UNIVERSIDADE D
COIMBRA

Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Licenciatura em Engenharia Informática

Sistemas Distribuídos

2022/2023

Projeto realizado por:

Miguel Fazenda – 2019222229

Tiago Henriques – 2020237060

Professor orientador:

Hugo Amaro

Índice

1. Introdução 3
2. Arquitetura do Software 3
   2.1 Descrição geral 3
   2.2 Funcionamento de threads e sockets 4
   2.3 Funcionamento da componente Multicast 5
   2.4 Funcionamento da componente RMI 5
3. Mecanismos de failover 6
4. Distribuição de tarefas 6
5. Testes realizados 6

1. Introdução

O objetivo deste projeto foi criar um motor de pesquisa de páginas web com um conjunto limitado de funcionalidades semelhantes aos oferecidos pelo Google.com, Bing.com e DuckDuckGo.com. O programa permite que os utilizadores realizem pesquisas e obtenham informações organizadas sobre as páginas que contêm os termos pesquisados.

2. Arquitetura do Software

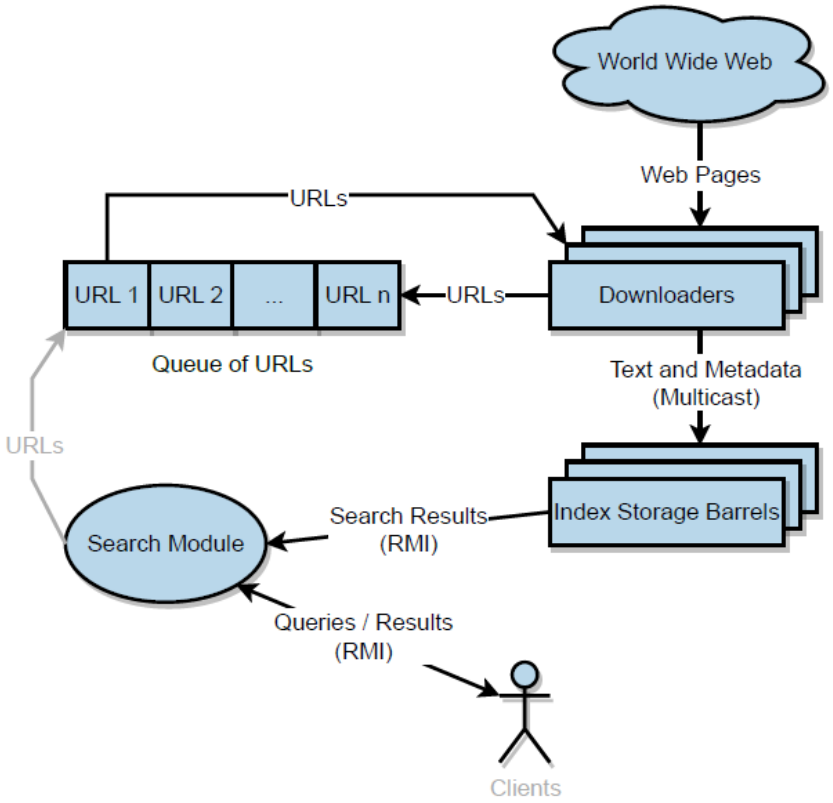

Figura 1 – Arquitetura do software

2.1 Descrição geral

O projeto é composto por 5 componentes, as quais desempenham as seguintes funções:

- Downloader
  - São os componentes que obtêm as páginas web em paralelo, as analisam (usando jsoup) e atualizam o índice através de Multicast.
  - Comunicam com a URL_Queue usando uma ligação TCP para buscar os url a analisar.

- URL_Queue
    - Guarda os url a pesquisar em ficheiro de objetos, periodicamente.
    - Comunica com os Downloaders para enviar os url a analisar através de uma ligação TCP.
    - Comunica com o RMI_Search_Module através de uma ligação TCP.
- RMI_Search_Module
    - Realiza uma ligação RMI com o cliente para que este possa dar uso aos seus métodos.
    - Realiza uma ligação RMI com os Storage_Barrels para puder utilizar os seus métodos.
    - Em caso de failover de um dos Storage_Barrels permite a continuação do programa.
    - Permite que os RMI_Clients se registem e deem login, guardando os seus dados em ficheiros de objetos.
    - Permite ao utilizador enviar um url para ser analisado.
    - Comunica com o URL_Queue através de uma ligação TCP para puder enviar os urls dados pelo utilizador.
- Storage_Barrel
    - Guarda as informações recolhidas pelos Downloaders que foram transmitidas através de Multicast, guardando-as em ficheiros de objetos periodicamente.
    - Permitem pesquisar por urls que estejam relacionados com as keywords dadas pelos RMI_Clients através de ligações RMI com o RMI_Search_Module.
- RMI_Client
    - Permitem aos utilizadores interagirem com o programa através de ligações RMI com o RMI_Search_Module.

2.2 Funcionamento de threads e sockets

Sockets: os servidores são implementados usando a API Java Socket. A classe ServerSocket é usada para criar uma socket de escuta em cada port. Quando um cliente se conecta a um port, o método accept() da classe ServerSocket é chamado para aceitar a conexão e criar um novo socket dedicado ao cliente. As classes BufferedReader e PrintWriter são usadas para enviar e receber mensagens de texto pelo socket. Estas sockets são usadas entre os Downloaders e a URL_Queue e entre esta e o RMI_Search_Module.

O projeto também utiliza MulticastSocket, que seré aprofundada no próximo ponto.

- Downloader

- Utilizamos threads nos Downloaders de modo a pudermos analisar urls em paralelo.

- URL_Queue
  - Utilizamos threads para permitir que a URL_Queue esteja simultaneamente a vários ports (os ports 1234 e 1235 para Downloaders e o port 1236 está reservado para quando o utilizador envia um url).

2.3 Funcionamento da componente Multicast

Utilizamos um processo Multicast baseado em UDP, as sockets Multicast são inicializadas através do "new MulticastSocket()", estas depois tem que fazer "InetAddress.getByName(MULTICAST_ADDRESS)" para achar o grupo Multicast por onde as mensagens serão enviadas. Esse grupo é utilizado para criar um DatagramPacket que depois será enviado por "socketM.send()", os Storage_Barrels, utilizam o "socketM.recieve()" para receber as mensagens, as mensagens estão divididas em 3 partes, sendo a primeira utilizada para dar informação ao Storage_Barrel onde deve guardar a informação contida no resto da mensagem.

2.4 Funcionamento da componente RMI

No nosso projeto utilizamos RMI em duas instâncias para conectar o RMI_Client ao RMI_Search_Module e para conectar este ao Storage_Barrel, isto permite ao RMI_Search_Module utilizar métodos do Storage_Barrel que estejam definidos na interface RemoteInterfaceSB, isto implica que o RMI_Client pode utilizar os métodos presentes no Storage_Barrel e no RMI_Search_Module através da Interface RemoteInterface.

Os métodos remotos implementados são o search, send_url, register, e login. O método search é primeiro chamado pelo RMI_Client, este faz "RemoteInterface ri = (RemoteInterface) LocateRegistry.getRegistry(7000).lookup("RMI_Server")", para fazer a ligação com o RMI_Search_Module, e, de seguida, faz "List<UrlInfo> urls = ri.search(searchPhrase)" para chamar o método do RMI_Search_Module, este método por sua vez faz "RemoteInterfaceSB ri = (RemoteInterfaceSB) LocateRegistry.getRegistry(nBarrel+7001).lookup(barrels[nBarrel])" e "urls = ri.search(searchPhrase)" para chamar o método do Storage_Barrel e este sim faz o search pelos hash maps para encontrar os urls relacionados com os termos dados pelo utilizador.

O método send_url é primeiro chamado pelo RMI_Client este faz "RemoteInterface ri = (RemoteInterface) LocateRegistry.getRegistry(7000).lookup("RMI_Server")", para fazer a ligação com o RMI_Search_Module, e, de seguida, faz "ri.send_url(url)" para chamar o método do RMI_Search_Module, este método por sua vez faz uma ligação TCP com a URL_Queue e envia o url para o queue.

Ambos métodos register e login são primeiro chamados pelo RMI_Client este faz "RemoteInterface ri = (RemoteInterface) LocateRegistry.getRegistry(7000).lookup("RMI_Server")", para fazer a ligação com o RMI_Search_Module, e, de seguida, faz "ri.registerl()"/"ri.login()" respetivamente para chamar o método do RMI_Search_Module, estes métodos pro sua vez verificam se o utilizador e a password são validos e se sim, guardam os numa HasMap, no caso do register, ou no caso do login da acesso a aplicação.

3. Mecanismos de failover

Os métodos remotos em RMI_Search_Module primeiro guardam o numero do Storage_Barrel com que vai tentar fazer ligação primeiro, se isto resultar numa connectException() o RMI_Search_Module vai tentar com os Storage_Barrels restantes até que um resulte ou que volte ao que falhou em primeiro lugar, isto permite ao projeto funcionar desde que pelo menos um strorrage_barrel esteja em funcionamento.

4. Distribuição de tarefas

A distribuição de tarefas foi feita de maneira colaborativa pelos membros do grupo, e durante todo o projeto ambos os membros trabalharam juntos fisicamente no mesmo local.

5. Testes realizados

Todas as funções em ambiente estável.

Resultado: Todas as funções operam como esperado.

Failover de um Storage_Barrel.

Resultado: O RMI_Search_Module faz ligação com o próximo Storage_Barrel.

Start dos Downloaders sem começar a URL_Queue.

Resultado: Os Downloaders enviao erro de comunicação e esperam 5 segundos antes de tentar novamente.