

ucBusca: Motor de pesquisa de páginas Web

Sistemas Distribuídos 2019/20 — Meta 1

Licenciatura em Engenharia Informática

António Marques Maria – 2017265346

Diogo Santos Paula – 2017267117

**1.Arquitetura do Software**

A figura 1 mostra a arquitetura geral do sistema. Este é constituído por 3 programas: o servidor RMI, o servidor Multicast e o cliente.

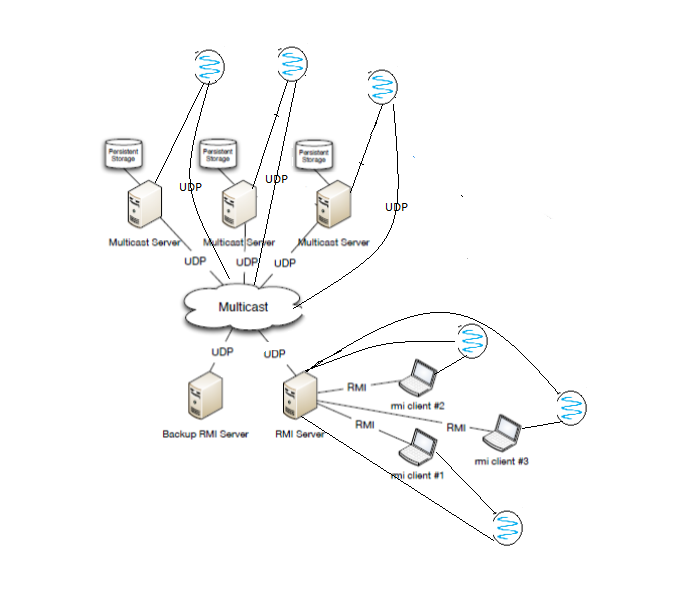


Figura 1- Arquitetura do sistema

Cada cliente conecta-se ao servidor RMI de cada vez que quer realizar alguma operação, e este posteriormente, liga ao servidor Multicast que realiza o que for pretendido e envia uma resposta para verificar se a ação foi bem realizada ou não.

O utilizador ao solicitar a ação chama um método através do servidor RMI que, de seguida, usando MulticastSockets comunica com o servidor Multicast. O socket do RMI envia a informação para o endereço do grupo Multicast, que no nosso caso é 224.0.224.0, e para o port 4321. Os servidores dentro desse grupo recebem essa informação e respondem para o RMI, enviando para um endereço de um grupo Multicast criado para os servidores RMI, sendo este 224.0.224.1, e essa resposta é posteriormente mostrada no Client.

Para as notificações, implementámos duas threads que estão sempre a correr, no Client, a thread está a enviar mensagens Multicast através de um método remoto para ele ir verificar na base de dados se tem alguma notificação pendente, se tiver, o servidor envia a notificação encontrada para a thread do Client que a pediu, apaga-a da base de dados, e o Client posteriormente imprime-a. Para a comunicação entre as threads usamos um endereço multicast diferente dos utilizados nos processos e portos diferentes.

O utilizador tem acesso ao menu de administrador ao realizar uma ação depois de receber a notificação. Se receber a notificação enquanto a ação está a ser realizada, tem logo acesso ao menu de administrador.

**2.Protocolo UDP Multicast**

O protocolo que utilizámos para as mensagens Multicast foi definir um type como primeiro parâmetro da mensagem, podendo ser este login, register,etc…, e de seguida as variáveis necessárias para o servidor realizar as operações, tal como, username, password, etc.

2.1.Login

Login username password

Enviamos o type login para o servidor saber o que tem de chamar, seguida do username e password para verificar na base de dados

A resposta é uma string para confirmar se o login foi bem sucedido ou não e se o utilizador que está a realizar login é admin.

2.2.Register

Register username password

Enviamos o type register para o servidor saber que tem de registar um utilizador, seguido dos seus credenciais.

A resposta é uma string a verificar se o registo foi bem sucedido ou não.

2.3.Pesquisa

Pesquisar username termo

Enviamos o type pesquisar para o servidor ir correlacionar as palavras que se encontram no termo de pesquisa com os websites indexados que contêm esse mesmo termo. Também enviamos o username do Client que está a efetuar a pesquisa para podermos guardar o seu histórico de buscas. Se o Client não tiver login efetuado o termo de busca irá para um ficheiro geral que assimila o username como null.

A resposta é o conjunto de URL’s que estão indexados na palavra.

2.4.Indexar

indexar website

Enviamos o type indexar para o servidor executar as funções da biblioteca jsoup e de seguida guardar esses resultados num Hashmap.

A resposta é a confirmação de que o site foi indexado no motor de busca.

2.5.Histórico

historico username

Enviamos o type historico para o servidor ir ler o ficheiro de histórico de buscas do username que está associado à mensagem, que é o mesmo que pediu a ação.

A resposta é cada termo de pesquisa que aquele determinado utilizador já pesquisou, por ordem da mais recente para a mais antiga, que depois é agregada no servidor RMI e mostrada no Client.

2.6.Tornar Admin

admin username

Enviamos o type admin para o servidor ir ler o ficheiro da base de dados de utilizadores e alterar o atributo do username especificado na mensagem de “0” para “1”, passando este a ser admin e tendo acesso ao menu de administrador da próxima vez que iniciar sessão.

A resposta é a confirmar se o utilizador foi tornado admin com sucesso ou se ele já era admin.

2.7.Notifica

notifica username

Enviamos o type notifica para a thread do servidor ir ler o ficheiro da base de dados de notificações do utilizador especificado e enviar o seu conteúdo para o Client.

A resposta é a notificação que se encontra no ficheiro ou uma String que não é impressa no Client.

2.8.Info

info

Enviamos o type info para o servidor ir buscar a informação das palavras mais pesquisadas tal como os sites mais importantes.

A resposta são duas string’s, uma com as 10 palavras mais pesquisadas e outra com os 10 sites mais imporantes.

**3.Servidor RMI**

Os métodos disponibilizados remotamente aos Clientes são:

*3.1. public String envia\_login(String username, String Password)*

Este método recebe uma String username e uma String Password do Cliente RMI e envia através de um MulticastSocket e de um PORT pré -definido para o endereço do grupo Multicast pretendido a mensagem “login username password“. De seguida, fica à espera da resposta do servidor através do mesmo PORT. Esta resposta é recebida pelo método através do endereço do grupo Multicast definido para os servidores RMI. No final, retorna a string que recebeu do servidor Multicast. A string recebida pode ser a dizer que fez login como admin, login como user, username não existe ou password errada.

*3.2. public String register(String username, String Password)*

Este método recebe uma String username e uma String Password do Cliente RMI e envia através de um MulticastSocket e de um PORT pré -definido para o endereço do grupo Multicast pretendido a mensagem “register username password“. De seguida, fica à espera da resposta do servidor através do mesmo PORT. Esta resposta é recebida pelo método através do endereço do grupo Multicast definido para os servidores RMI. No final, retorna a string que recebeu do servidor Multicast. A string recebida pode ser a dizer que o user foi registado com sucesso ou que o user já existe.

*3.3. public ArrayList<String> pesquisa(String pesquisa, String username)*

Este método recebe uma String pesquisa e uma String username do Cliente RMI e envia através de um MulticastSocket e de um PORT pré -definido para o endereço do grupo Multicast pretendido a mensagem “pesquisar username pesquisa“. De seguida, fica à espera das respostas do servidor através do mesmo PORT, até receber “FIM”. Estas respostas são recebidas pelo método através do endereço do grupo Multicast definido para os servidores RMI. No final, retorna a lista agregada de URL’s em que a palavra foi encontrada, juntamente com o seu título e uma breve descrição.

*3.4. public String indexar(String website)*

Este método recebe uma String website do Cliente RMI e envia através de um MulticastSocket e de um PORT pré -definido para o endereço do grupo Multicast pretendido a mensagem “indexar website“. De seguida, fica à espera da resposta do servidor através do mesmo PORT. Esta resposta é recebida pelo método através do endereço do grupo Multicast definido para os servidores RMI. No final, retorna a string que recebeu do servidor Multicast. A string recebida é a informar que o URL foi bem indexado.

*3.5. public ArrayList<String> historico(String username)*

Este método recebe uma String username do Cliente RMI e envia através de um MulticastSocket e de um PORT pré -definido para o endereço do grupo Multicast pretendido a mensagem “historico username“. De seguida, fica à espera das respostas do servidor através do mesmo PORT, até receber a string “FIM”. Estas respostas vão ser recebidas pelo método através do endereço do grupo Multicast definido para os servidores RMI. No final, agrega o histórico do utilizador, que são as várias respostas do servidor, numa ArrayList e retorna o array para o Cliente RMI.

*3.6. public String tornar\_admin(String Username)*

Este método recebe uma String username do Cliente RMI e envia através de um MulticastSocket e de um PORT pré-definido para o endereço do grupo Multicast pretendido a mensagem “admin username“. De seguida, fica à espera das respostas do servidor através do mesmo PORT. Esta resposta é recebida pelo método através do endereço do grupo Multicast definido para os servidores RMI. No final, retorna a string a informar que o utilizador foi tornado admin com sucesso.

*3.7. public String recebe\_notificacao(String Username)*

Este método recebe uma String username do Cliente RMI e envia através de um MulticastSocket e de um PORT2 pré-definido, diferente do port usada para a comunicação dos restantes métodos, para o endereço do grupo Multicast pretendido a mensagem “notifica username“. De seguida, fica à espera das respostas do servidor através do mesmo PORT2. Esta resposta é recebida pelo método através do endereço do grupo Multicast definido para os servidores RMI. No final, retorna a notificação que se encontrava no ficheiro ou uma String redundante que não é impressa no cliente.

*3.6. public ArrayList <String> info()*

Este método envia através de um MulticastSocket e de um PORT pré-definido para o endereço do grupo Multicast pretendido a mensagem “info“. De seguida, fica à espera das respostas do servidor através do mesmo PORT. Esta resposta é recebida pelo método através do endereço do grupo Multicast definido para os servidores RMI. No final, retorna um array de string’s com a informação das palavras mais pesquisadas, dos sites mais importantes e com os endereços e portos dos servidores Multicast. Este método também é responsável por ir buscar estes endereços e portos.

*3.8. Failover*

*Ao ligar o servidor RMI, este vai ver se existe já algum registo criado e a correr, caso exista ele fica à espera até que este vá abaixo ou seja desligado. O Backup corre 5 vezes o comando de verificar o status do outro servidor, se após estas tentativas der sempre exception, este liga-se, e o próximo servidor RMI que se ligar vai ter que esperar que este se desconecte para conseguir voltar a ligar aos clientes.*

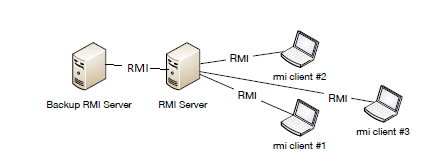


Figura 2- Ao ligar os servidores

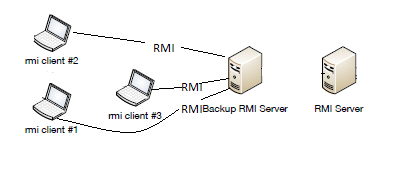


Figura 3- Após o servidor primário falhar

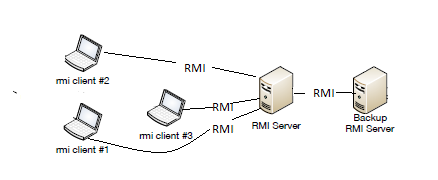


Figura 4- Quando o antigo primário (agora backup) se volta a ligar

**4.Distribuição de tarefas**

Elemento 1 - António

Elemento 2- Diogo

O elemento 1 ficou responsável pelo RMI Server, RMI Client, a implementação das notificações e do relatório

O elemento 2 ficou responsável pelo Multicast Server.

Nota: A grande maioria do trabalho foi feito em conjunto nas salas de estudo, por isso houve colaboração de ambos os elementos nas partes de cada um.

**5.Descrição dos testes realizados**

|  |  |
| --- | --- |
| Testes | Pass/Fail |
| Registar novo utilizador | Pass |
| Acesso protegido com password | Pass |
| Indexar URL | Pass |
| *Indexar iterativamente/recursivamente todos os URL’s* | Pass |
| Pesquisar páginas que contenham um conjunto de palavras | Pass |
| Resultados ordenados por número de ligações para cada pagina | Fail |
| Consultar lista de paginas com ligações para uma pagina especifica | Fail |
| Consultar lista de pesquisas feitas pelo próprio utilizador | Pass |
| Dar privilégios de administrador a um utilizador | Pass |
| Página de administração em tempo real | Pass |
| Notificação imediata de privilégios de administrador | Pass |
| Entrega posterior de notificações | Pass |
| Avaria de um servidor RMI não tem efeito visível nos clientes | Pass |
| Servidor RMI secundário substitui o primário em caso de avaria longa | Pass |
| Em caso de avaria longa os clientes RMI ligam ao secundário | Pass |
| Avarias temporárias (<30s) dos servidores multicast são invisíveis para clientes |  |
| Pedidos são garantidamente processados por N>=1 servidores multicast |  |
| Pedidos de indexação são respondidos por um servidor multicast | Pass |
| O serviço funciona se houver pelo menos um servidor multicast | Pass |
| Os servidores multicast recuperam de disco o seu estado se avariarem |  |
| Cada servidor multicast replica a sua parte do índice por outros servidores (TCP) |  |
| Cada servidor distribui URLs para serem indexados por outros servidores |  |
| O servidor RMI original, quando recupera, torna-se secundário | Pass |