FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Sistemas Distribuídos | Projeto 1

# Serverless Distributed Backup Service

MIEIC 2020/21

11 de Abril de 2021


**T1 G05**

Francisco Gonçalves | 201704790

Luís André Assunção | 201806140

# 1. Introdução

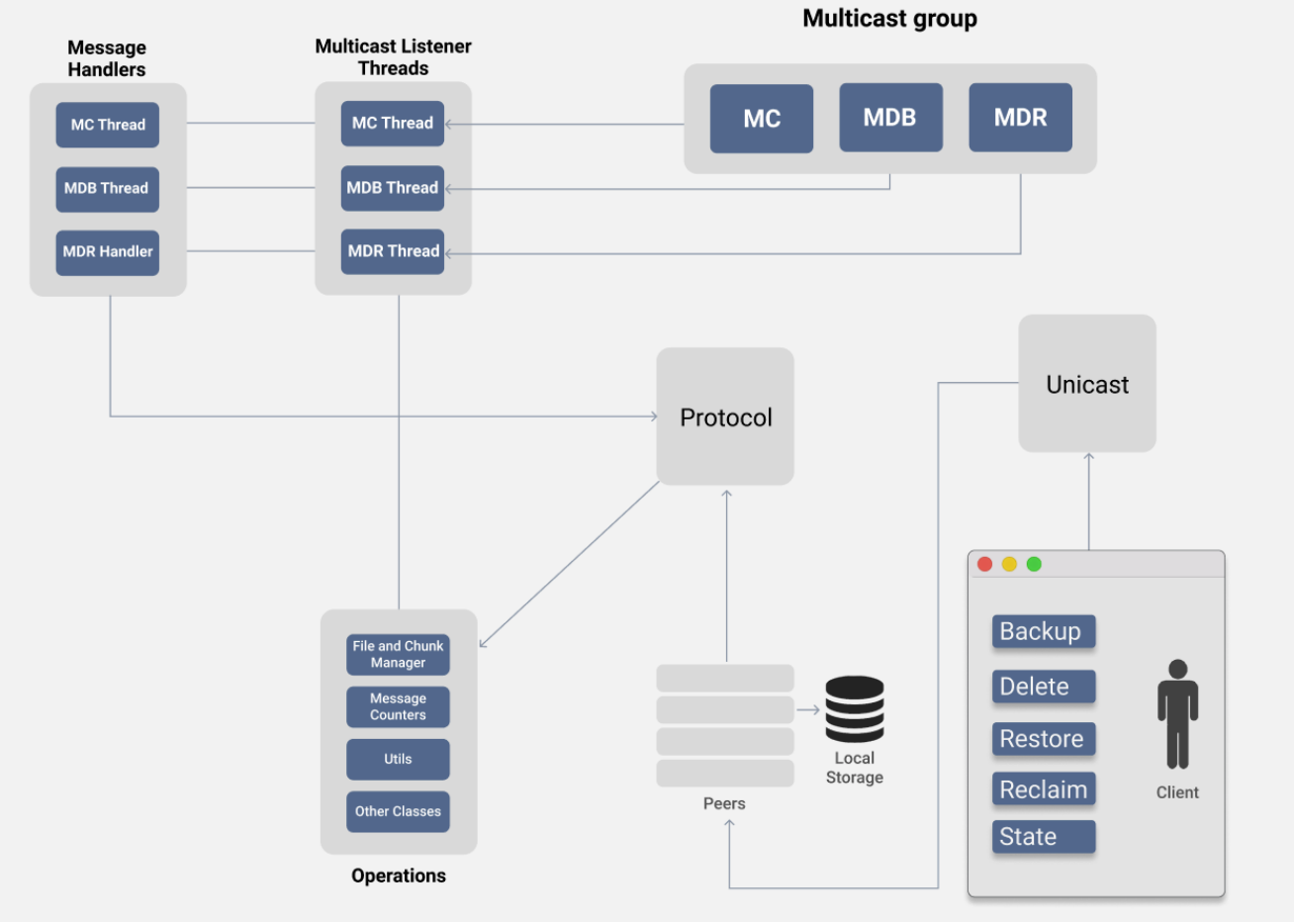
Este é o relatório para o 1º projeto de Sistemas Distribuídos do 3º ano, 2º semestre do MIEIC, com o objetivo de clarificar a implementação e mostrar os detalhes mais relevantes sobre a mesma, entre os quais:

- **Overview da implementação**: Descrição da organização e responsabilidade das classes.
- **Execução concorrente de protocolos**: Descrição das escolhas, estruturas e mecanismos que sustentam execução concorrente de protocolos.
- **Enhancements**: Descrição dos melhoramentos feitos aos protocolos.

# 2. Implementação

## 2.1 Visão geral da estrutura

A estrutura da implementação pode ser vista no diagrama abaixo. O diagrama é composto por classes e elementos, permitindo ter uma ideia do funcionamento geral do programa.



## 2.2 Classes principais

- **Peer**: classe encarregue de receber e processar os pedidos da TestApp
- **MC, MDB, MDR**: classes que estendem *MulticastThread* e representam a Thread que recebe a informação de cada canal de comunicação.
- **FileChunkManager**: classe que trata do processamento de chunks.
- **Utils**: classe com funções recorrentes, entre elas:
    - **readFile**(filepath)
    - **hash**(string)
    - **byteArrayToString**(byteArray)
    - **getFileSizeBytes**(filepath)
    - **mergeFiles**(chunks)
- **MessageHandlers (MC, MDB, MDR)**: classes responsáveis por lidar, processar e agir em função das mensagens recebidas
- **StoredCounter** e **StoredHandler**: classes auxiliares responsáveis por contar as mensagens stored e lidar com o que estas significam.

### 2.3 Concorrência

Para lidar com a concorrência entre os protocolos, tivemos alguns cuidados. Entre eles destacamos alguns nesta secção.

- MC, MDB estão sempre a ouvir mensagens, o que permite executar protocolos diferentes simultaneamente. Para além disso, como não há recursos partilhados e como cada protocolo assegura que apenas trata de mensagens coincidentes com o ChunkNo e FileID em questão. Para além disso, cada mensagem é enviada a partir de uma thread diferente, pelo que não há mensagens numa fila de espera para serem processadas.

- Cada vez que uma mensagem é recebida num dos canais (representados pelas classes MC, MDB e MDR), é iniciada uma nova *Thread* com as classes do tipo ***MessageHandler*** que implementam *Runnable*. Estas classes tratam de processar as mensagens recebidas. Por baixo podemos ver um excerto de código da classe ***MDBMessageHandler*** onde recorremos a um ***Single Thread Executor*** que permite submeter uma *callableTask* onde contamos o número de mensagens **STORED** recebidas, esperando um tempo aleatório entre 0 e 400ms.

```
int storedMessages = 0;
ExecutorService executor = Executors.newSingleThreadExecutor();
Future<String> future = executor.submit(new StoredCounter(port, ipAddress, header.getFileId(),header.getChunkNo(),header.getSenderId(), header.getReplicationDeg()));

try {
    int random = (int) (Math.random()*400);
    String ret = future.get(random, TimeUnit.MILLISECONDS);
    storedMessages = Integer.parseInt(ret);
    executor.shutdown();
} catch(InterruptedException | ExecutionException e) {
    e.printStackTrace();
} catch(TimeoutException ignored) {}
```

- A memória disponível de cada peer está guardada num ficheiro partilhado, que é manuseado usando *asynchronous file channels*.

## 3. Enhancements

**Protocolo de Backup**: A otimização que fizemos a este protocolo foi a de fazer com que os peers guardem um *chunk* se o *replication degree* ainda não tiver sido atingido. Assim, cada Peer mantém uma contagem de quantos Peers já guardaram o *chunk* e só o guardam se o valor de replicação ainda não estiver satisfeito. Esta solução ajuda bastante na eficiência do programa, já diminui bastante a probabilidade de um *chunk* ter um grau de replicação superior ao desejado.