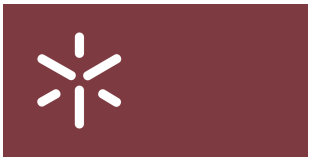

Universidade do Minho
Licenciatura em Engenharia Informática

Comunicações por Computador

Trabalho Prático nº2
PL3 - GRUPO 9

Transferência rápida e fiável de múltiplos servidores em simultâneo

Henrique Pereira (a100831) Maya Gomes (a100822) José Faria(a95255)

December 18, 2023

Conteúdos

1 Introdução **3**

2 Arquitetura da solução **3**
2.1 Esquema da solução 3
2.2 Diagramas temporais ilustrativos dos comportamentos 4
2.2.1 TCP 4
2.2.2 UDP 4
2.3 Funcionamento sobre TCP 5
2.4 Funcionamento sobre UDP 5

3 Especificação do(s) protocolo(s) propostos **6**
3.1 O formato das mensagens protocolares (sintaxe) 6
3.1.1 Mensagens TCP 6
3.1.2 Mensagens UDP 7
3.2 Interações 8

4 Implementação **8**

5 Testes e resultados **9**

6 Conclusões e trabalho futuro **10**

1 Introdução

O objetivo deste trabalho é desenhar, implementar e testar um serviço de partilha de ficheiros *peer-to-peer* de alto desempenho que representa um elemento fundamental no cenário contemporâneo da comunicação digital.

O serviço *peer-to-peer* é essencial para a rápida disseminação e acesso a dados, contribuindo não só para a eficiência na transferência, mas também para fiabilidade na integridade dos ficheiros. Deste modo, no relatório exploraremos a arquitetura do sistema, a implementação de um serviço de partilha de ficheiros *peer-to-peer* e as estratégias adotadas para garantir a fiabilidade baseado num protocolo de transferência sobre o protocolo de transporte UDP.

Para além disso abordaremos também os desafios enfrentados durante a realização do projeto bem como os resultados obtidos nos testes.

2 Arquitetura da solução

2.1 Esquema da solução

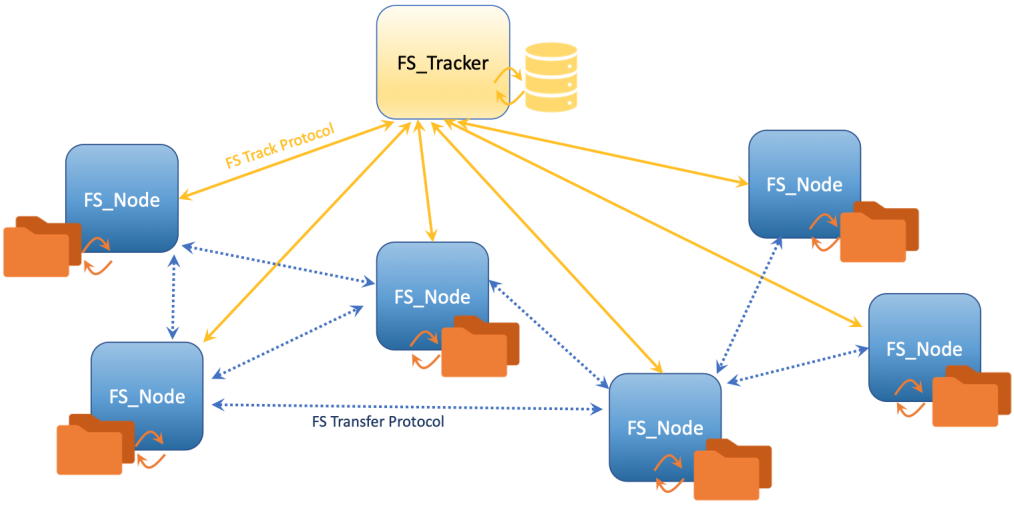

Figure 1: Esquema de funcionamento.

2.2 Diagramas temporais ilustrativos dos comportamentos

2.2.1 TCP

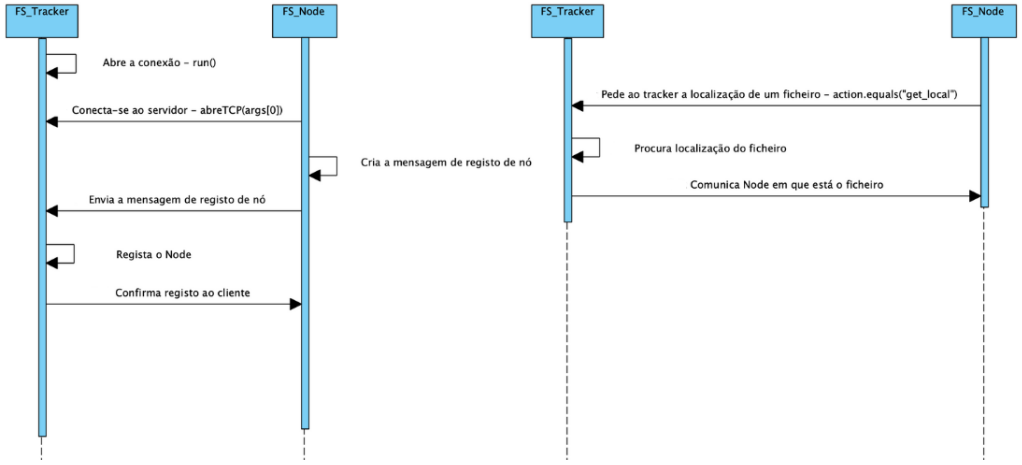

Figure 2: Diagrama temporal do registo do Node e Diagrama temporal do pedido da localidade de um ficheiro. (TCP)

2.2.2 UDP

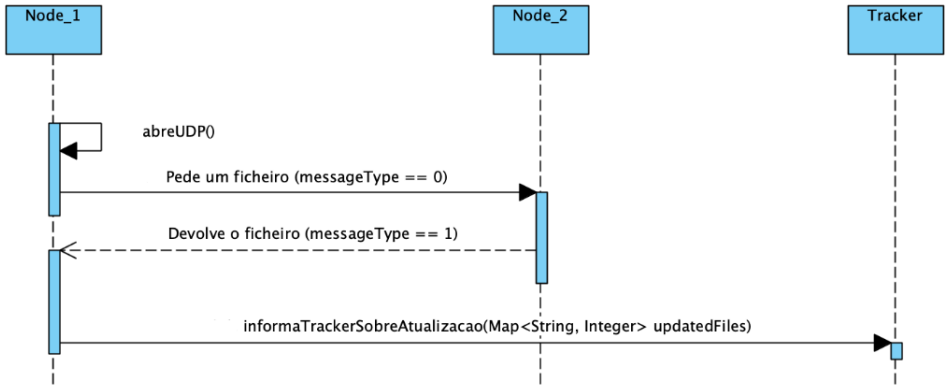

Figure 3: Diagrama temporal da transferência de ficheiros entre dois Nodes. (UDP)

2.3 Funcionamento sobre TCP

A implementação TCP estabelece uma conexão com o servidor, o nosso Tracker. Quando um FS_Node se conecta, o Tracker cria uma nova *thread* para lidar com a comunicação com esse FS_Node, assim o Tracker pode receber mensagens do FS_Node, processá-las no método *handle*, e desencadear ações/respostas com base no conteúdo da mensagem recebida.

Primeiramente, o FS_Node conecta-se ao Tracker através de um Socket. Aí o FS_Node efetua diretamente o seu registo no Tracker. O Tracker registra as informações do FS_Node, incluindo os ficheiros e respetivos blocos que este possui. Uma vez feito o registo, o FS_Node pode enviar mensagens ao Tracker para realizar diferentes ações, como obter a localização de ficheiros ou ainda encerrar a comunicação com o servidor.

O Tracker mantém uma lista de nós registados (*registeredNodes*) que contém as informações sobre cada Node. Desta forma, sempre que lhe for pedida uma informação sobre um ficheiro, ele acede a lista e devolve os Nodes que possuem o ficheiro pedido. Para além disso, sempre que um FS_Node sai da conexão (*exit*) as suas informações são removidas da lista dos nós registados, de forma a não haver troca de informação errada. O FS_Node pode ainda enviar uma mensagen do tipo "*update*" para informar o Tracker sobre mudanças nos seus ficheiros e blocos, nomeadamente depois da troca com outro FS_Node (UDP).

Finalmente, como referido acima, o Tracker cria uma nova thread para cada conexão recebida, permitindo que vários Nodes se conectem simultaneamente.

2.4 Funcionamento sobre UDP

A implementação UDP entre diversos FS_Node permite a transferência de ficheiros divididos por blocos. Desta forma, o FS_Node cria uma nova thread para lidar com a comunicação UDP, aguardando pela troca de mensagens. O servidor UDP (outro FS_Node) permanece num loop infinito, recebendo datagramas de outros Nodes na rede. De forma a diferenciar as diversas mensagens utilizamos o primeiro byte de cada datagrama, assim este funciona como um identificador de tipo de mensagem como será referido mais a frente. O servidor UDP, ao receber esses datagramas, identifica o tipo de mensagem com base neste primeiro byte.

Na troca de um ficheiro, quando um FS_Node envia um bloco para outro, o servidor UDP no destinatário recebe uma mensagem e valida o bloco por meio de um *checksum* baseado no algoritmo CRC32., comparando-o com o *checksum* esperado (calculado antes do envio do bloco). Com isto, permitimos verificar se os blocos recebidos correspondem aos blocos originais, assegurando a consistência dos dados transferidos. Se o bloco for válido, ele é armazenado na memória do FS_Node no mapa *fileArraysMap*. Após receber com sucesso um bloco, o FS_Node envia uma confirmação ao remetente. A medida que os blocos do ficheiro estão a ser recebidos, esses são colocados num `Map<String, byte[][]>` *fileArraysMap*, cada um na sua respetiva posição (index) de forma a ordena-los. Quando o array de bytes do ficheiro em questão esta preenchido com todos os

blocos, então o ficheiro esta completo e o FS_Node cria um ficheiro e copia os blocos recebidos, criando o ficheiro em questão. Finalmente ele informa o Tracker do *update* efetuado (a mensagem contém um *map* de ficheiros atualizados e os seus respetivos índices.).

É também importante referir que caso vários FS_Nodes possuem o ficheiro pedido, então o número de blocos total do ficheiro será dividido pelo número de FS_Nodes que o possuem de forma a tornar o envio equilibrado e mais eficiente dentro do possível. Teremos sempre em consideração o número de blocos do ficheiro que cada FS_Node possui (pois um FS_Node pode não ter o ficheiro inteiro mas somente parte dos blocos). Caso isso aconteça, os FS_Nodes que possuem o ficheiro inteiro irão acumular os blocos que os FS_Nodes "incompletos" não possuem e portanto, não conseguem enviar.

Cada bloco é enviado como um datagrama UDP incluíndo informações como o nome do ficheiro, o index do bloco, o checksum CRC32 e o conteúdo do bloco (1024 bytes). Caso o FS_Node destinatário recebe um bloco que foi mal copiado (*checksum* inválido), ele envia um novo pedido do bloco em questão.

Para garantir que uma resposta seja recebida num tempo razoável (20 segundos), foi criado um mecanismo de *timeout* (através do método checkPedido). Assim, se uma resposta não for recebida dentro do tempo especificado, o FS_Node (cliente) reenvia o pedido em questão.

Além disso, a implementação envolve o envio de ACKs (Acknowledgments) para confirmar o recebimento bem-sucedido de blocos. Caso ocorra algum problema na transmissão de um bloco, um mecanismo de retransmissão é acionado para solicitar novamente o bloco desejado.

3 Especificação do(s) protocolo(s) propostos

3.1 O formato das mensagens protocolares (sintaxe)

3.1.1 Mensagens TCP

```
public class NodeTrackerMessage implements Serializable {
    4 usages
    private String action;
    4 usages
    private Object conteudo;
```

Figure 4: Atributos da classe RegisterNodeMessage.

Sobre o significado dos campos (semântica) temos:

- action: indica a ação que o cliente deseja realizar (*register_node, get_local, update, exit*).
- conteudo: O conteúdo, sendo do tipo *Object* pode ter dois significados. Caso estejamos numa ação de registar o Node ("*register_node*") no Tracker, o conteúdo é um `Map<String nomeficheiro, Integer numblocos>`, isto

é, um *HashMap* cujo as *keys* são o nome do ficheiro e o *value* o respetivo número de blocos. Caso estejamos numa mensagem do tipo "*get_local*", isto é, uma mensagem em que o Node pede ao Tracker a localização de um ficheiro, o conteúdo da mensagem é uma *String* que representa o nome do ficheiro em questão.

3.1.2 Mensagens UDP

Para a resolução do nosso trabalho decidimos dividir as mensagens UDP em dois tipos genéricos. Desta forma o primeiro tipo de mensagens seriam as mensagem de pedido de ficheiros. O primeiro byte das mesmas será igual a zero de forma a podermos diferenciá-las. Para além disso, elas possuem 60 bytes para o nome do ficheiro que se está a pedir, 4 bytes para o index do primeiro bloco e 4 bytes para o index do último bloco que pretendemos receber.

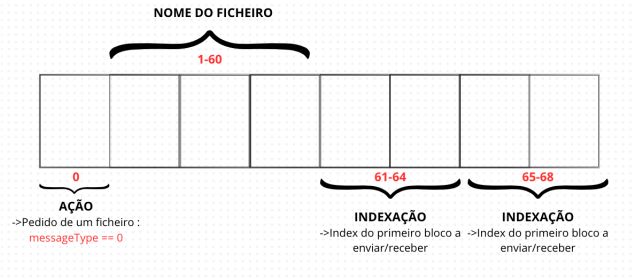

Figure 5: Esquema da representação do conteúdo da mensagem de pedido de um ficheiro.

Relativamente as mensagens de envio de um ficheiro. O primeiro byte das mesmas é igual a um de forma a diferencia-las das mensagens referidas acima. De seguida esta o nome do ficheiro que ocupa 60 bytes, a indexação, ou seja, o número do bloco que esta a ser enviado (4 bytes), o *checksum* da mensagem, de forma a compararmos mais tarde com o do bloco recebido(8 bytes) e o conteúdo do bloco do ficheiro que ocupa 1024 bytes, pois decidimos dividir os ficheiros em blocos de 1024 bytes.

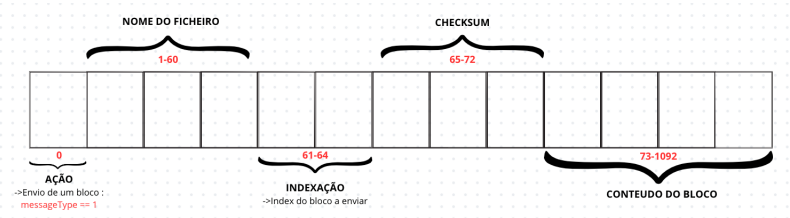

Figure 6: Esquema da representação do conteúdo da mensagem de envio de um ficheiro.

Para além disso, ainda temos um terceiro tipo de mensagens em que o primeiro byte é igual a dois. Este é utilizado para a confirmação (ACK). Esta mensagem confirma a receção de um determinado bloco sendo composta pelo nome do ficheiro e index do respetivo bloco.

3.2 Interações

Os protocolos definem os formato das mensagens trocadas entre o cliente e o servidor e as regras que governam a interação. O servidor, como habitualmente funciona, escuta numa porta específica e espera por solicitações de clientes. Após estabelecida a conexão as interações servidor-cliente e cliente-servidor podem ser feitas através de diferentes mensagens e que consequentemente fazem pedidos / dão respostas relativo ao que está a ser tratado.

Na comunicação TCP, em que o servidor é o FS_Tracker e o cliente um FS_Node, as várias interações possíveis, como referido anteriormente em 2.3, são:

- o registo do FS_Node no Tracker (register_node),
- a solicitação da localização de ficheiros (get_local),
- a atualização do conteúdo de um FS_Node no Tracker (update),
- encerrar a comunicação (exit).

Na comunicação UDP, em que o servidor e o cliente são dois FS_Nodes, as várias interações possíveis, como referido anteriormente em 2.4, são:

- o pedido de ficheiros ($messageType == 0$),
- o envio de ficheiros/blocos ($messageType == 1$),
- a confirmação da receção de blocos ($messageType == 2$).

4 Implementação

Detalhes, parâmetros, bibliotecas de funções, etc: Na realização do trabalho, foram usadas *Threads* para lidar com solicitações TCP e UDP concorrentes. Cada nova conexão TCP é tratada numa nova *thread*, e o servidor UDP é executado numa *thread* separada.

Para facilitar a diferenciação dos diversos tipos de mensagens, essas foram divididas em vários tipos.

Para além disso, o código trata exceções, relacionadas com a comunicação, como *IOException*, *ClassNotFoundException*, *EOFException* e *SocketException* garantindo uma desconexão adequada.

De forma a reduzir os erros de envio e de demora foram usados *checksums* e *timeouts* respetivamente. Para a confirmação de sucesso, o recetor envia uma mensagem a confirmar que recebeu um determinado bloco(ACK).

A comunicação faz uso extensivo de serialização de objetos (*ObjectOutputStream* e *ObjectInputStream*) permitindo que os objetos sejam transmitidos por meio de fluxos de entrada/saída.

Um *scheduler* (*ScheduledExecutorService*) é utilizado para lidar com tarefas agendadas, como *timeouts*. O método *shutDownScheduler* é fornecido para garantir um encerramento adequado.

Para uma boa apresentação dos dados no terminal, práticas de *logging* foram seguidas com o uso da classe *CustomLogger* para registar informações, avisos e erros.

5 Testes e resultados

No exemplo abaixo, temos o tracker no canto superior esquerdo, um FS_Node abaixo que possui o "arquivo2.java" e outro no canto superior direito que também possui o ficheiro. Como podemos ver, o FS_Node em baixo a direita pede o ficheiro e recebe metade de cada FS_Node, pois ambos possuem o ficheiro inteiro.

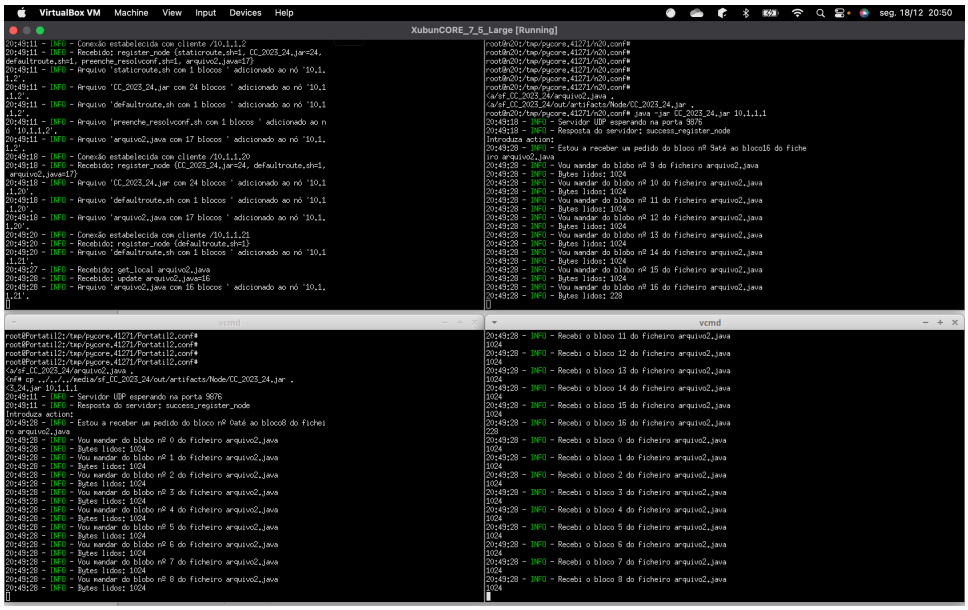

Figure 7: Captura dos terminais de três FS_Nodes (clientes) e um FS_Node (*tracker*) no core, trocando o ficheiro "arquivo.bin".

No exemplo abaixo conseguimos observar que quando o FS_Node deu *exit* o ficheiro "arquivo2.java" foi removido dos registos do tracker, pelo que, quando o outro FS_Node o pediu a mensagem "Ficheiro: arquivo.bin não encontrado." foi enviada.

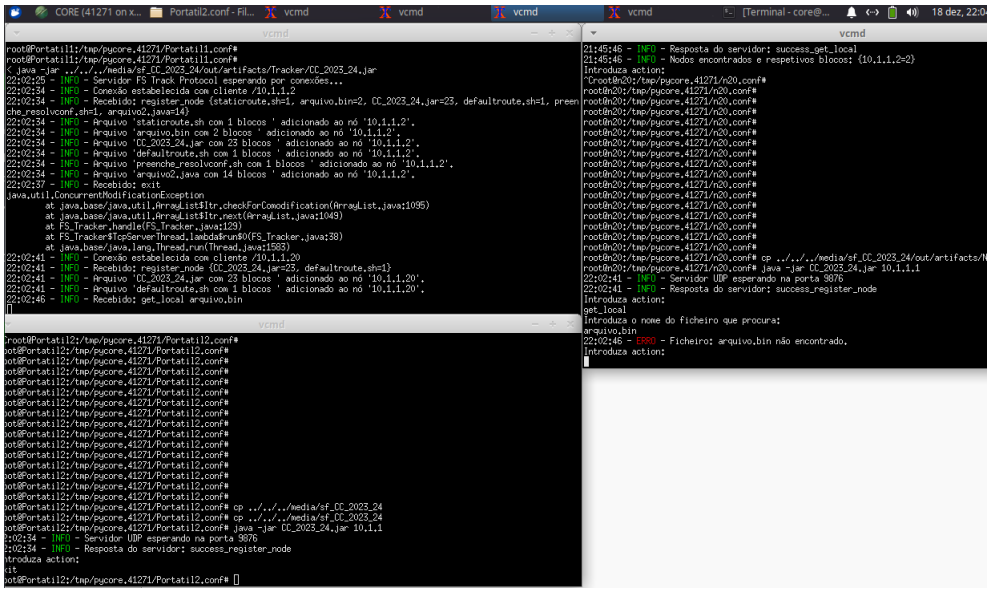

Figure 8: Captura dos terminais de dois FS_Nodes (clientes) e um FS_Node (*tracker*) no core.

6 Conclusões e trabalho futuro

Com a colaboração de todos os elementos do grupo fomos realizando as várias tarefas necessárias para a entrega deste trabalho prático. O processo foi gradual e faseado, ao longo das semanas e das aulas práticas o que permitiu um melhor acompanhamento e envolvência por parte de todos, apesar dos vários desafios.

Sentimos as maiores dificuldades na construção das mensagens UDP e toda a organização que as mesmas envolvem. O facto da transferência de blocos ser simultânea e por consequente ser necessária ordenar os diversos blocos também foi desafiante apesar de, no fim, conseguirmos uma solução eficaz e relativamente simples na nossa perspetiva.

Um ponto negativo é relativo à fase 3, não foi feita a implementação da resolução dos nomes, pois surgiram-nos várias duvidas naquilo que seria a transicção do IP para o nome.

Reconhecemos melhorias na implementação bem como na legibilidade do código em geral, no entanto consideramos que desenvolvemos com sucesso o trabalho.