Enunciado de PCD - 2024/25
IscTorrent: sistema de partilha de ficheiros distribuído

Luís Mota

Versão 3 *- 5 de novembro de 2024

Resumo

Neste projeto pretende-se desenvolver um sistema de descarregamento de ficheiros binários (neste caso ficheiros sonoros) distribuído. Devem existir vários utilizadores na rede P2P e todos podem fazer pedidos de ficheiros aos outros utilizadores ou responder aos pedidos recebidos enviando blocos dos ficheiros pretendidos. O foco do trabalho é na aplicação de programação concorrente e distribuída, sendo a parte gráfica das aplicações dos utilizadores considerada um aspeto secundário.

Para bem servir os propósitos de aprendizagem, as características do sistema foram adaptadas, por forma a serem um bom cenário de aplicação de questões habituais no domínio da concorrência.

1 Descrição genérica do projeto

1.1 Arquitetura

Em termos gerais, a solução deve apresentar uma arquitetura P2P (Peer-to-peer) onde os nós da rede trocam mensagens entre si sempre que pretendem procurar ou pedir o descarregamento de um ficheiro. A solução a implementar não deve recorrer a um servidor central que garanta a ligação entre as aplicações dos diferentes utilizadores.

*__Nota:__ o enunciado pode ser alterado para melhorar a clareza ou adequar-se melhor às necessidades de aprendizagem. Qualquer nova versão será publicada no Moodle e será alvo de uma notificação por email.

1.2 Nós

Tratando-se de uma rede ad-hoc, cada nó só poderá comunicar diretamente com os nós de que conheça o endereço. Por isso, será necessário a GUI dispor de uma ferramenta para introduzir o endereço de outros nós, com os quais se estabelecerá ligação. Para estar integrado na rede, cada nó deve estar ligado a pelo menos outro nó, podendo porém estar ligado a vários simultaneamente. As ligações com outros nós podem ser estabelecidas ativamente, através do envio de um pedido de ligação, através do envio de uma mensagem da classe `NewConnectionRequest`, ou passivamente, quando se recebe um desses pedidos. Logo que estabelecida esta ligação, deixa de ser relevante quem efetuou e quem recebeu o pedido, pois a comunicação poderá ser iniciada por qualquer um dos nós.

1.3 GUI

Os utilizadores podem fazer três ações diferentes, através de uma GUI associada ao seu nó:

1. estabelecer a ligação a outro nó;
2. fazer pesquisas por palavras chave, que serão enviadas a todos os nós com que se tem ligação;
3. pedir o descarregamento de um ficheiro que exista em pelo menos um utilizador encontrado na pesquisa.

1.3.1 Procura de ficheiros

Quando o Utilizador A pretender descarregar um ficheiro, terá de começar por saber quem está a disponibilizar esse ficheiro. Para isso deve pedir a lista dos ficheiros existentes que possuem uma determinada palavra chave, incluindo o endereço dos nós os detêm e podem, por isso, permitir o descarregamento. Esta pesquisa apenas será feita aos outros nós a quem este nó está diretamente ligado.

Para ilustrar o protocolo de procura junto dos nós existentes, descreve-se um caso, esquematizado na Figura 1. Neste exemplo, a procura é iniciativa do utilizador no nó A. Este nó apenas está ligado aos nós B e C, pelo que apenas a estes envia o pedido de procura. A procura deve ser feita através de um canal de objetos, usando uma classe específica com este fim, designada `WordSearchMessage`. A resposta deve incluir uma lista com uma instância de `FileSearchResult` para cada ficheiro que inclua no seu nome a palavra procurada. A classe `FileSearchResult` deve conter como atributos a

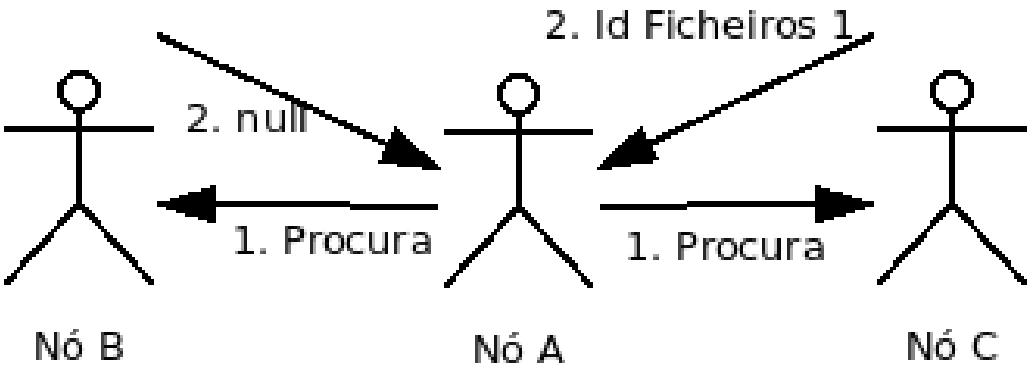

Figura 1: Exemplo de procura de ficheiros por palavra chave

instância de `WordSearchMessage`, o *hash* (descrito mais à frente), tamanho e nome do ficheiro, bem como o endereço e porta do nó. O nó B, não dispondo neste caso de nenhum ficheiro correspondente à palavra chave, responderá com uma lista vazia (ou `null`). Já o nó C, dispondo de ficheiros correspondentes à palavra chave procurada, enviará imediatamente uma mensagem com a lista dos dados dos ficheiros em questão, representados pela referida classe `FileSearchResult`.

Note-se que, para simplificar, um nó apenas deve assinalar que pode fornecer um ficheiro se o detiver na sua totalidade: se apenas detiver partes deste por ainda estar a descarregá-lo, não deve responder positivamente a uma procura.

1.3.2 Descarregamento de ficheiros

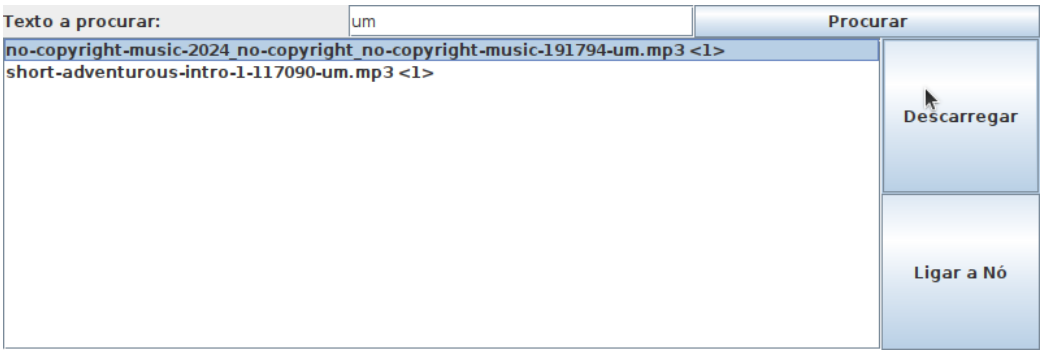

Figura 2: IG com resultados da pesquisa

Uma vez recebidos resultados da pesquisa, estes devem ser exibidos na GUI, em particular o nome do ficheiro e o número de nós que o disponibi-

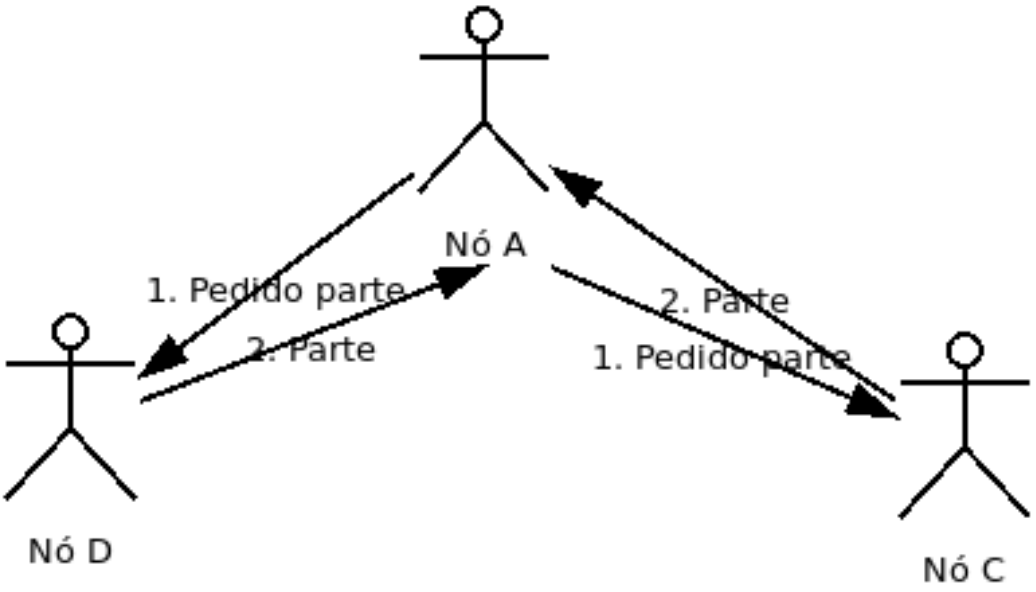

Figura 3: Exemplo de descarregamento de ficheiro

lizam, conforme visível na figura 2. Caso o utilizador selecione um destes ficheiros, proceder-se-á ao descarregamento propriamente dito, conforme esquema na Figura 3. Este descarregamento deve ser feito em blocos, para não sobrecarregar nenhum nó e para poder descarregar blocos em simultâneo a partir de vários nós. O tamanho de cada bloco a descarregar deve ser definida globalmente através de uma constante, e sugere-se que tenha o valor de 10240 bytes, sujeito a alteração. Os blocos devem ser sucessivamente pedidos aos diferentes nós que detenham o ficheiro através de uma mensagem do tipo `FileBlockRequestMessage`, que deverá identificar o `hash` do ficheiro, o "*offset*" - índice do byte onde começar; e "*length*" - número de bytes a ler, normalmente igual ao tamanho padrão do bloco. Porém, tem de se ter atenção ao facto de o último bloco a descarregar poder ter um tamanho mais pequeno. A resposta ao pedido deve devolver os dados em binário do bloco, encapsulados numa instância da classe `FileBlockAnswerMessage`, a desenvolver.

Antes de se iniciar o descarregamento, deve ser criada uma lista com todos os blocos a descarregar. Um novo pedido deve apenas ser enviado a um utilizador quando este terminar o envio do bloco anterior. Assim garante-se que os utilizadores mais eficientes vão receber um maior número de pedidos. O descarregamento deve ser feito por várias threads, uma para cada nó disponível para fornecer o ficheiro.

1.4 Detalhes de execução

A aplicação do utilizador (*IscTorrent*) deverá poder ser lançado num processo Java normal, por exemplo executando:

```
java IscTorrent 8081 dl1
```

Os argumentos de execução representam, por ordem, o porto em que este nó vai receber pedidos de outros utilizadores e o nome da pasta onde se encontram os ficheiros partilhados (designada pasta de trabalho, onde aliás também serão guardados os ficheiros descarregados). Para simplificar, sugere-se que esta pasta seja criada na raiz do próprio projeto Eclipse, sem ser dentro da pasta `src`.

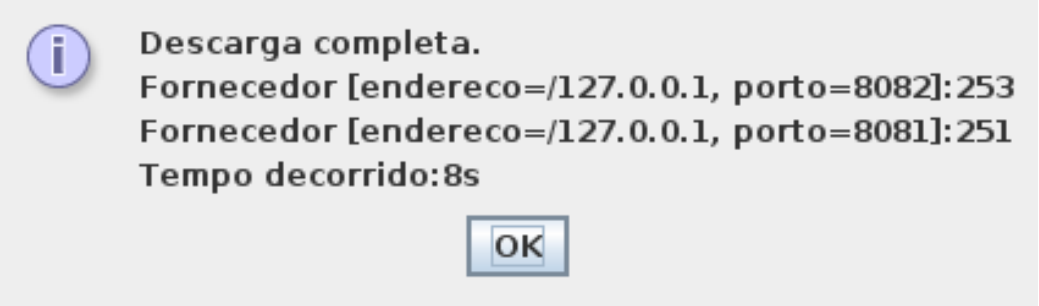

Figura 4: Mensagem exibida quando um descarregamento terminar

Uma tarefa de descarregamento deve ser executada em *thread* autónoma, de maneira a permitir que se possam fazer novas procuras enquanto um ficheiro é descarregado, e eventualmente lançar novos descarregamentos.

Quando um ficheiro estiver totalmente descarregado, deve ser indicado quantas partes foram descarregadas de cada outro utilizador, bem como o tempo consumido, conforme exemplo na Figura 4. A escrita do ficheiro em disco deve ser efectuado por uma `thread` dedicada que deve esperar (ficar em `wait`) até que todos os blocos do ficheiro tenham sido descarregadas.

Para não sobrecarregar os nós com pedidos excessivos, sempre que um utilizador recebe um pedido de uma parte de ficheiro este deve ser colocado numa lista de tarefas. As tarefas da lista devem ser tratadas por um número pré-definido de threads (por exemplo 5). Assim garantimos que se existirem por exemplo 20 utilizadores a pedirem ficheiros, a aplicação vai responder no máximo a 5 de cada vez, isto é, serão enviados no máximo 5 ficheiros simultaneamente. Deve implementar todas as estruturas necessárias para coordenar estas *threads* trabalhadoras bem como as *threads* que recebem os pedidos.

1.5 Ligações

Neste projecto a aplicação do utilizador vai desempenhar vários papéis:

- Como servidor, para receber a ligação inicial de outros nós;
- Como servidor, para receber, de outros nós, procuras ou pedidos de blocos de um determinado ficheiro;
- Como cliente, para se ligar a outros nós para fazer os pedidos anteriormente identificados.

Todos os intervenientes neste sistema deverão funcionar em *multi-threading*, assegurando a máxima disponibilidade para receber novas ligações, bem como reagir a pedidos das ligações já existentes.

1.6 Interface gráfica (GUI)

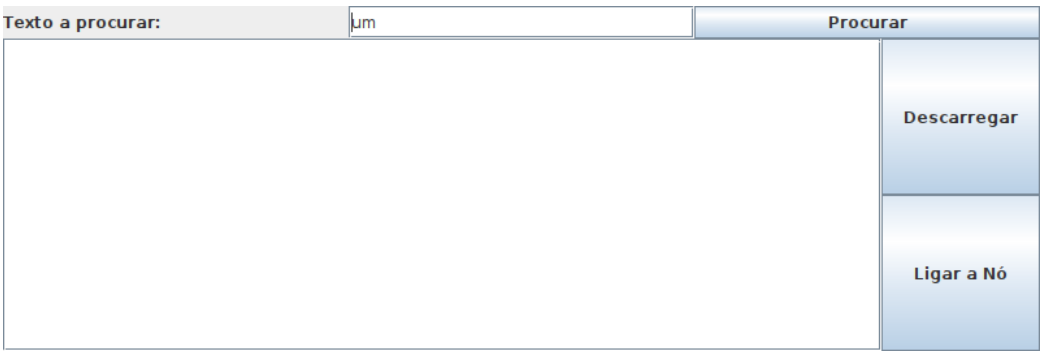

Figura 5: Interface gráfica a implementar

Deve criar uma GUI à imagem do exibido na Figura 5: no topo existe uma caixa de texto para inserir o texto a procurar e um botão ("Procurar") para lançar essa procura. Em baixo, serão exibidos, à esquerda, os resultados da procura, numa `JList`[1]. Do lado direito haverá dois botões: o primeiro para lançar o descarregamento do ficheiro selecionado à esquerda, e o segundo para poder introduzir o endereço e porta de um nó a que se queira fazer ligação.

[1] A classe `JList` será abordada num dos turnos práticos após a semana de publicação do enunciado.

2 Requisitos do projeto

O projeto é uma ferramenta de aprendizagem, pelo que deve servir para adquirir conhecimentos em vários tópicos do programa de PCD, que passamos a elencar e que devem ser escrupulosamente seguidos.

Todos os mecanismos de coordenação devem ser desenvolvidos pelo próprio grupo, não devendo ser usados os equivalentes disponibilizados nas bibliotecas padrão do Java. Como exceção, pode ser usada a `ThreadPool` do Java.

Lançamento inicial dos nós Os nós, aquando do início da execução, não estão ligados a nenhum outro nó. Só quando alguma dessas ligações for feita é que o nó estará plenamente funcional e parte integrante da rede.

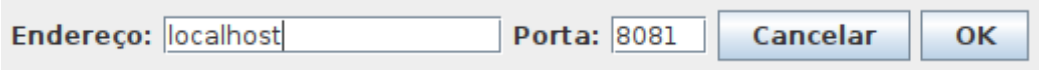

Figura 6: Interface gráfica para efetuar ligação a nó remoto

Ligação a nós remotos A ligação a outros nós será desencadeada ao premir um botão dedicado na GUI. Deve ser exibido um diálogo que permita introduzir o endereço e porta do nó remoto, conforme visível na figura 6

Sincronização de acesso às secções críticas Como sempre necessário em situações de acesso concorrente a informação partilhada, devem ser identificadas as secções críticas e devidamente protegidas de interferências, usando o mecanismo de sincronização ou equivalente, que assegurará a exclusão mútua de acessos por diferentes *threads*. A sincronização deve, naturalmente, ser tão localizada quanto possível, para permitir o máximo de concorrência.

Leitura do conteúdo da pasta de trabalho Para poder responder a procuras de ficheiros, é essencial proceder à leitura da pasta de trabalho quando a aplicação arrancar. Não é necessário, nesta fase, carregar o conteúdo de nenhum ficheiro: tal poderá ser feito apenas quando se receberem pedidos de descarregamento para ficheiros em particular.

Coordenação para receção de resultados Os resultados de uma pesquisa podem ser recebidos assincronamente, através de várias mensagens recebidas independentemente umas das outras. Esta situação deve ser tratada

corretamente, integrando informação recebida em vários momentos, permitindo esperar pelo fim do descarregamento para fazer a escrita do ficheiro para o disco. Estas tarefas de coordenação devem ser implementadas numa classe de coordenação (que é um recurso partilhado entre as *threads* que fazem o descarregamento) designada `DownloadTasksManager`.

Coordenação para a escrita em disco do ficheiro descarregado O conteúdo de um ficheiro descarregado apenas deve ser escrito para o disco quando o seu conteúdo tiver sido descarregado na sua totalidade.

Aplicação de `ThreadPool` A resposta a pedidos de descarregamento devem ser tratados por `threads` dedicadas: cada descarregamento deve ter um `thread` associada. A execução destas `threads` deve ser gerida por uma `ThreadPool`: em cada momento apenas devem estar em execução simultânea um número de `threads` definido numa constante, que se sugere que tenha o valor 5.

Função de `hash` Cada ficheiro terá um valor de `hash` associado. Este valor deve ser calculado pelo algoritmo `SHA-256`, sugerindo-se para tal a utilização da classe MessageDigest[2]

Comunicação em rede O sistema a desenvolver exige troca de mensagens entre diferentes programas, para fazer a pesquisa e descarregamento de ficheiros. Todas estas mensagens devem ser transmitidas sobre canais de objetos.

Extra: Descarregamento simultâneo de vários ficheiros para flexibilizar as tarefas que podem levar mais tempo a executar, seria útil que pudessem ser feitos vários descarregamentos em simultâneo. Para tal, teriam de ser replicados os objetos de coordenação `DownloadTasksManager` para cada descarregamento em curso.

Extra: Procura alargada para alargar a abrangência da procura, cada nó, quando recebesse uma procura por ficheiros, não se limitava a procurar nos próprios ficheiros: retransmitia também essa procura a todos os nós a que estivesse ligado, evitando, naturalmente, reenviar a procura a nós por que esta já tivesse passado. Esta valência é muito difícil de implementar

[2] `https://docs.oracle.com/javase/8/docs/api/java/security/MessageDigest.html`

e só deverá por isso ser explorada após a conclusão de todas as outras fases, e apenas por alunos particularmente motivados para explorar problemas de difícil resolução e compreender conceitos mais avançados e exigentes de programação concorrente e distribuída. Quem entender ter interesse neste desafio mais ambicioso deve contactar o coordenador da cadeira.

3 Fases de desenvolvimento

Nesta secção sugere-se um encadeamento de fases que permitem um desenvolvimento sustentado deste projeto. As primeiras quatro fases devem idealmente ser cumpridas no início, pois vão constituir a entrega intercalar. Não é obrigatório ter todas estas fases concluídas nessa altura, mas é desejável.

1. **Desenvolvimento da GUI:** desenvolver uma GUI funcional com as valências descritas em 1.3.

2. **Leitura dos conteúdos da pasta de trabalho:** No arranque da aplicação, deve ser lido o conteúdo da pasta de trabalho, informação que deve ser salvaguardada para posterior utilização. Sugere-se a consulta do exercício de visualização de fotografias da 1ª semana.

3. **Criação de lista de blocos:** a partir da informação de um ficheiro, criar uma lista de instâncias de `FileBlockRequestMessage` (classe a desenvolver), que servirão de base ao descarregamento. Esta lista deve ser guardada como atributo da classe `DownloadTasksManager`.

4. **Estabelecimento das ligações entre nós:** desenvolvimento da solução para o estabelecimento de ligações entre nós seguindo o modelo cliente/servidor. Após a ligação inicial, qualquer dos nós deve poder mandar mensagens para o outro nó.

5. **Desenvolvimento da capacidade de procura remota:** desenvolver o protocolo de envio de mensagens para obter os nós que têm em seu poder ficheiros correspondentes à procura efetuada.

6. **Descarregamento distribuído do ficheiro:** com base na lista de blocos desenvolvida atrás (ponto 3), organizar as tarefas de descarregamento a pedir e tratamento dos dados recebidos, com, no fim do descarregamento, a consequente escrita dos dados em disco. Estas relevantes tarefas de coordenação devem ser desenvolvidas no âmbito da classe `DownloadTasksManager`. Como indicado acima, deve ser criada uma *thread* de descarregamento de blocos para cada nó remoto que disponha desse ficheiro.

7. **Exibição de resultado do descarregamento:** deve impreterivelmente ser exibida uma janela semelhante à visível na Figura 4, para ser possível avaliar se o descarregamento foi de facto feito de forma concorrente.

8. **Descarregamento simultâneo de vários ficheiros (Extra):** para permitir maior flexibilidade, deve poder selecionar-se vários ficheiros simultaneamente na GUI, para serem descarregados concorrentemente.

9. **Utilização de variáveis condicionais (Extra):** para treinar este conteúdo relevante da matéria de concorrência, sugere-se que seja revista a implementação do `DownloadTasksManager`, para passar a utilizar cadeados explícitos e variáveis condicionais.

   Utilização de barreira ou *CountdownLatch* (Extra): para treinar a aplicação destas importantes estruturas de coordenação, faça uma nov a implementação do `DownloadTasksManager`, usando uma destas estruturas para suportar a coordenação desejada.

4 Entregas

Para favorecer o início atempado do desenvolvimento do projeto, existe uma entrega intermédia. Pede-se que seja feita uma apresentação das fases iniciais, referentes à versão concorrente, no turno de laboratório de 12 a 14 de novembro, onde deverá ser feita uma breve apresentação do trabalho feito. Esta entrega é obrigatória, e feita preferencialmente através do Zoom.

A entrega final será às 09:00 de 9.12.2023, em página no Moodle a disponibilizar oportunamente. As discussões finais, de que alguns alunos poderão eventualmente ser dispensados, decorrerão nos laboratórios de 10, 11 e 12 de dezembro e, se necessário, estender-se-ão eventualmente para a semana seguinte. Estas datas ainda estão sujeitas a validação pelo conselho de ano, pelo que não são definitivas e podem sofrer alterações.

O modelo da avaliação é o habitual nas cadeiras de programação: o projeto é obrigatório mas não conta diretamente para a nota, podendo apenas limitar a nota final, conforme critérios apresentados na primeira aula e constante da FUC.

5 Avaliação

Todas as fases descritas na secção 3 devem ser implementadas, pois o projeto é primordialmente uma ferramenta de aprendizagem e essas fases correspon-

dem a partes importantes da matéria a aprender e a ser avaliada. Caso haja lacunas menores nesta implementação, os alunos poderão mesmo assim ter aprovação com uma valoração qualitativa possivelmente mais baixa.

A implementação bem sucedida das últimas fases de desenvolvimento, marcadas como sendo **extra**, permitirá o acesso à avaliação máxima no projeto.

6 Histórico de versões

1 0. Versão original

2 Pequenas clarificações e acrescento de duas fase de trabalho extra opcionais.

3 Clarificação de que as tarefas de coordenação necessárias ao descarregamento devem ser feitas na classe `DownloadTasksManager`.