de multicast e enviar a mensagem de acknowledgement aos Downloaders através de um canal UDP. Para o index foram utilizadas duas estruturas, uma para o guardar o index e outra auxiliar para a criação do mesmo ambas do tipo "ConcurrentHashMap" devido ao facto de esta estrutura ser thread-safe.

O RMI SearchModel e o RMI Client não apresentam nenhuma thread pois achamos que não seriam necessárias. Comunicam entre si por ligação RMI Callback, sendo o RMI SearchModel responsável por satisfazer os pedidos feitos pelo cliente. O RMI SearchModel além de comunicar com o cliente, também comunica com os Index StorageBarrels através de ligações RMI Callback para realizar pesquisas e saber quantos Index StorageBarrels estão ativos no momento. Para alem disso foi necessário acrescentar uma ligação RMI Callback do Index StorageBarrel para os downloaders porque era necessário também saber quais/quantos Downloaders estavam ativos no momento para informar o Cliente quando este quisesse receber as estatísticas do sistema.

Funcionamento da Componente Multicast

Para a comunicação entre os Downloaders e os Index StorageBarrels foi nos pedidos que fosse implementado um protocolo Multicast Reliable.

Para a implementação dos Downloaders foi criada um thread responsável por pedir URL's a Queue, retirar informações e depois enviar através de multicast as informações aos utilizadores. No lado dos Index StorageBarrels também existe uma thread responsável por receber as informações através de mensagem enviadas por multicast e enviar ao Downloader o acknowledgment das mensagens recebidas.

Primeiro foi definido um protocolo de mensagem a serem enviadas pelos Downloaders aos Index StorageBarrels. Este protocolo foi desenhado baseado no protocolo presente no enunciado deste projeto.

A mensagem a ser enviada possui o url, o título, a citação, o conjunto de palavras e urls encontrados pelo Downloader. Para organizar esta informação na mensagem foi utilizado um conjunto não ordenado de pares chave/valor. Para os conjuntos de palavras e urls, além dos pares chave/valor, também é adicionado o tipo de cada conjunto (word_list e url_list, para palavras e urls respetivamente) acompanhados do número de elementos que cada possui.

Antes desta mensagem ser enviada pelo Downloader é enviada uma mensagem ao Index StorageBarrel, através desta mensagem onde o Downloader envia o seu id, seguido do tamanho da mensagem a ser enviada. Esta mensagem é importante para depois ser possível fazer acknowledgement das mensagens recebidas.

```
//handshake with the size
int UdpPort = sms_barrels.length();
String sms_size = identifier + ";" +Integer.toString(sms_barrels.length());
byte[] buffer_handshake = sms_size.getBytes();
DatagramPacket handshake = new DatagramPacket(buffer_handshake, buffer_handshake.length, group, PORT);
multicast_socket.send(handshake);
```

Imagem 2 - 1ª Mensagem enviado pelo Downloader

Apenas após a primeira mensagem, é enviada a mensagem que contem os dados a serem tratados e armazenados pelo Index StorageBarrel. Após o envio da mensagem também é calculado o tempo do envio utilizando com recurso a função 'currentTimeMillis()', que retorna a diferença, medida em milissegundos, entre a hora atual e a meia-noite de 1º de janeiro de 1970 UTC (horário universal coordenado). Este cálculo será mais á frente aprofundado.

Está mensagem referida e especificada anteriormente será processada e a sua informação armazenada.

```
//send mensagem
byte[] buffer = sms_barrels.getBytes();
DatagramPacket packet = new DatagramPacket(buffer, buffer.length, group, PORT);
multicast_socket.send(packet);
Long time = System.currentTimeMillis();
```

Imagem 3 - Mensagem enviada pelo Downloader

Após o processamento da mensagem será criada um socket UDP, onde será enviado o acknowledgement (uma mensagem contendo o id do Downloader que enviou as mensagens). Essa mensagem teria de ser enviada por uma porta especifica, e para resolver esse problema decidimos usar o tamanho da mensagem enviado no momento do handshake com porto para a conexão UDP.

```
try (DatagramSocket aSocket = new DatagramSocket()) {
    byte [] m = idSize[0].getBytes();
    int UDP_PORT = Integer.parseInt(idSize[1]);
    DatagramPacket request = new DatagramPacket(m,m.length,sms_packet.getAddress(),UDP_PORT);
    aSocket.send(request);
}catch (SocketException e){
    System.out.println("Socket: " + e.getMessage());
}catch (IOException e){
    System.out.println("IO: " + e.getMessage());
}
```

Imagem 4 - Acknowledgement enviado pelo Index StorageBarrel

```
try (DatagramSocket aSocket = new DatagramSocket(UdpPort)) {
    boolean flag = false;
    while(!flag){
        Long pastTime = System.currentTimeMillis();
        if(pastTime - time > 60000){
            multicast_socket.send(handshake);
            multicast_socket.send(packet);
            time = System.currentTimeMillis();
        }
        byte[] buffer_ack = new byte[1024];
        DatagramPacket request = new DatagramPacket(buffer_ack, buffer_ack.length);
        aSocket.receive(request);
        String s = new String(request.getData(),  offset: 0, request.getLength());
        if(s.equals(identifier)){
            flag = true;
        }
    }
```

Imagem 5 - Downloader á espera do Acknowledgement por parte dos Index StorageBarrels

Do lado do Downloader após o envio da mensagem, será criada uma socket UDP que ficará á espera do envio do acknowledgement por parte dos Index StorageBarrels.

Caso passado um minuto o Downloader não tenha recebido o acknowledgement por parte dos Index StorageBarrels, este voltará a enviar o handshake e a mensagem com os dados a serem processados, sendo outra vez calculado o tempo do envio da mensagem. Caso o acknowledgement tenha sido recebido ele prosseguirá o indexamento de novos URL's.

Em suma em mecanismo por nós utilizado, permite que só quando o Downloader receber a mensagem de acknowledgment este continue a indexar novos URL e com isso conseguimos garantir que as mensagens foram recebidas.

Mecanismos de FailOver da comunicação Queue/Downloaders

Foi necessário garantir os URL que iriam ser indexados não fossem perdidos, caso estes não fossem processados pelos Downloaders. Para isso decidiu se usar a comunicação TCP para o Downloader no momento do envio de novos URL a indexar, ou seja, depois de processar o URL enviado, enviei a o URL cujos novos URL'S foram encontrados. Caso o url da mensagem enviada não seja igual ao enviado pela queue, o URL volta a ser reposto na queue e novamente processado.

Funcionamento da Componente RMI

O RMI é uma API que permite a troca de dados entre máquinas através de sockets (mecanismo de comunicação que permite troca de mensagens). Com isto, são criados aplicativos cliente-servidor em que se pode invocar métodos em objetos remotos como se fossem locais.

Uma das técnicas para que haja a invocação de métodos tanto do cliente como do servidor é a utilização de RMI Callback permitindo assim uma comunicação bidirecional.

Uma vez que o RMI Callback tem esta vantagem e, era necessário que os clientes fossem notificados por parte do servidor das alterações de barrels ou downloaders ativos, optamos por esta abordagem.

Inicialmente, após a inicialização do SearchModule (server), o cliente tenta estabelecer uma ligação com um objeto remoto que está registado. Após o estabelecimento da ligação, o cliente invoca o método (subscribe_client()) da interface do servidor passando objeto do tipo "Client" e uma String.

```
public int subscribe_client(String name, ClientInterface client) throws RemoteException, ServerNotActiveException {
    System.out.println("Client Subscribing " + name);
    System.out.print("> ");
    clients.add(client);

    int key = IpClients.size();
    IpClients.put(key, Client.getClientHost());
    statistics.put(key, "false");

    return key;
}
```

Imagem 6 - Função para subscrever um cliente.

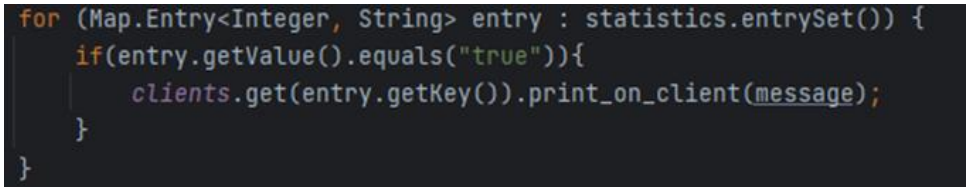
Quando invocado este método, o objeto é adicionado a um ArrayList de objetos vai permitir determinar quais e quantos clientes estão ativos de momento. Para se obter o seu endereço IP, é utilizado um HashMap denominado de IpClients em que a key (próxima posição livre de keys no HashMap) serve para atribuir um id ao cliente e consequentemente identificação, sendo retornado esse id para o mesmo e, servindo como forma de identificação em todas as comunicações estabelecidas.

Uma vez que, existe também a necessidade de o cliente, se desejar obter informação de alterações de barrels ou downloaders ativos, e para a informação não ser enviada para clientes que não a desejassem, foi criado outro HashMap "statistics" em que a chave é o id do cliente e, por defeito, o valor correspondente a cada chave era sempre introduzido como" false", que significa que o cliente não pediu para receber as estatísticas. Sendo assim, sempre que o cliente, identificado pelo seu id, envia

para o servidor a mensagem “stats”, é alterado o valor correspondeste à chave(id) para “true” e a partir desse momento, sempre que ocorre alterações, o mesmo é notificado até que deseje parar e volte a submeter a mesma mensagem, desencadeando o processo inverso.

Quando existe a inicialização quer de barrels, quer de downloaders o processo é idêntico ao do cliente, havendo apenas a diferença em que não existe um HashMap de estatísticas para atualizar, mas sim, a invocação de uma função local printStats() que sempre que fôr inicializado um novo barrel ou downloader, verifica quais os clientes que estão a desejar receber tal informação percorrendo o Hashmap statistics e envia a mensagem invocando um método remoto da interface do cliente(print_on_client()) que permite que a mensagem seja mostrada do lado do cliente.

Imagem 7.Funçao para subscrever um downloader.



Quando o cliente, downloader ou barrel encerra, toda a informação correspondente aos mesmos é eliminada e no caso do segundo e terceiro, caso requerido pelo cliente, é lhe atualizada a informação de quais ativos de momento.

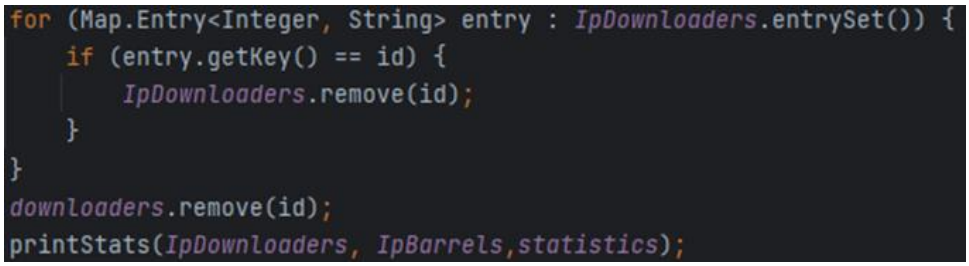


Imagem 8- Retirar informaçoes do downloader que terminou e printar aos user que requerem atualizaçao de stats.

->ShareInfoToBarrel

Este método da interface do barrel, é invocado pelo SearchModule sempre que o mesmo pretende realizar uma pesquisa ou se registar/autenticar, sendo escolhido um barrel da lista de barrels ativos de forma sequencial para efetuar estas ações.

```
if (counter < barrels.size()){
    message = barrels.get(counter).ShareInfoToBarrel(mm);
    counter++;
}else{
    counter = 0;
    message = barrels.get(counter).ShareInfoToBarrel(mm);
    counter ++;
}
```

Imagem 9 . Escolher sequencialmente os barrelsI

Já do lado do barrel, a informação recebida é toda processada e consoante o cariz de ação que seja necessário realizar, o barrel desencadeia ações de pesquisa, login ou de registo.

Se a ação for de registo, cria um objeto do tipo User em que guarda o nome e password e se escreve no ficheiro de objetos. No caso de login, abre o ficheiro e verifica se o utilizador inserido se encontra registado com a devida palavra pass, caso contrário, não é permitido o login. Por outro lado, se fôr de pesquisa, conforme o número de palavras inseridas, é realizado um "match" de links presentes em ambas as palvras, ou seja, na primeira palavra é guardado numa lista todos os urls correspondentes à mesma, e, conforme as sucessivas palavras, vai-se verificando quais links estão presentes em ambas palavras e vai-se descartando quais não o são, chegando ao final, e se obtendo unicamente os links que estão em comum de acordo com o input introduzido para pesquisa, sendo realizado o devido ordenamento.

->ShareInfoToServer

Este método, acente em RMI Callback, da interface do SearchModule, serviu como via de comunicação para clientes, downloaders e barrels e destes para o SearchModule.

Como referido anteriormente, sempre que um cliente, downloader ou barrel termina, para que haja a eliminação da informação correspondente ao mesmo, ou até para no caso da Queue e dos barrels seja armazenado em disco a informação que estes possuem, existe uma thread para apanhar o sinal Ctrl+C.

No caso da necessidade de eliminação de determinada informação devido ao término da aplicação, é invocado o método ShareInfoToServer que envia sempre o id da mesma, bem como, de uma mensagem elucidativa para indicar se se trata de um barrel (-1b), downloader (-1d) ou cliente (-1c) como é demostrado na figura acima.

```
Runtime.getRuntime().addShutdownHook(run() -> {
        try {
            h.ShareInfoToServer(c.id, s: "-1c");
        } catch (RemoteException e) {
        }
        System.out.println("Client ending...");
});
```

Imagem 10 . (Runtime (Java SE 17 & JDK 17) (oracle.com) thread para captar o sinal Ctrl+C)

No caso da mensagem recebida seja igual a “index <link>”, o SearchModule cria um socket TCP em em que envia o link que o cliente deseja que seja dado crawl para a queue.

Caso se queira registar ou efetuar login invoca o método explicado anteriormente (ShareInfoToBarrel()).

Distribuição das Tarefas pelos Elementos do Grupo

A divisão de trabalho foi feita segundo o enunciado, mas sempre que necessário o trabalho foi feito em conjunto.

Tomás Dias – Componente Multicast dos Barrels e pelos Downloaders.

Henrique Costa – Search Module e Componente RMI dos Barrels.

Testes Realizados

Requisitos Funcionais

	O cliente quer-se registar.
Resultado Esperado	O utilizador insere o comando register <nome> <palavra_passe> e o recebe uma mensagem de confirmação.
Resultado Obtido	O utilizador insere o comando register <nome> <palavra_passe> e o recebe uma mensagem de confirmação.
Conclusão	Passou.

	O cliente que iniciar sessão
Resultado Esperado	O utilizador insere o comando login <nome> <palavra_passe> e o recebe uma mensagem de confirmação.
Resultado Obtido	O utilizador insere o comando login <nome> <palavra_passe> e o recebe uma mensagem de confirmação.
Conclusão	Passou

Cliente indexa URL na Queue

Resultado Esperado	O cliente insere o comando index <URL> e receberá a mensagem "link indexado"
Resultado Obtido	Após inserir o comando o cliente recebe a mensagem "link indexado" e a queue contem mais um elemento.
Conclusão	Passou

	O Downloader indexa recursivamente todos os URL's encontrados.
Resultado Esperado	O Downloader após indexar um URL fará um pedido a queue de um novo URL para ser indexado.
Resultado Obtido	O Downloader após indexar um URL executa o pedido á Queue e continua a indexar novos URL's.
Conclusão	Passou

	Pesquisar um conjunto de termos e ordenar os resultados da pesquisa por relevância (Páginas com mais ligações de outras páginas são mais relevantes) agrupando o resultando em 10 a 10.
Resultado Esperado	Após o cliente inserir o comando search <conjunto de palavras> o cliente receberá agrupado de 10 a 10 e ordenado por relevância os resultados da pesquisa.
Resultado Obtido	O cliente recebe os resultados da pesquisa ordenados por relevância embora não venham agrupados 10 a 10.
Conclusão	Não Passou.

	Pesquisar um conjunto de termos e mostrar o link que apontam para os links resultantes da pesquisa.
Resultado Esperado	O cliente se logado obtém os links que apontam para outros corretamente
Resultado Obtido	O cliente obtém os links, mas os links estão a apontam mal uns para outros
Conclusão	Não Passou

Receber as estatísticas do Sistema quando existe alguma alteração e o top 10 pesquisas.

Resultado Esperado	Quando o cliente escreve o comando stats, e existe alguma alteração no sistema (Downloader ou Barrel ligar/desligar) o cliente recebe as estatísticas do Sistema.
Resultado Obtido	Após o cliente inserir o comando stats, e caso exista alguma alteração no sistema ele é informado, mas não recebe o top 10 pesquisas.
Conclusão	Não Passou. Embora mostre as estatísticas do sistema não mostra o top 10 pesquisas.

Requisitos Não-Funcionais

	FailOver - Mensagem não recebida por parte dos Index StorageBarrels
Resultado Esperado	A mensagem será reenvida e o downloader só prosseguirá o indexamento quando receber o acknowledgement
Resultado Obtido	A mensagem não foi entregue e o downloader só prosseguiu o indexamento após receber o acknowledgement.
Conclusão	Passou o teste embora não tenha sido feito testes intensivos.

	FailOver- Queue não recebe a lista de URL encontrados pelo downloader no URL enviado
Resultado Esperado	O URL voltará a queue e será novamente processado.
Resultado Obtido	O URL voltou para a queue, embora link que não conseguem ser processados devido ao jsoup não conseguir fazer o handshake com a página continuaram a ser colocados na queue.
Conclusão	Passou.

	FailOver- Queue não se encontra ligada e os downloaders tentam estabelecer comunicação
Resultado Esperado	Os downloaders recebem informação que a Queue não está em funcionamento e desligam.

Resultado Obtido	Os downloaders recebem informação que a Queue não está em funcionamento e desligam.
Conclusão	Passou.

	FailOver- Queue vai abaixo e recupera corretamente os links presentes na queue
Resultado Esperado	Queue faz backup dos dados em disco até ao término da sessão e quando inicia de novo, recupera-os.
Resultado Obtido	Queue armazena e recupera corretamente os links.
Conclusão	Passou.

	StorageBarrels vai abaixo e recupera corretamente os dados neles contidos.
Resultado Esperado	StorageBarrel faz backup dos dados em disco até ao término da sessão e quando incia de novo, recupera-os.
Resultado Obtido	StorageBarrels recupera as informações corretamente.
Conclusão	Não Passou

Conclusão

No início deste projeto foi pedido que fosse desenvolvido um motor de busca garantindo a aplicação através de redundância com FailOver.

Apesar de não termos implementado todos os requisitos funcionais e não funcionais, pensamos ter cumprido bem alguns dos objetivos. Não foram implementadas garantias de fiabilidade ao nível dos Index StorageBarrel caso um Barrel fosse abaixo. Este é sem dúvida um ponto a ter em conta em trabalho futuro para melhorar a plataforma.

Em suma tendo em conta os resultados dos testes apresentados acima podemos reforçar a afirmação, pois a maioria dos testes passou e aqueles que não passaram na grande generalidade chegou perto do resultado esperado.

Referências

- Material fornecido pelos docentes
- https://www.devmedia.com.br/hashmap-java-trabalhando-com-listas-key-value/29811
- https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Runtime.Version.html
- https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html
- https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/LinkedBlockingQueue.html
- https://stackoverflow.com/questions/3775483/ssl-handshake-exception