EasyTopic: Uma arquitetura para extração de dados e segmentação semântica de vídeos educacionais

Maxwell Souza de Carvalho

JUIZ DE FORA

MARÇO, 2021

EasyTopic: Uma arquitetura para extração de dados e segmentação semântica de vídeos educacionais

Maxwell Souza de Carvalho

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Eduardo Barrére

JUIZ DE FORA

MARÇO, 2021

EasyTopic: Uma arquitetura para extração de dados e segmentação semântica de vídeos educacionais

Maxwell Souza de Carvalho

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Eduardo Barrére
Doutor em Engenharia de Sistemas e Computação

Eduardo Pagani Julio
Doutor em Computação

Jairo Francisco de Souza
Doutor em Informática

JUIZ DE FORA
1 DE MARÇO, 2021

A minha família, pelo apoio e sustento.

Aos meus amigos, pelo suporte e noites em claro juntos.

Aos meus avós, pelo amor e proteção.

A Amanda, por me apoiar desde o início.

Resumo

O Aprendizado online se tornou vital nos últimos anos, principalmente durante a atual pandemia. Nele, se faz uso de vídeos e palestras educacionais, além de materiais de leitura. Entretanto, dado a vasta quantidade de material disponível, se torna árdua a tarefa de buscar um tópico especifico. A arquitetura propõe a segmentação em tópicos pesquisáveis de vídeos, de forma automática. Foi desenvolvido um sistema em pipeline o qual executa este processo, e o mesmo foi melhorado para possibilitar a criação e gerenciamento de novos pipelines de forma prática e intuitiva.

Palavras-chave: Videoaula, Educação, Segmentação em Tópicos.

Abstract

Online learning has become vital in recent years, especially during the current pandemic. It uses educational videos and lectures, as well as reading materials. However, given the vast amount of material available, the task of searching for a specific topic becomes arduous. The architecture proposes the segmentation in searchable video topics, automatically. A pipeline system was developed which performs this process, and it has been improved to enable the creation and management of new pipelines in a practical and intuitive way.

Keywords: Video lecture, educational, topic segmentation.

Agradecimentos

A minha família, pelo encorajamento e apoio, além de suporte financeiro.

Aos meus avós e a Amanda, que sempre me apoiaram nos momentos difíceis e adversidades

Ao professor Eduardo Barrére pela orientação, amizade e principalmente, pela paciência, sem a qual este trabalho não se realizaria.

Aos professores do Departamento de Ciência da Computação pelos seus ensinamentos e aos funcionários do curso, que durante esses anos, contribuíram de várias formas para o nosso enriquecimento pessoal e profissional.

"A água pode fluir ou pode colidir. Seja água, meu amigo."

Bruce Lee

Conteúdo

Lista de Figuras

Lista de Abreviações

DCC	Departamento de Ciência da Computução
UFJF	Universidade Federal de Juiz de Fora
EAD	Ensino a distância
TCC	Trabalho de conclusão de curso
ASR	Automatic Speech Recognition
OCR	Optical character recognition
AMQP	Advanced Message Queuing Protocol
API	Interface de programação de aplicações
JSON	JavaScript Object Notation
REST	Representational State Transfer
SQL	Structured Query Language
PIP	Python Package Index
VAD	Voice activity detector
ID	Identificador

Glossário

- **Framework:** Abstração que une códigos comuns em vários projetos de software, com o fim de fornecer uma funcionalidade genérica.
- **Pipeline:** Canalização, consistindo em uma cadeia de elementos de processamento, de forma a saída de um elemento ser a entrada do próximo.
- **EasyTopic:** Framework para segmentação de vídeos educacionais, por (BARRÉRE; SOUZA; SOARES, 2020).
- **Dashboard:** Ferramenta que disponibiliza informações de um sistema, visualização de dados e monitoramento.
- **Tesseract:** Software de reconhecimento ótico de caracteres de código aberto.
- **Job:** Representa um processamento inicializado no pipeline, possuindo um identificador e argumentos próprios.
- **Socket:** Tecnologia que permite comunicação bidirecional em tempo real.
- **Monorepo:** Estratégia de desenvolvimento de software que consiste em armazenar múltiplos projetos em um único projeto e repositório.
- **JavaScript:** Linguagem de programação interpretada estruturada de alto nível.
- **ReactJS:** Biblioteca JavaScript com foco em criar interfaces interativas em páginas web.
- **NodeJS:** Software de código aberto que executa códigos JavaScript no servidor
- **Python:** Linguagem de programação de alto nível, interpretada e imperativa
- **PIP:** Sistema de gerenciamento de pacotes e dependências da linguagem Python.
- **Axios:** Cliente HTTP para a linguagem JavaScript.
- **GitHub:** Plataforma de hospedagem de código-fonte e arquivos com controle de versão.

- **Docker:** Ferramenta de virtualização de sistemas operacionais para execução de softwares em pacotes chamados containers.
- **Docker Compose**: Ferramenta para orquestração de containers Docker em arquivo de configuração;
- **contêiner**: Pacote gerado na ferramenta Docker, que representa o código de uma aplicação executando em um sistema operacional
- **Express:** Framework para aplicações web em NodeJS.
- **HTTP:** Protocolo de comunicação utilizado como comunicação em sistemas web. Possui como uma das propriedades de sua requisição, o tipo, que tem como alguns usados comumente, os tipos GET, POST e DELETE.
- **SQL:** Linguagem de pesquisa declarativa usada em bancos de dados relacionais.
- **PostgresSQL:** Sistema gerenciador de banco de dados relacional
- **AMQP:** Protocolo da camada de aplicação para sistemas com orientação a mensagens.
- **RabbitMQ:** Software de mensagens de código aberto que implementa o protocolo AMQP.
- **API:** Interface de programação de aplicações que consiste em um conjunto de rotinas e padrões.
- **REST:** Arquitetura de software que define um conjunto de restrições e regras para criação de serviços web.
- **Stateless:** Quando uma aplicação não mantém estado local, recebendo todos os dados necessários para processamento via parâmetro ou de forma externa, e devolvendo seus resultados na conclusão.

1 Introdução

Na educação, são aplicadas metodologias de ensino variadas. Ao longo dos anos, novas formas de aprendizado são consolidadas. Vídeos e palestras educacionais são muito usadas hoje, por ser um formato acessível, reutilizável e de fácil compartilhamento.

Com a evolução da internet nas últimas décadas, a disseminação desse tipo de material se tornou extremamente simples e prática, via plataformas de vídeo e redes sociais como YouTube e Facebook. Plataformas de ensino a distância cada vez mais vem sendo adotadas, por instituições de ensino ou mesmo por pessoas que buscam auto aprendizado.

Com a vinda da pandemia de COVID-19, o ensino remoto se tornou essencial, visto que em muitos países foi declarado estado de quarentena, recomendando confinamento residencial e distanciamento social. Logo, a criação de material para estudo a distância como vídeos educacionais foi ainda mais acelerada.

O formato de vídeos e palestras educacionais permitem maior controle e divulgação do material, além da aplicação de várias metodologias didáticas. Com ele, o estudante pode rever múltiplas vezes uma aula ministrada, ou mesmo recuperar uma aula perdida.

1.1 EasyTopic

1.1.1 Problema

O uso de vídeos gera um problema que é a indexação e busca por conteúdos específicos (tópicos). Em outras mídias, como por exemplo texto, é comum a divisão de assuntos por tópicos e capítulos, sendo possível realizar buscas por palavras chaves e tópicos geralmente de forma simples. Na mídia vídeo, sem segmentação, isto geralmente se torna mais complexo.

Algumas palestras e videoaulas, possuem horas de duração, abrangendo assuntos mais densos. Ao se buscar por um tópico específico, o vídeo em questão pode não ser

encontrado, mesmo abordando aquele tópico. Além disso, encontrar o momento em que um tópico específico é abordado no vídeo pode se tornar uma tarefa árdua.

Uma solução para este problema é a segmentação do vídeo em tópicos. O vídeo possuiria como metadado, marcações temporais em que um novo tópico é abordado, facilitando na navegação e busca pelo mesmo. Porém, a segmentação manual é custosa, considerando o grande volume de informação atuai disponível, que cresce a todo momento.

1.1.2 Segmentação automática

A segmentação temporal automática tem sido um desafio nas áreas de multimídia e e recuperação de informações, devido a complexidade do problema, envolvendo análises semânticas, processamento de áudio e vídeo, além da variedade de formatos disponíveis, com aulas padronizadas ou em estilo livre.

Na literatura, existem diversas propostas (LIN et al., 2005) de abordagens para a automatização da segmentação em tópicos, porém a maioria depende de recursos externos específicos, como legendas, slides e livros. Isto limita muito o uso das abordagens em grande parte das aulas disponíveis. Além disso, o método aplicado varia de acordo com os parâmetros disponíveis, dificultando a escolha do método apropriado.

A framework EasyTopic (BARRÉRE; SOUZA; SOARES, 2020), proposta por Eduardo Soares, permite incorporar módulos para a obtenção de características e dados da video aula, além de permitir incorporar diversos algoritmos de segmentação. Estes módulos podem ser executados de forma individual ou em um fluxo, em sequência. Isto permite a extração de forma simples de dados do vídeo e auxilia pesquisadores na criação e avaliação de soluções para segmentação.

1.2 Objetivo

O objetivo deste TCC é estudar a framework proposta, seu funcionamento e recursos, e identificar pontos de melhoria e possibilidades de novos recursos, para sua implementação.

Após análise, foram identificados alguns pontos, para então, implementar estas

melhorias na framework, como desacoplação dos módulos da framework para uso individual, padronização do formato dos dados de entrada e saída dos módulos, simplificação do processo de criação de uma nova pipeline e disponibilização um dashboard, tanto em forma de API quanto em forma de interface WEB, para criação de novos jobs de processamento, bem como gerenciamento de processamentos anteriores.

Além disso, tem como objetivo a implementação de novos recursos para a framework EasyTopic, como um servidor de arquivos compartilhado de simples comunicação e um novo módulo para reconhecimento ótico de caracteres, a fim de testar e validar as melhorias anteriores.

1.3 Histórico no laboratório

Nesta sessão, serão apresentados recursos que desenvolvi anteriormente no laboratório, durante minha iniciação científica. Estes recursos são aplicados no projeto posteriormente

1.3.1 Módulo de OCR

Foi desenvolvido um módulo de reconhecimento ótico de caracteres, o qual executa o reconhecimento de caracteres em uma imagem, usando a ferramenta Tesseract. Este módulo foi integrado ao projeto, possibilitando seu uso de forma individual, ou mesmo dentro de uma pipeline pré-definida.

1.3.2 Servidor de arquivos

Também foi desenvolvido um servidor de arquivos local, com comunicação via API REST, no qual é possível cadastrar, obter e deletar múltiplos arquivos em massa. Este servidor tem como objetivo, simplificar os módulos da framework, o qual além da responsabilidade de executar sua dada tarefa, também tinham a responsabilidade do gerenciamento de arquivos, o qual era feito via um banco de dados.

No novo formato, os módulos apenas precisariam do identificador ou da URL do arquivo para download, e caso necessário, uma URL para upload. Além disso, foram disponibilizadas bibliotecas para o uso deste servidor de arquivo na linguagem Python,

a qual simplificava o processo de download e upload em métodos simples e reutilizáveis, facilitando a integração aos módulos já existentes.

1.4 Organização do texto

A seguir contém uma descrição de como este trabalho será organizado, descrevendo as sessões posteriores e seus objetivos.

No capítulo 2, é apresentada a revisão bibliográfica, analisando trabalhos relacionados na área de multimídia, os quais tem como objetivo a segmentação automática em tópicos, realizando um comparativo com o projeto EasyTopic.

No capítulo 3 é realizada uma análise da arquitetura proposta pelo projeto EasyTopic, analisando o seu funcionamento e arquitetura, além da análise do atual pipeline de execução para segmentação em tópicos. Também é analisado problemas que a atual arquitetura possui, bem como propostas para sua solução.

Em seguida, é descrito a metodologia aplicada, bem como os passos para solucionar os problemas levantados anteriormente, além de novas propostas para a arquitetura, a fim de simplificar seu funcionamento e utilização. Também são descritas propostas de modificações nos módulos já existentes, para permitir a sua reutilização e escalabilidade. É descrita uma proposta de configuração de pipelines, a qual abrange os atuais pipelines disponíveis e possibilita a criação de novos, de forma simplificada e descritiva.

Em sequência, é abordado a implementação do projeto, aspectos técnicos do EasyTopic e dos novos recursos implementados, bem como seu funcionamento. Também é abordada a validação da solução desenvolvida, com a criação de um novo pipeline utilizando um novo módulo desenvolvido.

No capítulo 4 são realizadas as considerações finais, analisando se o projeto atendeu as propostas e resolveu os problemas citados. Também são analisados melhorias que podem ser feitas no projeto e nos novos recursos adicionados, bem como outros meios e áreas de aplicação da solução

2 Revisão bibliográfica

O assunto deste trabalho é altamente relevante e estudado, e portanto, muitas abordagens e métodos foram propostos para a extração de tópicos em vídeos educacionais. Estas fazem uso de recursos dos materiais, como áudio, vídeo, texto, etc., em níveis variados. Nesta sessão, serão discutidas algumas destas abordagens.

Em (CHE et al., 2016), é proposta uma solução que, baseada na análise acústica das faixas de áudio da palestra, se destaca as sentenças-chaves nas transcrições dos vídeos. Considerando que um bom palestrante saiba quando enfatizar, estas enfases podem ser detectadas através de recursos acústicos de baixo nível(ARONS, 1994), analisando velocidade da fala, energia e afinação. Assim são geradas palavras-chave as quais podem ser usadas como fonte de indexação.

Já em (TOGASHI; YAMAGUCHI; NAKAGAWA, 2006), é aplicado o uso dos mesmos recursos acústicos de baixo nível, porém, combinados com recursos linguísticos de alto nível, analisando palavras-chave, repetições de palavras, frequência dos termos e uso das palavras nas frases. O uso em conjunto de recursos de baixo e alto nível, baseado nos relatos de (TOGASHI; YAMAGUCHI; NAKAGAWA, 2006), teve melhores resultados em relação ao uso destes recursos separadamente.

Em (LIN et al., 2005), foi proposto um método completamente baseado em linguística, no qual se extraem dois conjuntos de recursos. Estes são baseados em discurso e em conteúdo. Enquanto o baseado em conteúdo é composto por estruturas linguísticas, como classes verbais e frases substantivas, o baseado em discurso é baseado em pronomes e sugestões. A estrutura baseada em conteúdo está relacionada ao significado sintático do conteúdo, enquanto o discursivo se relaciona com a vizinhança dos tópicos, segundo (LIN et al., 2005).

Outro método de segmentação de tópicos, proposto por (YAMAMOTO; OGATA; ARIKI, 2003), considera um caso mais específico, onde a aula segue o conteúdo de um livro. O método consiste na associação da transcrição de áudio gerada pelo reconhecimento automático de fala (ASR) com os tópicos do resumo do livro. Este método permite obter

melhores resultados em casos em que um material do tipo é usado, porém, não é viável para casos em que o mesmo não está disponível.

Em alguns casos, videoaulas possuem recursos extras que podem ser analisados para melhorar o resultado da segmentação dos tópicos, como slides ou textos presentes no vídeo. Em (LEE et al., 2017), um método é proposto onde, baseado em técnicas de processamento de imagem para extração de caligrafia em um quadro negro no conteúdo do vídeo, é identificado pontos de corte na videoaula para se obter a segmentação de tópicos. Já em (SHAH et al., 2014), é proposto uma abordagem, onde se utiliza recursos combinados de visuais e de texto, sendo estes slides, vídeo e legenda. Em ambas as abordagens, é proposto a busca por dicas de transição, as quais indicam a mudança do tópico, os quais são combinados com os pontos de transição obtidos anteriormente, melhorando significativamente a taxa de avaliação, segundo os autores.

3 Arquitetura proposta

O projeto EasyTopic é usado como base, com foco em adicionar melhorias e correções no seu funcionamento, bem como simplificação de seu uso. Também são adicionados novos recursos ao projeto. É implementado uma API de dashboard, responsável por inicializar e gerenciar todos os pipelines do projeto, além de uma interface para controle e acesso a informação.

3.1 EasyTopic

O projeto EasyTopic (BARRÉRE; SOUZA; SOARES, 2020), por Eduardo Soares, implementa uma framework para segmentação de vídeos educacionais, na qual usa de recursos acústicos (de baixo nível, como som, ênfase e pausas) quanto de recursos linguísticos (de alto nível, como semântica).

A proposta diferencial do projeto, é sua flexibilidade, de variar o algoritmo ou técnica aplicada na segmentação, de acordo com os parâmetros disponibilizados, possibilitando o uso de apenas um recurso do projeto se necessário.

3.1.1 Arquitetura e comunicação

O projeto foi desenvolvido em Python, aplicando uma arquitetura de microsserviços, onde cada módulo roda como um servidor separado, possibilitando a comunicação entre eles. Cada módulo da segmentação é um microsserviço, executado em um contêiner Docker, o que funciona como um servidor virtual dedicado, com seus próprios arquivos e sistema operacional, permitindo maior configuração e compatibilidade, independente do ambiente.

O projeto e todos os seus módulos em containers, são inicializados via Docker Compose, o qual realiza a orquestração de todos os servidores virtuais. As especificações do funcionamento se dão através de um arquivo de configuração, o qual especifica protocolos de reinicialização, configuração dos módulos via variáveis de ambiente, parâmetros de inicialização, configuração de uma rede virtual interna para comunicação entre os módulos,

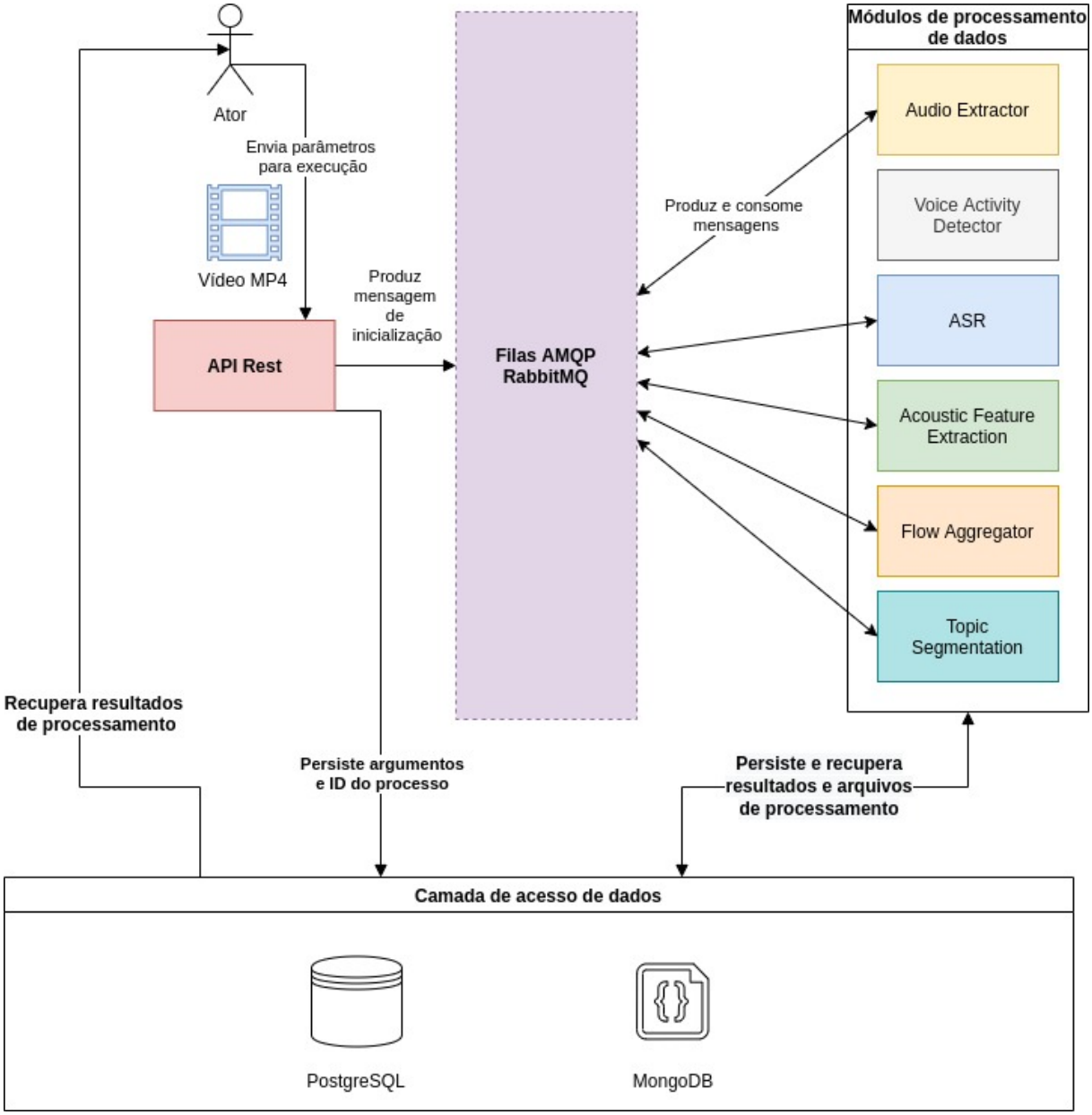

Figura 3.1: Arquitetura Original do projeto EasyTopic

e exposição de portas para acesso externo.

Além dos módulos do sistema, o Docker Compose também inicializa servidores de banco de dados PostgresSQL e MongoDB, junto de um servidor RabbitMQ para gerenciamento de filas AMQP.

A comunicação entre os módulos se dá via filas de mensagens no protocolo AMQP, o qual consiste em um servidor para gerenciamento das filas de mensagens, gerenciado pelo RabbitMQ. Os módulos se conectam a este servidor como consumidores e provedores de mensagens, pois cada módulo possui uma fila de entrada e uma de saída.

Cada mensagem na fila de entrada representa um novo dado a ser processado pelo microsserviço. O Conteúdo da mensagem consiste em um texto no formato JSON, o qual possui os parâmetros necessários para a execução do processamento. Após o processamento, é gerado uma mensagem na fila de saída, com os resultados do processamento, os quais servem de parâmetro para o próximo módulo, que recebe em sua fila de entrada.

Cada módulo também possui uma conexão com servidores de persistência de dados. A mensagem recebida na fila de entrada contém um identificador do processo, atual, o qual é usado para realizar uma busca no banco de dados, para se obter os arquivos resultantes de módulos anteriores.

Com exceção do módulo de API, Todos os módulos do sistema são microsserviços para processamento de dados, que fazem parte do pipeline. O módulo de API REST fornece uma API HTTP para conexão externa, a qual recebe os parâmetros para processamento e inicializa um novo job. Esta dispara na primeira fila de entrada do pipeline os argumentos necessários, e ouve a última fila de saída para disponibilizar o resultado do processamento.

A figura 3.1 ilustra o processo de execução e arquitetura do EasyTopic. Um ator envia para a API REST o arquivo de parâmetro a ser processado, a qual persiste em banco o arquivo e gera um processo com um identificador, para se m seguida produzir a mensagem de inicialização.

Cada Módulo de processamento possui sua fila de entrada, onde recebe os parâmetros e recupera dados e arquivos necessários para o processamento. Após o fim do seu processamento, o mesmo persiste os seus resultados e em seguida, produz uma mensagem para

a fila do próximo item do pipeline.

Enquanto o processamento ocorre, o ator consome diretamente o banco de dados, buscando pelo item enviado, analisando seu status de conclusão, realizando buscas periódicas, até o status retornar concluído, para então obter os resultados do processamento.

3.1.2 Pipeline de execução

O Projeto EasyTopic possuí três pipelines por padrão, ilustrados na figura 3.2 a qual é uma sequência de instruções, onde cada instrução do fluxo é um módulo do sistema. Todos os pipelines são inicializados pelo módulo de API REST, o qual recebe os argumentos para o processamento, e inicializa na fila de entrada do módulo necessário. Cada módulo registra em banco de dados os resultados do seu processamento, como status de conclusão, dados obtidos e arquivos gerados. Estes dados podem ser consultados externamente para verificar se o processamento foi concluído.

O pipeline principal é o de segmentação em tópicos. A partir de uma entrada de vídeo obtida pela API REST, é executada a extração da faixa de áudio da entrada, a qual e seguida pela detecção de atividade de voz (VAD). Em sequência, duas instruções são executadas paralelamente: O reconhecimento automático de fala, e a extração de recursos de baixo nível (como som, ênfase e pausas).

Ambos os resultados obtidos são enviados para um agregador, o qual ao receber o resultado de uma instrução, o armazena e aguarda a finalização da outra instrução, para em seguida enviar os resultados agregados para o próximo módulo. Por fim, é executado o algoritmo de segmentação em tópicos, e seu resultado é salvo em banco de dados, para consulta externa.

O pipeline de ASR executa apenas o módulo de reconhecimento automático de fala. A partir de uma entrada de áudio, esta e passada como parâmetro para o módulo de ASR, o qual executa seu processamento e armazena os resultados em banco de dados, para consulta externa.

Por último, o pipeline de VAD executa apenas o módulo de detecção de atividade de voz. A partir de uma entrada de áudio, esta e passada como parâmetro para o

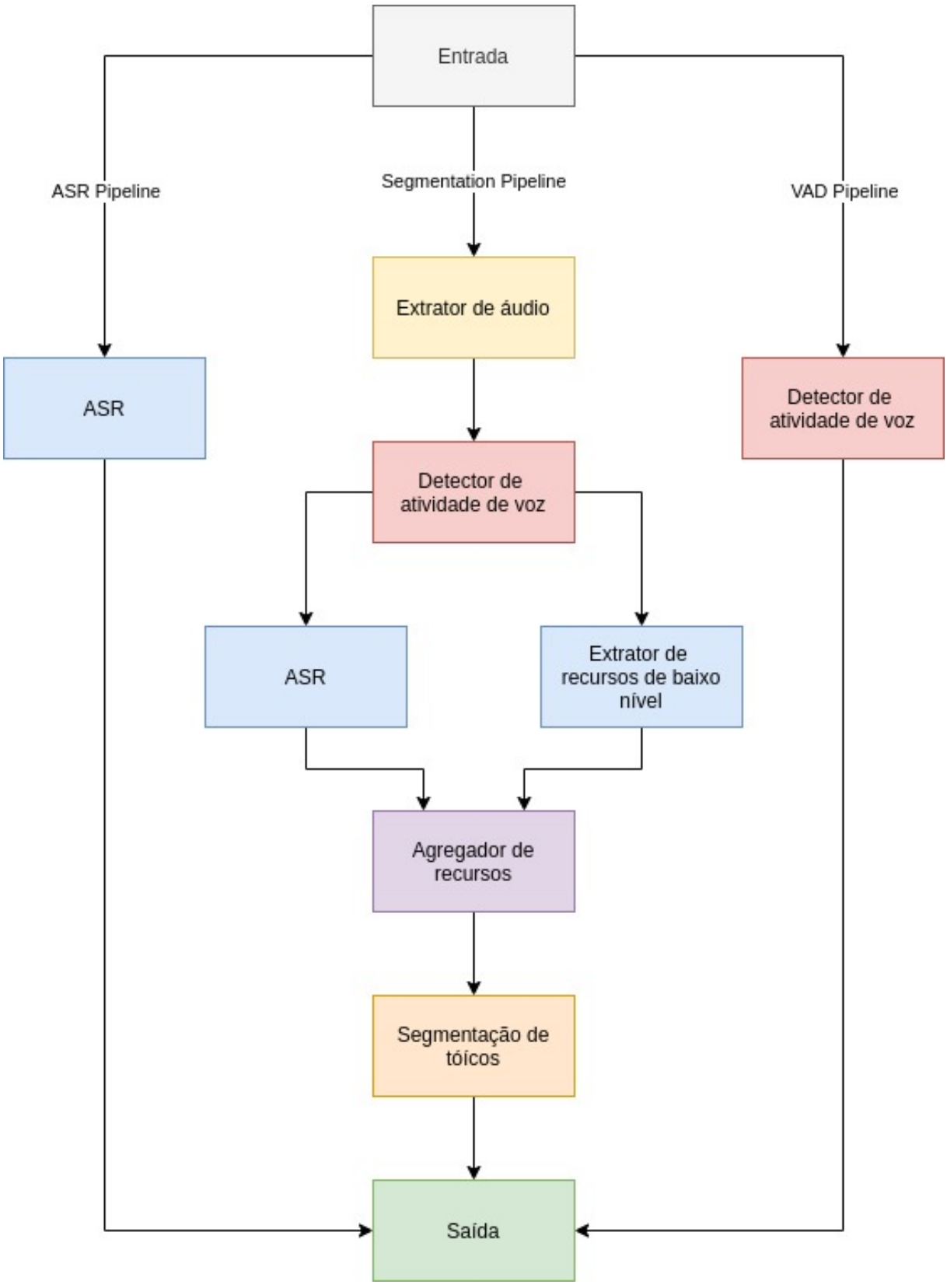

Figura 3.2: Pipeline Original de execução do EasyTopic

módulo de detecção de atividade de voz, o qual executa seu processamento e armazena os resultados em banco de dados, para consulta externa.

3.1.3 Problemas e pontos de melhorias

Entretanto, a framework EasyTopic possui alguns pontos de melhoria. Um deles é, para adicionar um novo módulo ao projeto, é necessário alteração de múltiplos módulos. Isto ocorre porque atualmente, os módulos não estão totalmente desacoplados, pois possuem uma dependência entre si, tendo como requisito ter conhecimento de qual o próximo passo do pipeline e seu formato de entrada e saída.

Isto também impossibilita o reuso e uso singular dos módulos, visto que eles estão sempre em um pipeline fixo. Atualmente, os pipelines que fazem uso de módulos singulares contam com o recurso de que o módulo armazena em banco os seus resultados, para extrair a saída. Porém, o módulo dá continuidade ao processo do pipeline principal, gerando dados e processamento extra desnecessários.

Quanto a obtenção dos resultados do processamento, hoje se dá realizando buscas no banco de dados. Para descobrir se um processamento foi concluído, é necessário realizar consultas periódicas ao banco, verificando o campo de identificador do job e o campo de status concluído. Isto além de gerar um processamento e consumo de recursos desnecessários, dificulta na obtenção em tempo real dos resultados.

Outro ponto é a forma de troca de dados dos módulos, que necessitam gerenciar bancos de dados e arquivos, dos dados de outros serviços, o que não segue o princípio de responsabilidade única em microsserviços. Além disso, os dados trocados entre os módulos não são padronizados, diminuindo o reuso dos módulos.

Por fim, no formato atual, a criação de um novo pipeline é uma tarefa complexa, visto que, além de os módulos necessitarem ter conhecimento do módulo que o sucede no pipeline, o que os impossibilita de serem reaproveitados, o processo de criação de um novo pipeline atualmente seria manual, necessitando alterar o módulo que inicializa um pipeline, bem como todos os módulos que participam do pipeline. Não está disponível atualmente uma forma simples de configurar um novo pipeline, como por exemplo, um arquivo de definição de pipeline.

3.2 Metodologia

Para solucionar os pontos apresentados, era necessário a implementação de melhorias nos módulos da framework, para enfim usa-los de forma mais eficiente. Também era necessário a criação de novos recursos e módulos para serem usados dentro dos pipelines, além de uma solução para a criação simplificada de um novo pipeline. Para isto, foi criada uma nova proposta de arquitetura.

3.2.1 Proposta de nova arquitetura

A arquitetura atual possui problemas que impossibilitaria algumas evoluções e dificultam a modificação. Logo, foi levantado uma nova proposta de arquitetura. Nesta novas proposta, alguns módulos são substituídos, e a comunicação entre os módulos foi modificada.

Na organização de comunicação e estrutura, representada na figura 3.3, foram realizadas mudanças para atender as novas demandas. O módulo de API REST, o qual apenas inicializava o processamento recebendo os parâmetros, foi substituído por um novo módulo chamado Dashboard. Este módulo seria responsável por gerenciar todos os pipelines, além de gerenciar a persistência de dados dos processos, como resultados de processamentos.

Além disso, foi proposto um microsserviço para gerenciamento de arquivos, o qual iria tratar todas as iterações que envolviam arquivos e binários, tanto de entrada quanto de saída. Desta forma, os módulos não precisariam ter conexão com um banco de arquivos, tornando-os mais simples e fáceis de serem integrados. Este módulo também é acessível pelo ator, para upload dos parâmetros de processamento e download dos arquivos gerados.

Os módulos de processamento de dados seriam alterados, de forma que não dependeriam de conexões com bancos de arquivos e de dados persistentes. Estes módulos funcionariam de forma stateless, recebendo quaisquer informação necessária para processamento pela fila de entrada, ao invés de buscar em um banco de dados. A fila de entrada deve conter links para download dos arquivos necessários para o processamento, simplificando a obtenção dos mesmos. Quaisquer resultado de saída devem ser enviados na fila de saída, os quais anteriormente eram armazenados em banco. Arquivos gerados pela saída

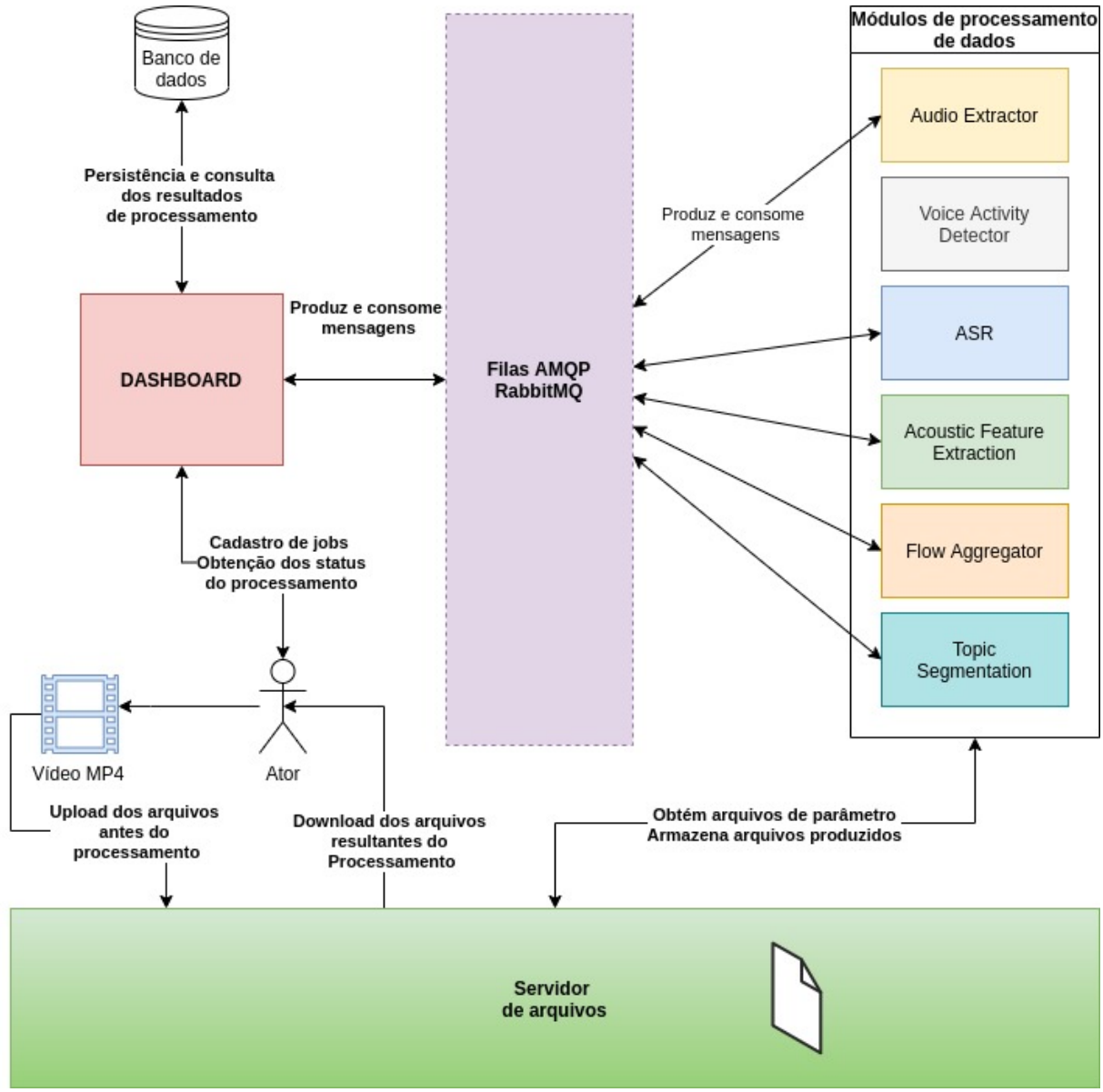

Figura 3.3: Nova arquitetura do projeto EasyTopic

devem ser armazenados em um servidor de arquivos, enviados via um link de download na fila.

para o início do processamento, o ator deve enviar para o dashboard os parâmetros para processamento, os quais são tratados e persistidos pelo dashboard. Para arquivos, o ator deve enviar links para download do mesmo, os quais podem ser enviados para o servidor de arquivos. O ator também não tem mais acesso ao banco de dados para obtenção dos resultados, ele deve obtê-los diretamente do dashboard.

Com a arquitetura proposta, seria possível manter os pipelines originais, representados na figura 3.2. Para sua implementação, foram realizadas correções iniciais nos módulos atuais, a fim de preparar para a integração com a nova arquitetura.

3.2.2 Responsabilidade única e servidor de arquivos

Primeiramente, foi abordado o problema da responsabilidade única dos módulos, pois com este ponto corrigido, seria mais fácil reaproveitá-los em outras áreas. Como citado anteriormente, os módulos também possuem a responsabilidade de gerenciar a conexão e entradas do banco de dados, tornando-se mais complexos.

Cada módulo tinha, além da conexão com as filas AMQP e seus algoritmos de processamento, uma conexão com um banco de dados para upload e download de arquivos, além de uma cópia de uma classe de conexão com este banco.

Para solucionar este problema, foi proposto um novo módulo, com a responsabilidade única de gerenciar arquivos. Este módulo tem sua comunicação via API REST, disponibilizando rotas para cadastro, leitura e deleção de arquivos em massa. O módulo também geraria metadados arquivo, como identificador, data de criação, formato do arquivo e URL de download, de forma que estes dados seriam enviados entre os módulos via mensagens.

Com estas informações, todos os módulos teriam acesso simplificado aos arquivos necessários para sua execução. Para facilitar a integração deste novo módulo aos serviços, foi desenvolvida uma biblioteca simples de conexão, a qual resume os comandos de cadastro e download a simples métodos, e que poderia ser instalada e reusada em quaisquer módulo. Com isto, foi possível substituir o gerenciamento de banco de dados por algo mais simples.

3.2.3 Gerenciador de pipelines

Com os módulos agora simplificados, o outro problema a ser resolvido era a complexidade da criação de novos pipelines, além do desacoplamento dos módulos para sua reutilização.

O problema era causado porque, na implementação atual, ocorria um processo descentralizado, onde iniciado um pipeline, os módulos conheciam seu passo anterior e o próximo passo, comunicando diretamente entre si, tendo apenas o ponto inicial em comum e armazenando seus resultados em banco, para consulta posterior.

Desta forma, para a criação de um novo pipeline, era necessário modificar todos os módulos que este pipeline iria necessitar, para terem conhecimento dos passos

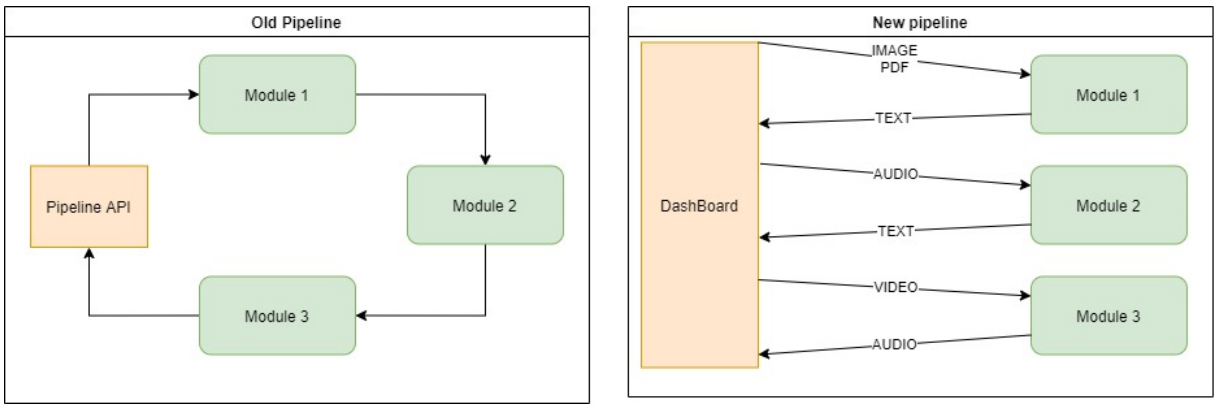

Figura 3.4: Comparativo do antigo e novo pipeline

necessários. Isto torna o processo mais complexo e menos escalável, além de diminuir o reaproveitamento dos módulos do sistema.

Foi proposta então uma modificação na arquitetura, a qual iria ter um novo ponto de centralização, um módulo chamado de dashboard. Este módulo tem dois objetivos principais: Gerenciar os passos do pipeline, desacoplando os módulos, e simplificar a criação de novos pipelines.

A figura 3.4 representa o funcionamento do novo fluxo de pipeline, em comparação com o antigo, onde cada linha representa uma fila AMQP. No novo fluxo, o dashboard tem a responsabilidade de enviar os argumentos do processamento para os módulos e receber os resultados, os encaminhando para o próximo item do pipeline.

O dashboard se comunica via filas AMQP com os módulos do sistema, mas também fornece uma API REST para comunicação externa, para inicialização e obtenção de pipelines. Este dashboard também gerencia processos inicializados, registrando-os em banco de dados, para garantir persistência e escalabilidade.

Para o gerenciamento dos passos do pipeline, o dashboard teria conhecimento de todos os pipelines e de seus passos, inicializando o processo, enviando mensagens de entrada para os módulos, e ouvindo suas saídas, para então usar como parâmetro para o próximo passo.

Enquanto isso, os módulos apenas teriam como requisito ouvir uma fila de entrada, e enviar o processamento para uma fila de saída, se tornando desacoplado dos outros módulos. O único requisito a ser seguido pelos módulos, era um padrão nas mensagens AMQP, onde todas deviam ser no formato JSON, e também devem transitar um

ID, o qual seria o identificador do processo iniciado.

Para isto funcionar de forma reaproveitável, foi necessário padronizar os dados de entrada e saída dos módulos, de forma a cada módulo ter uma lista de tipos de dados que aceita como entrada, e uma lista de tipos de dados que tem como saída. Isto possibilita que o dashboard saiba quais saídas dos módulos podem ser usados como parâmetros para outros.

Neste ponto, era possível criar pipelines individuais, com todos os módulos do sistema, além da geração de pipelines mais complexos. Também era possível o escalonamento de todo o sistema, criando novas instâncias dos módulos, já que os mesmos não necessitam mais do conhecimento de outros passos do pipeline.

A obtenção dos resultados do processamento não mais dependia da consulta periódica de um banco de dados, mas sim do consumo de uma fila pelo dashboard, o qual armazenava os resultados em banco. Isto também possibilita o dashboard notificar atores interessados no momento da finalização do processo.

Outro problema resolvido foi o reuso dos módulos, pois o mesmo não enviava o seu resultado em uma fila de saída que era a de entrada de outro módulo, mas sim para uma fila de saída padronizada, a qual era consumida pelo pipeline. A responsabilidade de encaminhar a saída para outro módulo era do pipeline, removendo esta responsabilidade do módulo em si.

Também foi simplificado o processo para adicionar um novo módulo, pois não seriam necessárias alterações em outros módulos do sistema, visto que não precisavam ter conhecimento do ambiente de execução.

3.2.4 Arquivo de configuração de pipelines

Para a simplificação de novos pipelines, além da padronização dos dados de entrada e saída, foi definido um arquivo de configuração de pipelines, o qual tem como objetivo, descrever um pipeline e seu funcionamento.

Para a configuração, foi definido o formato JSON, por ser altamente usado para troca de dados e pela sua flexibilidade e facilidade de leitura. Este arquivo possui três campos principais: Entradas, saídas e passos do pipeline.

Na entrada, são definidos quais os parâmetros necessários para o pipeline, descrevendo seus tipos e formatos. Os tipos variam desde arquivos, os quais se originam do módulo de arquivos citado anteriormente, até marcações de tempo ou texto simples. Isto permite a flexibilização das entradas, possibilitando de forma mais simplificada a integração de novos módulos a um pipeline. Na saída, define quais serão os arquivos gerados ao fim do pipeline, com sua descrição de tipo, similar a entrada. Estes dados podem ser usados por quem deseja consumir um pipeline, para gerar um formulário de cadastro por exemplo.

Já na configuração dos passos do pipeline, são definidos quais as sequências de passos necessários para o funcionamento do pipeline. Cada passo possui um identificador único, e suas filas de entrada e saída, as quais podem ser estáticas ou obtidas por variáveis de ambiente.

Também é feito um mapeamento de parâmetros do passo, onde é mapeado os parâmetros que serão enviados para o módulo, a partir dos parâmetros recebidos pelo pipeline ou pela saída de outro módulo. Por fim, é feito um mapeamento das saídas dos módulos do pipeline para a saída do pipeline em si.

Um exemplo de arquivo de configuração válido é apresentado a seguir, para a criação de um pipeline de OCR, usando como parâmetro um arquivo de imagem e retornando um texto simples com o OCR produzido.

```
{
    "version": "1.0",
    "id": "ocr",
    "name": "OCR",
    "description": "Runs OCR on an image files",
    "input": [
        {
            "id": "image",
            "name": "Image",
            "description": "The image from the OCR will be
               generated",
            "type": "file",
            "required": true,
            "accept": [
                "image/*"
            ]
        }
    ],
    "output": [
        {
            "id": "ocr",
            "type": "text",
            "name": "OCR Result",
            "description": "The result of the OCR processing"
        }
    ],
    "pipeline": {
        "steps": [
            {
                "id": "ocr-service",
                "queues": [
                    {
                        "env": "OCR_INPUT_QUEUE",
                        "default": "ocr-in"
                    },
                    {
                        "env": "OCR_OUTPUT_QUEUE",
                        "default": "ocr-out"
                    }
                ],
                "arguments": {
                    "file": "image"
                },
                "output": [
                    "ocr"
                ]
            }
        ]
    }
}
```

Exemplo de arquivo de configuração de pipeline

Cada arquivo de configuração também possui alguns dados extras opcionais, como nomes e descrições dos pipelines, argumentos e valores, além de um identificador de versão do arquivo de configuração. Esta versão atualmente não possui função, porém seu objetivo é dar suporte a melhorias no arquivo de configuração, sendo retro compatível com arquivos antigos. Novas funcionalidades adicionadas ao arquivo de configuração gerariam uma nova versão, de forma que um arquivo em uma versão mais antiga seria traduzido para uma nova em tempo de execução.

O dashboard lê os arquivos de configuração, e inicializa os mesmos. Para cada arquivo de configuração, é inicializado uma rota HTTP, para o registro de um novo job, recebendo e validando os parâmetros definidos na configuração. Esta rota, após a validação, inicializa o processo de pipeline, salvando os dados e parâmetros em banco de dados, e então enviando mensagens AMQP na fila de entrada dos módulos descritos na configuração.

Além da rota de registro, também é executado, para cada fila de saída, o consumo da mesma, aguardando respostas de um processamento finalizado. Para cada resposta recebida, a mesma é atualizada no identificador do processo em banco, e então, o próximo passo do pipeline é inicializado, enviando na fila de entrada do mesmo, os seus argumentos necessários.

Por fim, este dashboard disponibiliza rotas de acesso aos pipelines registrados, possibilitando acompanhar a progressão do pipeline até sua finalização. Todos os dados gerados pelo pipeline, de entrada e saída, bem como as saídas dos módulos, são armazenadas em banco, para futura consulta.

Com este recurso, era possível criar pipelines facilmente, demandando apenas da criação de um arquivo de configuração, descrevendo os parâmetros e resultados que serão produzidos, bem como os passos necessários para a execução e quais as filas de entrada e saída de cada passo.

3.3 Implementação

O projeto tem como base a arquitetura de microsserviços, sendo os mesmos inicializados usando a ferramenta Docker, e gerenciados pela ferramenta Docker Compose. Cada

microsserviço funciona como um projeto independente, tendo padronizado apenas a comunicação entre si, a qual se da por filas AMQP, gerenciadas pela plataforma RabbitMQ. Todos os módulos possuem variáveis de ambiente para configuração, como parâmetros de processamento, portas para execução e nome de filas de entrada e saída.

Por conta disso, limitações de linguagem e tecnologias não costumam ocorrer. Logo, a implementação dos novos módulos foi realizada em NodeJS, na linguagem JavaScript, tanto por proficiência na linguagem, quanto pela facilidade de implementação, além de desempenho. Entretanto, os módulos iniciais do EasyTopic foram implementados originalmente em Python, com gerenciamento de dependências feito na ferramenta PIP, logo, modificações nos mesmos mantiveram a linguagem.

3.3.1 Execução da aplicação

Tanto o servidor RabbitMQ quanto os bancos de dados, eram executados em contêineres Docker dedicados. A ferramenta Docker Compose, que foi usada para orquestrar os contêineres, era usada através de um arquivo de configuração, o qual continha uma lista dos serviços que deveriam ser gerados e sua configuração.

Cada módulo do sistema é uma entrada na lista de serviços, possuindo um nome, o qual era usado para ser acessado por outros módulos, um arquivo com o script para geração da máquina virtual, uma lista de portas que deveriam ser expostas para acesso externo e uma lista de parâmetros de configuração, passados por variáveis de ambiente.

Todos os módulos de processamento de dados recebem o endereço e configurações de acesso do servidor AMQP, bem como o endereço e configurações de acesso dos bancos de dados. Os módulos também podem ter parâmetros extras para sua execução se necessário.

A ferramenta Docker Compose, ao executada, inicializa todos os módulos do seu arquivo de configuração, bem como a rede entre eles. Entretanto, tanto o uso das ferramentas Docker e Docker Compose é opcional, pois seu objetivo é criar uma rede de servidores virtuais locais. Cada módulo do sistema pode ser executado sem problemas em um servidor dedicado próprio, sendo necessário na execução, informar as configurações do módulo, para conexão e comunicação com os outros módulos do sistema, via variáveis de ambiente.

3.3.2 Módulo de gerenciamento de arquivos

O módulo de arquivos, gera uma API REST HTTP, usando a biblioteca Express para NodeJS. Este módulo faz o gerenciamento de arquivos de forma local, onde arquivos cadastrados são salvos em uma pasta temporária do sistema, porém possibilita a expansão para o uso de um servidor dedicado de arquivos, bem como um serviço de arquivos em nuvem, devido a sua natureza de microsserviço. O Objetivo deste módulo é gerenciar arquivos de multimídia gerados por outros módulos, ou mesmo arquivos passados como parâmetros para o processamento.

O módulo disponibiliza uma rota HTTP do método POST, para o cadastro de novos arquivos, recebidos no corpo da requisição no formato de "multipart/form-data". Cada campo do corpo da requisição pode conter um arquivo, possibilitando múltiplos uploads em uma só requisição. O módulo responde a requisição no formato JSON, enviando um objeto com múltiplas chaves, onde cada chave representa um campo recebido no corpo da requisição, e o seu valor contém as descrições do arquivo cadastrado, como ID, formato, data de registro e URL para download.

Também é disponibilizada uma rota HTTP do método GET, para a obtenção de um arquivo cadastrado, onde na rota é passado o identificador do arquivo. O módulo inicializa nesta rota um servidor de arquivos estáticos, retornando o arquivo correspondente aquele identificador.

Na conclusão do processamento pelos microsserviços, os mesmos devem enviar uma requisição HTTP do método DELETE, passando o identificador do arquivo, para realizar a deleção do mesmo, se necessário. Atualmente, os arquivos são mantidos de forma permanente, para fins de testes, mas os mesmos podem ser configurados para auto deleção periódica.

Foi implementado para este módulo, uma biblioteca em Python para consumo, a qual é usada nos módulos iniciais. Esta biblioteca é instalada via PIP, possibilitando a configuração e atualização de forma independente.

3.3.3 Módulo de dashboard

O módulo de dashboard, feito em NodeJS, disponibiliza uma API REST HTTP usando a biblioteca Express. Atualmente, os dados de banco são salvos em memória, para agilizar nos testes e modificações, porém o mesmo será salvo em banco de dados em uma versão futura, no formato SQL ou MongoDB. As filas AMQP são consumidas usando a biblioteca amqplib.

O módulo analisa os arquivos de configuração de pipelines, no formato JSON, inicializando todas as filas de entrada e saída. Para cada arquivo, são inicializadas Rotas HTTP do método POST, para criação de um novo job do pipeline atual. Ao acessada, a rota obtém os parâmetros recebidos no corpo da requisição, em formato JSON, e executa uma validação dos tipos dos dados, retornando erro caso necessário.

Também é disponibilizado uma rota para a obtenção dos pipelines cadastrados, onde é retornado parte do arquivo de configuração do pipeline, a fim de descrever quais são as entradas necessárias e as saídas geradas pelo pipeline. Isto é usado para gerar dinamicamente um formulário de cadastro de um novo job para o pipeline.

Além disso, são disponibilizadas rotas para a obtenção de jobs executados anteriormente, bem como a obtenção de um job específico. Estas rotas retornam todas as informações salvas sobre o processamento, como parâmetros recebidos, hora de criação e conclusão, status de conclusão e saídas geradas. Arquivos de entrada e saída são salvos no servidor de arquivos, sendo apenas referenciado o endereço de acesso dos mesmos.

3.3.4 Módulo de OCR

Foi desenvolvido um novo módulo de OCR em NodeJS, o qual gera o reconhecimento ótico de caracteres de uma imagem fornecida. Este consome uma fila AMQP usando a biblioteca amqplib, e executa o processamento de OCR usando a ferramenta Tesseract, inicializada através da biblioteca node-tesseract-ocr.

Na fila de entrada, é recebido o link HTTP para download de um arquivo (preferencialmente do servidor de arquivos), o qual é salvo localmente, para então executar um processamento de OCR simples no mesmo. É gerado como resultado, o reconhecimento do texto contido na imagem, e então é enviado para a fila de saída. Este módulo gera um

contêiner Docker, o qual permite a execução de forma independente, além de configuração via variáveis de ambiente, as quais são configuradas preferencialmente em um arquivo de Docker Compose, as quais configuram parâmetros para execução da ferramenta Tesseract e o nome das filas de entrada e saída. O módulo está disponível em um repositório no GitHub, e pode ser instalado via Docker Compose pela URL do repositório, o qual automaticamente baixa e compila uma imagem Docker

Por fim, foi desenvolvida uma interface web para o uso do dashboard, em outro projeto de iniciação científica. Esta foi desenvolvida usando a biblioteca ReactJS, e consumindo a API do dashboard pela biblioteca Axios. Após consumir a lista de pipelines disponíveis, são gerados dinamicamente formulários para o cadastro de um novo job, o qual ao cadastrado, gera uma entrada no banco, o qual é consultado periodicamente para se obter o status da conclusão. Em uma versão futura, esta atualização do status do job será obtida em tempo real, via socket.

3.4 Validação

Para validar o funcionamento da solução desenvolvida, foi usado o módulo de OCR, o qual já foi desenvolvido de forma desacoplada e compatível com a nova proposta. Foram criados múltiplos pipelines com o mesmo, e a criação de cada pipeline se resumiu a gerar um arquivo no formato JSON, descrevendo o funcionamento do pipeline, de forma rápida e prática. O dashboard conseguiu lidar com o reaproveitamento de um mesmo módulo em múltiplos pipelines diferentes sem problema, possibilitando o escalonamento do mesmo.

Esta solução, auxiliaria um pesquisador a testar uma solução que foi implementada, necessitando apenas a criação de um arquivo de configuração mapeando as configurações do seu módulo, e de pequenas adaptações no módulo criado, como uso de filas AMQP de entrada e saída, com mensagens no formato JSON.

Foram também realizados testes de integração dos módulos existentes no EasyTopic. Um problema enfrentado, foi que o projeto EasyTopic foi implementado no formato monorepo. Isto significa que todos os módulos do sistema, mesmo sendo projetos separados, estão em um mesmo repositório, dificultando a sua instalação em outros projetos.

Também foram necessárias modificações para a remoção das conexões de banco

de dados existentes, pois as mesmas não possuem mais função. Logo, uma refatoração no projeto EasyTopic é necessária para tirar melhor proveito destes novos recursos, sendo uma delas a separação de cada módulo em seu próprio projeto.

Entretanto, em testes locais, foi possível a adição de novos pipelines utilizando os módulos do EasyTopic, com poucas alterações, o que significa que é possível integrar novos módulos de forma simplificada, garantindo seu funcionamento.

3.5 Adição de um novo módulo ou pipeline

Após implementados os novos recursos, o processo para a adicionar um novo pipeline ou módulo ao sistema foi altamente simplificado, em comparação ao formato original do EasyTopic. Para exemplificar o processo, será descrito quais os procedimentos para a adição de um novo módulo e pipeline.

O módulo adicionado será um módulo de processamento de imagem, no qual ao recebido uma imagem como parâmetro, uma marca d'água é adicionada a imagem e retornada como resultado. Também será adicionado um novo pipeline, o qual executa este módulo apenas. Partiremos do pressuposto que o módulo já foi implementado, para uso em outros projetos, sendo descrito então quais as alterações necessárias para o funcionamento no EasyTopic.

Primeiramente, o módulo deverá possibilitar a conexão a um servidor AMQP, escutando duas filas, uma de entrada e uma de saída. Preferencialmente, o módulo deve receber os endereços de conexão e nome das filas via variáveis de ambiente ou arquivo de configuração, permitindo maior flexibilização na configuração. O módulo também deve receber como parâmetro, o endereço do servidor de arquivos do projeto.

Todos os argumentos necessários para o processamento devem ser recebidos via mensagem da fila de entrada, no formato JSON, e o arquivo para execução do processamento deve ser um dos argumentos, recebendo o identificador e endereço de download do mesmo. Outro argumento que deve ser recebido é um identificador de processo, o qual deverá ser enviado junto com o resultado na fila de saída.

O módulo deve realizar o processamento necessário, realizando a recuperação da imagem se necessário, produzindo assim o arquivo processado. Em seguida, deve persistir

o arquivo produzido no servidor de imagens, gerando assim, um endereço para acesso do arquivo.

Por fim, o módulo deve enviar na fila de saída, o resultado do seu processamento, no formato JSON, com o endereço de acesso do arquivo produzido, e o identificador de processo, recebido como argumento.

Com isto, o módulo está pronto para ser usado juntamente com o projeto Easy-Topic. Agora, basta adicioná-lo na lista de serviços do arquivo de configuração do Docker Compose, informando os argumentos para seu funcionamento, para ser executado juntamente com os outros módulos.

Para a criação de um pipeline com o novo módulo, é necessário criar um arquivo, no formato JSON, contendo as configurações do pipeline, armazenando-o juntamente dos outros arquivos de configuração. No arquivo, é necessário informar os parâmetros de entrada e saída do pipeline, bem como os passos do pipeline e o mapeamento de parâmetros. Também é recomendado fornecer descrições do pipeline e seus parâmetros, para facilitar o entendimento do mesmo.

Considerando que o módulo descrito tem as seguintes configurações:

- As filas de entrada e saída do módulo tem como identificadores "watermark-in"e "watermark-out"respectivamente
- A imagem a ser processada é recebida no parâmetro "processing-img"
- O resultado do processamento é enviado no parâmetro "processed-img"

O seguinte arquivo de configuração seria valido.

```
{
    "version": "1.0",
    "id": "watermark",
    "input": [
        {
            "id": "processing-img",
            "type": "file",
            "required": true,
            "accept": [
                "image/*"
            ]
        }
    ],
    "output": [
        {
            "id": "processed-img",
            "type": "file",
            "format": "image/*"
        }
    ],
    "pipeline": {
        "steps": [
            {
                "id": "watermark-module",
                "queues": [
                    "watermark-in",
                    "watermark-out"
                ],
                "arguments": [
                    "processing-img"
                ],
                "output": [
                    "processed-img"
                ]
            }
        ]
    }
}
```

Exemplo de arquivo de configuração de pipeline para o processamento do módulo de marca d'água

4 Conclusão e trabalhos futuros

4.1 Solução aplicada

Foi visto que a solução aplicada melhorou consideravelmente o uso e aplicação do projeto. A solução simplifica todos os módulos do sistema, melhorando assim sua manutenção e aumentando o reuso. A solução aplicada também simplificou o processo de adição e teste de um novo módulo ao sistema, além de fornecer de forma simples a obtenção de informações sobre o processamento.

O novo sistema de arquivos além de diminuir a complexidade dos módulos, também permite seu uso por novos módulos no sistema, sendo um recurso genérico e desacoplado, pode até ser reusado em outros projetos.

Entretanto, para concluir a solução, serão necessárias alterações no projeto Easy-Topic. Com a possibilidade de desacoplamento dos módulos, é necessário separá-los em projetos separados, para possibilitar sua instalação e uso de forma independente. Com isso, será possível usar todos os módulos dentro de qualquer pipeline e inclusive recriar o pipeline original, usando os arquivos de configuração definidos.

O novo módulo de OCR validou a integração da nova arquitetura, além de ser um módulo de fácil reuso e configuração.

4.2 Aplicações na área da educação

O projeto em geral, pode ser aplicado em áreas variadas, como pesquisa e educação. Para pesquisadores, é uma ferramenta extremamente útil para testar algoritmos e propostas para segmentação de vídeos, pois são disponibilizados módulos comumente usados na área de multimídia e segmentação, como reconhecimento automático de fala, extração de faixa de áudio, detecção de atividade de voz e reconhecimento ótico de caracteres. Além disso, a configuração de um novo pipeline de processamento é relativamente simples, possibilitando a integração de um módulo sem grandes dificuldades

Para a área da educação, considerando o pipeline de segmentação de vídeos educacionais, pode ser integrado a uma plataforma de ensino a distância, para segmentação automática de videoaulas e palestras, possibilitando a indexação e busca por tópicos.

4.3 Aprimoramentos

A solução aplicada pode ser evoluída, tanto para aplicação em outras áreas de processamento de dados, quanto para adicionar mais recursos ao gerenciamento. Melhorias que podem ser feitas atualmente são a adição de um banco de dados para persistência dos jobs, os quais hoje são salvos em memória, e de um socket para obtenção em tempo real do status de um job em execução.

Uma melhoria futura que pode ser feita no projeto, é a adição a suporte a agregadores de dados, em uma configuração de pipeline. Este tem como objetivo, possibilitar o processamento em paralelo de dados, por dois ou mais módulos diferentes. Após concluírem seus processamentos, os resultados seriam agregados e enviados para o próximo passo, o qual recebe por parâmetro ambos estes resultados.

O módulo de arquivos pode ter duas melhorias vistas atualmente. Uma delas, é a integração com serviços de armazenamento em nuvem. Hoje isto é possível configurando o ambiente de execução, porém também é possível adicionar a integração de forma interna, recebendo via parâmetros chaves de API para conexão com tais serviços de armazenamento. Outra melhoria disponível, é o agrupamento de arquivos. Hoje, todos os arquivos são salvos sem classificação ou agrupamentos em pastas, considerando que o interessado no arquivo possui seu ID ou link. Porém, uma melhoria seria o armazenamento em subpastas, para agrupar arquivos de acordo com o identificador de um job. Assim, Seria possível realizar o download de todos os arquivos gerados por um job específico, bem como a deleção de todos após o job ser concluído.

Outra melhoria futura no projeto, seria uma ferramenta para geração de novos pipelines. O Dashboard teria uma lista pré-definida de Módulos, com suas entradas e saídas, possibilitando o cadastro de um novo pipeline em banco de dados, ao invés de um arquivo estático. Para isso, seria adicionado a interface web, uma ferramenta visual para a criação de pipelines, onde seriam definidos os passos e alocado os seus parâmetros e

saídas.

Atualmente, o projeto tem como objetivo ser executado em ambientes locais, controlados, para testes. Porém, o mesmo pode ser expandido para possibilitar a execução online. Para isto, seriam necessários adicionar algumas medidas de segurança ao projeto, como a adição de uma camada de segurança as filas AMQP, além de um sistema de gerenciamento de permissões por usuário no módulo de dashboard, podendo este inclusive ser um módulo independente.

O projeto possui suporte atualmente, apenas para comunicação via filas AMQP. No geral, este formato simplifica a comunicação, pois faz tratamento de persistência e escalonamento. Entretanto, dependendo da demanda, como a nova proposta de arquitetura oferece um ponto de centralização, novos formatos de comunicação podem ser adicionados. É possível adicionar suporte para comunicação com servidores em formato de API REST por exemplo, definindo no arquivo de configuração de pipeline, os parâmetros para a comunicação. Isto pode facilitar a integração de módulos que não estão no formato de filas AMQP, ou que rodam externamente como serviços.

Por fim, com foco na segmentação de vídeos educacionais, uma grane melhoria seria a adição de módulos auxiliarias, os quais são módulos que permitem a execução de tarefas comuns na área de multimídia, como extração de metadados ou conversões de arquivos, e que podem ser usados para auxiliar na criação de um novo módulo, sendo reaproveitáveis. Alguns exemplos de módulos possíveis são os que executam tarefas como extração de áudio, transcrição de áudio, extração de texto de documentos, etc. Estes funcionariam como ferramentas, fazendo parte dos pipelines e sendo usado como parâmetros para outros módulos.

Bibliografia

ARONS, B. Pitch-based emphasis detection for segmenting speech recordings. In: *Third International Conference on Spoken Language Processing*. [S.l.: s.n.], 1994.

BARRÉRE, E.; SOUZA, J.; SOARES, E. R. Framework para segmentação temporal de vídeos educacionais. In: SBC. *Anais do XXXI Simpósio Brasileiro de Informática na Educação*. [S.l.], 2020. p. 972–981.

CHE, X. et al. Sentence-level automatic lecture highlighting based on acoustic analysis. In: IEEE. *2016 IEEE International Conference on Computer and Information Technology (CIT)*. [S.l.], 2016. p. 328–334.

LEE, G. C. et al. Robust handwriting extraction and lecture video summarization. *Multimedia Tools and Applications*, Springer, v. 76, n. 5, p. 7067–7085, 2017.

LIN, M. et al. Automated video segmentation for lecture videos: A linguistics-based approach. *International Journal of Technology and Human Interaction (IJTHI)*, IGI Global, v. 1, n. 2, p. 27–45, 2005.

SHAH, R. R. et al. Atlas: automatic temporal segmentation and annotation of lecture videos based on modelling transition time. In: ACM. *Proceedings of the 22nd ACM international conference on Multimedia*. [S.l.], 2014. p. 209–212.

TOGASHI, S.; YAMAGUCHI, M.; NAKAGAWA, S. Summarization of spoken lectures based on linguistic surface and prosodic information. In: IEEE. *2006 IEEE Spoken Language Technology Workshop*. [S.l.], 2006. p. 34–37.

YAMAMOTO, N.; OGATA, J.; ARIKI, Y. Topic segmentation and retrieval system for lecture videos based on spontaneous speech recognition. In: *Eighth European Conference on Speech Communication and Technology*. [S.l.: s.n.], 2003.