Distributed Computing Project

Bachelor's Degree in Computer Engineering - Distributed Computing

Distributed Music Editor

Teachers:

- Diogo Gomes (dgomes@ua.pt)
- Nuno Lau (nunolau@ua.pt)
- Mario Antunes (mario.antunes@ua.pt)
- Alfredo Matos (alfredo.matos@ua.pt)

Term:

June 5th – 24h00

Concepts to address:

- sockets
- Marshalling
- Message Broker
- Fault Tolerance

Introduction

Advanced Sound Systems (ASS) is developing a karaoke application for musicians. This application distinguishes itself by not only removing the vocals from songs, but also removing individual instruments, allowing a musician to replace the original musician's performance with his own. This new service will be made available online through a web portal where the musician can upload a music file, analyze the instruments that make up the music, select several instruments and finally receive a new file in which the music contains only those instruments.

The task of separating a song into several tracks per instrument is intensive from a processing point of view, so ASS hired Distributed Computing students to develop an online service capable of meeting the company's needs with a high quality of user experience. final (quick identification of the instruments and construction of the new file without the instrument(s) selected).

For this task, they must develop all the necessary code to create a Web service that can serve multiple clients in parallel quickly and efficiently using parallel computing (here achieved through parallelization in multiple independent processes/workers that can be on the same computer or not).

This script and a set of music tracks available at [4] are provided for this task.

project https://classroom.github.com/a/q9wGcN9U. to clone your at the GitHub Classroom using Must

The repository already includes example code [1][2][3] on how you can process MP3 files from music and detect 4 different tracks (number that will be enough for this project).

Objectives

With this work it is intended that students develop a system that divides the processing task into multiple parallelizable sub-tasks with the aim of increasing the system's performance.

Operation

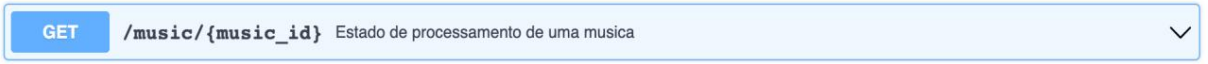
This is an open job in which they will have to develop a system for distributing computing tasks (separation of instruments) and scheduling tasks to serve the backend to the ASS portal.

The system will have a REST API as an interface, through which the portal will be able to:

Submit audio files in MP3 format. In response to the audio file submission, the service must return a JSON file containing a work identifier (music_id), and a list of instrument track identifiers (each identifier must correspond to an instrument). This method only stores and analyzes the song's metadata.

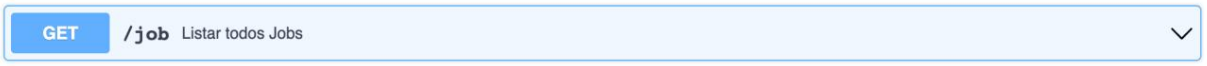
List all submitted songs

Request asynchronous processing (this method should return immediately) of the song <music_id> with only the instruments listed via their identifier in a submitted JSON file. The system must separate the music into its various instruments, resulting in a file per instrument. In the end, the system should mix the selected instruments into a single file.

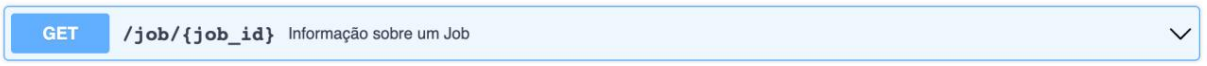


Requesting the processing status of a song, when the status is 100% the response must include the url of the final file. When an instrument's track is complete, it should return the link to the file for each instrument (allowing the user to download and play the file). At the end there will also be a link to the final mixed file (containing the selected instruments).

To facilitate the use of this system, you should also implement the endpoints:



List all JOBs (JOB is a task distributed to a worker) submitted and their statistics (file size, link to the various files, processing times, etc.)



List JOB's used to process, processing time, number of instruments

Delete all temp files and music (system clean), cancel all workers still running.

The API is documented using the OpenAPI 3.0 specification. You can find both the source file and PDF and HTML versions in their code repository (api.yaml, api.pdf, api.html respectively). Please note that there may be a need for updates before the project's delivery date, so be aware of the announcements on the #cd channel at detiuaveiro.slack.com

You should also create a **basic** Web portal (frontend) for this project for the purposes of **demonstration.** For development and testing purposes have audio files in [3].

Assessment

The evaluation of this work will be done through the submission of code on the GitHub classroom platform and a report in PDF format with no more than 5 pages placed in the same repository along with the code and with the name reportrio.pdf .

The defined and documented protocol is under evaluation, as well as the *features* implemented according to the objectives:

• Web service with REST API following the work requirements •
Distributed solution for processing audio files o Separation of the
original file into multiple sections o Identification of the
instruments by separating the sub-tracks o Creation of a new complete
file • Efficiency of the solution
developed (the aim is to have a system capable of responding as quickly as possible to
requests from end customers) • Serving multiple customers
simultaneously. • Attend to the possibility of
failures occurring

Comments

1. The example code makes use of all physical CPU's by default, for local tests it is necessary we limit it to 1 CPU. For this purpose, add the following code:

   ```
   import torch
   torch.set_num_threads(1)
   ```
2. You can use https://editor.swagger.io website to consult all the details of the file api.yaml
3. To get the best performance from your system you should break the asynchronous music processing task into smaller pieces of music and process them in parallel.

References

[1] https://github.com/facebookresearch/demucs
[2] https://pytorch.org/
[3] https://torchmetrics.readthedocs.io/en/stable/
[4] https://filesender.fccn.pt/?s=download&token=cd4fcd29-b3f1-4a4d-9da3-50aae00e702d