Meta 1 - Relatório Técnico

Licenciatura em Engenharia Informática
Sistemas Distribuídos

Carlos Soares	2020230124, uc2020230124@student.uc.pt
Miguel Machado	2020222874, uc2020222874@student.uc.pt

March 25, 2025

Contents

1. Introdução

Este relatório apresenta o desenvolvimento de um sistema distribuído para indexação e pesquisa de páginas web, desenvolvido no contexto da unidade curricular de Sistemas Distribuídos da Licenciatura em Engenharia Informática. O principal objetivo do projeto é aplicar, de forma prática, conceitos fundamentais da disciplina como comunicação remota, concorrência, tolerância a falhas, replicação e modularidade.

A solução foi implementada em Java, utilizando RMI (Remote Method Invocation) como tecnologia base de comunicação entre os diversos módulos distribuídos. A arquitetura do sistema é composta por vários componentes independentes e cooperantes: cliente de pesquisa, gateway de distribuição, servidores de armazenamento (barrels), gestores de fila centralizada de URLs e crawlers responsáveis pela recolha e análise de páginas web.

O sistema foi concebido para funcionar de forma concorrente e tolerante a falhas, com múltiplos crawlers a operar em paralelo e os dados replicados de forma atómica entre os diferentes barrels. Os dados são também guardados localmente, assegurando persistência e recuperação após falhas ou reinícios.

Neste documento são descritas as principais decisões de arquitetura, os mecanismos de replicação e sincronização entre nós, os testes realizados, bem como o funcionamento detalhado de cada componente. O trabalho desenvolvido cumpre, na sua maioria, os objetivos definidos para a primeira meta do projeto, demonstrando uma abordagem sólida.

2. Objetivos do Projeto

O desenvolvimento deste sistema teve como propósito a construção de uma aplicação distribuída robusta, escalável e tolerante a falhas, com funcionalidades completas de recolha, indexação, armazenamento e pesquisa de páginas web.
Para tal, foram definidos os seguintes objetivos específicos:

- **Permitir a pesquisa de termos por parte dos utilizadores**, com apresentação dos resultados de forma ordenada consoante a sua relevância, baseada no número de ligações recebidas de outras páginas (backlinks);
- **Implementar um processo de indexação recursiva e automática**, a partir de URLs inseridos pelo utilizador, de modo a explorar e processar todos os links internos presentes nas páginas web;
- **Disponibilizar funcionalidades adicionais de consulta**, tais como a visualização da lista de páginas que referenciam uma determinada URL (backlinks), e estatísticas de utilização do sistema, incluindo os termos mais pesquisados e o tempo médio de resposta;
- **Assegurar tolerância a falhas**, garantindo que a aplicação se mantém funcional mesmo perante a indisponibilidade temporária de alguns dos seus componentes, nomeadamente os servidores de armazenamento (barrels);
- **Distribuir a carga entre múltiplos servidores de armazenamento**, implementando mecanismos de balanceamento que otimizem a utilização dos recursos disponíveis;
- **Permitir a execução paralela de múltiplos crawlers**, com isolamento e independência, assegurando a eficiência na recolha e indexação de conteúdos web;
- **Assegurar persistência dos dados**, garantindo que toda a informação indexada é armazenada localmente de forma segura e é recuperada automaticamente após falhas ou reinicializações do sistema.

3. Componentes do Sistema

O sistema foi estruturado de forma modular e distribuída, promovendo independência entre componentes e comunicação remota via Java RMI. Os principais módulos são:

- **SearchClient**: Interface textual utilizada pelo utilizador para pesquisar termos, consultar backlinks, estatísticas, adicionar links e visualizar barrels ativos.
- **SearchGateway**: Responsável por encaminhar pedidos para os barrels disponíveis, realizando balanceamento de carga e deteção de falhas.
- **IndexStorageBarrel**: Servidores de armazenamento que mantêm o índice invertido, backlinks e estatísticas. Executam persistência local e replicação de dados para garantir tolerância a falhas.
- **WebCrawler**: Agentes responsáveis por recolher conteúdo web e enviar para os barrels. Suportam execução paralela, scraping com JSoup e envio atómico para múltiplos barrels.
- **CentralURLQueue**: Fila central de URLs partilhada por todos os crawlers. Garante que cada URL é processado uma única vez e permite adição de novos links.
- **SearchService e SearchGateway (Interfaces RMI)**: Definem as operações remotas disponíveis nos barrels e na gateway, respetivamente.
- **SearchServiceImpl e SearchGatewayImpl**: Implementações concretas das interfaces RMI, com tratamento de concorrência e falhas.
- **LinkAdder**: Aplicação auxiliar que permite ao utilizador adicionar URLs à fila. Cada novo link origina o lançamento de um `WebCrawler` dedicado.
- **run.bat**: Script de arranque que compila o projeto e inicia automaticamente todos os componentes do sistema em terminais distintos.

4. Funcionamento Geral

O sistema segue uma arquitetura modular com comunicação remota entre componentes, permitindo a indexação e pesquisa de páginas web de forma distribuída, tolerante a falhas e paralela.

4.1. Indexação

1. O utilizador insere um URL no cliente (`SearchClient`).
2. O link é adicionado à `CentralURLQueue`, que evita duplicações.
3. Um `WebCrawler` é lançado automaticamente numa nova thread.
4. O crawler obtém a página com JSoup, extrai o texto e os links.
5. Os dados são enviados para todos os barrels (multicast atómico).
6. Cada barrel atualiza o seu índice invertido e backlinks.

Este processo é concorrente: múltiplos crawlers podem trabalhar em paralelo sem conflitos, graças à centralização da fila.

4.2. Pesquisa

1. O utilizador pesquisa um termo no cliente.
2. O pedido é enviado para a `SearchGateway`.
3. Um barrel é escolhido automaticamente (round-robin).
4. O índice é consultado e os resultados ordenados por número de backlinks.
5. Os resultados são apresentados em grupos de 10.

A gateway reencaminha o pedido se um barrel estiver offline, garantindo disponibilidade.

4.3. Outras Funcionalidades

- Consulta de backlinks por URL;
- Estatísticas: termos mais pesquisados e tempo médio de resposta;
- Visualização de barrels ativos em tempo real;
- Dados persistentes entre reinícios;
- Crawlers executados em paralelo com threads independentes.

5. Tolerância a Falhas e Confiabilidade

O sistema foi concebido para manter a sua integridade e operabilidade mesmo perante falhas parciais ou interrupções temporárias. Foram implementados vários mecanismos que asseguram a continuidade dos serviços, a consistência dos dados e a recuperação automática em caso de falhas.

5.1. Replicação Fiável entre Barrels

Todos os *storage barrels* mantêm cópias consistentes do índice invertido. Para garantir a coerência entre os nós, o processo de indexação recorre a **multicast atómico**: o *WebCrawler* envia a página a todos os barrels e apenas considera a indexação concluída após todos confirmarem a receção. Esta abordagem assegura consistência mesmo em presença de falhas temporárias de comunicação.

5.2. Recuperação de Estado

Cada barrel persiste periodicamente os seus dados localmente em ficheiros `.ser` (binário) e `.txt` (legível). Ao reiniciar, o barrel tenta importar dados atualizados de outro barrel ativo através dos métodos `exportIndexData()` e `importIndexData()`. Na ausência de barrels disponíveis, carrega os dados locais salvos. Este processo de sincronização é realizado de forma assíncrona, sem bloquear o arranque do servidor.

5.3. Alta Disponibilidade

A *SearchGateway* monitoriza dinamicamente o estado dos barrels. Caso um barrel esteja indisponível, os pedidos são automaticamente redirecionados para outro ativo. Esta redundância assegura que o sistema continua funcional desde que pelo menos um barrel esteja operacional, sem impacto para o utilizador.

5.4. Distribuição de Carga

As pesquisas são distribuídas entre os barrels de forma equilibrada com uma política de **round-robin**, implementada na gateway. Esta técnica contribui para a escalabilidade e evita sobrecarga de servidores específicos.

5.5. Execução Concorrente de Crawlers

A cada URL inserido na fila de indexação, é criado um novo *WebCrawler* numa thread independente. Este paralelismo permite acelerar o processo de recolha e indexação de páginas, garantindo eficiência e utilização ótima dos recursos disponíveis.

5.6. Resiliência da Gateway

A *SearchGateway* é projetada para continuar operacional independentemente do estado dos barrels. Quando todos os barrels estão offline, notifica o utilizador e permanece em execução, retomando automaticamente as funcionalidades normais assim que algum barrel se torne novamente acessível.

6. RMI - Replicação e Multicast Fiável

A replicação entre barrels segue o princípio de **multicast atómico**:

- O `WebCrawler` envia cada nova página para todos os barrels ativos através de chamadas RMI ao método `indexPage()`.
- A indexação só é considerada completa após todos os barrels confirmarem sucesso. Isto é implementado no ciclo com verificação de `confirmed.contains(address)` na classe `WebCrawler`.
- Em caso de falha de comunicação com um barrel, é feito `retry` com `Thread.sleep(1000)` até obter sucesso.
- Quando um barrel reinicia, importa o estado de outro barrel usando `exportIndexData()` e `importIndexData()`. Ver exemplo na classe `IndexStorageBarrel1`:

```
1     InvertedIndex.IndexData data = other.exportIndexData();
2     index.importIndexData(data);
```

Listing 1: Interface remota SearchService

7. Interfaces RMI e Componentes Distribuídos

O sistema utiliza RMI para permitir que os objetos distribuídos comuniquem entre si de forma transparente:

- `SearchGateway`: utilizado pelo cliente. Define métodos como `search()`, `getTopSearches()`, `getActiveBarrels()`.
- `SearchService`: implementado pelos barrels. Inclui os métodos `indexPage()`, `search()`, `getBacklinks()`, `exportIndexData()`, entre outros.
- `CentralURLQueue`: usado pelos crawlers. Expõe `getNextUrl()` e `addUrl()`.
- Os métodos RMI estão protegidos por `synchronized`, garantindo concorrência segura (ex: `public synchronized void indexPage(...)`).
- Todos os acessos remotos são protegidos com blocos `try/catch` e os servidores utilizam `Naming.rebind()` para registo.

Exemplo de JAVA RMI: Interface remota `SearchService.java`

O sistema utiliza **Java RMI (Remote Method Invocation)** como base para a comunicação entre os seus componentes. Cada serviço define uma interface remota que é implementada e registada num servidor. Seguem exemplos reais da implementação.

```
1 public interface SearchService extends Remote {
2     void indexPage(String url, String content) throws RemoteException;
3     List<String> search(String term) throws RemoteException;
4     Set<String> getBacklinks(String url) throws RemoteException;
5     IndexData exportIndexData() throws RemoteException;
6 }
```

Listing 2: Interface remota SearchService

8. Exemplo Prático de Execução do Sistema

8.1. Iniciar o Sistema

1. Abrir um terminal na raiz do projeto.
2. Executar o seguinte comando:

```
.\run.bat
```

Este script compila e inicia todos os componentes em terminais separados: fila central, gateway, barrels, cliente e crawler.

8.2. Adicionar um URL para Indexação

No terminal do cliente `SearchClient`, o menu apresenta várias opções. O utilizador escolhe a opção `4`:

Uma nova janela é aberta com o programa `LinkAdder`. O utilizador introduz um link, por exemplo:

```
Enter URL to index: https://en.wikipedia.org/wiki/Distributed_computing
```

Este link é enviado para a fila central. Um `WebCrawler` é lançado automaticamente numa nova thread, e começa a processar o conteúdo da página.

Nos terminais dos barrels, surgem mensagens como:

```
[Barrel][Indexing] https://en.wikipedia.org/wiki/Distributed_computing
[Barrel][Backlink] Registered: https://en.wikipedia.org/... -> ...
```

8.3. Pesquisar um Termo

De volta ao `SearchClient`, o utilizador escolhe a opção `1` para realizar uma pesquisa:

```
Choose an option: 1
Enter search term: distributed
```

O sistema apresenta os resultados paginados (10 por página), ordenados por número de backlinks:

```
Results [1  10  ] of 23:
1. https://en.wikipedia.org/wiki/Distributed_computing
2. https://en.wikipedia.org/wiki/Cloud_computing
...
Type 'n' for next page or 'q' to quit:
```

8.4. Estatísticas e Funcionalidades Extra

O utilizador pode ainda ver os termos mais pesquisados e o tempo médio de resposta, consultar backlinks de uma URL, etc.

8.5. Encerrar o Sistema

Todos os componentes podem ser encerrados manualmente ou fechando as janelas de terminal. Os barrels guardam automaticamente os dados em ficheiros `.ser` e `.txt`, garantindo persistência.

9. Divisão de Trabalho

- **Carlos Soares**: Downloaders (WebCrawler) e componente multicast fiável dos Barrels.
- **Miguel Machado**: Gateway e componente RPC/RMI dos Barrels.
- **Ambos**: Relatório e código em geral.

10. Tabela de Testes

10.1. Gateway

Descrição	Resultado
Conexão RMI em localhost	Passou
Conexão RMI entre diferentes máquinas	Falhou
Ligar/Desligar barrels e confirmar atualização de estados	Falhou
Múltiplos clientes conectados simultaneamente	Falhou
Pedidos de menu (Cliente → Gateway → Downloader / URLQueue / ISB)	Passou
Balanceamento de carga entre barrels	Passou
Redirecionamento de pedidos em caso de falha	Passou

10.2. Downloader

Descrição	Resultado
Integração URLQueue → adicionar e consumir da fila	Passou
Indexação de URL	Passou
Indexação recursiva (exploração de links internos)	Passou
Integração com ISB	Passou
Execução concorrente e multithreading	Passou
Particionamento do índice / mensagens entre barrels	Falhou
Envio via multicast para todos os barrels	Passou

10.3. IndexStorageBarrel (ISB)

Descrição	Resultado
Resultados de pesquisa ordenados por backlinks	Passou
Integração com Downloader e sincronização via multicast	Passou
Stress test com múltiplos downloaders	Passou
Comunicação Cliente → Gateway → ISB	Passou
Deteção de falhas ou perda de pacotes	Falhou
Vários barrels ativos em simultâneo de forma correta	Passou
Recuperação de dados após falha e reinício de um barrel	Passou

10.4. URLQueue

Descrição	Resultado
Integração com Gateway e Downloaders (conexão + RMI)	Passou
Concorrência / bloqueios: evitar duplicações e condições de corrida	Passou

10.5. Cliente

Descrição	Resultado
Conexão via RMI à Gateway	Passou
Página de administração (menu) atualizada em tempo real	Passou
Testes de input/output e navegação no menu	Passou

11. Conclusão e Considerações Finais

O sistema desenvolvido cumpre os objetivos propostos para a Meta 1, incluindo indexação recursiva, pesquisa distribuída, tolerância a falhas, concorrência e persistência de dados. A arquitetura modular baseada em RMI demonstrou-se eficaz e extensível, e o sistema comportou-se corretamente em testes com múltiplos crawlers e falhas simuladas.

Durante o desenvolvimento, foram adotadas várias decisões técnicas que reforçaram a robustez do sistema:

- **Multicast atómico na indexação**: A indexação é considerada concluída apenas após todos os barrels confirmarem receção, garantindo consistência entre réplicas.
- **Sincronização assíncrona entre barrels**: A recuperação de estado após falha é feita em background, permitindo arranque rápido mesmo com falhas temporárias.
- **Evitar RMI em blocos sincronizados**: As chamadas RMI foram mantidas fora de secções críticas para evitar *deadlocks* e melhorar desempenho.
- **Logs com identificação por thread**: Permitiram validar o paralelismo entre crawlers e facilitaram a depuração.
- **Script `run.bat`**: Automatiza a compilação e arranque de todos os componentes, simplificando testes e utilização.
- **Persistência dual (`.ser` + `.txt`)**: Além do armazenamento binário, é gerado um ficheiro de texto para verificação manual.
- **Modularização do `LinkAdder`**: Mantido como componente separado para facilitar testes e simplificar o cliente.
- **Lançamento de crawlers em threads dedicadas**: Garante concorrência real e escalabilidade na indexação de múltiplos URLs.

Estas opções, embora não obrigatórias, aumentaram a fiabilidade, clareza e desempenho do sistema, contribuindo para um projeto robusto e preparado para evolução futura.