universidade de aveiro

Projeto Final

Editor de Música Distribuído

Licenciatura em Engenharia Informática
Computação Distribuída

Filipe Obrist 107471 - 50%
Jose Mendes 107188 - 50%

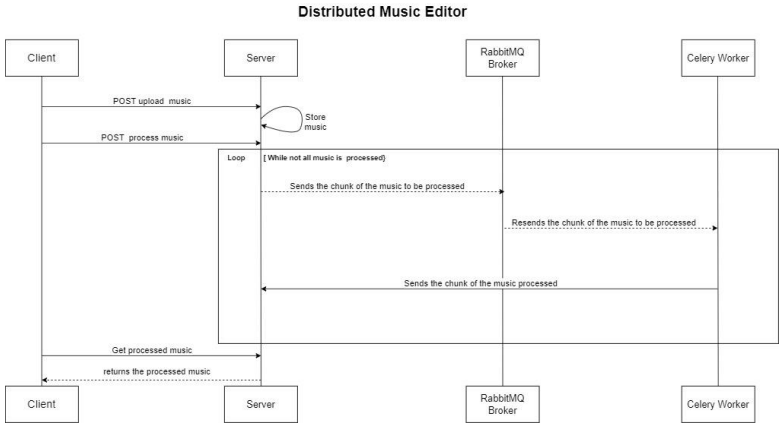
Introdução

Neste trabalho pretende-se desenvolver um sistema distribúido que gere a separação de um ficheiro de música nos seus vários instrumentos, dividindo a tarefa de processamento em múltiplas sub-tarefas paralelizáveis com o objetivo de aumentar a performance do sistema.

Tecnologias Utilizadas

Para a concretização dos objetivos deste projeto, as principais tecnologias utilizadas foi a web framework **Flask**, a Library **Celery** e o broker para ser utilizado neste, o **RabbitMQ**.

Protocolo:



Implementação

Inicialmente, com o intuito de implementar as funcionalidades que este sistema deve suportar, foi utilizado o **Flask**, que utiliza o protocolo **HTTP**, permitindo a realização de pedidos **POST** e **GET** do cliente para o servidor.

De seguida, sempre que é inserido uma música (**POST** /music), é gerado um ID único com o auxilio da função generateID() e um objeto Lock da biblioteca threading, garantindo a exclusão mútua, não permitindo que duas músicas recebam o mesmo ID (também foi aplicado cada vez que surge um novo Job ou Track).

Posteriormente, quando é realizado um requisito para o processamento da música (**POST** music/{musicID}), esta é dividida em múltiplas partes e enviada para uma Task Queue, no caso, **AMQP** (Advanced Message Queuing Protocol) implementada pelo broker **RabbitMQ**. Cada uma destas tarefas vai ser enviada para os workers ativos que realizam o processamento da música. O **Celery** possui politicas de load balancing permitindo para fazer a distribuição de tarefas pelos workers, garantindo a concorrência dos processos.

Por fim, após o processamento de cada pedaço da música, estes são enviados de volta para o servidor utilizando o método de callback fornecido pelo **Celery**. A música é reconstituída e são gerados os links permitindo download dos instrumentos individuais e da música pretendida.

Nota: Em todos os instantes que a música ou os seus pedaços são enviados entre servidor e worker é utilizado o mecanismo de serialização **JSON**, passando as imagens em bytes codificadas em **base64**.

Tolerância a Falhas e Redundância

Ao utilizar o Celery e RabbitMQ, a tolerância a falhas e redundância foi alcançada por meio da colocação da variável **task_acks_late** a true, o que fará com que as tarefas só serão "acknowledged" após serem concluídas com sucesso e também por meio de **filas duráveis** (caso qualquer worker falhe, o RabbitMQ irá guardar a mensagem no fim da fila, sendo depois processada por outro worker - redundância passiva).

Estruturas (entre outras)

id_usados - Lista de id usados para verificação da unicidade dos ids.
musicas - Lista de ids das músicas uploaded
idBytes - Dicionário para guardar os bytes associados ao id da música
idTracks - Dicionário para guardar as tracks selecionadas pelo cliente de uma música
callbacks - Dicionário para guardar as callbakcs de cada música
jobCallback - Dicionário para guardar as callbacks de cada job
jobs - Lista para guardar variáveis de classe Job
tracks - Lista onde guardamos todas as tracks criadas
IdProgress - Dicionário para guardar o progresso de cada música pelo seu id
Classes - Criação de várias classes (Music, Job, Track, Instrument, Progress)

Resultados

Para testar o sistema, o grupo decidiu colocar uma música teste fornecida de 59 minutos e 4 segundos (music.mp3) com o intuito de obter apenas o baixo e a bateria.

Com apenas um worker o processo demorou cerca de 1 hora, 27 minutos e 5 segundos, com dois workers cerca de 40 minutos e 54 segundos e, por fim, com quatro workers, 30 minutos e 25 segundos, sendo desta forma percetível a melhoria significativa.

Conclusão

Concluindo, este trabalho apresentou a implementação de um sistema distribuído utilizando tecnologias Flask, Celery e RabbitMQ. No geral, estas tecnologias possibilitaram a construção de um sistema distribuído robusto e confiável, capaz de lidar com o processamento de músicas.