

Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia de Software

**Um estudo sobre requisitos técnicos para uma
biblioteca digital de música brasileira**

Autores: João Pedro Moura Oliveira e Thiago Sampaio de Paiva
Orientador: Prof. Dr. Fernando William Cruz

Brasília, DF
2023



João Pedro Moura Oliveira e Thiago Sampaio de Paiva

Um estudo sobre requisitos técnicos para uma biblioteca digital de música brasileira

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Prof. Dr. Fernando William Cruz

Brasília, DF

2023

João Pedro Moura Oliveira e Thiago Sampaio de Paiva

Um estudo sobre requisitos técnicos para uma biblioteca digital de música brasileira/ João Pedro Moura Oliveira e Thiago Sampaio de Paiva. – Brasília, DF, 2023-

119 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Fernando William Cruz

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2023.

1. Biblioteca digital. 2. Música brasileira. I. Prof. Dr. Fernando William Cruz.
II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Um estudo sobre
requisitos técnicos para uma biblioteca digital de música brasileira

CDU 00:000:000.0

João Pedro Moura Oliveira e Thiago Sampaio de Paiva

Um estudo sobre requisitos técnicos para uma biblioteca digital de música brasileira

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Prof. Dr. Fernando William Cruz
Orientador

Prof. Dr. Bruno César Ribas
Convidado 1

Prof. Dr. Daniel Sundfeld Lima
Convidado 2

Brasília, DF
2023

Resumo

Bibliotecas digitais são um recurso interessante para efeito de preservação da cultura de sociedades. Por outro lado, a implementação de um *software* dessa natureza pressupõe uma série de requisitos que merecem investigação. Este trabalho relata um estudo sobre requisitos tecnológicos para criação de uma biblioteca digital de música brasileira. No geral, bibliotecas digitais de música devem se comportar como uma aplicação de larga escala (*large-scale application*). Com base nessa premissa, foram feitos estudos sobre a criação de clusters, considerando máquinas reais e virtuais, a fim de gerar comparações e permitir dar uma visão mais efetiva sobre as diversas configurações possíveis. Ou seja, esses ambientes devem funcionar em ambiente com alto desempenho, permitindo a oferta performática e escalável – por explorar os recursos de memória, núcleos de processamento e disco espalhados entre diferentes máquinas –, e elasticidade – por garantir que a aplicação se adapte às demandas de uso. Em seguida, foram elencadas duas características essenciais de uma aplicação de música para poderem ser testadas em ambiente de cluster. A primeira delas refere-se à capacidade de oferta de *streaming* de áudio com qualidade, considerando uma grande audiência. A segunda é relativa à capacidade da aplicação permitir recomendação musical com base em parâmetros subjetivos, como, por exemplo, similaridade musical ou nível de afinidade. Para o *streaming* de áudio, foram feitos estudos e testes em servidores de mídia e protocolos de rede. Para a recomendação, foram feitos testes com filtragem por conteúdo e colaborativa em um corpus musical. Ademais, foi modelado um banco de dados e construído um protótipo de biblioteca digital conforme uma ontologia de músicas brasileiras. O resultado mais evidente desta pesquisa é o nível de conhecimento adquirido, que permite o credenciamento para avançar nas pesquisas em direção à construção da biblioteca digital de música brasileira.

Palavras-chave: Biblioteca digital. Música brasileira. Sistema de Recomendação. *Streaming* de áudio. Infraestrutura escalável. Ontologias.

Abstract

Digital libraries are an interesting resource for preserving the culture of societies. On the other hand, the implementation of such software presupposes a series of requirements that deserve investigation. This paper reports a study on technological requirements for the creation of a digital library of Brazilian music. In general, digital music libraries should behave as a large-scale application. Based on this premise, studies were carried out on the creation of clusters, considering real and virtual machines, in order to generate comparisons and allow a more effective view of the various possible configurations. In other words, these environments must operate in a high-performance environment, allowing performance and scalable – by exploiting memory resources, processing cores and spaced disk between different machines –, and elasticity – to ensure that the application adapts to usage demands. Then, two essential characteristics of a music application were listed so that they could be tested in a cluster environment. The first of these refers to the ability to offer quality audio streaming, considering a large audience. The second is related to the ability of the application to allow music recommendation based on subjective parameters, such as musical similarity or affinity level. For audio streaming, studies and tests were carried out on media servers and network protocols. For the recommendation, tests were made with content-based and collaborative filtering on a musical corpus. Furthermore, a database was modeled, and a digital library prototype was built according to an ontology of Brazilian music. The most evident result of this research is the level of knowledge acquired, which allows accreditation to advance in research towards the construction of the digital library of Brazilian music.

Key-words: Digital library. Brazilian music. Recommendation System. Audio streaming. Scalable infrastructure

Lista de ilustrações

Figura 1 – Fases da metodologia para a infraestrutura	22
Figura 2 – Fases da metodologia para a biblioteca digital	22
Figura 3 – Cronograma de atividades	24
Figura 4 – [a] Arquitetura de máquinas virtuais tipo 1 e [b] Arquiteturas de máquinas virtuais tipo 2	27
Figura 5 – Arquitetura simplificada do NFS	31
Figura 6 – Arquitetura simplificada do HDFS	37
Figura 7 – Arquitetura simplificada do Mesos	42
Figura 8 – Arquitetura Kubernetes	44
Figura 9 – Representação do <i>cluster</i>	47
Figura 10 – Representação da estrutura de pastas do Apache Hadoop instalado no <i>cluster</i>	48
Figura 11 – Configuração simplificada do Apache <i>Datanode</i> na máquina Zeus	49
Figura 12 – Representação da estrutura de pastas do Apache Spark instalado no <i>cluster</i>	49
Figura 13 – Configuração do Spark <i>Worker</i> na máquina Atenas	50
Figura 14 – Configuração do escalonador do Spark <i>Master</i> na máquina Zeus	50
Figura 15 – Configuração do Idle-culler no Jupyterhub	50
Figura 16 – Tela das métricas coletadas em um nó	51
Figura 17 – Arquitetura para o servidor de <i>streaming</i>	53
Figura 18 – Exemplo do método RTSP SETUP	54
Figura 19 – Exemplo do método RTSP PLAY e PAUSE.	55
Figura 20 – Exemplo de requisição de mídia com HTTP	56
Figura 21 – Comparação da taxa de transferência de pacotes entre CUBIC e BBR	58
Figura 22 – Fluxo do servidor de <i>streaming</i>	60
Figura 23 – Código-fonte do endpoint do servidor de streaming em Python	61
Figura 24 – Ambiente base para a experimentação do servidor de <i>streaming</i>	62
Figura 25 – Resultados do teste de estresse do servidor de <i>streaming</i> em diferentes ambientes	63
Figura 26 – Ambiente onde os usuários têm acesso direto aos servidores de <i>streaming</i>	64
Figura 27 – Resultados do teste de estresse com e sem acesso direto dos usuários aos servidores de <i>streaming</i>	65
Figura 28 – Rotina de extração das informações necessárias	67
Figura 29 – Código-fonte simplificado da filtragem colaborativa	68
Figura 30 – Código-fonte simplificado da recomendação por filtragem colaborativa	68
Figura 31 – Arquitetura da filtragem baseada em conteúdo	69

Figura 32 – Código-fonte simplificado da filtragem baseada no conteúdo	71
Figura 33 – Código-fonte simplificado da recomendação por filtragem baseada no conteúdo	72
Figura 34 – Visão geral das relações presentes no modelo conceitual de música brasileira	77
Figura 35 – Modelo de banco de dados para autoria.	78
Figura 36 – Modelo de banco de dados para versões de ideia musical.	79
Figura 37 – Modelo de banco de dados para instrumentação.	80
Figura 38 – Modelo de banco de dados para performance.	81
Figura 39 – Modelo de banco de dados para gênero.	83
Figura 40 – Arquitetura da biblioteca digital.	86
Figura 41 – Código-fonte da criação da visão para filtragem colaborativa	89
Figura 42 – Apresentação da recomendação por filtragem colaborativa ao usuário	89
Figura 43 – Código-fonte da criação da visão para filtragem baseada em conteúdo	91
Figura 44 – Página inicial	92
Figura 45 – Busca por músicas	93
Figura 46 – Detalhes de uma música	94
Figura 47 – Cadastros	95
Figura 48 – Cadastro de gênero	95
Figura 49 – Detalhes do gênero	108
Figura 50 – Detalhes do álbum	109
Figura 51 – Detalhes do Artista	110
Figura 52 – Detalhes do Grupo Musical	111
Figura 53 – Cadastro de Álbum	112
Figura 54 – Cadastro de Música	113
Figura 55 – Variáveis de ambiente do <i>back-end</i>	114
Figura 56 – Diagrama do modelo conceitual para autoria	116
Figura 57 – Diagrama do modelo conceitual para versões de ideia musical	117
Figura 58 – Diagrama do modelo conceitual para instrumentação	117
Figura 59 – Diagrama do modelo conceitual para performance.	118
Figura 60 – Diagrama do modelo conceitual para genero	119

Lista de tabelas

Tabela 1 – Configuração e resultados obtidos pelo Linpack	47
Tabela 2 – Exemplo da matriz de utilidade para a filtragem colaborativa	66
Tabela 3 – Objetivos específicos do trabalho	96

Lista de abreviaturas e siglas

ALS	Alternating Least Squares
AVP	Audio-Video Protocol
BBR	Bottleneck Bandwidth and Round-trip propagation time
CDN	Content Delivery Network
DASH	Dynamic Adaptive Streaming over HTTP
D-Streams	Discretized Streams
HDFS	Hadoop Distributed File System
HLS	HTTP Live Streaming
IFLA	International Federation of Library Associations and Institutions
ISBD	International Standard Bibliographic Description
LRM	Library Reference Model
MPI	Message Passing Interface
NDCG	Normalized Discounted Cumulative Gain
NIS	Network Information Service
NFS	Network File System
RDF	Resource Description Framework
RPC	Remote Procedure Call
RTP	Real-time Transport Protocol
RTSP	Real-time Streaming Protocol
SNIA	Storage Networking Industry Association
SPARQL	SPARQL Protocol and RDF Query Language
TF	Term Frequency
IDF	Inverse Document Frequency
TF-IDF	Frequency–Inverse Document Frequency
WEMI	Work, Expression, Manifestation, and Item

Sumário

1	INTRODUÇÃO	17
1.1	Justificativa	18
1.2	Questão de pesquisa	18
1.3	Objetivos	19
1.3.1	Objetivo geral	19
1.3.2	Objetivos específicos	19
1.4	Classificação da pesquisa	19
1.5	Organização da monografia	20
2	METODOLOGIA	21
2.1	Metodologia de desenvolvimento	21
2.1.1	Etapas do desenvolvimento	23
2.1.2	Cronograma	23
3	INFRAESTRUTURA BÁSICA	25
3.1	Formação do <i>cluster</i>	25
3.1.1	Implantação em <i>bare-metal</i>	25
3.1.2	Implantação em máquinas virtuais	26
3.1.3	Tecnologias auxiliares para armazenamento	28
3.1.3.1	Armazenamento confiável com RAIDs	28
3.1.3.2	Disponibilizando conteúdos geograficamente com CDNs	29
3.1.3.3	Network File System	30
3.1.4	Serviços de diretório	31
3.1.5	Testes da infraestrutura	32
3.1.5.1	Avaliando o desempenho com o Linpack	33
3.1.6	Monitoramento da infraestrutura	34
3.1.6.1	Coletando e processando informações do sistema	35
3.1.6.2	Análise e apresentação gráfica	36
3.2	Camada de implantação distribuída	36
3.2.1	Apache Hadoop	37
3.2.1.1	Apache Spark	38
3.2.1.2	OpenMP e OpenMPI	40
3.3	Contêineres	41
3.4	Orquestradores	41
3.4.1	Apache Mesos	42
3.4.2	Docker Swarm	43

3.4.3	Kubernetes	44
3.5	Experimentos	45
4	REQUISITOS ESPECÍFICOS DA APLICAÇÃO	52
4.1	Oferta de <i>streaming</i>	52
4.1.1	RTSP	54
4.1.2	HTTP	56
4.1.3	Algoritmos de congestionamento	56
4.1.4	Testes de carga	58
4.1.5	Experimentos	59
4.1.5.1	<i>Benchmark</i>	61
4.2	Serviço de recomendação	65
4.2.1	Filtragem colaborativa	65
4.2.1.1	Experimentos	66
4.2.2	Filtragem baseada no conteúdo	68
4.2.2.1	Experimentos	70
5	BIBLIOTECA DIGITAL	73
5.1	Ontologia	73
5.1.1	Web Semântica	74
5.1.2	Modelos Ontológicos	74
5.1.3	Música Brasileira e Ontologia	75
5.2	Modelos	76
5.2.1	Doremus	82
5.2.2	Variations	84
5.2.3	MusicBrainz	84
5.3	Implementação da biblioteca	85
5.3.1	Arquitetura	85
5.3.2	Coleta de dados	87
5.3.3	Recomendação	88
5.3.4	Uso do sistema	90
6	CONSIDERAÇÕES FINAIS	96
6.1	Resultados alcançados	96
6.2	Contribuições	97
6.3	Trabalhos futuros	98
	REFERÊNCIAS	100

APÊNDICES	107
APÊNDICE A – INTERFACES	108
APÊNDICE B – MANUAL DE INSTALAÇÃO	114
ANEXOS	115
ANEXO A – DIAGRAMAS DO MODELO CONCEITUAL DE MÚ- SICA BRASILEIRA	116

1 Introdução

Aspectos como ontologia, recomendação por meio de inteligência artificial, serviços de mídia, ambientes *clusterizados*, testes e monitoramento de infraestrutura são alguns dos elementos que orientaram a elaboração desta monografia. Em seguida, serão abordados a Justificativa, a Questão de pesquisa, os Objetivos, Geral e Específicos, e, a estrutura organizacional do texto.

Percebe-se que desde a antiguidade até os tempos atuais a música é utilizada não apenas como entretenimento, mas ainda persiste como uma forma de preservação da história e identidade de uma sociedade, refletindo suas tradições locais e a organização da sua cultura (ALVES, 2011). Um exemplo bastante notável são as produções artísticas do período da ditadura brasileira instaurada no ano de 1964, que passaram a contar não apenas a história, mas serviram como uma forma de protesto ao regime instituído no país.

Os registros culturais, portanto, se tornam uma fonte rica e extremamente relevante para o entendimento das diversas populações do globo terrestre, fortalecendo a necessidade de sua preservação para a difusão pelas diversas gerações. Apesar dessa importância, no Brasil, poucos investimentos são feitos para resguardar essas evidências, se tornando esse um dos pontos essenciais para o desenvolvimento desta monografia.

Este trabalho, portanto, insere-se no contexto dos estudos de mapeamento ontológico da música brasileira (PADRON; CRUZ; SILVA, 2020; PADRON et al., 2023). Observa-se que a organização da ontologia traz vantagens significativas para o armazenamento e indexação dos registros culturais, especialmente no âmbito musical, com a incorporação dos novos metadados. Essa nova modelagem dos dados também capacita as aplicações a realizarem consultas mais complexas, permitindo o desenvolvimento de filtros mais robustos.

Além disso, destaca-se a aplicação dos conhecimentos de inteligência artificial na área de recomendação. Conceitos como a filtragem colaborativa, que utiliza a avaliação dos usuários e suas preferências para gerar recomendações (KARAU et al., 2015), e a filtragem baseada no conteúdo (LESKOVEC; RAJARAMAN; ULLMAN, 2020), apoiada pelo uso de ontologias, proporcionaram uma nova perspectiva na visualização e interpretação dos dados.

O processamento de informações mencionado requer um ambiente tecnológico robusto e flexível, capaz de se adaptar à carga do sistema. Nesse sentido, o conhecimento acerca de ferramentas e tecnologias que permitem a execução distribuída e paralela desempenha um papel relevante e motivador para o desenvolvimento desta monografia. Tecnologias como NFS, Apache Hadoop, Apache Spark e Kubernetes foram exploradas,

utilizadas no desenvolvimento e avaliadas neste estudo. A análise dessas tecnologias contribuiu para a compreensão e aprimoramento do ambiente tecnológico necessário para o processamento eficiente das informações.

Considerando o enfoque deste trabalho na seleção de uma infraestrutura adequada para o desenvolvimento da aplicação, torna-se de extrema importância a realização de estudos sobre testes de estresse e testes de carga no *cluster* de máquinas implantado, como destacado por (NAIK, 2008). Além disso, os conceitos discutidos por Tsai et al. (2018) sobre monitoramento desempenham um papel fundamental na manutenção e compreensão do estado de um conjunto de máquinas. A compreensão desses conceitos é crucial para garantir a eficiência e a disponibilidade da infraestrutura utilizada durante o desenvolvimento do projeto.

1.1 Justificativa

Conforme enfatizado por Cruz (2008), à medida que as tecnologias evoluem, torna-se cada vez mais crucial o armazenamento, indexação e acesso de mídias musicais em plataformas digitais. Impulsionado pelo campo do conhecimento ontológico, o desenvolvimento dessas aplicações se torna cada vez mais iminente.

Além disso, o avanço dos modelos de inteligência artificial traz uma nova capacidade para as bibliotecas digitais. A razão por trás disso reside na possibilidade de recomendações mais precisas com base nos usuários e na criação e utilização de filtros mais complexos, permitindo a descoberta e divulgação ampliada dessas mídias musicais.

Todo esse conjunto de conhecimentos culmina na necessidade de uma infraestrutura escalável, tanto horizontalmente quanto verticalmente, que seja tolerante a falhas, eficiente e capaz de suportar um grande volume de processamento em tempo real. Além disso, para reforçar essas características, a realização de testes e a implementação de sistemas de monitoramento são essenciais.

Portanto, este trabalho propõe a construção de uma infraestrutura elástica e com alta disponibilidade, capaz de suportar todas as funcionalidades de uma biblioteca digital de músicas brasileiras, utilizando conhecimentos em ontologia e inteligência artificial.

1.2 Questão de pesquisa

Com a finalização deste trabalho, visa solucionar o seguinte questionamento de pesquisa: Como construir bibliotecas virtuais de músicas escaláveis, tolerantes a falhas e capazes de atender altas demandas, provendo interoperabilidade, inferências e recomendações personalizadas ao usuário?

1.3 Objetivos

Na intenção de responder à Questão de Pesquisa apresentada anteriormente, os objetivos gerais e específicos foram elaborados, sendo apresentados nas seções seguintes.

1.3.1 Objetivo geral

O objetivo geral desta dissertação é investigar alternativas tecnológicas para viabilizar a construção de uma biblioteca digital de música brasileira.

1.3.2 Objetivos específicos

Para atender o objetivo geral proposto em 1.3.1 os seguintes objetivos específicos foram elaborados:

- Explorar o funcionamento de *clusters* como um requisito para aplicações de larga escala;
- Explorar o funcionamento de tecnologias distribuídas como um requisito para a escalabilidade de aplicações de larga escala;
- Explorar o funcionamento de orquestradores de software como um requisito para elasticidade de aplicações de larga escala;
- Explorar alternativas tecnológicas para transmissão de *streaming* de áudio;
- Explorar alternativas para sistemas de recomendação musical;
- Explorar a modelagem de um banco de dados mediante uma ontologia;
- Explorar o desenvolvimento de um sistema web seguindo os requisitos de escalabilidade, elasticidade e larga escala;
- Explorar o desenvolvimento de modelos de recomendação em tempo real;

1.4 Classificação da pesquisa

Segundo Gerhardt e Silveira (2009), nas diversas classificações de pesquisa, quanto ao objetivo, este trabalho pode ser considerado principalmente uma pesquisa de natureza **explicativa**. Tal classificação se justifica pela necessidade desta monografia em identificar e determinar os fatores e ações necessárias para a criação de ambientes escaláveis e tolerantes a falhas, com o propósito de sustentar aplicações que demandam alto poder de processamento.

Ainda conforme Gerhardt e Silveira (2009), além de sua natureza explicativa, essa pesquisa pode ser também classificada como experimental e de levantamento. A abordagem **experimental** se evidencia pelo processo que se inicia com a formulação de problemas e hipóteses, seguido pela definição dos atributos que podem influenciar os resultados a serem coletados e avaliados ao término da pesquisa. Por outro lado, a caracterização de pesquisa **de levantamento** se deve à coleta de dados para fins de avaliação e análise estatística.

1.5 Organização da monografia

Este documento se organiza nos seguintes capítulos:

- Capítulo 2 - Metodologia: Definição das metodologias adotadas para a elaboração dessa monografia, enfatizando os aspectos metodológicos referentes a investigação bibliográfica, desenvolvimento e análise dos resultados;
- Capítulo 3 - Infraestrutura básica: Contextualização e experimentos sobre a construção de ambientes escaláveis e elásticos.
- Capítulo 4 - Requisitos específicos da aplicação: Contextualização e experimentos sobre a oferta de *streaming* de áudio e recomendação.
- Capítulo 5 - Biblioteca digital: Contextualização dos conceitos da aplicação que esse trabalho visa realizar.
- Capítulo 6 - Considerações finais: Apresentação dos resultados obtidos, contribuições e possíveis trabalhos futuros.

2 Metodologia

Este capítulo aborda a definição e implementação das metodologias empregadas ao longo do desenvolvimento do trabalho. Ele está dividido em seções dedicadas às metodologias de desenvolvimento e ao cronograma de execução das tarefas. Na seção de metodologia de desenvolvimento, são detalhadas as estratégias adotadas para a construção da infraestrutura e da biblioteca digital de músicas brasileiras. Já na seção do cronograma, são apresentadas as diferentes fases do desenvolvimento, acompanhadas do tempo dedicado à execução de cada uma delas.

2.1 Metodologia de desenvolvimento

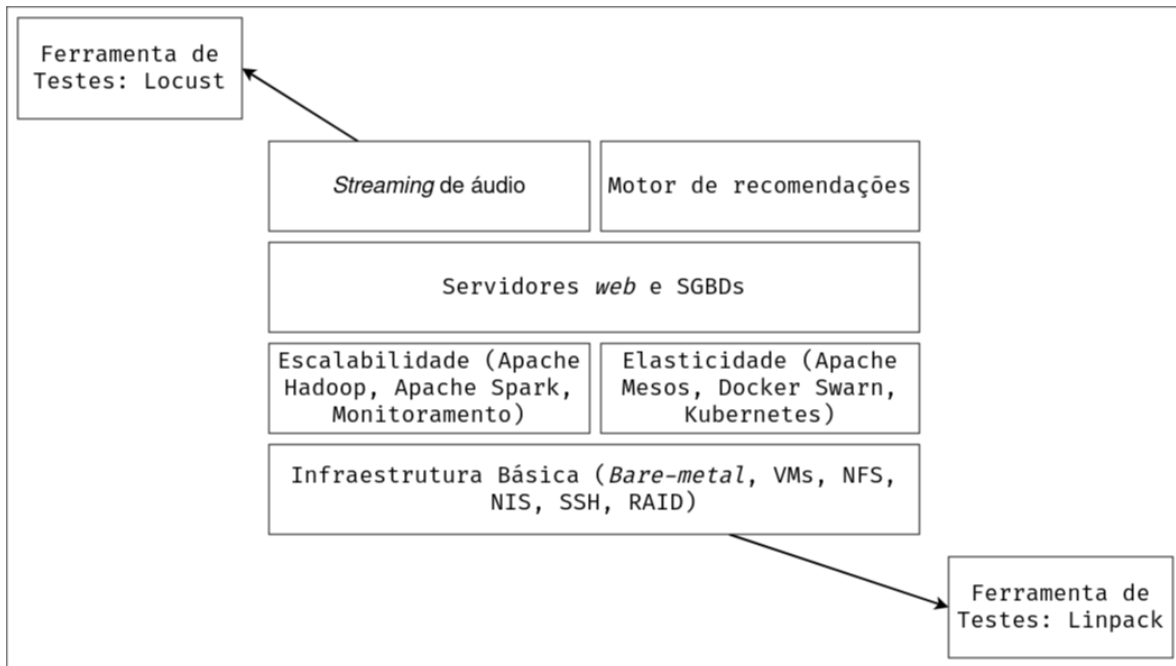
Para o desenvolvimento da infraestrutura, adotou-se uma abordagem em camadas, na qual as tarefas necessárias para alcançar os objetivos específicos foram divididas em quatro categorias distintas: a infraestrutura básica do cluster, a camada de tecnologias distribuídas, as ferramentas e conceitos fundamentais para a aplicação, e a camada de aplicação e motor de recomendação. Essa divisão utilizou ainda a aplicação de conceitos e testes em camadas específicas, especialmente na infraestrutura básica e na camada de aplicação, a fim de validar e assegurar a efetivação dos requisitos, conforme pode ser visto na Figura 1.

O desenvolvimento, conforme apresentado na Figura 1, será conduzido iniciando-se pela infraestrutura básica e avançando até a camada de aplicação e motor de recomendação. Em todas essas etapas, será realizado um estudo inicial para obter uma visão geral das tecnologias e ferramentas, considerando o ponto de vista baseado nos requisitos. Em seguida, serão fornecidas descrições teóricas embasadas na literatura, a fim de respaldar a execução prática. Por fim, serão realizadas a instalação, configuração e experimentação das tecnologias, visando proporcionar uma avaliação completa sobre o aprendizado.

Além disso, para a validação e análise dos resultados obtidos, serão empregados testes de estresse e carga, acompanhados do monitoramento contínuo do ambiente, garantindo, assim, uma base sólida para as discussões finais. Essa avaliação abordará a escalabilidade do sistema, sua capacidade de lidar com falhas, desempenho e suporte a altas demandas. Como ferramentas, serão utilizados o Linpack para os testes de estresse, o Locust para a validação da carga e o Grafana para o monitoramento contínuo do estado do cluster.

De maneira análoga, para a criação da biblioteca digital, será adotada uma abordagem estruturada em quatro categorias distintas: desenvolvimento do banco de dados,

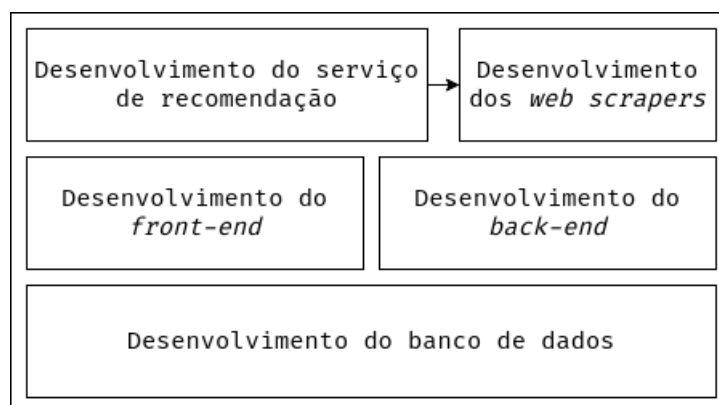
Figura 1 – Fases da metodologia para a infraestrutura



Fonte: Autores

implementação do *front-end*, programação do *back-end* e elaboração do serviço de recomendação. Além disso, dentro desta última camada, no serviço de recomendação, serão concebidos *web scrapers* destinados a alimentar o banco de dados com informações relevantes para o treinamento dos modelos desenvolvidos. Estas fases podem ser visualizadas na Figura 2.

Figura 2 – Fases da metodologia para a biblioteca digital



Fonte: Autores

É relevante ressaltar que, para o avanço de ambas as fases, serão adotadas duas metodologias específicas de desenvolvimento: o Kanban e o XP, com foco especialmente em testes, pareamento e refatorações. A escolha pelo Kanban decorre da necessidade de organizar as tarefas, proporcionando uma visualização clara do escopo definido, do *status*

atual e do progresso, reforçando, assim, o cumprimento dos prazos estipulados para as atividades, uma vez que essa metodologia é eficaz na solução dessas questões (AHMAD et al., 2018).

A escolha do XP como metodologia de desenvolvimento possui um viés estratégico, uma vez que grande parte das tarefas foi planejada para ser desenvolvida em pareamento. Ademais, a utilização de testes, conforme preconizado pelo XP (WELLS, 2009), será de suma importância para validar a análise dos resultados da implementação da infraestrutura. Por fim, no desenvolvimento da biblioteca digital, espera-se realizar refatorações evolutivas nas APIs, no banco de dados e nos modelos de recomendação, reforçando a importância da utilização do XP no desenvolvimento.

2.1.1 Etapas do desenvolvimento

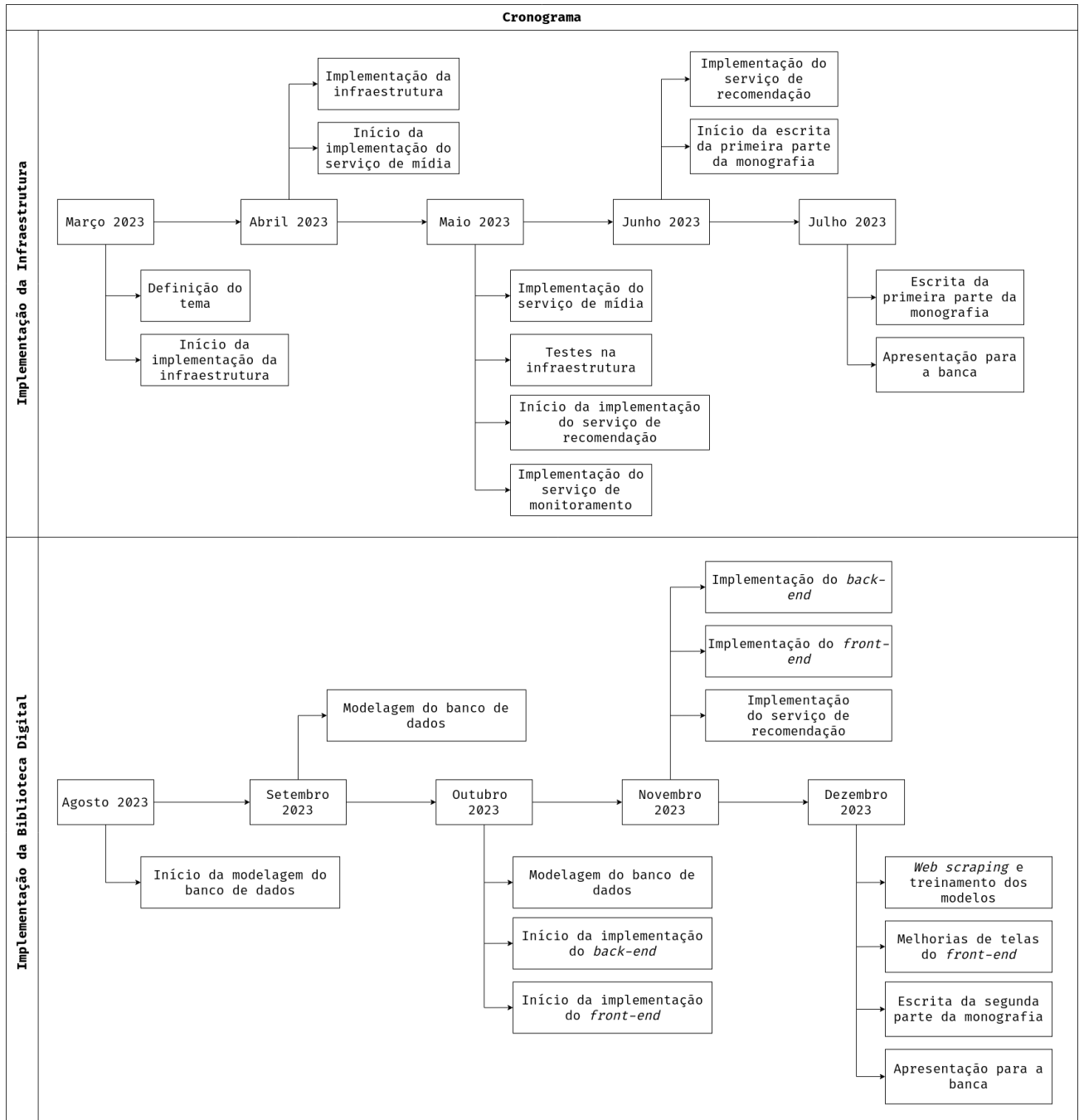
Para realizar a implantação e implementação do protótipo da biblioteca virtual de músicas, serão seguidos os seguintes passos:

- **Etapa 1:** Prototipação da infraestrutura, buscando explorar as diferentes possibilidades para criação de ambientes distribuídos;
- **Etapa 2:** Avaliação de infraestrutura, visando validar os pontos positivos e negativos das infraestruturas disponibilizadas na etapa 1;
- **Etapa 3:** Instalação e disponibilização da camada responsável por realizar a distribuição dos recursos físicos, com tecnologias como o Hadoop e o Spark;
- **Etapa 4:** Prototipação do mecanismo de pesquisa, avaliando a viabilidade de implementação nos componentes físicos disponíveis;
- **Etapa 5:** Prototipação do serviço de *streaming*, como uma visão inicial dos serviços que disponibilizam mídia, e
- **Etapa 6:** Implementação e testagem da aplicação, utilizando os resultados da etapa 5 concomitantes com os requisitos levantados para a aplicação.

2.1.2 Cronograma

Para proporcionar uma compreensão mais clara do processo de elaboração e execução das tarefas desta monografia, foi criado o cronograma apresentado na Figura 3. Nesse cronograma, são contempladas as duas fases do desenvolvimento deste projeto: a implementação da infraestrutura e a implementação da biblioteca digital.

Figura 3 – Cronograma de atividades



Fonte: Autores

3 Infraestrutura básica

Visando fornecer uma infraestrutura capaz de suportar altas demandas de carga, torna-se necessário um estudo sobre as tecnologias de criação de *clusters*. Independentemente de ser em ambiente *bare-metal* ou em máquinas virtuais, é de fundamental importância disponibilizar uma arquitetura que seja escalável, por meio da utilização de tecnologias distribuídas, e elástica, através do uso de orquestradores, a fim de garantir um atendimento eficaz em larga escala. Essas medidas são essenciais para garantir que a infraestrutura possa lidar com as demandas de carga esperadas e assegurar um desempenho adequado mesmo em situações de grande volume de trabalho.

3.1 Formação do *cluster*

É importante ressaltar que todas as vantagens de uma aplicação escalável podem ser limitadas ou até mesmo não aproveitadas caso o ambiente desenvolvido não consiga acompanhar o aumento na demanda de processamento adequadamente. Portanto, se torna crucial a utilização de ambientes e tecnologias que permitam a escalabilidade das aplicações de maneira eficiente, garantindo o atendimento das necessidades de processamento em constante evolução.

Além da escalabilidade, os serviços que atendem a uma grande quantidade de usuários também requerem resiliência e capacidade de recuperação de falhas. Em particular, esses sistemas devem ser capazes de detectar falhas e gerenciar recursos de forma eficiente, evitando perdas e a degradação da qualidade dos serviços oferecidos. Além disso, a escolha do ambiente adequado envolve a gestão e manutenção dos serviços, enfatizando a importância de ferramentas de fácil utilização que, ao mesmo tempo, possam fornecer todos os recursos necessários para a construção de ambientes robustos. Portanto, duas arquiteturas se destacam: *bare-metal* e máquinas virtuais.

3.1.1 Implantação em *bare-metal*

A primeira e mais antiga arquitetura é conhecida como *bare-metal*. Nesse modelo, todas as aplicações e serviços são gerenciados e executados diretamente no *hardware* subjacente dos servidores, sem a necessidade de camadas adicionais responsáveis pelo controle, divisão e gestão dos recursos distribuídos entre as aplicações. Ao utilizar diretamente os componentes físicos disponíveis, essa arquitetura permite uma alta eficiência e desempenho, uma vez que não há sobrecarga causada por camadas intermediárias.

No entanto, é importante ressaltar que a implantação *bare-metal* também apre-

senta algumas limitações. A falta de camadas de abstração adicionais pode tornar o gerenciamento e escalabilidade dos serviços mais complexos, uma vez que a responsabilidade recai inteiramente sobre o desenvolvedor e a equipe operacional. Além disso, essa arquitetura pode ser menos flexível em termos de adaptação a mudanças nos requisitos ou na infraestrutura.

Para superar esse desafio imposto, surgiram ferramentas específicas voltadas para a configuração, implantação, automação e gerenciamento, a fim de auxiliar na manutenção e desenvolvimento de sistemas baseados nessa arquitetura. Duas tecnologias principais destacam-se nesse contexto:

- Chef: Ferramenta de automatização da infraestrutura como código com funcionamento baseado no modelo cliente-servidor. Nesse sentido, o cliente é instalado em cada máquina ou nó responsável pela execução dos comandos, enquanto o servidor atua como uma estrutura centralizada que abriga todos os livros de receitas (*cookbooks*) disponíveis para utilização pelos clientes. Cada livro de receitas representa a infraestrutura, descrevendo comandos e ações em forma de código, que serão interpretados e executados nos nós designados (CORPORATION, 2023).
- Ansible: O Ansible é uma ferramenta que visa a automatização da infraestrutura por meio do uso de código. Assim como o Chef, o Ansible permite descrever a infraestrutura como um conjunto de instruções que podem ser executadas de forma programática. No entanto, o Ansible apresenta uma estrutura menos complexa em comparação com o Chef. Uma das diferenças fundamentais é que o Ansible pode ser configurado com apenas um nó principal responsável por executar os comandos em máquinas selecionadas, utilizando o protocolo SSH. Além disso, o Ansible permite descrever a infraestrutura em manuais (*playbooks*), os quais são interpretados pelo nó principal e executados nos nós escolhidos. (HAT, 2023).

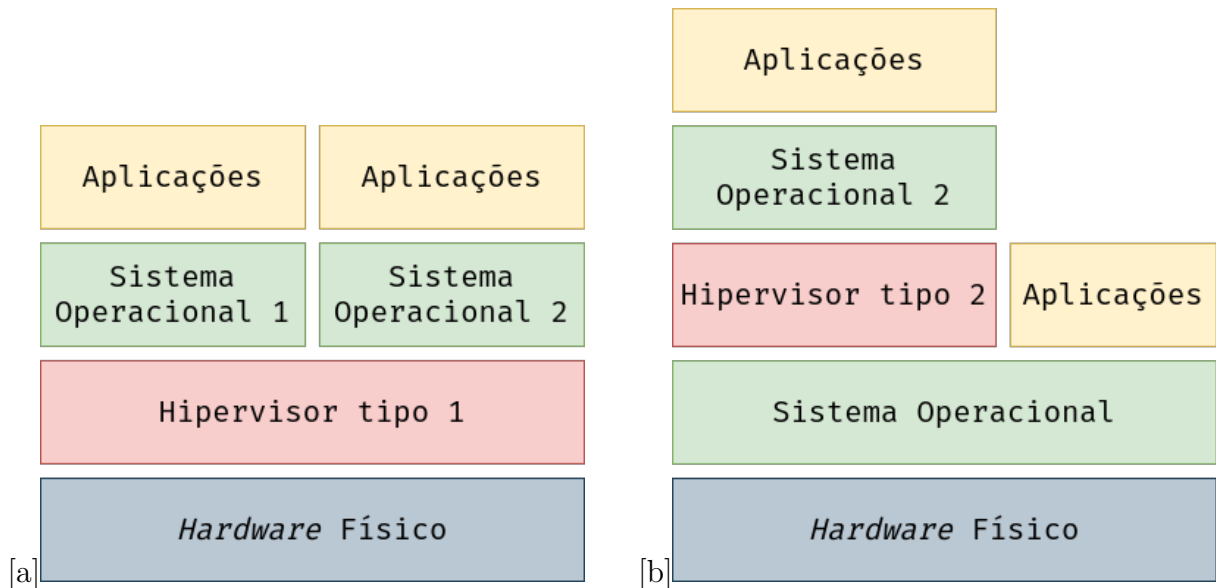
3.1.2 Implantação em máquinas virtuais

Com o avanço da tecnologia, ocorreram importantes avanços conceituais e implementações no campo da distribuição de serviços *web* em *cluster*. A virtualização, em particular por meio do uso de máquinas virtuais, emergiu como uma ferramenta fundamental que começou a ser desenvolvida na década de 1960, conforme relatado por Varian (1997 apud DANIELS, 2009). Com essa nova camada de tecnologia, pesquisadores e desenvolvedores passaram a repensar como as aplicações são organizadas, a fim de aproveitar todos os benefícios proporcionados pela virtualização e oferecer produtos de maior qualidade aos clientes.

De acordo com Daniels (2009) e demonstrado na Figura 4, uma máquina virtual é um ambiente ou camada de abstração localizada entre o hardware físico e o usuário final.

Nesse sentido, espera-se que uma máquina consiga executar múltiplas máquinas virtuais, gerenciando o compartilhamento eficiente de recursos, tais como CPU, controladoras, discos, memória (RAM) e periféricos de entrada e saída, como mouses e teclados. Essa camada de abstração, conhecida como hipervisor (*hypervisor*), é responsável por facilitar as interações entre as máquinas virtuais e os componentes físicos do sistema eficientemente.

Figura 4 – [a] Arquitetura de máquinas virtuais tipo 1 e [b] Arquiteturas de máquinas virtuais tipo 2



Fonte: Tanenbaum e Bos (2016)

Esse novo formato de implantação de serviços oferece às empresas a oportunidade de reduzir os custos e aproveitar melhor os recursos disponíveis (DANIELS, 2009). No entanto, apesar desses benefícios, é importante ressaltar que o desempenho das aplicações pode ser afetado devido à introdução de uma camada adicional entre o *hardware* e o *software*. Portanto, tecnologias como o KVM¹ permitem a virtualização com perdas mínimas de desempenho, se tornando essenciais para a implantação desse modelo.

Além disso, problemas de gerenciamento e escalabilidade em ambiente *clusterizados*, frequentemente encontrados no modelo *bare-metal*, são mitigados com o surgimento de ferramentas capazes de fornecer interfaces intuitivas e facilitar o controle dos ambientes nos múltiplos nós, como o OpenNebula e o Proxmox. Essas soluções permitem gerenciar de forma mais eficiente os recursos e escalar a infraestrutura conforme necessário, proporcionando uma maior flexibilidade e facilidade de administração.

- OpenNebula: Plataforma de código-fonte aberto experimentada nesse trabalho, que oferece facilidade de uso na construção e gerenciamento de servidores de nuvem.

¹ Mais informações sobre a tecnologia podem ser encontradas em: <https://www.linux-kvm.org/page/Main_Page>

Essa ferramenta permite a orquestração de máquinas virtuais, podendo lidar tanto com sistemas simples quanto com sistemas complexos que exigem regras sofisticadas de escalonamento. Um componente essencial dessa ferramenta é o *OpenNebula Daemon*, o qual pode ser instanciado em múltiplas máquinas, proporcionando uma distribuição eficiente e paralelismo em *clusters* de máquinas (SYSTEMS, 2023).

- Proxmox: Ferramenta semelhante ao OpenNebula, que foi estudada durante a monografia, projetada para o gerenciamento e manutenção de servidores de nuvem. Assim como o OpenNebula, o Proxmox também oferece recursos de orquestração de máquinas virtuais, fornecendo ambientes de alta disponibilidade, escalabilidade e desempenho para as aplicações. Ao contrário do OpenNebula, o Proxmox adota uma abordagem de virtualização direta em *hardware*, considerando-se como um sistema operacional em si. Essa característica proporcionando um desempenho otimizado e uma melhor eficiência em termos de utilização dos recursos disponíveis, se destacando, portanto, o Proxmox como uma solução completa e robusta para virtualização (GmbH, 2023).

3.1.3 Tecnologias auxiliares para armazenamento

Com o contínuo crescimento da Internet, o armazenamento se torna um aspecto essencial para os sistemas, em especial para bibliotecas digitais de música que armazenam diversas informações. Características como resistência a falhas, adaptabilidade e eficiência são altamente valorizadas nos sistemas responsáveis por essa função. Os dispositivos de *hardware* utilizados, como HDs e SSDs, possuem a capacidade de escalar horizontal e verticalmente, permitindo a incorporação de novas máquinas ao conjunto já existente.

No intuito de alcançar essas escalabilidades, são utilizadas diversas ferramentas, tais como o NFS, em conjunto com conceitos como Redundant Array of Inexpensive Drives (RAIDs) e Content Delivery Networks (CDNs). Ao combinar essas soluções, é possível estabelecer um ambiente de armazenamento resiliente, que satisfaz as crescentes demandas de dados. Essas tecnologias oferecem mecanismos eficazes para lidar com o aumento do volume de informações, garantindo a confiabilidade, a disponibilidade e a eficiência necessárias para os sistemas de armazenamento.

3.1.3.1 Armazenamento confiável com RAIDs

Ao longo da evolução dos sistemas, é observada uma assimetria no progresso entre diferentes componentes físicos. Em muitos casos, os avanços nos processadores superaram significativamente as melhorias nos mecanismos de armazenamento. Como resultado, ocorre uma disparidade entre essas tecnologias, resultando em situações em que o desempenho do processamento é comprometido pela limitação na capacidade de leitura e

escrita dos discos. Visando mitigar esses problemas, foi introduzida a ideia de conjuntos redundantes de discos baratos, conhecidos como RAIDs.

Centrada na distribuição e paralelização, essa estratégia visa aprimorar a confiabilidade e o desempenho do armazenamento por meio da utilização de vários discos rígidos para a alocação dos dados. Como ressaltam [Chen et al. \(1994, tradução nossa\)](#):

O agrupamento de discos é a solução natural para o problema do armazenamento. [...] Os dados serão divididos entre múltiplos discos e acessados paralelamente para alcançar uma maior taxa de transferência em grandes acessos e uma maior taxa de entrada e saída de dados para acessos em menor escala.

Outra relevância dos sistemas RAID está na capacidade de recuperação de falhas. Ao organizar os discos em grupos, é possível evitar a perda de dados em situações de falhas catastróficas, por meio da duplicação das informações entre os discos presentes nos sistemas RAID. No entanto, é importante ressaltar que essa duplicação pode resultar em uma diminuição de desempenho ([CHEN et al., 1994](#)).

Portanto, para atender às necessidades específicas de cada caso de uso, diversas configurações de RAID podem ser adotadas. Variações como o RAID 0, cujo objetivo é garantir a velocidade das operações, e o RAID 1, que, por outro lado, assegura a preservação redundante dos dados, são exemplos de configurações formalizadas e padronizadas pelo Storage Networking Industry Association (SNIA) ([ASSOCIATION, 2009](#)). Esses padrões estabelecidos pelo SNIA fornecem diretrizes e práticas recomendadas para a implementação de sistemas RAID, contribuindo para a segurança e eficiência do armazenamento de dados.

3.1.3.2 Disponibilizando conteúdos geograficamente com CDNs

A distribuição de conteúdos é uma questão amplamente abordada no contexto dos servidores. Especificamente, soluções eficientes para enfrentar os desafios relacionados ao compartilhamento de objetos da *web*, dados estáticos e mídias têm sido objeto de estudos que culminaram no desenvolvimento das Redes de Distribuição de Conteúdo (CDNs).

Por meio de uma abordagem distribuída, as CDNs desempenham um papel crucial ao aliviar a carga sobre os servidores principais, reduzir a latência percebida pelos usuários, aumentar a taxa de transferência de pacotes e viabilizar uma maior escalabilidade dos serviços oferecidos ([VAKALI; PALLIS, 2003](#)). Essas soluções proporcionam benefícios significativos no desempenho e na disponibilidade de conteúdos, aprimorando a experiência do usuário e garantindo uma entrega eficiente dos recursos solicitados.

Além disso, outra vantagem significativa das CDNs reside na sua capacidade de distribuição geográfica dos conteúdos. Essa abordagem envolve a instalação de múltiplos

servidores em diferentes regiões do globo terrestre, permitindo que os usuários acessem os conteúdos a partir do servidor mais próximo de sua localização geográfica. Ainda de acordo com [Vakali e Pallis \(2003\)](#), existem três técnicas comumente utilizadas para viabilizar a distribuição de conteúdo nas CDNs: redirecionamento por DNS, reescrita de URLs e CDN *peering*.

De acordo com [Vakali e Pallis \(2003\)](#), no redirecionamento por DNS, a CDN atua como um sistema central que possui conhecimento sobre todos os servidores disponíveis. Utilizando métricas como latência e perda de pacotes, a CDN identifica e redireciona o usuário para o servidor mais adequado com base nessas métricas. Por sua vez, a reescrita de URLs consiste em modificar as URLs presentes nas páginas da *web*, tanto de forma estática quanto dinâmica, direcionando o usuário para o servidor que melhor atende às suas necessidades. Por fim, o CDN *peering* envolve a comunicação entre múltiplas CDNs, permitindo o redirecionamento de conteúdos por meio de acordos de compartilhamento.

Diversas tecnologias se destacam no contexto das CDNs e descobrimento de serviços, porém para fins de estudo desse TCC foram avaliadas metodologicamente apenas as seguintes:

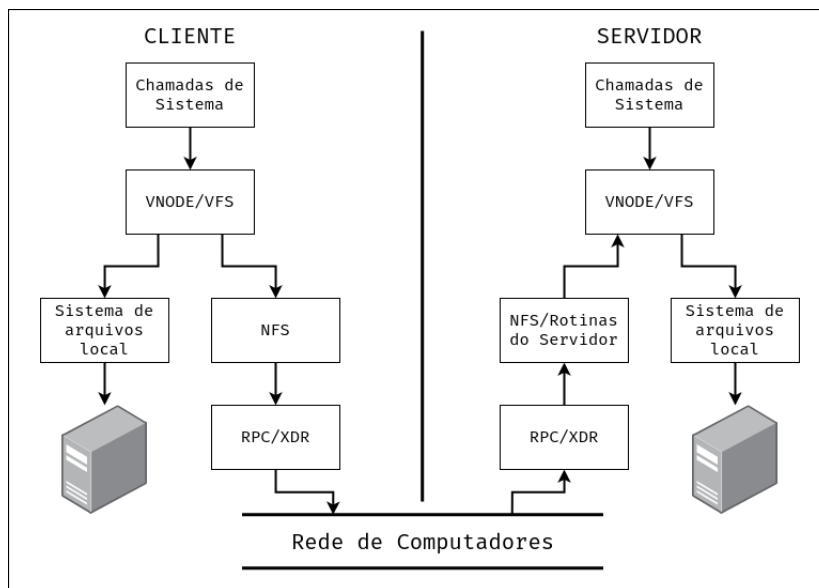
- Consul: Tecnologia para descobrimento de serviços em uma rede de computadores, que provê uma malha de serviços, gerenciamento de tráfego e atualizações automatizadas para dispositivos da infraestrutura de rede ([HASHICORP, 2023](#)).
- Eureka: Ferramenta originada na Netflix para descobrimento de serviços e balanceamento de carga. Sua arquitetura é fundamentada no modelo de *peers*, e disponibiliza uma API REST para a gestão das máquinas presentes na CDN ([NETFLIX, 2014](#)).

3.1.3.3 Network File System

Apesar da relevância dos conceitos e ferramentas, como os sistemas de RAID e as CDNs, ainda existem diversas necessidades de armazenamento que não são totalmente atendidas. Para suprir algumas dessas lacunas, novas tecnologias, como o Network File System (NFS), foram desenvolvidas. O NFS, conforme mencionado por [Sandberg et al. \(1985\)](#), é uma camada que possibilita o acesso remoto e transparente ao sistema de arquivos de um servidor central de forma distribuída, com ênfase na independência entre máquinas e sistemas operacionais, recuperação de falhas e desempenho. A Figura 5 ilustra a arquitetura simplificada do NFS.

A implementação do NFS emprega o mecanismo conhecido como Remote Procedure Call (RPC) para facilitar a comunicação entre o cliente e o servidor. Essa tecnologia, caracterizada como *stateless* (sem estado), possibilita ao NFS adotar uma abordagem simplificada para a recuperação de falhas. Nesse contexto, o NFS entra em um estado de ciclos que reenvia a mesma requisição até o recebimento com sucesso, assegurando, desse modo,

Figura 5 – Arquitetura simplificada do NFS



Fonte: Sandberg et al. (1985, tradução nossa)

resiliência e eficiência no tratamento de falhas durante a comunicação (SANDBERG et al., 1985).

Conforme destacado por Sandberg et al. (1985), o NFS adota o modelo cliente-servidor para a troca de mensagens. O servidor, encarregado de responder e executar os comandos, adota a abordagem sem estado mencionada anteriormente, garantindo que comandos que geram alterações no sistema de arquivos sejam executados antes de retornar aos clientes. Por sua vez, os clientes atuam como interfaces transparentes que permitem aos usuários utilizar o sistema de arquivos disponível no servidor.

3.1.4 Serviços de diretório

Ao ter um ambiente com vários computadores, segundo Eisler, Labiaga e Stern (2001), um grande problema é manter a cópia das configurações de usuários, senhas, grupos e *hosts* sincronizadas, além da autenticação e autorização em sistemas e aplicações distribuídas, de forma que a alteração da configuração em uma das máquinas seja replicada para as demais. Dessa forma, os serviços de diretório consistem em associar um nome a outro, tornando a configuração acessível a várias máquinas.

Os serviços de diretório possuem amplo suporte pelos sistemas operacionais. Por exemplo, no Linux, há o arquivo de configuração “/etc/nsswitch.conf”, onde é possível especificar a origem dos nomes de *passwd*, *group*, *shadow*, *gshadow*, *hosts*, etc, associando esses nomes a um serviço de diretório torna o uso transparente ao usuário independente da máquina. Logo, a adição de um novo *host*, usuário e grupo é facilitada, uma vez que a configuração é replicada, mantendo a consistência entre os nós. (EISLER; LABIAGA;

[STERN, 2001](#))

Dentre as implementações possíveis, duas ferramentas se destacam, o Network Information Service (NIS) e o Lightweight Directory Access Protocol (LDAP), e são detalhadas a seguir.

O NIS é um serviço de diretório, desenvolvido pela Sun Microsystems na década de 80, baseado no modelo cliente-servidor em RPC, onde visa centralizar a configuração e distribuir entre máquinas em uma rede. O *master* é o servidor central que possui o mapa com as informações. Já o *worker* é responsável por conter uma réplica das informações e atender às requisições do cliente. Assim que uma modificação é feita, na configuração armazenada no *master*, é distribuída para os *workers* e atualizada com o cliente na próxima consulta. ([EISLER; LABIAGA; STERN, 2001](#))

O LDAP é um protocolo aberto para serviço de diretório, sendo uma extensão mais leve do Directory Access Protocol e descendente do X.500. Permite centralizar mais informações do que o NIS, como configuração de rede, preferências de aplicações, certificados e controle de acesso. A arquitetura segue o modelo cliente-servidor, onde a comunicação entre o cliente, *worker* e *master* acontece por meio do próprio protocolo LDAP. ([KOUTSONIKOLA; VAKALI, 2004](#))

3.1.5 Testes da infraestrutura

Visando garantir a escalabilidade, confiabilidade, adaptabilidade, desempenho e resiliência do *cluster* de máquinas, é essencial planejar e executar uma série de testes nas infraestruturas. Várias áreas podem ser avaliadas, incluindo o consumo de CPU, o uso da memória volátil (RAM), operações de leitura e escrita nos discos físicos, transmissão e recebimento de dados pelas placas de rede, além de várias outras características relevantes.

Os testes de *performance* são de fundamental importância para a avaliação da eficiência de um sistema, além de exercerem um papel crucial na identificação de lacunas que não são reveladas por meio de testes funcionais ([HENDAYUN; GINANJAR; IHSAN, 2023](#)). Duas metodologias podem ser empregadas para a validação do desempenho, nomeadamente os testes de *load* e os testes de *stress*.

Não somente por meio de testes, mas também por meio do monitoramento contínuo das máquinas, busca-se assegurar a observância das características previamente abordadas. A supervisão, adicionalmente, facilita a detecção de problemas que possam surgir ao longo do tempo, capacitando os profissionais para respostas rápidas e eficazes, além de simplificar a fiscalização de várias áreas, especialmente aquelas críticas, da infraestrutura ou de um sistema específico.

3.1.5.1 Avaliando o desempenho com o Linpack

Conforme mencionado por Naik (2008), os testes de estresse têm a finalidade de avaliar o comportamento de um sistema quando exposto a uma carga que ultrapassa sua capacidade projetada. Com base nisso, busca-se validar o funcionamento adequado do sistema diante de condições extremas esperadas durante picos de carga. Esses testes permitem verificar não apenas se o sistema atinge seu limite e falha, mas também avaliar os mecanismos de recuperação implementados, abordando três pontos cruciais: vazamentos e gravações de memória, bem como alocações de buffers.

Ainda segundo Naik (2008), a execução do teste comumente se inicia com a avaliação da escalabilidade do sistema diante da carga introduzida, sendo finalizado quando ocorre uma falha. Portanto, ao submeter o sistema a uma carga acima do limite esperado, diversos gargalos passam a ser identificados em diferentes partes do sistema. Essa análise serve, dessa maneira, como base para discussões sobre a robustez e a capacidade de recuperação do sistema testado em caso de falhas.

Com base na teoria dos testes de estresse, surgiram diversas ferramentas para comparação de desempenho, sendo o Linpack um dos destaques. Proposta por Dongarra (1992), essa ferramenta consiste na resolução de um denso sistema de equações com pontos flutuantes, problema comumente encontrado na computação. Sua evolução, o High Parallel Computing Linpack Benchmark ou HPL (PETITET et al., 2018), possibilitou a execução em paralelo dos cálculos garantindo maior escalabilidade e controle dos testes.

Para padronizar as medições, Dongarra, Luszcz e Petit (2002) definiram o número de operações requeridas para as resoluções das equações como sendo:

$$\frac{2}{3n^3} + 2n^2 + O(n) \quad (3.1)$$

Bem como, para evitar a excessiva otimização em favor da assertividade, a acurácia necessária nos resultados:

$$\frac{\|Ax - b\|}{\|A\| \cdot \|x\| \cdot n \cdot \epsilon} = O(1) \quad (3.2)$$

Onde n representa o tamanho do problema, $A \in \mathbf{R}^{n \times n}$ e $x, b \in \mathbf{R}^n$.

Em resposta às regulamentações e ao interesse em obter informações sobre a localização e o desempenho dos supercomputadores nas diversas instituições, foi criada em 2002 a lista TOP500². Essa iniciativa surgiu como uma solução para suprir tais necessidades, fornecendo um espaço adequado para essas comparações.

² A lista pode ser encontrada em <<https://www.top500.org/>>

A lista TOP500, ao adotar o Linpack como método de avaliação e por ser amplamente reconhecida e referenciada como uma fonte autorizada neste contexto, oferece uma base sólida e argumentos consistentes para a utilização da ferramenta de avaliação durante a segunda fase deste trabalho. Além de sua relevância como referência, a lista TOP500 também apresenta uma comparação do poder computacional da infraestrutura construída. Portanto, sua inclusão reforça ainda mais a validade e a abrangência dos resultados obtidos.

3.1.6 Monitoramento da infraestrutura

Apesar da disponibilidade de diversas ferramentas de teste, há uma demanda constante para garantir e manter a qualidade dos serviços distribuídos aos usuários. Essa necessidade se reflete na utilização e no desenvolvimento de aplicações e camadas responsáveis pelo monitoramento abrangente do sistema. De acordo com [Hasan, Jaafar e Hassan \(2012\)](#), a garantia da qualidade é um acordo estabelecido entre as partes envolvidas, ou seja, os usuários e os provedores, visando garantir que os usuários recebam o que pagaram, enquanto os provedores forneçam os recursos necessários para atender às demandas.

O monitoramento desempenha um papel essencial no controle, entendimento e manutenção da qualidade, tornando-se necessário avaliar individualmente cada subsistema para uma análise mais precisa e aprofundada dos vários níveis da aplicação. Uma das camadas comumente monitoradas é a camada de redes, responsável por estabelecer as conexões do sistema.

[Tsai et al. \(2018\)](#) entende que o monitoramento dessa camada proporciona uma nova perspectiva sobre os estados das redes e seus comportamentos, beneficiando a gerência, a engenharia de tráfego, a qualidade do serviço e a detecção de possíveis anomalias no sistema. Além disso, [Tsai et al. \(2018\)](#) propõe uma metodologia em etapas para a coleta, avaliação e apresentação do monitoramento de redes, que pode ser aplicada de forma generalizada e expandida para os demais níveis do software. As etapas dessa metodologia, incluem:

- Coleta: Etapa de coleta dos dados monitorados, focando em três características principais: meios, alvo e frequência. Os meios representam como os dados serão coletados, o alvo são os dispositivos observados e a frequência o espaço de tempo entre a execução das coletas.
- Pré-processamento: Fase que os dados originais são agregados e transformados em algum formato estatístico. Esse processamento agrega no monitoramento especificando e rastreando os resultados da medição.

- Análise: Momento de análise dos dados pré-processados para geração de estatísticas e identificação de eventos específicos. Essa detecção se torna crucial na descoberta e notificação sobre o estado e possíveis falhas do sistema monitorado.
- Apresentação: Etapa final de extração dos dados analisados em gráficos para facilitar a sua visualização.

Além da camada de redes, outro subsistema de extrema importância para o monitoramento é o servidor de banco de dados. A supervisão desse subsistema permite que os Administradores de Bancos de Dados (DBAs) procedam com eficiência e reduzam os custos, avaliando o tempo de execução e a quantidade de chamadas das consultas (CHAUDHURI; KÖNIG; NARASAYYA, 2004). Além disso, informações sobre o consumo de recursos também podem ser extraídas, fornecendo dados relevantes para avaliações do sistema e auxiliando nas tomadas de decisões.

Por fim, é relevante ressaltar que os testes de carga e estresse obtêm resultados mais precisos por meio da implementação das camadas de monitoramento. Essa abordagem é viabilizada pela coleta, processamento e análise de dados relacionados à utilização dos recursos físicos (*hardware*), permitindo a identificação de falhas que podem não ser prontamente detectadas apenas por meio dos testes.

3.1.6.1 Coletando e processando informações do sistema

O monitoramento, conforme já discutido anteriormente, tem início com a etapa de coleta de informações do sistema. Nessa fase, dois tipos de dados distintos se destacam: as métricas e os registros (*logs*). As métricas consistem em medidas do sistema obtidas em intervalos de tempo específicos, enquanto os registros são conjuntos de dados, geralmente em formato textual, relacionados a eventos específicos ocorridos no sistema.

Portanto, devido à natureza distinta dessas informações e às diferentes origens e momentos de coleta, as ferramentas utilizadas para o monitoramento também variam, realizando processamentos específicos nos dados extraídos. Nesse contexto, duas dessas ferramentas se destacam:

- Prometheus: Ferramenta de código aberto para monitoramento e alerta. Construída em cima de um modelo de dados multidimensional de séries temporais identificados pelo nome da métrica e pares chave/valor, com uma linguagem de busca específica, o PromQL (LABS, 2023c).
- Elasticsearch: Mecanismo potente, de código aberto, para armazenar, procurar e gerenciar *logs*. Por ser construído em cima do Apache Lucene³, o Elasticsearch

³ Mais informações sobre a tecnologia podem ser encontradas em: <<https://lucene.apache.org/>>

obtêm todo o poder de processamento, quase em tempo real, de textos estruturados ou não, dados numéricos ou geoespaciais (ELASTIC, 2023a).

3.1.6.2 Análise e apresentação gráfica

Com os dados coletados e processados, avança-se para as etapas finais do monitoramento, que envolvem a análise e representação gráfica das informações. Nesses estágios, as ferramentas se concentram na capacidade de realizar consultas avançadas nos dados disponibilizados em tempo real, fornecendo painéis (*dashboards*) para a visualização gráfica. Muitos desses filtros complexos são oferecidos pelas tecnologias responsáveis pelo processamento dos dados, estabelecendo, assim, uma forte interconexão entre essas ferramentas. Alguns *softwares* que se destacam são:

- Grafana: Ferramenta de código-fonte aberto focado em consultas, observação e alerta das métricas. Suas principais capacidades incluem a consolidação das informações sem a necessidade de um banco de dados centralizado, possibilitando a criação de notificações personalizadas. Além disso, a ferramenta oferece interfaces para a execução de consultas avançadas por meio de integrações externas (LABS, 2023a).
- Kibana: *Software* de código aberto dedicado à análise e representação gráfica de registros (*logs*). Assim como o Grafana, o Kibana é uma ferramenta que permite a visualização de dados previamente extraídos e processados. No entanto, ao contrário da ferramenta anterior, o Kibana concentra-se especificamente na representação gráfica dos *logs* provenientes do Elasticsearch (ELASTIC, 2023b).

3.2 Camada de implantação distribuída

À medida que o tempo avança, a demanda pelos recursos das máquinas tem se tornado cada vez mais intensa, resultando em problemas de desempenho ou, em casos extremos, na impossibilidade de fornecer os serviços desejados devido à escassez dos recursos físicos. Essas questões têm impulsionado crescentemente a adoção de agrupamentos de máquinas em junção a tecnologias distribuídas, que podem prover, coletivamente, os recursos necessários para as tarefas computacionais esperadas.

Essa utilização dos recursos distribuídos viabiliza a conquista de um dos requisitos fundamentais para a criação da biblioteca digital: a escalabilidade. Além disso, a elasticidade proporcionada pelos *clusters* permite a expansão do acesso aos serviços, possibilitando atender a uma quantidade maior de usuários. Sendo possível, dessa forma, satisfazer outro requisito essencial da aplicação: a capacidade de lidar com altas demandas.

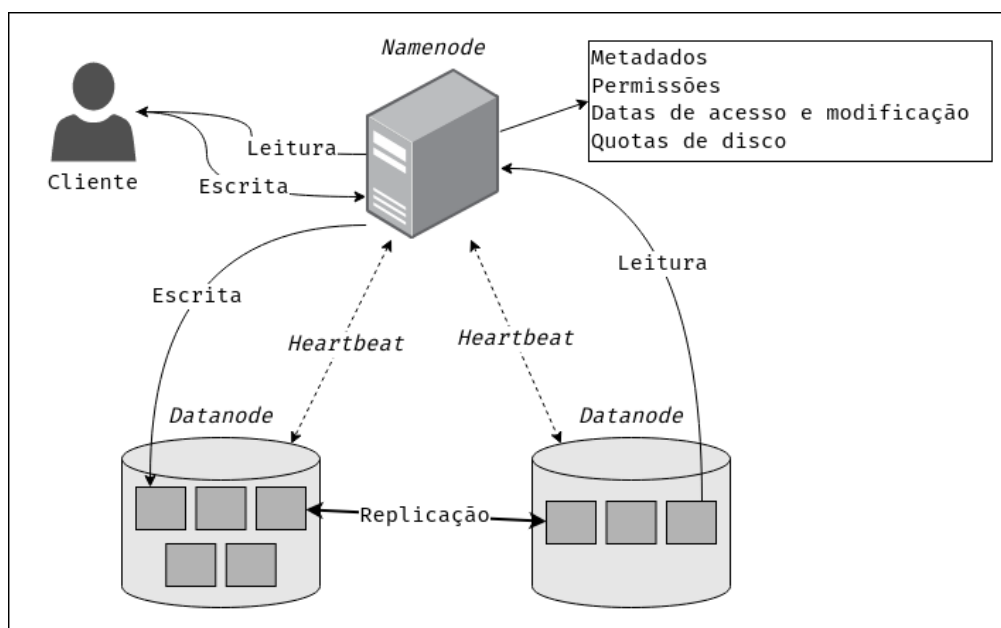
Portanto, com a adição desta nova camada de ferramentas distribuídas, torna-se viável o desenvolvimento e a implantação de aplicativos mais sofisticados e avançados, que aproveitam plenamente o poder computacional de alto desempenho. Tecnologias como Apache Hadoop, Apache Spark, OpenMP, OpenMPI, entre outras, destacam-se na criação e estruturação dessas infraestruturas poderosas, como a biblioteca digital de música proposta neste trabalho.

3.2.1 Apache Hadoop

Uma tecnologia amplamente reconhecida por sua capacidade de processamento e armazenamento distribuído e paralelo de dados é o Apache Hadoop. Essa poderosa ferramenta é composta por quatro módulos principais: Hadoop Common, Hadoop Distributed File System (HDFS), Hadoop YARN e Hadoop MapReduce. O Apache Hadoop tem sido amplamente adotado na construção de sistemas distribuídos e paralelos, devido à sua capacidade de lidar eficientemente com grandes volumes de dados e oferecer um alto desempenho computacional.

Diferentemente de outras soluções voltadas para o armazenamento em ambientes distribuídos, o Apache Hadoop, em particular o HDFS, apresenta características distintas. Conforme apontado por [Foundation \(2023\)](#), o HDFS se destaca pela sua notável tolerância a falhas e pela capacidade de ser implantado em *hardwares* de baixo custo. Essa solução proporciona um acesso de alto rendimento, tornando-a especialmente adequado para o processamento de grandes volumes de dados.

Figura 6 – Arquitetura simplificada do HDFS



Fonte: Adaptado de [Foundation \(2023\)](#) e [Shvachko et al. \(2010\)](#)

A arquitetura do HDFS, mostrada na Figura 6, é composta por dois componentes principais: o *namenode* e o *datanode*, conforme descrito por Shvachko et al. (2010). O *namenode* desempenha um papel fundamental ao gerenciar os arquivos e diretórios armazenados no HDFS. Ele armazena informações sobre permissões, datas de acesso e modificação, *namespaces* e quotas de disco. Além disso, o *namenode* é responsável pela distribuição e replicação dos blocos de dados nos *datanodes*.

Os *datanodes*, por sua vez, desempenham um papel distribuído nos diversos computadores de um *cluster*, como mencionado por Shvachko et al. (2010). Sua principal responsabilidade é o armazenamento dos dados em blocos, selecionados pelo *namenode*. Além dos dados em si, o *datanode* também mantém dois metadados, o *checksum* e um selo de geração, utilizados na validação e na transmissão de informações para o *namenode* por meio de *heartbeats* (validações periódicas).

Todas essas características de armazenamento distribuído fornecidas pelo Apache Hadoop, em particular pelo HDFS, desempenham um papel fundamental no desenvolvimento de uma camada distribuída que pode eficientemente separar e disponibilizar recursos. Portanto, a utilização desse sistema de arquivos, juntamente com a camada mais interna da infraestrutura, o NFS, desempenha um papel significativo na consecução de uma infraestrutura escalável capaz de atender às demandas de gerenciamento de arquivos.

Essa camada, em particular, desempenha um papel fundamental no armazenamento intermediário dos modelos e dos conjuntos de dados (*datasets*) propostos neste trabalho. A capacidade de armazenamento distribuído possibilita um armazenamento e acesso eficientes desses dados, que são volumosos, além de conferir resiliência e escalabilidade, características essenciais para tais operações.

3.2.1.1 Apache Spark

Apesar do serviço de arquivos distribuídos do Apache Hadoop, tarefas como o processamento de *stream* de dados em tempo real ainda não são atendidas. Atualmente, muitas aplicações *web* se baseiam nesse conceito para oferecer recomendações mais precisas, identificar tendências em redes sociais e até monitorar sistemas e aplicações. Portanto, é necessário contar com uma ferramenta robusta e escalável capaz de atender a esses usuários, aproveitando todo o poder do processamento paralelo e distribuído nos *clusters* de máquinas.

Uma das ferramentas que se destaca nesse contexto é o Apache Spark. Ele adota um modelo em micro *batch*, onde a carga de dados ocorre em intervalos curtos de tempo. Por meio das Discretized Streams (D-Streams), o Spark é capacitado a operar de maneira eficiente, escalável e resiliente, sendo capaz de se recuperar de falhas (ZAHARIA et al., 2013).

Nesse sentido, o Apache Spark utiliza a estrutura de comunicação *Master-Worker*. Ainda segundo Zaharia et al. (2013), o *master* é responsável por controlar os D-Streams e agendar novas tarefas, enquanto os *workers* recebem as informações, processam-nas e executam as funções designadas. Essa abordagem, portanto, permite fazer o uso do alto poder computacional distribuído e paralelizado dos agrupamentos de computadores.

Portanto, por meio do Apache Spark, torna-se viável a execução de tarefas computacionalmente intensivas. Além disso, a estrutura de *Master-Worker* capacita essa tecnologia a ser escalável, atendendo ao requisito fundamental para aplicações em grande escala. Essa característica tem uma relevância direta no tema central desta monografia, permitindo que as bibliotecas digitais atendam em larga escala, através do uso distribuído dos recursos de *hardware*.

Com as vantagens proporcionadas pelo Spark para o processamento iterativo e distribuído de dados, surge um ambiente propício para a aplicação de técnicas de inteligência artificial. Esse contexto impulsionou o desenvolvimento da MLlib⁴, uma biblioteca de aprendizado de máquina voltada para sistemas distribuídos, que se beneficia do paralelismo para operar nos modelos (MENG et al., 2015). Essa abordagem possibilita aos desenvolvedores a realização de análises em tempo real, de maneira escalável e com alto desempenho.

Ainda segundo Meng et al. (2015), o MLlib apresenta três principais funcionalidades distintas: suporte a métodos e utilitários de inteligência artificial, otimizações de algoritmos e uma API para linhas de processamento (*pipelines*), além de fornecer suporte distribuído em ambientes Spark. Desse modo, o MLlib disponibiliza a implementação distribuída de uma variedade de algoritmos comuns de inteligência artificial, tais como modelos lineares, árvores de decisão, classificação, regressão e filtragem colaborativa, entre outros.

Além das versões distribuídas, foram realizadas diversas otimizações em vários algoritmos, visando aproveitar ao máximo os recursos disponíveis nos múltiplos nós. O MLlib obteve melhorias significativas ao utilizar recursos algébricos da linguagem C++ e ao empregar métodos de comunicação do Spark para distribuir as tarefas. Adicionalmente, a existência de uma API para pipelines permite aos pesquisadores a construção de modelos de aprendizado em múltiplos estágios, simplificando e reduzindo o custo de execução dessas tarefas (MENG et al., 2015).

Dessa forma, assim como ocorre no processamento de *stream* de dados, os modelos de inteligência artificial e suas execuções são amplamente beneficiados pela capacidade de processamento do Spark. Em particular, devido à escalabilidade alcançada, torna-se possível construir, treinar e utilizar sistemas de recomendação, foco central deste trabalho,

⁴ Mais informações sobre a tecnologia podem ser encontradas em: <<https://spark.apache.org/mllib/>>

de forma mais eficiente e tolerante a falhas.

Adicionalmente, para fornecer suporte à programação e execução dos códigos relacionados às ferramentas do Apache Spark, optou-se pela instalação do Jupyterhub como ambiente apropriado para essa finalidade. O Jupyterhub oferece uma interface flexível e intuitiva para o desenvolvimento e execução de programas, especialmente voltados à ciência de dados, por meio tanto do Jupyter Notebook quanto do JupyterLab⁵. Portanto, para viabilizar a integração entre o Spark e os notebooks, é necessário utilizar a biblioteca PySpark⁶, responsável por disponibilizar as funções e módulos que se comunicam com o Apache Spark.

3.2.1.2 OpenMP e OpenMPI

Além do Apache Spark, mais duas ferramentas se destacam no compartilhamento de recursos, tanto verticalmente quanto horizontalmente: OpenMP e MPI. A primeira delas, OpenMP, é uma tecnologia voltada para o paralelismo em CPUs, utilizando o conceito de *threads*. De acordo com [OpenMP \(2021\)](#), seu modelo adota a abordagem *Fork-Join* para a execução paralela de códigos em um único processador. Isso significa que várias bifurcações (ou *threads*) podem ser criadas para executar atividades e, ao serem concluídas, são agrupadas novamente na tarefa principal.

Similarmente, o protocolo Message Passing Interface (MPI) desempenha um papel crucial ao fornecer uma infraestrutura para a computação paralela entre CPUs, mas agora de maneira horizontal, ou seja, entre múltiplas máquinas ou nós de um *cluster*. Diversas ferramentas, portanto, fazem uso desse protocolo de comunicação para criar ambientes robustos, escaláveis e de alta performance para execuções paralelas.

Uma das tecnologias relevantes nesse contexto é o OpenMPI ([GABRIEL et al., 2004](#)), que apresenta uma arquitetura abrangente, composta por um conjunto diversificado de funcionalidades. Essas funcionalidades incluem recursos como controle de processos, gerenciamento e monitoramento de latência, tolerância a falhas e otimização de operações para padrões comuns de comunicação. Desse modo, por meio desse conjunto de utilitários, os *clusters* de máquinas podem aprimorar suas capacidades de processamento, por meio da execução distribuída e paralela.

Para fins de experimentação e exploração, embora as tecnologias ofereçam capacidade de processamento distribuído e paralelo escalável, elas foram apenas instaladas no *cluster*. Portanto, não serão utilizadas na aplicação final, uma vez que não se alinham com o objetivo principal de construir uma biblioteca digital, considerando que tecnologias como o Apache Spark estão mais direcionadas ao processamento de *streaming* de dados.

⁵ Mais informações sobre as tecnologias em: [<https://jupyter.org/>](https://jupyter.org/)

⁶ Integração do Spark com Python: [<https://spark.apache.org/docs/latest/api/python/>](https://spark.apache.org/docs/latest/api/python/)

No entanto, vale ressaltar a importância da instalação do MPI no *cluster*, uma vez que ela é utilizada pela ferramenta de testes Linpack para realizar suas avaliações.

3.3 Contêineres

Conforme mencionado previamente, a virtualização trouxe uma série de vantagens para o desenvolvimento de aplicações em ambientes *clusterizados*. Um desses benefícios é a utilização eficiente dos recursos, que anteriormente não eram totalmente explorados nos ambientes em *bare-metal*. Além disso, a virtualização permitiu a execução de diferentes sistemas, possibilitando o compartilhamento dos mesmos recursos de máquina. Nesse contexto, além dos hipervisores (máquinas virtuais), outra tecnologia que se destaca pela sua capacidade de isolamento dos recursos são os contêineres, onde em ambas ferramentas é permitido o isolamento dos sistemas e a multilocação de *software* (SILVA; KIRIKOVA; ALKSNIS, 2018).

Ainda segundo Silva, Kirikova e Alksnis (2018), os contêineres, assim como os hipervisores, são uma camada de abstração que empacota, código e suas dependências. Dessa forma, o contêiner fornece uma forma de independência dos recursos no nível do sistema operacional, semelhante à virtualização por hipervisores do tipo 2, em que recursos do kernel são distribuídos para processos isolados. Além disso, assim como as máquinas virtuais, os contêineres conseguem limitar recursos, como CPU, memória, disco e conexões de rede, isolando as aplicações de acordo com suas necessidades.

No entanto, conforme evidenciado pelo estudo comparativo realizado por Zhang et al. (2018), os contêineres têm se tornado consideravelmente mais populares devido a sua conveniência em encapsular, implantar e isolar aplicações de forma mais eficiente, leve e flexível, utilizando os recursos disponíveis nas máquinas. Além disso, ainda de acordo com Zhang et al. (2018), outro fator que impulsiona esse crescimento é o desempenho superior dos sistemas executados em contêineres em comparação com as máquinas virtuais, devido à camada responsável pela virtualização em ambientes de contêineres serem significativamente mais leves e eficientes se comparados com a virtualização tradicional.

3.4 Orquestradores

Similar às máquinas virtuais que possuem ferramentas para auxiliar na organização e gerenciamento de ambientes, os contêineres também evoluíram, incorporando tecnologias capazes de orquestrar e escalar, sendo denominadas como orquestradores. Sistemas como Apache Mesos, Docker Swarm e Kubernetes têm ganhado popularidade crescente na orquestração de aplicações, em consonância com o aumento do uso de contêineres. Dessa forma, o emprego dessas tecnologias tem se tornado essencial para a automação e

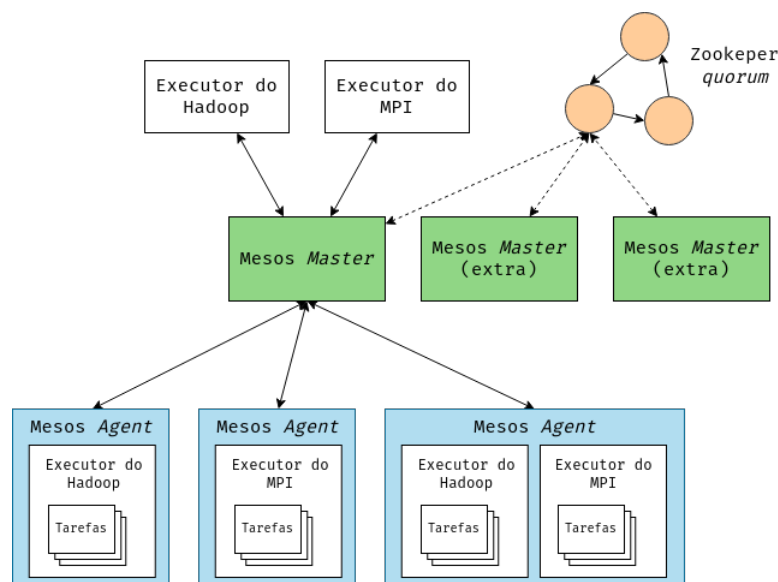
implantação eficiente, proporcionando redução de custos, escalabilidade e principalmente elasticidade para os *softwares*.

De acordo com [Casalicchio e Iannucci \(2020\)](#), os orquestradores de contêineres permitem que os desenvolvedores definam de forma dinâmica as configurações de construção, implantação, execução e monitoramento de sistemas multi-contêineres. O autor destaca que essas ferramentas devem ser capazes de controlar o consumo de recursos, como o número de CPUs e a quantidade de memória, agendar tarefas, equilibrar a carga de trabalho e escalar automaticamente a quantidade de contêineres segundo a demanda. Além disso, é necessário que os orquestradores realizem verificações contínuas sobre o estado dos ambientes e sejam tolerantes a falhas.

3.4.1 Apache Mesos

O Apache Mesos é um orquestrador projetado especificamente para o gerenciamento de contêineres em ambientes *clusterizados*. [Hindman et al. \(2011\)](#) enfatizam que o principal objetivo do Mesos é fornecer uma abordagem para a construção de sistemas escaláveis, eficientes e resilientes a falhas. Além disso, o Apache Mesos se destaca por sua versatilidade ao distribuir diversas tarefas, utilizando múltiplos algoritmos de agendamento de processos, em diferentes computadores presentes no *cluster*, garantindo o isolamento necessário para as aplicações.

Figura 7 – Arquitetura simplificada do Mesos



Fonte: [Hindman et al. \(2011\)](#), tradução nossa)

A arquitetura do Apache Mesos é baseada no conceito de mestre-agente (*master-agent*), como pode ser visto na representação simplificada da arquitetura na Figura 7, em que o mestre gerencia os processos agentes que são executados de forma independente

em cada nó do *cluster*. O mestre também armazena uma lista de recursos disponíveis e é responsável por distribuí-los entre os algoritmos de agendamento, segundo a política de divisão selecionada. Além dessas duas entidades, o Mesos apresenta outros dois conceitos fundamentais para a distribuição de tarefas: o escalonador (*scheduler*) e o executor (*executor*) (HINDMAN et al., 2011).

O escalonador, conforme descrito por Hindman et al. (2011), é registrado no mestre e é responsável por distribuir os recursos conforme o algoritmo de agendamento definido. Por outro lado, os executores são instanciados nos nós agentes, com a responsabilidade de processar as tarefas atribuídas a eles, utilizando os recursos alocados. Portanto, devido à importância do mestre na execução e no gerenciamento dos recursos, a tolerância a falhas se torna fundamental. Essa característica é alcançada por meio da utilização do ZooKeeper⁷ para gerenciar múltiplos mestres e garantir alta disponibilidade do ambiente.

3.4.2 Docker Swarm

Assim como o Apache Mesos, o Docker Swarm é um orquestrador projetado para fornecer ambientes de containerização altamente disponíveis, escaláveis e de fácil manutenção. Sua arquitetura é baseada no conceito de nós gerenciadores (*manager nodes*) e nós trabalhadores (*worker nodes*), que colaboram entre si para executar contêineres Docker de forma eficiente e coordenada, provendo uma solução robusta para as aplicações.

Os nós gerenciadores possuem duas funções principais: manter o estado atual do *cluster* e agendar serviços. Para garantir a tolerância a falhas, o Docker Swarm utiliza a implementação do Raft (ONGARO; OUSTERHOUT, 2014 apud DOCKER, 2023), um algoritmo de consenso, permitindo que múltiplos nós gerenciadores possam atuar coordenadamente evitando falhas catastróficas em caso de falhas individuais. Por sua vez, os nós trabalhadores têm como único propósito executar os contêineres que lhes foram orquestrados, promovidos a nós gerenciadores, caso necessário, por meio de comandos específicos (DOCKER, 2023).

Ainda segundo Docker (2023), assim como o Apache Mesos, o Docker Swarm incorpora os conceitos de tarefas, escalonador e executor. O escalonador tem a função de enviar instruções aos nós trabalhadores, enquanto os executores efetivam as tarefas atribuídas. No contexto do Docker Swarm, as tarefas representam os passos necessários para a implantação dos contêineres Docker nos nós trabalhadores. Esses contêineres, ou serviços, podem então ser replicados tanto na mesma máquina quanto em outras máquinas do *cluster*, possibilitando uma distribuição mais eficiente da carga nas aplicações.

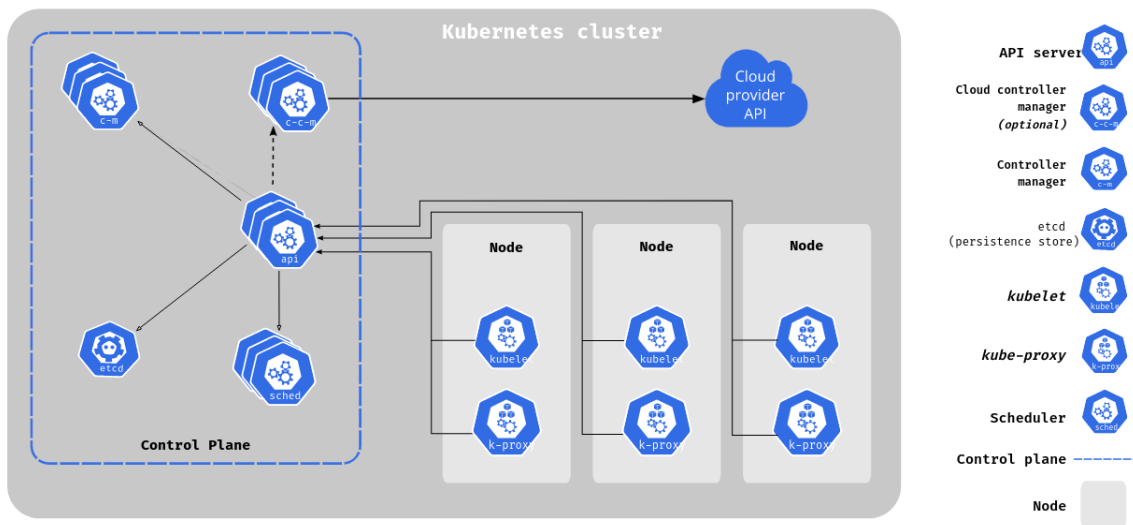
⁷ Mais informações sobre a tecnologia podem ser encontradas em: <<https://zookeeper.apache.org/>>

3.4.3 Kubernetes

Por último, entre os orquestradores selecionados, destaca-se o Kubernetes, que representa uma evolução do Borg (VERMA et al., 2015). Assim como as outras tecnologias mencionadas anteriormente, o Kubernetes concentra-se em fornecer ambientes distribuídos de alto desempenho, escalabilidade e capacidade de atualização contínua. Com o intuito de facilitar e otimizar a implantação de aplicações, o Kubernetes emprega conceitos de imutabilidade, configurações declarativas e ambientes autocorretivos, resultando em uma alta disponibilidade e tolerância a falhas (BURNS; BEDA; HIGHTOWER, 2019).

A imutabilidade é alcançada no Kubernetes ao impedir alterações nos contêineres em execução, exigindo a reconstrução dos mesmos para efetivar as mudanças. Além disso, ao utilizar configurações declarativas que descrevem os estados desejados, o Kubernetes mantém um histórico preciso das alterações, permitindo a fácil reversão a estados anteriores. Segundo Burns, Beda e Hightower (2019), a arquitetura desacoplada do Kubernetes é outro aspecto crucial, por possibilitar uma escalabilidade mais precisa, permitindo que os microsserviços sejam escalados individualmente sem afetar todo o sistema.

Figura 8 – Arquitetura Kubernetes



Fonte: Kubernetes (2023)

Para assegurar todas essas características e funcionalidades, o Kubernetes adota uma arquitetura complexa, ilustrada na Figura 8. Baseada nos conceitos de plano de controle (*control plane*) e nós trabalhadores (*worker nodes*), a arquitetura do Kubernetes envolve seis componentes principais: o servidor API (*API server*), o etcd, o escalonador (*scheduler*), o gerenciador de controladores (*controller-manager*), o kubelet e o kube-proxy (KUBERNETES, 2023).

- Componentes executados no plano de controle

- *API server*: Componente responsável por expor a API do Kubernetes, ou seja, é a porta de entrada para a comunicação entre os elementos presentes no plano de controle. Por ser essencial no fluxo de mensagens, pode ser escalado horizontalmente entre múltiplos nós do *cluster*.
 - *Etcd*: Banco de dados de chave-valor usado para sincronizar estados e armazenar informações do sistema.
 - *Scheduler*: Componente que observa novos grupos de contêineres (Pods) e os designa para um nó trabalhador para serem executados. Considera diversas informações presentes no *cluster* para realizar a distribuição dos pods.
 - *Controller-manager*: Elemento responsável por executar processos controladores que monitoram estados específicos dos componentes do Kubernetes. Em particular, os controladores de nós tem a função de detectar e responder quando um nó fica inativo, enquanto os controladores de tarefas observam as tarefas e criam os pods conforme especificado.
- Componentes executados nos nós trabalhadores
 - *Kubelet*: Agente que roda em cada nó trabalhador do *cluster*, recebendo e certificando que cada pod orquestrado pelo escalonador está executando em um contêiner.
 - *Kube-proxy*: Componente responsável por manter e gerenciar as regras de rede em cada nó trabalhador.

3.5 Experimentos

Portanto, para fins de experimentação desta monografia, ambas as infraestruturas foram utilizadas em três ambientes distintos. O primeiro ambiente consiste no servidor *bare-metal* Chococino⁸, utilizado para a experimentação de tecnologias distribuídas. Em seguida, foi empregado um ambiente em nuvem de máquinas virtuais no servidor da Azure⁹, direcionado à exploração de orquestradores. Por fim, foi implementada a infraestrutura definitiva em *bare-metal*, que servirá como base para a biblioteca virtual a ser desenvolvida na segunda etapa deste trabalho. As configurações das máquinas são as seguintes:

- Infraestrutura da Chococino
 - Máquinas cm1, cm2, cm3 e cm4: CPU i7-8700 com 4 cores e 16 GB de memória

⁸ O *cluster* Chococino, idealizado por Bruno C. Ribas, foi criado com o intuito de fornecer um ambiente com vários computadores para os alunos de Engenharia de Software da FGA-UnB para a realização de experimentos.

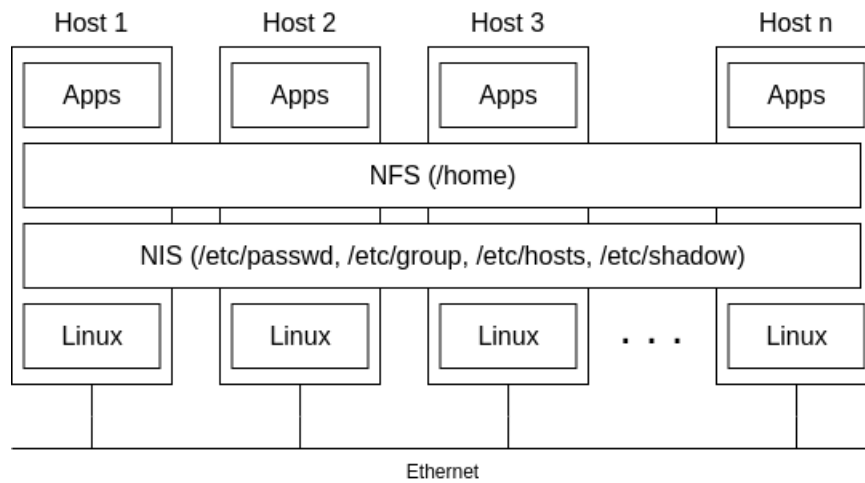
⁹ Mais informações em: <<https://azure.microsoft.com/pt-br/>>

- Máquinas pos1 e pos2: CPU i7-9700 com 8 cores e 16 GB de memória
- Máquinas gpu1, gpu2 e gpu3: CPU Ryzen 7 2700 com 16 cores, 32 GB de memória e placa gráfica GTX1650
- Infraestrutura da Azure
 - 10 máquinas: 2 CPUs virtuais e 4 GB de memória
- Infraestrutura construída para a monografia
 - Máquina Zeus: CPU Xeon X7460 com 24 cores, 125 GB de memória e cerca de 572 GB de armazenamento, disponível para uso;
 - Máquina Atenas: CPU Xeon E5620 com 16 cores, 23 GB de memória e cerca de 98 GB de armazenamento, disponível para uso;
 - Máquina Hera: CPU Xeon E5504 com 8 cores, 11 GB de memória e cerca de 815 GB de armazenamento, disponível para uso;
 - Máquina Apolo: CPU Xeon X7460 com 24 cores, 125 GB de memória e cerca de 480 GB de armazenamento, indisponível para uso devido a problemas de resfriamento, e
 - Máquina Medusa: CPU Xeon E5620 com 16 cores, 23 GB de memória e cerca de 98 GB de armazenamento, indisponível para uso devido a problemas com a controladora de discos.

Além disso, a criação de um ambiente de *cluster* que se assemelha a um único sistema é imprescindível para a facilitação e integração dos módulos do sistema. Aspectos como autenticação e armazenamento foram unificados, no contexto desse trabalho, permitindo o usuário navegar com uma única credencial e acessar o armazenamento compartilhado entre as máquinas. A representação da base do *cluster* utilizado nos experimentos de sistema de recomendação e fornecimento de mídia é mostrada na Figura 9.

A unificação, conforme a Figura 9, acontece em duas camadas, composta pelo NIS e o NFS. O NIS tem a responsabilidade de compartilhar as informações do usuário pela rede, permitindo que todas as máquinas possuam os mesmos usuários, grupos, senhas e *hosts*. Já o NFS tem a responsabilidade de compartilhar os arquivos pela rede, sendo utilizado para compartilhar a pasta “/home”, que contém as pastas dos usuários. Desta forma, as informações e dados dos usuários são compartilhados entre as máquinas.

Para avaliar o desempenho das máquinas presentes na infraestrutura final, após a sua implantação, utilizou-se o Linpack para testar a carga do sistema. Devido à incompletude da implantação, foi realizada uma avaliação parcial do desempenho, analisando individualmente cada nó. Para conduzir a análise, foi necessário configurar o tamanho do

Figura 9 – Representação do *cluster*

Fonte: Adaptado do *cluster* Chococino

problema executado pelo Linpack para obter dados o mais fidedignos possível, segundo as capacidades do nó.

Tabela 1 – Configuração e resultados obtidos pelo Linpack

Máquina	Tamanho do problema	do	<i>Grid</i> utilizado	Tempo de execução (s)	Gflop/s
Zeus	102144		4x5	9564.36	7.4285e+01
Atenas	28800		2x7	364.61	4.3681e+01
Hera	24960		2x2	474.79	2.1836e+01

Fonte: Autores

A Tabela 1 apresenta as principais configurações utilizadas para o tamanho do problema e a *grid* de processos em cada máquina. Essas configurações foram calculadas¹⁰ para deixar no máximo 4 núcleos e 10 GB de memória livre, a fim de estressar ao máximo cada nó. Na tabela, também são registrados o tempo, em segundos, necessário para a conclusão do teste, bem como a taxa de execução, em *gigaflop* por segundo, para resolver o problema.

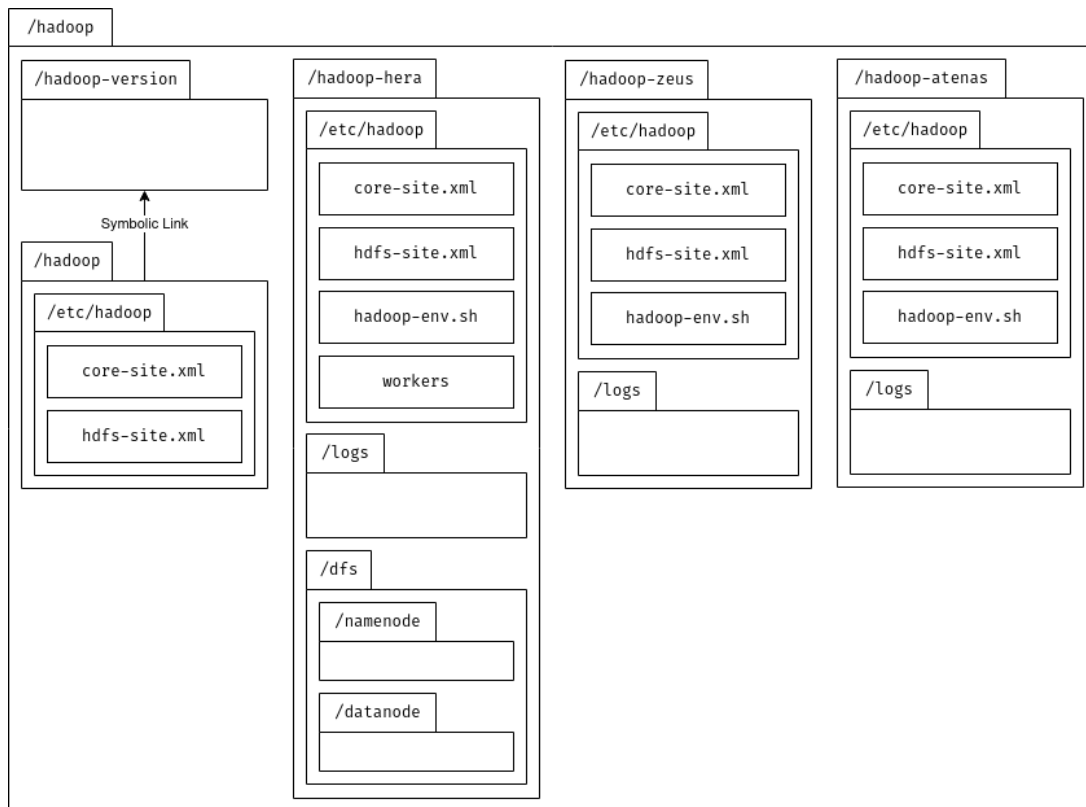
A partir desses resultados, pôde-se observar que o nó Zeus apresenta a maior capacidade de processamento, seguido por Atenas e, por fim, Hera. Isso indica que para tecnologias e ferramentas que demandam um uso intensivo da CPU, o desempenho será melhor se executadas na máquina Zeus em comparação com as outras.

Com o intuito de disponibilizar a camada distribuída para a construção da biblioteca digital, foram instaladas no cluster as tecnologias Apache Hadoop, Apache Spark e Jupyterhub. Em particular, o Apache Hadoop, com foco no HDFS, foi configurado para

¹⁰ Calculadora utilizada está disponível em: https://www.advancedclustering.com/act_kb/tune-hpl-dat-file/

ser distribuído entre todas as máquinas disponíveis no *cluster*, resultando em um sistema de arquivos distribuído com uma capacidade de armazenamento de aproximadamente 1,45 TB. De maneira semelhante, o Apache Spark também foi distribuído em todos os nós, alocando um total de 18 núcleos de CPU e 63 GB de memória para o processamento dos dados.

Figura 10 – Representação da estrutura de pastas do Apache Hadoop instalado no *cluster*



Fonte: Autores

Como pode ser observado na Figura 10, que ilustra a organização do Hadoop no *cluster*, é evidente a presença dos seguintes diretórios: um *link* simbólico para a pasta que contém a versão instalada no sistema, denominada “/hadoop”, a pasta do *namenode*, que armazena os *logs*, suas configurações específicas e os metadados do *cluster*, e as pastas dos *datanodes*, que armazenam seus próprios logs e configurações.

Para configurar o funcionamento do *namenode* e dos *datanodes*, foram utilizados os arquivos “hdfs-site.xml” de cada nó presente no *cluster*, conforme exemplificado nas Figuras 11. Nesses arquivos, foram definidas as localizações de armazenamento dos arquivos (linhas 7 e 8), devidamente particionados para cada máquina, assim como a estratégia de replicação entre os nós disponíveis (linhas 11 e 12). Além dessas configurações, também foram realizados ajustes em portas, URLs de conexão e variáveis de ambiente, a fim de assegurar a correta configuração do ambiente.

Assim como o Hadoop, o Apache Spark também segue uma estrutura de pastas

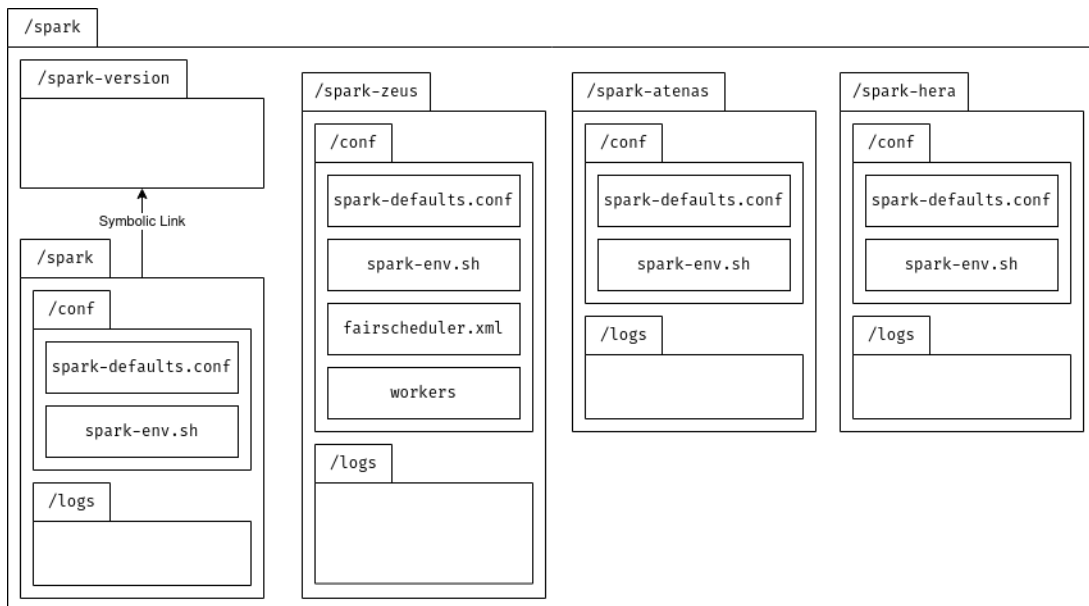
Figura 11 – Configuração simplificada do Apache *Datanode* na máquina Zeus

```

1 <configuration>
2   <property>
3     <name>dfs.secondary.http.address</name>
4     <value>zeus:9868</value>
5   </property>
6   <property>
7     <name>dfs.datanode.data.dir</name>
8     <value>/home/hadoop/.dfs/datanode</value>
9   </property>
10  <property>
11    <name>dfs.replication</name>
12    <value>3</value>
13  </property>
14 </configuration>

```

Fonte: Autores

Figura 12 – Representação da estrutura de pastas do Apache Spark instalado no *cluster*

Fonte: Autores

semelhante, conforme ilustrado na Figura 12. De forma análoga, é criado um *link* simbólico para a versão instalada no *cluster* e as pastas são organizadas conforme o papel de *Master* ou *Worker*. No diretório da máquina *Master* são armazenadas as configurações do escalonador, as variáveis de ambiente e os nós *Workers* correspondentes. Por sua vez, nos diretórios dos *Workers*, são mantidas apenas as configurações e as variáveis de ambiente específicas de cada nó.

Ainda similarmente ao Apache Hadoop, o Spark também segue padrões de configuração, definidas nos arquivos “`spark-defaults.conf`”. Na Figura 13, é apresentada a

Figura 13 – Configuração do Spark *Worker* na máquina Atenas

```

1 spark.dynamicAllocation.enabled      true
2 spark.shuffle.service.enabled        true
3 spark.ui.reverseProxy                true
4 spark.ui.proxyBase                   /spark
5 spark.ui.reverseProxyUrl              http://zeus:8310/spark

```

Fonte: Autores

configuração de um nó *Worker* do *cluster*, onde são definidas as configurações de *proxy* para conexão e acesso (linhas 3 a 5), além da participação do *Worker* no processo de alocação dinâmica de recursos (linhas 1 e 2).

Figura 14 – Configuração do escalonador do Spark *Master* na máquina Zeus

```

1 <allocations>
2   <pool name="production">
3     <schedulingMode>FAIR</schedulingMode>
4     <weight>1</weight>
5   </pool>
6 </allocations>

```

Fonte: Autores

Por sua vez, na Figura 14, é apresentada a configuração do arquivo “fairscheduler.xml”, responsável pelo controle do escalonador de recursos utilizando o modelo *FAIR* (linha 3). Nesse modelo, a *pool* de recursos denominada “production” (linha 2) divide os recursos de forma mais equitativa possível entre as tarefas. Para o caso de uso da biblioteca digital, esse modelo de divisão se mostra adequado para atender às necessidades.

Figura 15 – Configuração do Idle-culler no Jupyterhub

```

1 c.JupyterHub.services = [
2   {
3     "name": "jupyterhub-idle-culler-service",
4     "command": [
5       sys.executable,
6       "-m", "jupyterhub_idle_culler",
7       "--timeout=7200",
8     ],
9   }
10 ]

```

Fonte: Autores

Por fim, para completar toda a camada de desenvolvimento e processamento distribuído do sistema, foi realizada a instalação do Jupyterhub. A escolha dessa ferramenta baseou-se na facilidade de instanciar *notebooks* sem a necessidade de executar comandos de linha. Além disso, a fim de evitar o desperdício desnecessário dos recursos de *hardware* dos nós, foi configurado o Idle-culler (linhas 3 a 7) para encerrar os *notebooks* que permanecerem inativos por duas horas (linha 7), conforme demonstrado na Figura 15.

Já com o intuito de compreender e monitorar o estado do cluster de máquinas, a ferramenta selecionada para essa finalidade foi o Grafana. Caracterizada por uma interface moderna, o Grafana consegue apresentar gráficos de séries temporais, linhas do tempo, histórico de estados, mapas de calor e outras representações visuais. Para coletar as métricas necessárias, foi utilizado o Prometheus, que foi instalado em cada nó do conjunto de máquinas.

Figura 16 – Tela das métricas coletadas em um nó



Fonte: Autores

Na fase de prototipação, foram selecionadas apenas algumas métricas relevantes, como ilustrado na Figura 16. Essas métricas incluem informações sobre o uso da CPU, memória, leitura e escrita do disco, além do número de pacotes enviados e recebidos pelas interfaces de rede. É esperado que novas métricas sejam adicionadas posteriormente, a fim de obter uma visão mais precisa não apenas do estado do cluster, mas também das ferramentas em execução.

Portanto, com todo esse arcabouço de tecnologias instaladas na infraestrutura construída, se torna possível ofertar um ambiente que atenda às demandas de uma aplicação como uma biblioteca digital. Além disso, assegura-se a confiabilidade, escalabilidade, elasticidade e resiliência da aplicação desenvolvida sobre essa arquitetura.

4 Requisitos específicos da aplicação

Esse capítulo aborda os requisitos específicos da aplicação. Primeiramente, a oferta de *streaming* é abordada, explorando a arquitetura, protocolos de comunicação, controle de congestionamento, testes de carga e experimentos de desempenho e elasticidade em ambientes de cluster. A seguir, os serviços de recomendação são investigados, com ênfase na filtragem colaborativa e na filtragem baseada no conteúdo. Por fim, é discutido, a partir dos experimentos, as melhores opções para atender a biblioteca digital.

4.1 Oferta de *streaming*

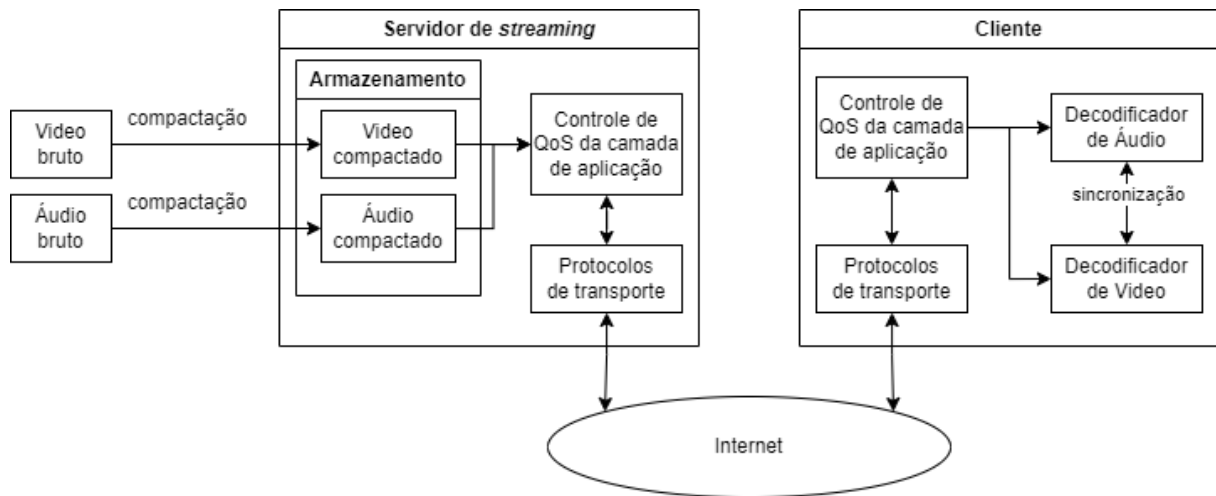
Fornecer um meio que permita a reprodução é um elemento necessário para a preservação, recurso esse que por muito tempo não estava disponível. Segundo [Tanenbaum \(2010\)](#), o fornecimento de áudio e vídeo é o Santo Graal da rede mundial de computadores, uma vez que a popularização da Internet, com velocidade de transmissão nas ordens de gigabits por segundo, e de computadores com alto poder de processamento ainda é recente.

[Tanenbaum \(2010\)](#) apresenta duas formas de fornecimento de mídia, sendo a mídia armazenada, que já foi gravada e contida em algum arquivo, e ao vivo, que está sendo criada no momento em que está sendo reproduzida. Como o objetivo dessa monografia é sobre a conservação da música brasileira, então o fornecimento de uma música já gravada e armazenada será mais explorado.

De acordo com [Tanenbaum \(2010\)](#), a oferta de mídia pode acontecer de duas formas. Na primeira, todos os dados da mídia são enviados para o cliente e só no final acontece a reprodução, isso pode acarretar aumento do armazenamento e demora no início da reprodução. Na segunda, a mídia é fracionada em várias partes e enviada para o usuário sob demanda, diminuindo o tempo de início de reprodução e uso de armazenamento, mas pode sofrer com travamentos devido à lentidão e instabilidades da conexão.

Um serviço de mídia sob demanda, também chamado de serviço de *media streaming*, permite a transmissão e reprodução a partir do particionando da mídia em vários blocos, visando oferecer um início imediato e reprodução contínua, minimizando os impactos da instabilidade da rede. Deste modo, para uma melhor experiência de usuário, questões como compactação e descompressão dos dados, problemas de transmissão, atrasos e navegação são transparentes ao usuário ([WU et al., 2001](#)). A arquitetura de um servidor de *streaming* é mostrada na Figura 17.

Conforme representado na Figura 17, o sistema possui a arquitetura cliente-servidor, onde o servidor contém acesso às mídias armazenadas e oferece para o cliente por meio

Figura 17 – Arquitetura para o servidor de *streaming*

Fonte: Adaptado de Wu et al. (2001)

da Internet. Uma vez que é um sistema I/O Bound, onde a saída de dados pela rede é intensa, o armazenamento e a forma de disponibilização eficiente é imprescindível para uma experiência de reprodução contínua. Desta forma, Wu et al. (2001) detalha os módulos da arquitetura a seguir:

- Compactação: A mídia deve ser comprimida anteriormente para uma transmissão eficiente e de menor tamanho.
- Protocolos de comunicação: A comunicação entre o cliente e o servidor deve estar em acordo. Protocolos de transporte como TCP, UDP e RTP podem ser utilizados, além do RTSP e HTTP na camada de aplicação.
- Controle de Quality of Service (QoS) da camada de aplicação: Para lidar com a instabilidade na rede, técnicas, como controle de congestionamento, controle de erros, retransmissão, codificação resiliente e ocultação de erros - alguns desses métodos já se encontram inclusos no protocolo TCP - devem ser levadas em consideração.

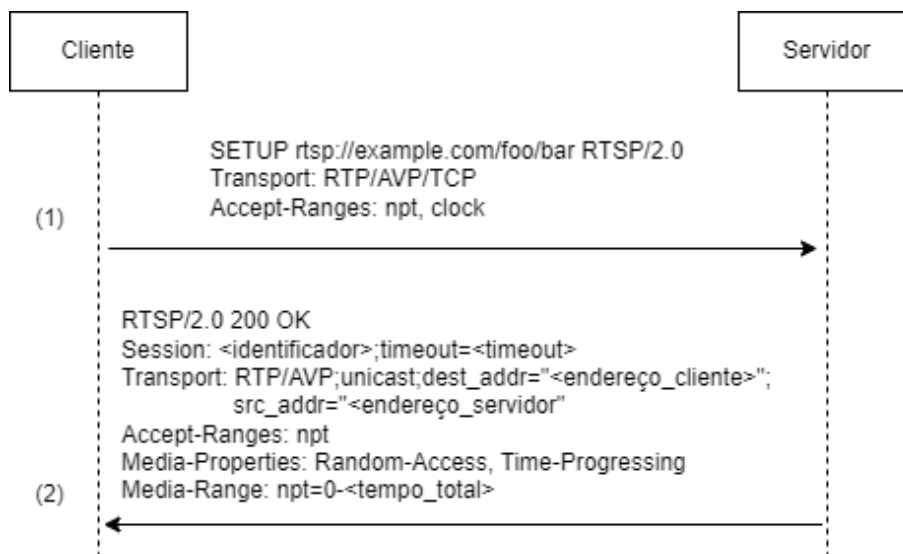
O fornecimento da mídia acontece por meio do modelo cliente-servidor, onde o cliente inicia a requisição. Essa interação exige a necessidade de protocolos de comunicação que atendam as operações de um *streaming server*, como pausar, continuar, prosseguir e voltar a reprodução (WU et al., 2001). As seções a seguir abordam o Real-Time Streaming Protocol (RTSP) e o Hypertext Transfer Protocol (HTTP) como possibilidade de protocolo de comunicação entre o servidor e o cliente.

4.1.1 RTSP

O Real-Time Streaming Protocol (RTSP) é um protocolo da camada de aplicação que controla o fornecimento de mídia, seja em tempo real ou sob demanda. O RTSP é um protocolo bidirecional, composto por requisição e resposta, onde, inicialmente, é estabelecido um contexto com os metadados da mídia requisitada e, a partir disso, o servidor controla o fornecimento enviando os dados para o cliente. Ademais, é composto por três partes principais: o estabelecimento de sessão; entrega de dados; e extensões. (SCHULZRINNE et al., 2016)

O estabelecimento de sessão acontece por meio da ação do cliente para o servidor, utilizando o método SETUP do protocolo. A requisição é formada pela localização da mídia, juntamente com a unidade de tempo e informações de transporte, como porta de retorno e protocolo de transporte. Já a resposta é composta pelo *status code*, código da sessão, informações de transporte, unidade de tempo e a natureza da mídia. Um exemplo do método SETUP é apresentado na Figura 18. (SCHULZRINNE et al., 2016)

Figura 18 – Exemplo do método RTSP SETUP

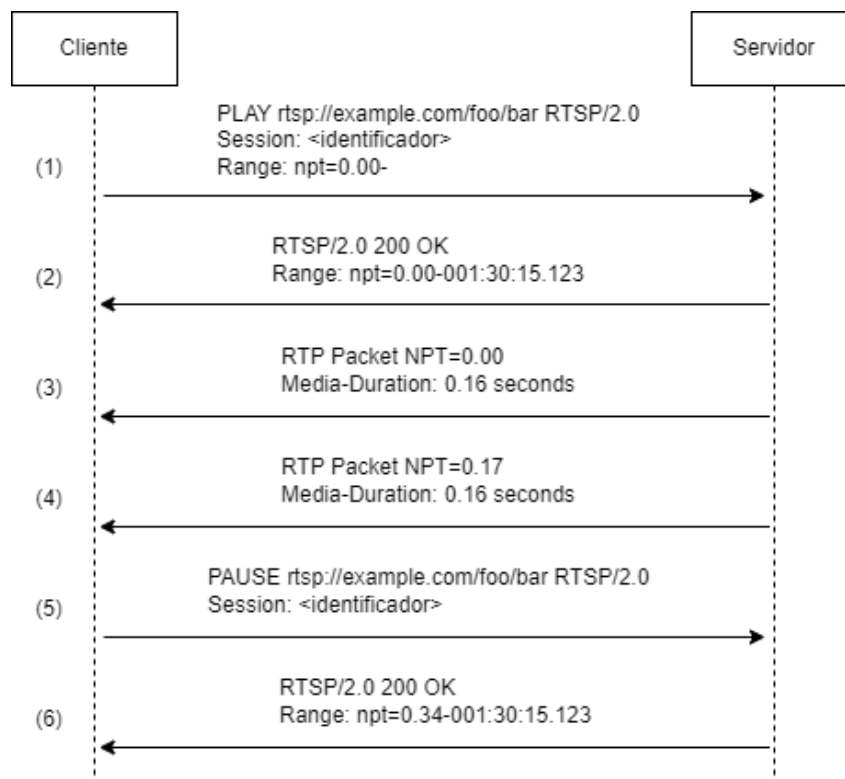


Fonte: Schulzrinne et al. (2016)

Como é possível observar na Figura 18, em (1), o cliente inicia a sessão RTSP com o método SETUP, onde a mídia solicitada se encontra em "rtsp://example.com/foo/bar", aceitando os protocolos de transporte Real-time Transport Protocol (RTP), Audio-Video Protocol (AVP) ou TCP. Já em (2), como resposta do servidor, é retornado o *status code* 200, significando que a mídia está disponível, o código da sessão, *timeout*, informações de transporte, unidade de tempo "npt", a natureza da mídia, o "Random-Access" informa ao cliente que há a possibilidade de acessar qualquer intervalo de tempo, e, por fim, o intervalo com duração total.

Com a sessão estabelecida, a conexão é mantida e o cliente pode realizar operações de reprodução, como iniciar, pausar, avançar e voltar. Isso acontece por meio dos métodos RTSP PLAY e PAUSE. No método PLAY, o cliente envia o intervalo de tempo que deseja reproduzir, e o servidor, como resposta, devolve uma confirmação com o intervalo de tempo e, periodicamente, conforme a demanda, os pacotes contendo os bytes do intervalo da mídia requisitada. Já o método PAUSE para imediatamente o fluxo de envio dos dados. O método PLAY e o PAUSE são exemplificados na Figura 19. (SCHULZRINNE et al., 2016)

Figura 19 – Exemplo do método RTSP PLAY e PAUSE.



Fonte: Adaptado de Schulzrinne et al. (2016)

Conforme observado na Figura 19, em (1), o cliente envia uma requisição com o método PLAY, passando o intervalo de tempo que será reproduzido no atributo "Range". O servidor responde, em (2), com a confirmação da requisição e, imediatamente, em (3) e (4), são enviados os dados da mídia. Para interromper o fluxo de dados, o cliente utiliza o método PAUSE, passando a sessão e o recurso que será pausado.

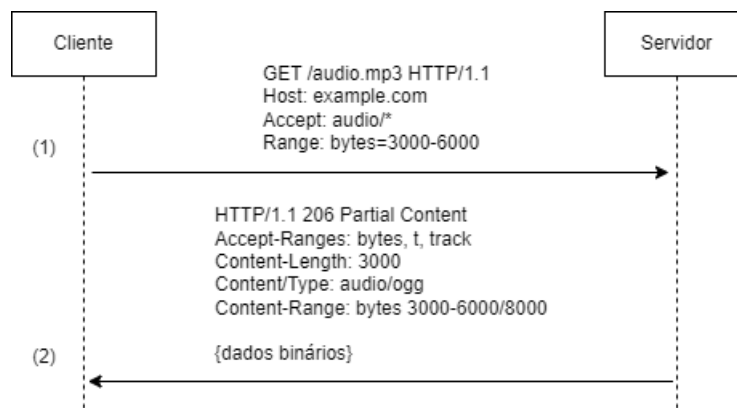
Esse protocolo, por fornecer a mídia por uma única conexão, acaba tendo um baixo *overhead* em seus pacotes. No entanto, em uma implantação em alta escala, por precisar gerenciar a sessão e conexão de cada usuário, a necessidade por *hardware* é amplificada. Além disso, muitas das CDNs, responsáveis por distribuir a mídia em locais próximos aos usuários, e *firewalls* não fornecem suporte e permissão ao RTP e RTSP. (SODAGAR, 2011)

4.1.2 HTTP

Outro protocolo bastante utilizado para fornecimento de mídia é o Hypertext Transfer Protocol (HTTP), uma vez que, segundo [Sodagar \(2011\)](#), a infraestrutura de internet mundial tem evoluído na ampliação de seu suporte, principalmente, em *firewalls*, CDNs e navegadores.

Para isso acontecer por meio do HTTP, é necessário de um módulo que converta o intervalo de tempo em bytes. Assim, foi criada a especificação Media Fragments URI 1.0 ([HAUSENBLAS et al., 2012](#)), que padroniza, tanto do lado do cliente, como do servidor, as formas de endereçar a mídia na *web*, seja de modo temporal, espacial ou *track*. Desta forma, a Figura 20 apresenta um exemplo de requisição de mídia por meio do HTTP.

Figura 20 – Exemplo de requisição de mídia com HTTP



Fonte: Adaptado de [Lancker et al. \(2011\)](#)

A requisição, em (1), é iniciada pelo cliente, passando o caminho para a mídia e, no cabeçalho, o atributo "Range" com o intervalo de tempo, juntamente com o tipo de resposta aceito. Já a resposta do servidor, em (2), retorna o *status code* 206 Partial Content, mais os tipos aceitos de endereçamento do intervalo de tempo, o tamanho do bloco, o tipo da mídia, o intervalo e o tamanho total e, por fim, no corpo da resposta, os bytes do intervalo solicitado.

Devido à versatilidade do HTTP, surgiram diversas extensões, como o Dynamic Adaptive Streaming over HTTP (DASH) ([SODAGAR, 2011](#)) e o HTTP Live Streaming (HLS) ([PANTOS; MAY, 2017](#)), que possuem a capacidade de ajustar a qualidade da transmissão e a taxa de bytes durante a transmissão, maximizando a reprodução contínua mesmo com instabilidades de rede.

4.1.3 Algoritmos de congestionamento

Adicionalmente, é crucial mencionar outra etapa significativa na comunicação entre dispositivos por meio do protocolo TCP, que diz respeito aos algoritmos de conges-

tionamento empregados. Esses algoritmos têm a responsabilidade de operar na janela de congestionamento, ajustando a quantidade de pacotes transmitidos durante uma comunicação, a fim de otimizar o desempenho ao garantir a utilização máxima do tráfego disponível. Nesse contexto, destacam-se dois algoritmos principais, o CUBIC (HA; RHEE; XU, 2008) e o Bottleneck Bandwidth and Round-trip propagation time (BBR) (CARDWELL et al., 2016).

O CUBIC, segundo Ha, Rhee e Xu (2008), foi desenvolvido como uma evolução do algoritmo de congestionamento conhecido como BIC-TCP. Construído em cima do protocolo TCP, o CUBIC atua quando existe a perda de um pacote durante a comunicação, ajustando sua janela de congestionamento conforme a fórmula cúbica mostrada na seguinte equação.

$$W(t) = C(t - K)^3 + W_{max} \quad (4.1)$$

Onde C é a constante de escalonamento, t é o tempo entre a última redução de janela, W_{max} é a maior janela possível onde ocorreu a perda de pacote e K é o tempo necessário para a função sair da menor janela W para a maior W_{max} , calculada pela seguinte equação.

$$K = \sqrt[3]{\frac{W_{max}\beta}{C}} \quad (4.2)$$

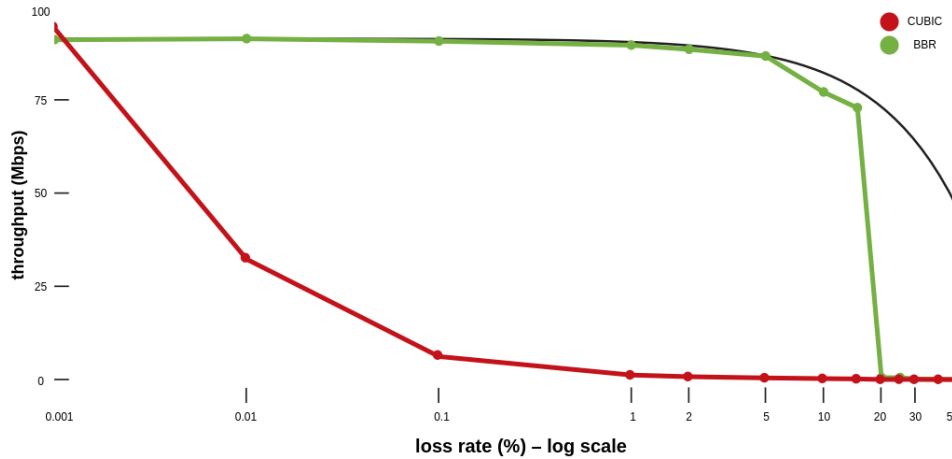
Em que β representa o fator constante de decréscimo entre as janelas. Ha, Rhee e Xu (2008) propõem, portanto, um algoritmo de congestionamento eficiente e rápido para a comunicação utilizando o protocolo TCP.

Com o avanço das tecnologias, a relação entre perda de pacotes e o conceito de congestionamento, que serviram como base para o desenvolvimento do algoritmo CUBIC, perderam sua relevância, e o impacto positivo passou a representar um problema nas comunicações. Uma das implicações dessa situação é que a utilização do CUBIC em conexões com grandes *buffers* resulta em superlotação, ocasionando altos níveis de latência e gargalos na comunicação. Por outro lado, a aplicação do algoritmo em conexões com *buffers* reduzidos também gera efeitos adversos, uma vez que a perda de um único pacote é interpretada como congestionamento pelo algoritmo, afetando a taxa de transferência (CARDWELL et al., 2016).

Para solucionar essas questões, uma abordagem alternativa foi apresentada por Cardwell et al. (2016) por meio do algoritmo de congestionamento BBR. Nesse modelo, medidas como o tempo de propagação e a largura de banda do gargalo são utilizadas como indicadores de congestionamento, resultando em detecções mais eficientes e comunicações

mais otimizadas, como demonstrado na Figura 21, onde o eixo horizontal representa a taxa de perda de pacotes em escala logarítmica e o eixo vertical o *throughput* em Mbps.

Figura 21 – Comparação da taxa de transferência de pacotes entre CUBIC e BBR



Fonte: Cardwell et al. (2016)

Na Figura 21 é possível observar que rapidamente o *throughput* do CUBIC reduz, em comparação com o BBR. O BBR consegue se manter estável até que a taxa de pacotes se aproxima de 20% e após isso se assemelha ao CUBIC. Segundo Cardwell et al. (2016), isso acontece devido a não utilização da taxa de perda como indicador de congestionamento, permitindo o *throughput* se manter próximo ao limite do *buffer*.

O BBR, segundo Cardwell et al. (2016), já está sendo implantado no YouTube e substituindo progressivamente o CUBIC, devido ao aumento das métricas da qualidade da experiência do usuário. Desta forma, explorar os algoritmos de congestionamento no uso do servidor de *streaming* é imprescindível para a escalabilidade, estabilidade e desempenho no uso em alta escala.

4.1.4 Testes de carga

Diferentemente dos testes de estresse do subcapítulo 3.1.5, os testes de carga têm por objetivo avaliar formalmente a estabilidade de um sistema quando este se encontra sob condições de carga máxima por um extenso período (NAIK, 2008). Durante esses testes, é possível observar uma degradação no desempenho do sistema, falhas em determinadas funcionalidades ou um comportamento inesperado das aplicações, podendo inclusive ocorrer falhas totais.

Os testes de carga são realizados por meio da simulação de múltiplos usuários virtuais, a fim de replicar situações de alta demanda ao sistema. Nesse sentido, o planejamento desses testes requer uma compreensão aprofundada do público-alvo do sistema, visando representar realisticamente as características do software em análise.

Essa abordagem permite identificar e abordar potenciais problemas de carga, assegurando, assim, uma maior estabilidade do sistema (NAIK, 2008). Ao antecipar e enfrentar essas questões, é possível garantir um desempenho consistente e confiável, atendendo às expectativas dos usuários e promovendo a qualidade do sistema na totalidade.

Os testes de carga apresentam ferramentas que se concentram em avaliar formalmente o desempenho do sistema, seguindo os princípios estabelecidos por esse tipo de teste. Em sua maioria, consistem em utilitários destinados à validação de aplicações *web*, com foco na execução de múltiplas requisições em uma ou mais APIs do sistema em análise, conforme perfis de usuários previamente definidos.

Ao realizar testes de carga, é possível identificar gargalos de desempenho, detectar problemas de escalabilidade e medir a capacidade de resposta do sistema. Essas informações são cruciais para otimizar o desempenho do sistema, melhorar a eficiência e garantir uma experiência de usuário satisfatória.

Dentro desse contexto, alguns *softwares* se destacam para a realização de testes de carga, são eles:

- Apache JMeter: Aplicação de código aberto utilizada para realizar testes de desempenho em recursos estáticos e dinâmicos. Suas funcionalidades abrangem a execução de testes em vários protocolos, uma IDE para planejamento e implementação de planos de testes, bem como suporte ao *multithreading*, entre outras características (APACHE, 2022, tradução nossa).
- Locust: Ferramenta de código aberto para testes de carga de fácil utilização, programável e escalável. Dentre suas funcionalidades, destaca-se a capacidade de desenvolvimento dos testes em código Python, a distribuição e escalabilidade em múltiplas máquinas, bem como uma interface gráfica *web* intuitiva e de fácil utilização (HEYMAN et al., 2023, tradução nossa).
- Grafana K6: *Software* de código aberto voltado para a garantia de qualidade dos sistemas. Suas principais capacidades envolvem a programação em JavaScript, definição de verificações e limites (*thresholds*) e a utilização como uma ferramenta de linha de comando (CLI) (LABS, 2023b, tradução nossa).

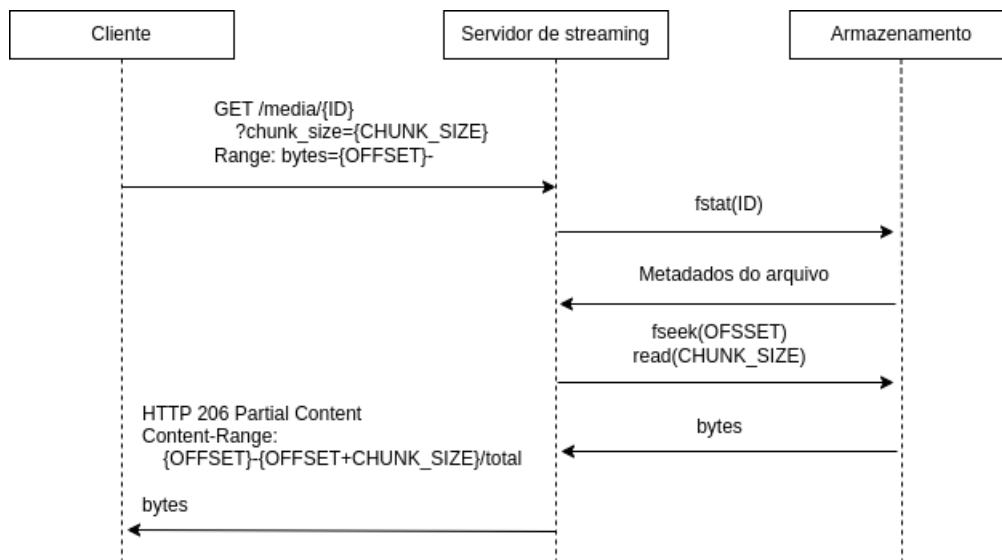
4.1.5 Experimentos

A experimentação da oferta de mídia aconteceu, primeiramente, desenvolvendo um servidor de *streaming* e, posteriormente, é realizado um teste de carga, visando a comparação da capacidade de servir uma grande quantidade de usuários com implantação nos ambientes detalhados no Capítulo 3.

No HTML, há as *tags* `<video>` e `<audio>`, implementadas pelos navegadores como um reprodutor de mídia. Deste modo, o navegador realiza a requisição seguindo o protocolo mostrado na Figura 20, onde é especificado a identificação e o intervalo de tempo em bytes. Assim, foi criado um *endpoint* no servidor *web* que interpreta o intervalo de tempo e fornece os bytes da mídia.

O servidor de *streaming* foi implementado em duas linguagens, Python e Rust, visando a comparação do desempenho do servidor em uma linguagem interpretada e a outra compilada. Ambos, possuem o *endpoint* `"/media/"`, recebendo a identificação, intervalo de tempo em bytes e o tamanho do bloco. O fluxo do *endpoint* é mostrado na Figura 22.

Figura 22 – Fluxo do servidor de *streaming*



Fonte: Autores

A partir da requisição do cliente para o *endpoint*, é verificado os metadados do arquivo no armazenamento, verificando a existência e tamanho, por conseguinte, é iniciado a leitura dos bytes no OFFSET e tamanho CHUNK_SIZE requisitado. Desta maneira, o navegador consegue reproduzir a mídia em qualquer intervalo de tempo sob demanda.

A aplicação em Python foi implementada utilizando o *framework* FastApi, já em Rust, Actix. Ambas possuem implementações similares, possuindo um *endpoint* que recebe as informações da mídia e do bloco e retorna os bytes do intervalo. O código-fonte simplificado do servidor de *streaming* em Python é mostrado na Figura 23.

Na linha 1 da Figura 23, o *endpoint* recebe a identificação da mídia na variável "name", o tamanho do bloco "chunk_size" e o intervalo de bytes "range". O tamanho do bloco é especificado em KiB, então, na linha 3, é transformado em bytes, multiplicando por 1024. Na linha 8, o arquivo é aberto e é recebido o descritor e o tamanho total dele.

Figura 23 – Código-fonte do endpoint do servidor de streaming em Python

```
1 @app.get("/media/{name}")
2 async def media_endpoint(name: str, chunk_size: int = None, range: str = Header(None)):
3     chunk_size = max(min(chunk_size * 1024, MAX_CHUNK_SIZE), MIN_CHUNK_SIZE)
4     start, end = range.replace("bytes=", "").split("-")
5     start = int(start)
6     end = int(end) if end else start + chunk_size
7
8     with storage.open(name, "rb") as (media, size):
9         media.seek(start)
10        data = bytes(media.read(end - start))
11        end = start + len(data) - 1
12        headers = {
13            'Content-Range': f'bytes {start}-{end}/{size}',
14            'Accept-Ranges': 'bytes'
15        }
16        return Response(
17            data,
18            status_code=206,
19            headers=headers,
20            media_type="application/octet-stream"
21        )
```

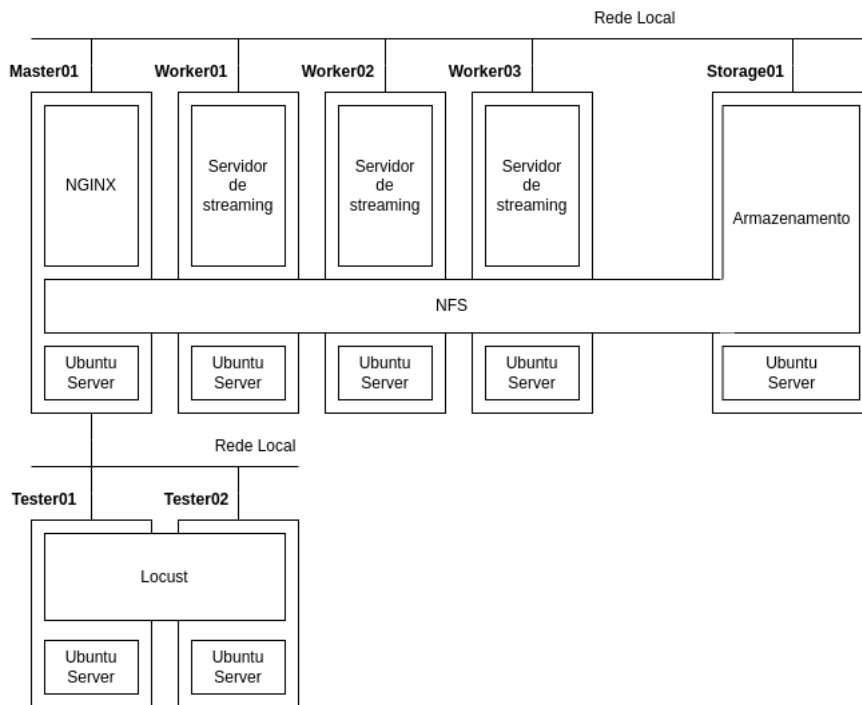
Fonte: Autores

Na linha 9 e 10, a posição de leitura é direcionada para a posição de "start" e os bytes são lidos no intervalo solicitado. Por fim, nas linhas 11 até 21, a resposta para o cliente é construída contendo um cabeçalho com as informações do intervalo e um corpo com os bytes.

4.1.5.1 Benchmark

O *benchmark* do servidor de *streaming* acontece no ambiente de *bare-metal*, máquina virtual e contêineres, utilizando orquestradores para gerenciar as réplicas. O objetivo do *benchmark* é medir a capacidade de fornecer a mídia para os usuários, utilizando a métrica de requisições por segundo e evidenciando a evolução com a adição de novas réplicas no sistema.

Para a realização dos testes, o ambiente inicial foi criado em uma infraestrutura com 7 máquinas virtuais na Azure com o Ubuntu Server. Cada máquina virtual tem 2 VCPU e 4 GiB de memória RAM. Das 7 VMs, 2 são para a ferramenta de teste, 1 para o armazenamento das mídias, 3 para o servidor de *streaming* e 1 para ser a porta de entrada e saída do sistema. O ambiente é mostrado na Figura 24.

Figura 24 – Ambiente base para a experimentação do servidor de *streaming*

Fonte: Autores

Conforme mostrado na Figura 24, o ambiente é formado pelas seguintes VMs:

- Storage01: VM responsável pelo armazenamento físico de todas as mídias. O compartilhamento das mídias acontece por meio do NFS.
- Master01: VM que contém o servidor *web* NGINX e é responsável por ser a porta de entrada para o serviço de streaming, distribuindo a carga de trabalho por meio de *load balancing*.
- Worker[01..03]: VMs que contém as réplicas do servidor de streaming.
- Tester[01..02]: VMs que possuem o Locust, ferramenta utilizada para o teste de estresse.

Para o teste em *bare-metal*, o ambiente base já é o suficiente. As máquinas *workers* contêm os múltiplos processos do servidor de *streaming*. O acesso ao armazenamento acontece por meio de uma montagem local do NFS. Desta forma, o armazenamento remoto fica transparente na aplicação, que trata como se fosse uma pasta local.

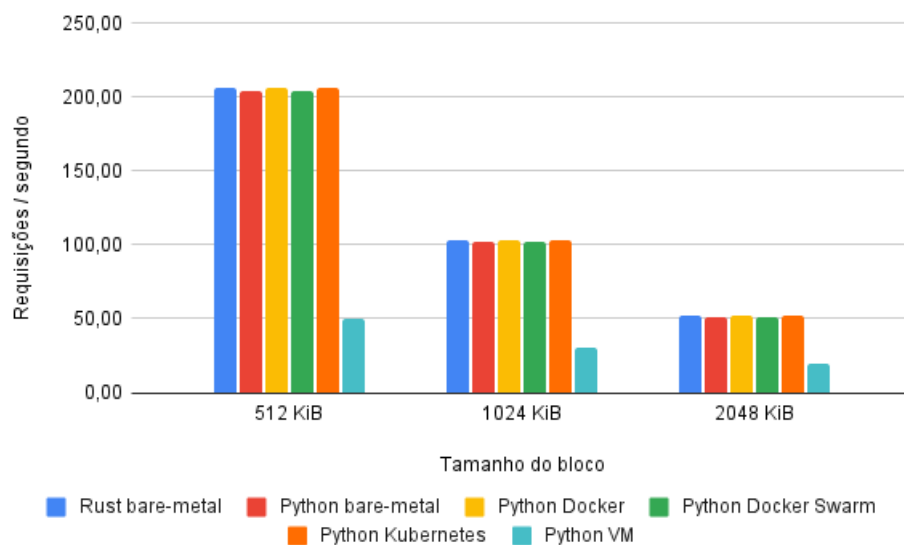
Já o teste em *contêineres* é realizado utilizando duas ferramentas, sendo o Docker Swarm e o Kubernetes. Ambas possuem a responsabilidade de gerenciar as réplicas dos contêineres, decidindo em qual máquina o contêiner será alocado, criando um proxy para isolamento da comunicação, e fornecendo um volume virtual que se conecta com o

armazenamento das mídias. Desta maneira, as máquinas *workers* possuem a capacidade de executar os contêineres do servidor de *streaming*, já a Master01 gerencia o *cluster*.

Por fim, para o teste em *máquinas virtuais*, é utilizado o Libvirt e QEMU. O KVM não é suportado pelas máquinas, então pode haver uma queda de desempenho que deve ser considerada. Cada VM possui a responsabilidade de executar uma única instância do servidor de *streaming*. Apenas as máquinas *workers* possuem VMs que executam o servidor de *streaming*. Não há um proxy para comunicação entre as réplicas, então o acesso ao servidor acontece por meio da associação de uma porta da VM com uma porta da máquina que executa a VM.

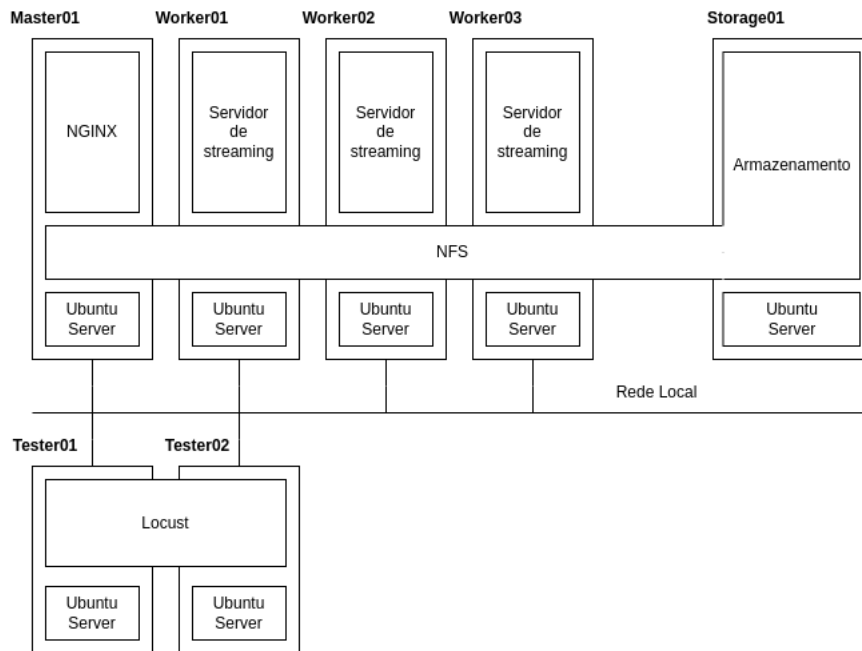
O teste acontece de forma que 100 usuários, divididos entre a máquina Tester01 e Tester02, realizam requisições para a reprodução de um áudio. Cada usuário consome a mídia de forma sequencial, simulando um caso de uso real. O tamanho do bloco pode variar entre 512, 1024 e 2048 KiB. O teste é aplicado em 8 réplicas do servidor de *streaming* em Rust e em Python, em que o ambiente varia entre bare-metal, Docker, Docker Swarm, Kubernetes e máquinas virtuais. A Figura 25 mostra os resultados do teste.

Figura 25 – Resultados do teste de estresse do servidor de *streaming* em diferentes ambientes



Fonte: Autores

Conforme observado, em cada tamanho de bloco, as aplicações atendem de maneira semelhante a um valor aproximado de 200, 100 e 50 requisições por segundo, com exceção do ambiente em máquinas virtuais, que devido à falta de suporte a KVM, a capacidade de transferência tenha sido limitada. Essa semelhança pode ser devida a um gargalo do sistema, uma vez que o ato de fornecer uma mídia é um sistema *I/O bound*, onde a entrada e saída de dados é intensiva, visto que a mídia é constituída por um agrupamento de bits que necessitam ser transferidos do servidor ao cliente.

Figura 26 – Ambiente onde os usuários têm acesso direto aos servidores de *streaming*

Fonte: Autores

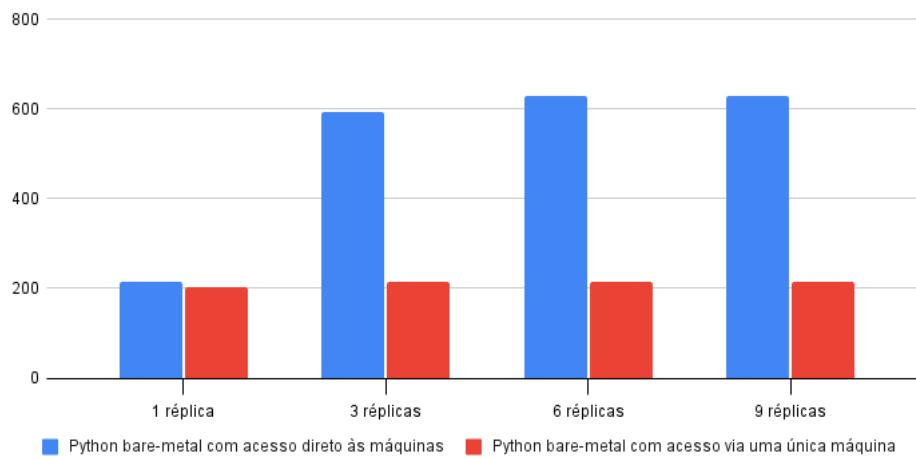
A limitação do sistema pode estar ocorrendo devido ao fato de que todos os dados passam pela Master01, que é a porta de entrada e saída do ambiente, realizando o *load balancing* entre os servidores de *streaming*. Todas as máquinas possuem a mesma largura de banda. Assim, a Master01 não conseguiria transportar todos os dados que as outras máquinas possuem capacidade de transmitir ao mesmo tempo.

Para verificar essa limitação, o teste foi realizado novamente, mas comparando com um ambiente onde os usuários conseguem acesso diretamente aos servidores de *streaming*, sem a necessidade de um *proxy*. O ambiente de acesso direto aos servidores é mostrado na Figura 26.

Dessa forma, os usuários, no caso as máquinas com o *framework* de teste Locust, estão na mesma rede que os servidores de *streaming*, sem a necessidade de requisitar somente pela Master01. O teste foi realizado comparando o ambiente com acesso direto aos servidores com o ambiente inicial, mostrado na Figura 24, com 1, 3, 6 e 9 réplicas e tamanho de bloco de 512 KiB, em um ambiente de bare-metal, e é apresentado na Figura 27.

Como observado, a partir de 3 réplicas, onde cada um dos 3 *workers* possui pelo menos uma réplica, o acesso direto aos servidores de *streaming* fornece, aproximadamente, 3 vezes mais requisições por segundo, quando comparadas a servidores acessados por meio de um *proxy*. Adicionalmente, é possível aferir que, neste tipo de aplicação, se uma réplica consegue usufruir de toda a largura de banda, então a adição de novas réplicas na mesma máquina não garante um aumento substancial de *throughput*.

Figura 27 – Resultados do teste de estresse com e sem acesso direto dos usuários aos servidores de *streaming*



Fonte: Autores

Portanto, a partir dos resultados, ambientes que utilizam *proxy* para a distribuição entre os *servidores de mídia*, no caso o Kubernetes, Docker Swarm e NGINX, possuem uma limitação de *throughput*, dependendo, por sua vez, da largura de banda das máquinas que são a entrada e saída do sistema. Dessa forma, para atender em larga escala, uma das opções é distribuir os *servidores de mídia*, com replicações do armazenamento, em locais geograficamente próximos aos usuários e com uma alta largura de banda, utilizando, por exemplo, serviços de CDNs.

4.2 Serviço de recomendação

Neste subcapítulo, serão abordados os detalhes dos algoritmos de recomendação e o seu funcionamento. Inicialmente, sobre os algoritmos que recomendam com base nas preferências de usuários semelhantes, inseridos na categoria de filtragem colaborativa, e, por conseguinte, sobre os algoritmos que recomendam a partir do conteúdo e características próprias da música.

4.2.1 Filtragem colaborativa

A filtragem colaborativa é um tipo de sistema de recomendação que se baseia na interação dos usuários com os itens para a recomendação. Esse algoritmo necessita apenas de três tipos de dados: a identificação do usuário; a identificação do item; por fim, a avaliação do usuário sobre o item. Deste modo, a recomendação acontece identificando usuários com preferências similares e sugerindo novos itens que os usuários similares também apreciam. (KARAU et al., 2015)

Segundo [Leskovec, Rajaraman e Ullman \(2020\)](#), para a filtragem, é necessário abstrair os dados em uma matriz esparsa de utilidade, onde cada par de usuário e item pode ter um valor associado, representando a preferência do usuário sobre o item, sendo possível o valor ser em branco caso o usuário nunca tenha avaliado ou acessado o item. Um exemplo é mostrado na Tabela 2, onde é representado a avaliação de usuários sobre gêneros musicais em uma escala de 1 a 5.

Tabela 2 – Exemplo da matriz de utilidade para a filtragem colaborativa

	Choro	Samba	MPB	Sertanejo	Forró	Bossa Nova	Axé
A	4	1		5	1		
B	5	5	4				
C		1		4	2	1	
D		3					3

Fonte: Adaptado de [Leskovec, Rajaraman e Ullman \(2020\)](#)

Como é possível observar na Tabela 2, há vários campos em branco, que significam que o usuário ainda não avaliou aquele item. Na prática, segundo [Leskovec, Rajaraman e Ullman \(2020\)](#), a quantidade de avaliação que um usuário realiza representa uma pequena fração de todas as possíveis escolhas, deixando a matriz mais esparsa. Dessa forma, o objetivo da recomendação é tentar prever os valores em branco. Com dados suficientes, seria possível inferir que os usuários que não gostam de Samba também tendem a não gostar de Bossa Nova. Desta maneira, por exemplo, como o usuário A se assemelha com o C, então, provavelmente, o usuário A também não gosta de Bossa Nova.

Há diversos algoritmos para a filtragem colaborativa, dentre eles destacam-se os algoritmos de fatoração de matrizes, popularizado pela competição da Netflix ([BELL; KOREN; VOLINSKY, 2009](#)), que evidenciou a sua eficiência. O Alternating Least Squares (ALS) é um dos métodos que se destaca pela paralelização e eficiência no processamento de grande quantidade de dados ([ZHOU et al., 2008](#)).

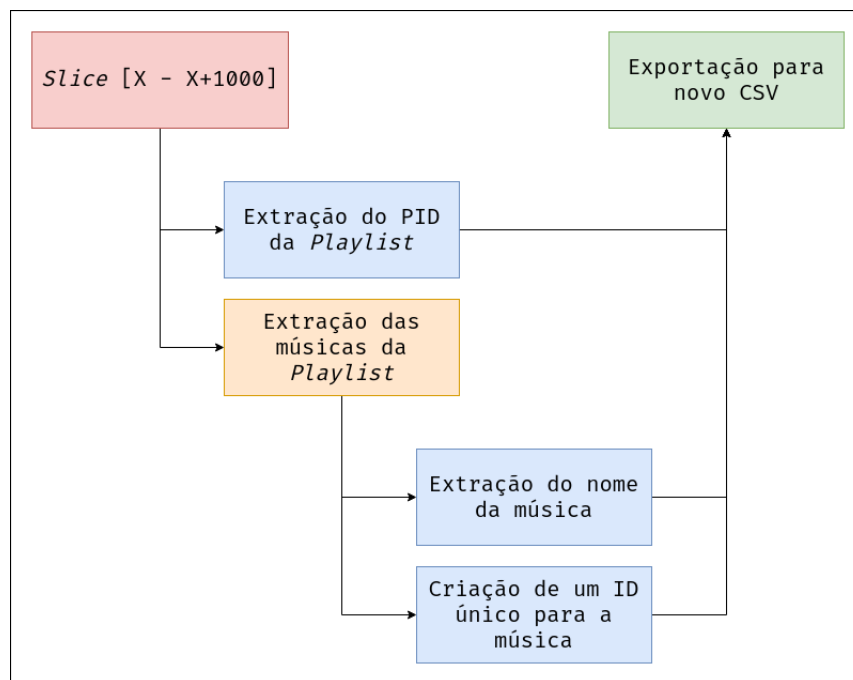
O ALS é um algoritmo de fatoração de matrizes, ou seja, visa derivar da matriz de utilidade duas outras matrizes que, ao serem multiplicadas, se aproximam da matriz original, o erro entre os valores são minimizados de maneira iterativa sendo calculado por meio do método dos mínimos quadrados. Como há células sem valores, a matriz resultante se aproxima dos valores da avaliação dos usuários sobre um item. ([ZHOU et al., 2008](#))

4.2.1.1 Experimentos

A experimentação da filtragem colaborativa aconteceu com o uso do Apache Spark, com o uso da biblioteca MLlib, e do Apache Hadoop, para o armazenamento do conjunto de dados. Já há a implementação pronta do ALS na MLlib, então o experimento abordou o seu uso com o *dataset* do Spotify, simulando os dados da futura biblioteca digital.

O conjunto de dados utilizado é o Spotify Million Playlist Dataset Challenge (CHEN et al., 2018), que contém 1.000.000 de playlists criadas por usuários do Spotify no período entre 2010 e 2017. Esse conjunto é ideal para análise e recomendação baseada na filtragem colaborativa. Devido à grande quantidade de dados nesse conjunto e considerando que se trata de um protótipo do mecanismo de recomendação, apenas uma parte foi utilizada, totalizando cerca de 2.504.485 músicas. Esse conjunto foi devidamente tratado, conforme demonstrado na Figura 28, para extrair apenas as informações necessárias.

Figura 28 – Rotina de extração das informações necessárias



Fonte: Autores

Assim, os dados são estruturados de forma que para cada par de usuário e música há um valor que representa a quantidade de vezes que o usuário escutou aquela música. A partir disso, é possível aplicar o ALS, implementado no Apache Spark, para gerar o modelo de recomendação. A Figura 29 exemplifica¹ os passos para o treinamento do modelo.

Na linha 1 da Figura 29, é instanciado a classe do ALS, passando como parâmetro o "maxIter", que determina o limite de iteração da minimização do erro. Na linha 3, 4 e 5, é informado o nome das colunas do conjunto de treinamento que será utilizado. Por fim, na linha 7 e 8, acontece o treinamento e o modelo é salvo no HDFS.

A partir disso, é possível realizar recomendação de músicas para um ou múltiplos usuários. Para isso, basta construir a matriz de utilidade, aplicar no modelo já treinado e ordenar pela avaliação prevista. A Figura 30 mostra esses passos.

¹ O código completo pode ser encontrado em: https://github.com/TCCJoaoThiago/recommender-system/blob/tcc/modelos/collaborative_filtering.ipynb

Figura 29 – Código-fonte simplificado da filtragem colaborativa

```

1 als = ALS(
2     maxIter=5,
3     userCol="user",
4     itemCol="track_name",
5     ratingCol="count",
6 )
7 model = als.fit(training)
8 model.save(MODEL_NAME)

```

Fonte: Autores

Figura 30 – Código-fonte simplificado da recomendação por filtragem colaborativa

```

1 musics_ids = spark.read.options(
2     delimiter=";", header=True
3 ).csv(MUSIC_ID).select(col("id").cast("int"), "name")
4 user_musics = training.filter(training.user == 413999)
5 prediction = model.recommendForUserSubset(user_musics, 20)
6
7 # Músicas relevantes do usuário
8 user_musics.join(
9     musics_ids, user_musics.track_name == musics_ids.id, "inner"
10 ).select("user", "id", "name").show()
11
12 # Músicas recomendadas
13 prediction.select(
14     "user", explode("recommendations")
15 ).select("user", "col.track_name", "col.rating") \
16 .join(
17     musics_ids, col("track_name") == musics_ids.id, "inner"
18 ).select("user", "id", "name", "rating") \
19 .sort(col("rating").desc()).show()

```

Fonte: Autores

Na linha 1 até a 4 da Figura 30, é construído a matriz de utilidade de um usuário qualquer do *dataset*, aplicado no modelo já treinado na linha 5. Por fim, para efeito de comparação, na linha 8, é mostrado as músicas escutadas pelo usuário e, na linha 13, as músicas recomendadas pelo modelo, ordenadas pela avaliação.

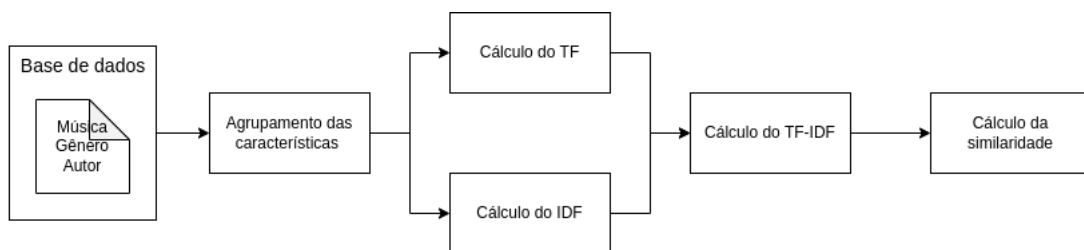
4.2.2 Filtragem baseada no conteúdo

A filtragem baseada no conteúdo, por sua vez, baseia-se na recomendação por meio das propriedades do item, onde os semelhantes são recomendados. De acordo com Leskovec, Rajaraman e Ullman (2020), é necessário, para cada item, um perfil que contém as principais características. Desta maneira, a utilização de características provenientes

de modelos ontológicos já validados, como o da música brasileira proposto por [Padron, Cruz e Silva \(2020\)](#), garante que a recomendação se aproxima do que o item é do esperado pelo usuário.

De acordo com [Leskovec, Rajaraman e Ullman \(2020\)](#), a recomendação acontece agrupando as características dos itens em termos, possibilitando o cálculo da similaridade. A semelhança pode ser obtida por meio da distância de Jaccard ou pela transformação dos termos em vetores e o cálculo através da similaridade de cosseno. Na Figura 31, é mostrada uma arquitetura para a filtragem baseada em conteúdo.

Figura 31 – Arquitetura da filtragem baseada em conteúdo



Fonte: Adaptado de [Ikhsanudin e Winarko \(2019\)](#)

Como é possível observar na Figura 31, as características da música são agrupadas, transformadas em vetores, onde os principais termos são amplificados por meio do cálculo do Frequency–Inverse Document Frequency (TF-IDF), e, por fim, utilizadas para o cálculo da similaridade de cosseno.

De acordo com [Karau et al. \(2015\)](#), o TF-IDF revela o quão importante uma palavra é para um determinado documento, considerando a raridade para todo o conjunto de documentos, onde o cálculo é feito por meio do produto entre o Term Frequency (TF) e o Inverse Document Frequency (IDF). O TF representa a importância de um termo t em um documento d , sendo calculado pela divisão da frequência do termo $f(t, d)$ pelo total de termos no documento $n(d)$. Já o IDF representa a importância do termo t em um conjunto de documento d , calculado dividindo-se a quantidade de documentos n pela quantidade de documentos que o termo aparece df . O TF-IDF é mostrado na Equação 4.3.

$$TFIDF(t, d) = \frac{f(t, d)}{n(d)} * \log\left(\frac{n}{df}\right) \quad (4.3)$$

Já a similaridade de cosseno representa a distância entre dois vetores. A semelhança entre dois itens x e y é mostrada na Equação 4.4, onde $w_{j,x}$ é o peso do j -ésimo elemento do vetor x e $w_{j,y}$ é o j -ésimo elemento do vetor y .

$$\text{Similarity}(x, y) = \frac{\sum_{j=1}^U w_{j,x} \cdot w_{j,y}}{\sqrt{\sum_{j=1}^U w_{j,x}^2 \sum_{j=1}^U w_{j,y}^2}} \quad (4.4)$$

Assim, segundo [Ikhsanudin e Winarko \(2019\)](#), já que os vetores representam as características dos itens, quanto menor for a distância entre os vetores, maior será a similaridade entre eles. Possibilitando, dessa forma, a obtenção de itens similares para a recomendação.

4.2.2.1 Experimentos

A experimentação da filtragem baseada em conteúdo aconteceu com o uso do Apache Spark e do Apache Hadoop, para o armazenamento do conjunto de dados. Diferentemente da filtragem colaborativa, o Apache Spark não fornece a implementação pronta, então a experimentação reproduziu os passos do TF-IDF e da similaridade de cosseno manualmente sobre o conjunto de dados do Spotify.

O conjunto de dados utilizado é o "Top Spotify songs from 2010-2019 - BY YEAR"², que contém 584 músicas. Nesse conjunto de dados, foram adicionados metadados às músicas, incluindo informações como o nome do artista, gênero, ano, popularidade, batidas por minuto e outras características relevantes. A escolha desse conjunto foi motivada pela sua semelhança com a proposta de ontologia para avaliação das músicas, o que o torna ideal para ser utilizado no modelo de filtragem baseado no conteúdo proposto nesse trabalho.

O primeiro passo é construir a *pipeline* que calcula o TF-IDF do conjunto de dados. Primeiramente, é necessário separar os termos em palavras. Seguidamente, as palavras consideradas *stop words*, palavras vazias e comuns que não agregam no sentido do termo de recomendação, são removidas. Por conseguinte, os termos são transformados em *hash* e posteriormente em um vetor, onde cálculo do TF é realizado. Com o resultado do TF, é possível calcular o IDF. Por fim, a *pipeline* é treinada com o conjunto de dados e os vetores resultantes são normalizados para facilitar o cálculo da similaridade de cosseno, uma vez que o produto vetorial com vetores normalizados resulta no cosseno entre eles. A Figura 32 mostra o código-fonte simplificado³.

Na linha 1 até 15 da Figura 32, é construído a *pipeline* para o cálculo do TF-IDF, composta pela transformação `Tokenizer`, que separa em palavras, `StopWordsRemover`, que remove as palavras vazias, `HashingTF`, que cria um vetor dos termos pelo seu *hash* e calcula o TF, e, por fim, o IDF. O modelo é treinado na linha 15 e, posteriormente, os vetores do treinamento são normalizados, na linha 17, 18 e 19.

² O *dataset* pode ser encontrado em: <https://www.kaggle.com/datasets/leonardopena/top-spotify-songs-from-20102019-by-year>

³ O código completo pode ser encontrado em: https://github.com/TCCJoaoThiago/recommender-system/blob/tcc/modelos/content_based.ipynb

Figura 32 – Código-fonte simplificado da filtragem baseada no conteúdo

```
1 tokenizer = Tokenizer(inputCol="features", outputCol="feature_tokens")
2 stop_words_remover = StopWordsRemover(
3     inputCol=tokenizer.getOutputCol(),
4     outputCol="feature_filtered"
5 )
6 hashing = HashingTF(
7     inputCol=stop_words_remover.getOutputCol(),
8     outputCol="features_hash"
9 )
10
11 idf = IDF(inputCol=hashing.getOutputCol(), outputCol="features_idf")
12
13 pipeline = Pipeline(stages=[tokenizer, stop_words_remover, hashing, idf])
14
15 model = pipeline.fit(training_features)
16
17 features = model.transform(training_features)
18 normalizer = Normalizer(inputCol=idf.getOutputCol(), outputCol="features_normalized")
19 normalized_model = normalizer.transform(idf_features)
```

Fonte: Autores

Com o modelo treinado, é possível realizar a recomendação das músicas similares. Para isso, as características das músicas bases para a recomendação devem ser filtradas. Por conseguinte, é aplicado TF-IDF e a normalização sobre as músicas. Após, é realizado o cálculo da similaridade de cosseno, por meio do produto vetorial entre as músicas normalizadas. Com o cálculo da similaridade do par de música escolhida com todas as músicas, é possível filtrar as músicas similares. A Figura 33 exemplifica os processos para a predição.

Na linha 2 e 3 da Figura 33, as características das músicas selecionadas para a recomendação de similares filtradas, no caso as músicas com ID 4, 400, 525 e 41. Na linha 6 e 7, as músicas são submetidas ao TF-IDF e a normalização. Para o cálculo da similaridade de cosseno, na linha 11, é feito um CrossJoin das músicas selecionadas com todo o conjunto de música. Para cada par de música, na linha 17, é calculado a similaridade de cosseno por meio do produto vetorial definido na função da linha 15. Desta forma, é possível agrupar as recomendações das músicas similares e ordenar pelos pares de música que possuem a maior similaridade, isso está acontecendo na linha 26.⁴

⁴ O resultado desse exemplo pode ser encontrado em: <https://github.com/TCCJoaoThiago/recommender-system/blob/tcc/modelos/content_based.ipynb>

Figura 33 – Código-fonte simplificado da recomendação por filtragem baseada no conteúdo

```

1  # Músicas para a recomendação de semelhantes
2  IDS = ["4", "400", "525", "41"]
3  music = training_features.filter(training_features.id.isin(IDS))
4
5  # Aplicação das músicas no modelo
6  music_tf_idf = model.transform(music)
7  music_normalized = normalizer.transform(music_tf_idf).selectExpr(
8      "id as other_id", "features_normalized as other_features_normalized")
9
10 # Cálculo da similaridade de cosseno
11 similarity_data = normalized_data.crossJoin(
12     music_normalized
13 ).filter(col("id") != col("other_id"))
14
15 dot_udf = udf(lambda x,y: float(x.dot(y)), DoubleType())
16
17 music_similarity = similarity_data.select(
18     "id", "other_id",
19     dot_udf(
20         "features_normalized", "other_features_normalized"
21     ).alias("similarity")
22 )
23
24 # Apresentação da das 20 melhores recomendações
25 TOP_MUSICS = 20
26 music_similarity.join(training_features, ['id', 'id']) \
27     .select("id", "title", "features", "other_id", "similarity") \
28     .withColumn("music_rank", row_number().over(
29         Window.partitionBy("other_id") \
30         .orderBy(music_similarity.other_id, music_similarity.similarity.desc())
31     ) \
32     .filter(col("music_rank") <= TOP_MUSICS) \
33     .select("title", "other_id", "features") \
34     .show(TOP_MUSICS * len(IDS), False)

```

Fonte: Autores

5 Biblioteca digital

Com o considerável avanço dos sistemas ontológicos e o contínuo desenvolvimento dos sistemas de recomendação, torna-se viável a criação de bibliotecas digitais com capacidade elevada de armazenamento, classificação e recuperação de informações. De acordo com Cruz (2008), as bibliotecas digitais desempenham um papel fundamental, especialmente no contexto musical, ao proporcionarem não apenas um ambiente virtual para interação musical, mas também ao atuarem como espaços para preservação, implementação de melhorias na acessibilidade, integração de diversos formatos e disponibilização de ferramentas voltadas para a educação musical.

Os elementos contidos em uma biblioteca digital são denominados artefatos digitais, que, além de seu conteúdo principal, são enriquecidos com metadados que auxiliam e orientam os usuários. Essa característica não apenas permite a integração das bibliotecas com múltiplas fontes externas, ao assegurar a presença dos metadados necessários, como também estimula a adoção de modelos ontológicos para a construção de uma base de informações mais abrangente e descritiva.

5.1 Ontologia

Com o avanço da tecnologia, é presenciado o surgimento de novas abordagens para lidar com o armazenamento, entendimento e classificação das mídias digitais. Nesse contexto, despontam como protagonistas a Web Semântica e as ontologias, que desempenham um papel fundamental na organização e interpretação do vasto volume de dados presentes na era digital. Taniar e Rahayu (2006, tradução nossa) reforçam:

A web semântica é a chave para a próxima geração dos sistemas de informações da web, onde o conhecimento recebe um significado bem definido, permitindo melhor que pessoas e programas trabalhem em cooperação entre si. [...] A web semântica motivou o compartilhamento de conhecimento, através da qual existe a necessidade em tornar a busca por conteúdos mais eficiente e significativa, provendo informações contextuais e estruturais sobre os conteúdos apresentados.

Essas inovações, especialmente na área da ontologia, expandem as possibilidades de armazenamento das expressões artísticas, permitindo a inclusão de informações além dos dados brutos, os valiosos metadados. Atributos esses que conseguem enriquecer as pesquisas, contribuindo na busca por conteúdos similares, facilitando o entendimento e a divulgação sobre os diversos períodos históricos de um país.

5.1.1 Web Semântica

O avanço tecnológico, especialmente na área da Ciência da Computação, impulsionou o surgimento e desenvolvimento de diversos conceitos de grande relevância, sendo a Web Semântica um dos mais destacados. Esse termo adquiriu importância devido à sua capacidade de facilitar a recuperação, processamento e mediação da informação em benefício dos usuários, de maneira estruturada e significativa (SEGUNDO; CONEGLIAN; LUCAS, 2017).

Outra premissa relevante introduzida por esse termo diz respeito à capacidade de processamento que os computadores adquiriram, permitindo a realização de consultas mais precisas conforme o contexto em que estão inseridos. Portanto, para as máquinas e usuários conseguirem fazer o uso desse potente mecanismo se faz necessário que as informações, manifestações de arte ou quaisquer outros tipos de conhecimento estejam representados de uma forma estruturada e tenham a disposição um conjunto de regras (BERNERS-LEE; HENDLER; LASSILA, 2001).

Com o intuito de atender às exigências e princípios estabelecidos na Web Semântica, foram desenvolvidas diversas tecnologias capazes de representar o conjunto de regras essenciais para o seu adequado funcionamento. Entre essas tecnologias destacam-se o Resource Description Framework (RDF), que serve como uma ferramenta para expressar semanticamente o conhecimento, e o SPARQL Protocol and RDF Query Language (SPARQL), uma linguagem de consulta utilizada para recuperar informações de bancos de dados RDF. Além disso, tecnologias já existentes, como o Extensible Markup Language (XML), sendo uma forma de notação amplamente empregada na *web* e possui elevada capacidade computacional, foram empregadas para suprir as necessidades e os objetivos propostos.

5.1.2 Modelos Ontológicos

Apesar de existirem representações estruturadas e tecnologias capazes de descrever a Web Semântica, é imprescindível estabelecer especificações para definir os vocabulários utilizados e estipular um conjunto de regras bem definido. Para esses problemas, Berners-Lee, Hendler e Lassila (2001, tradução nossa) definem a ontologia como sendo:

Um documento ou arquivo que formalmente define as relações entre os termos. O tipo mais comum de ontologia para a *Web* tem uma taxonomia e um conjunto de regras de inferência associadas. A taxonomia define classes de objetos e a relações entre si. [...] Regras de inferência em ontologias fornecem mais poder.

A utilização de modelos ontológicos resolve diversos problemas relacionados a múltiplos bancos de informação com conteúdos idênticos ou semelhantes, por meio do estabelecimento de regras e definições para as terminologias. Essa habilidade fortalece a Web

Semântica, permitindo buscas complexas em conjuntos de informações e criando vínculos entre diferentes bases de dados, proporcionando resultados e explicações mais abrangentes sobre os objetos de pesquisa.

De acordo com [Berners-Lee, Hendler e Lassila \(2001\)](#), esse poder de pesquisa é derivado da contextualização das páginas *web*, ou seja, da sua marcação. Essa característica adiciona metadados às páginas, viabilizando questionamentos estruturados e garantindo uma definição clara com uma melhor compreensão dos termos pelos computadores.

5.1.3 Música Brasileira e Ontologia

Apesar do amplo arcabouço de princípios subjacentes à Web Semântica, a possibilidade de desentendimentos persiste, mesmo com a aplicação dos modelos ontológicos. Para mitigar essa problemática, a Federação Internacional de Associações e Instituições Bibliotecárias ([IFLA, 2011](#)) implementou a norma de descrição bibliográfica padronizada conhecida como International Standard Bibliographic Description (ISBD), com o propósito de estabelecer diretrizes que promovam a compatibilidade global na catalogação descritiva.

Com o surgimento dessa bibliografia, diversos modelos começaram a ser formalizados e utilizados mundialmente, conforme evidenciado no contexto das produções musicais. Uma dessas ontologias que pode ser citada é a International Federation of Library Associations and Institutions Library Reference Model (IFLA LRM) ([RIVA et al., 2017](#)), constituída por conceitos abrangentes capazes de descrever tanto elementos concretos quanto abstratos.

Essa modelagem tem, como um de seus fortes, a representação de múltiplas entidades e seus relacionamentos através do conceito de Trabalho, Expressão, Manifestação e Item (WEMI). Ainda segundo [Riva et al. \(2017\)](#), os seguintes conceitos foram definidos:

- Trabalho: Conteúdo intelectual ou artístico de uma criação distinta. Entidade abstrata que representa um conjunto de ideias e conceitos sobre uma expressão.
- Expressão: Uma combinação de sinais distintos que convêm para um conteúdo artístico ou intelectual.
- Manifestação: Um conjunto de portadores que supostamente compartilham das mesmas características artísticas ou intelectuais.
- Item: Um ou múltiplos objetos responsáveis por transmitir informações artísticas e intelectuais.

Embora as evoluções trazidas pela modelagem IFLA LRM tenham sido bastante vantajosas e ainda serem utilizadas em diversas ontologias, ela não consegue trazer o

significado ao nível mais específico do conceito. Isso é devido ao modelo ser voltado para descrever a memória cultural e bibliográfica, além de visar a publicação de dados na Web Semântica (PADRON et al., 2023). Esses motivos impedem que a música brasileira, foco deste trabalho, seja ontologicamente descrita de maneira detalhada, requerendo assim a utilização de um modelo mais específico para sua representação.

O modelo recentemente proposto (PADRON; CRUZ; SILVA, 2020; PADRON et al., 2023) vai além da aplicação dos conhecimentos e definições previamente descritos no IFLA LRM, ao introduzir novos elementos concretos e abstratos em sua definição ontológica. Ao estabelecer conexões com o modelo mencionado anteriormente, várias generalizações são eliminadas para dar lugar a uma descrição mais precisa e detalhada.

O escopo apresentado visa proporcionar, em um amplo sentido, o entendimento e a comunicação de conceitos da música popular brasileira, que compreende os aspectos do contexto cultural na qual a música está inserida, sua natureza, sua concepção, execução e registro, não se restringindo aos objetos de informações que o registram na forma de áudio ou notação musical (PADRON; CRUZ; SILVA, 2020, tradução nossa).

Esse novo modelo de representação passa, portanto, a compreender diversos aspectos evidenciados nas músicas populares brasileiras, através da definição de alguns elementos como composição, arranjo e interpretação, demonstrados na figura do modelo (Figura 34). Com essas novas características, diversas músicas brasileiras que possuem interpretações diferentes, sejam instrumentais, musicais ou performáticas passam a se relacionar diretamente na Web Semântica.

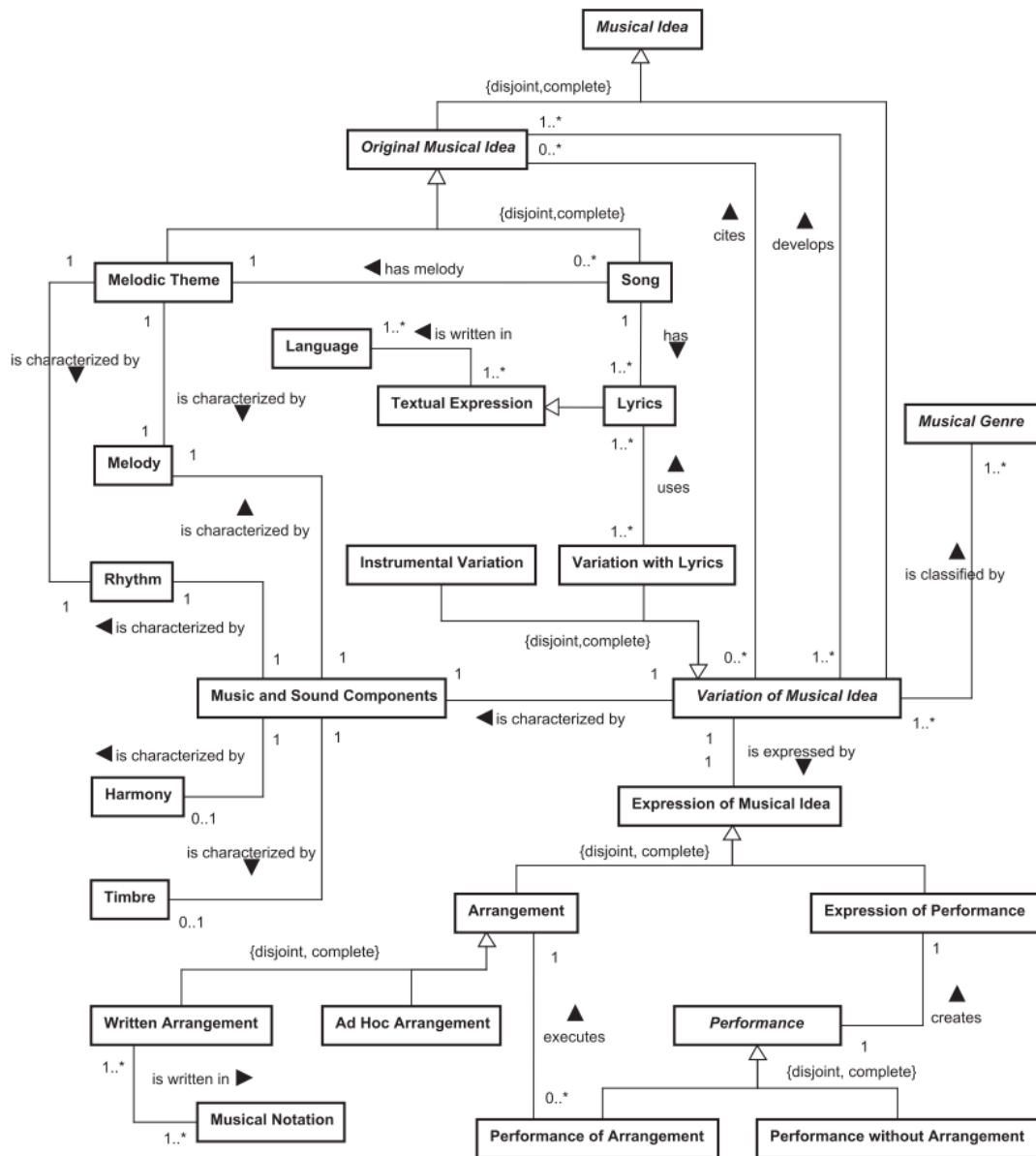
As novas associações entre as músicas possibilitam a ampla utilização das vantagens da Web Semântica e da ontologia. Isso se reflete não apenas em filtros de busca mais poderosos, mas também em inovações no armazenamento da história brasileira em bibliotecas virtuais. Além disso, os sistemas de recomendação se beneficiam dos conceitos ontológicos, gerando sugestões de músicas mais similares e coerentes com o contexto desejado.

5.2 Modelos

Com base no abrangente estudo sobre a ontologia das músicas brasileiras proposto por Padron (2019), torna-se viável desenvolver um modelo de banco de dados que estabelece a conexão entre a teoria proposta e sua aplicação prática por meio da implementação de um sistema capaz de incorporar esse artifício para a criação de uma biblioteca digital.

A concepção desse modelo demandou um laborioso processo de transposição dos termos técnicos do domínio ontológico para expressões de software viáveis para implementação. Ademais, essa tarefa exigiu uma execução meticulosa a fim de prevenir possíveis

Figura 34 – Visão geral das relações presentes no modelo conceitual de música brasileira



Fonte: Padron et al. (2023)

discrepâncias entre o modelo conceitual e o banco de dados, evitando representações divergentes ou mesmo incorretas das informações.

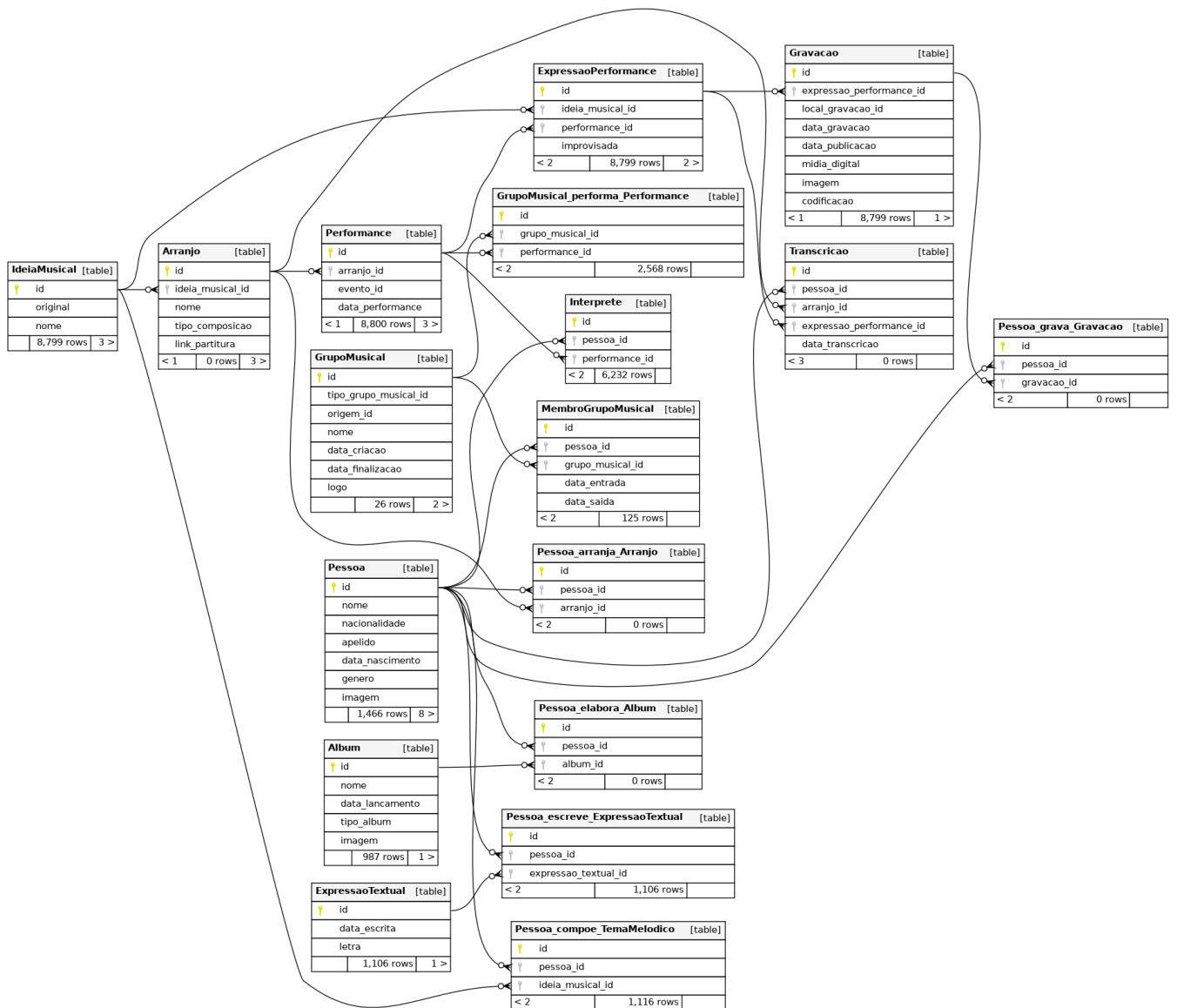
Assim, a tradução desse domínio iniciou mediante uma análise minuciosa de cada modelo e fundamentação teórica apresentados por Padron (2019), complementados por Padron, Cruz e Silva (2020) e aprimorados por Padron et al. (2023). Esse aprofundamento permitiu a compreensão dos inter-relacionamentos entre as entidades, conferindo maior confiabilidade a esse processo de conversão.

A primeira parte do modelo escolhida para a conversão, disposta na Figura 56 do Anexo A, trata dos relacionamentos e entidades associados à autoria de uma produção

musical. Neste modelo, é delineado que uma pessoa pode desempenhar distintos papéis na elaboração de uma música, incluindo sendo um compositor, letrista, arranjador, intérprete individual ou de um grupo, e produtor. Adicionalmente, o modelo retrata como essas distintas contribuições originam entidades separadas que se inter-relacionam para formar o produto final.

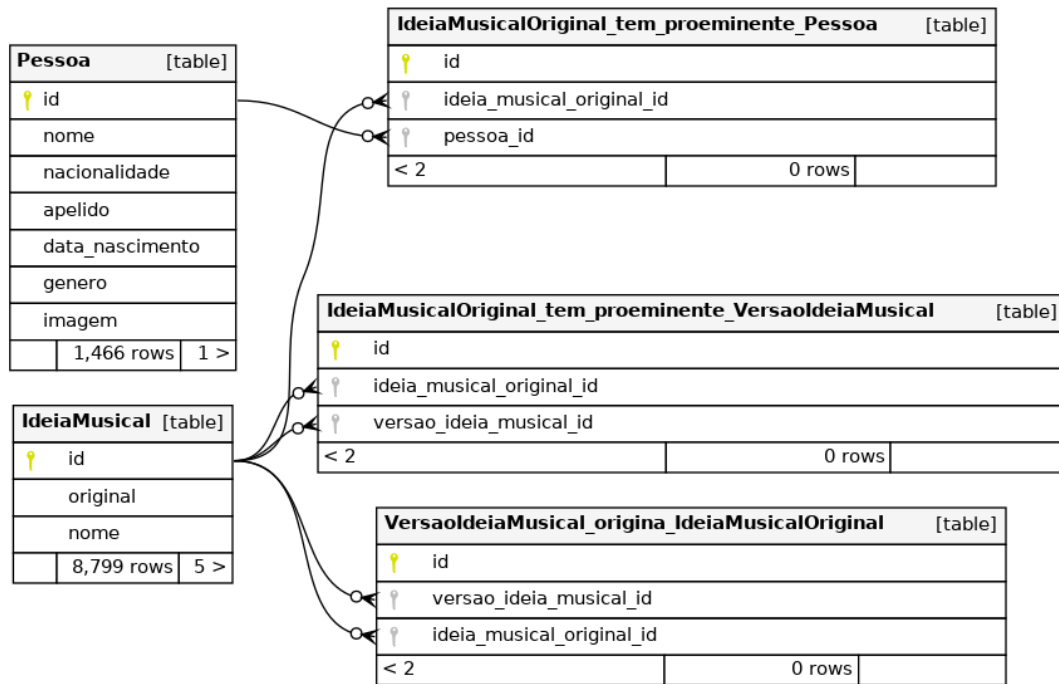
O resultado, evidenciado na Figura 35, revela que muitos dos papéis que uma pessoa pode desempenhar foram transformados em relacionamentos N:M, estabelecendo uma conexão entre a entidade fundamental de pessoa e o produto final da respectiva atividade profissional. Ademais, observa-se a inclusão de diversos atributos conforme as exigências para uma representação mais precisa do papel da entidade no contexto.

Figura 35 – Modelo de banco de dados para autoria.



Nessa primeira representação, foi estabelecido o conceito de ideia musical, uma das entidades centrais do banco de dados. Em conjunto com essa definição, foram pensados e modelados os conceitos de ideia original, versão de ideia, versão representativa e *medley*. Portanto, uma modelagem distinta foi elaborada para abranger esses novos conceitos, conforme demonstrado na Figura 36.

Figura 36 – Modelo de banco de dados para versões de ideia musical.



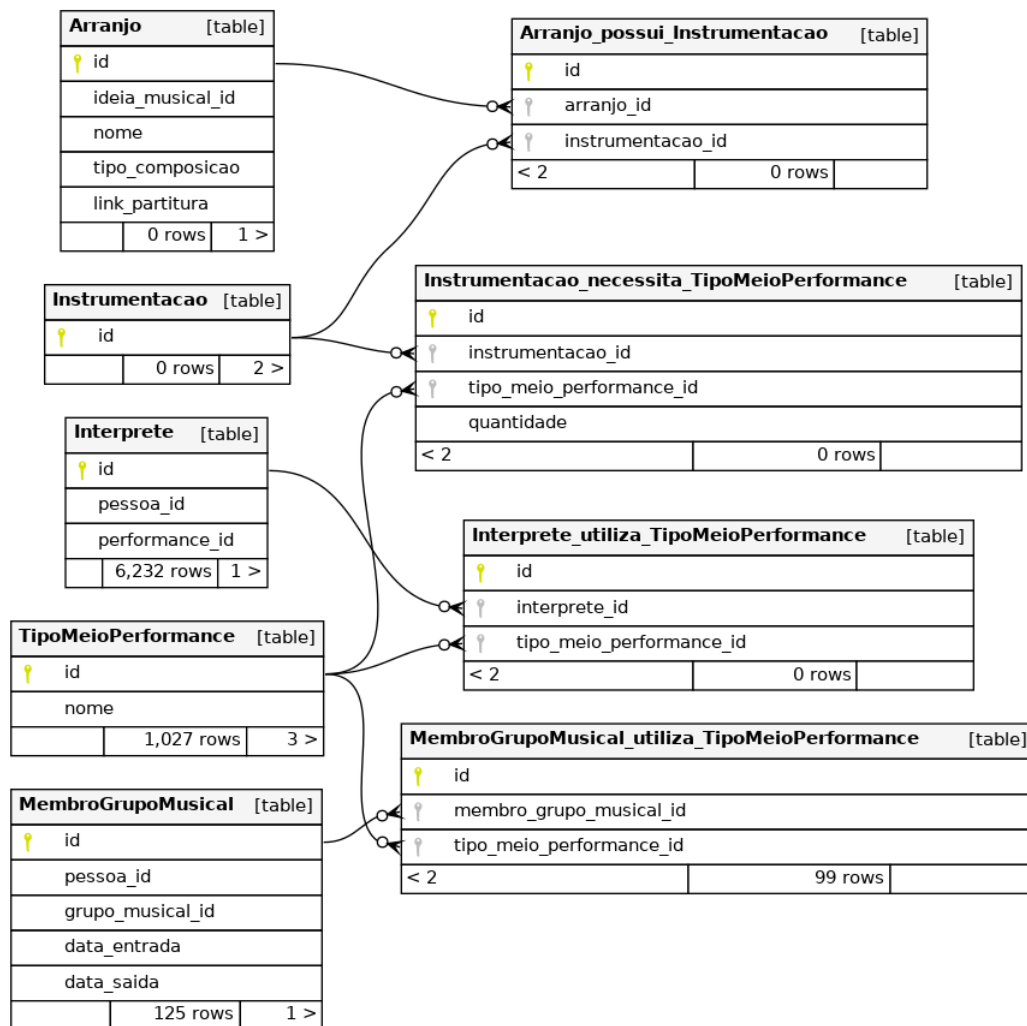
Fonte: Autores

É relevante enfatizar que determinados conceitos, a exemplo do *medley* e suas variações com e sem improvisação, inicialmente delineados no modelo apresentado na Figura 57 do Anexo A, não foram incorporados ao modelo definitivo do banco de dados. Tal decisão decorreu de limitações temporais e da complexidade intrínseca à sua implementação na biblioteca digital, resultando, assim, em uma representação simplificada.

Com a conclusão da definição completa de autoria, procedeu-se à elaboração das entidades associadas à instrumentação dos artistas, denominada como "Meio de Performance" (PADRON et al., 2023). Essa modelagem engloba os conceitos relacionados ao nome do instrumento, à sua utilização por um membro do grupo ou por um intérprete individual, bem como ao seu vínculo com um arranjo. Semelhantemente à ontologia de autoria, a conversão da ontologia base (Figura 58, Anexo A) transcorreu sem grandes obstáculos, resultando em uma representação (Figura 37) bastante análoga à modelagem original.

Com a Ideia Musical e os Meios de Performance devidamente definidos, procedeu-

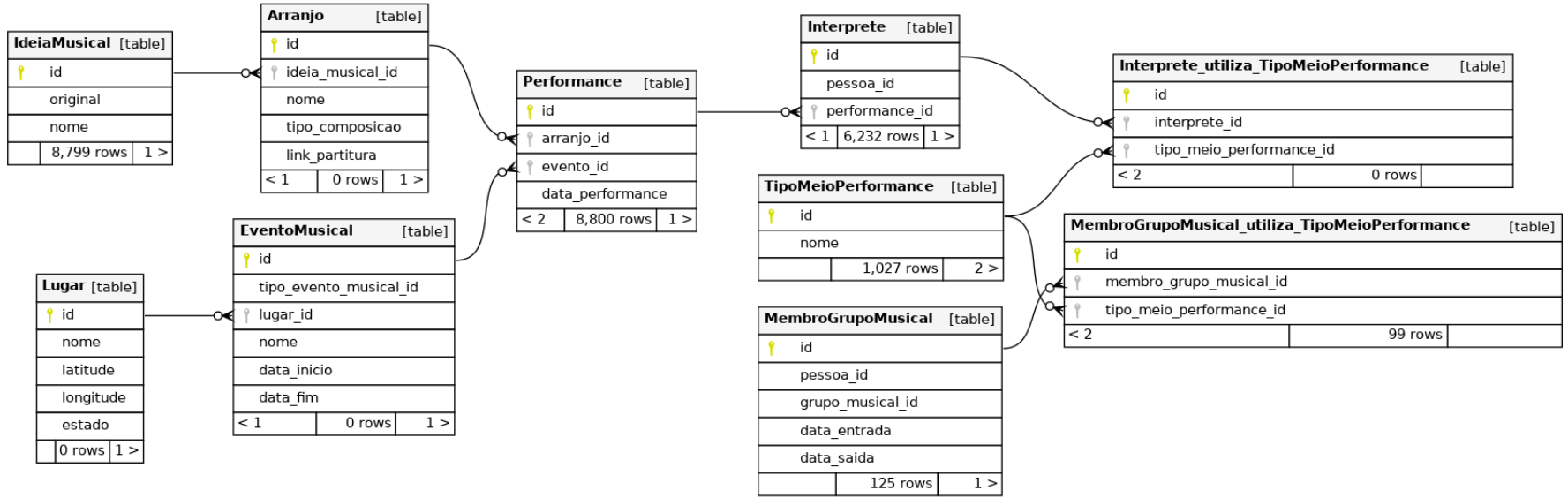
Figura 37 – Modelo de banco de dados para instrumentação.



Fonte: Autores

se à modelagem concernente à performance (Figura 59, Anexo A). Também considerada uma das entidades centrais no modelo conceitual, essa entidade interage com o local por meio de um evento musical, com o arranjo, com uma gravação, com os intérpretes e com os meios de performance. A Figura 38 evidencia o resultado desse processo, no qual houve a conversão e simplificação de relacionamentos em atributos ou entidades agrupadas, buscando representar de maneira mais coesa o modelo em um banco de dados.

Figura 38 – Modelo de banco de dados para performance.



Fonte: Autores

Por fim, uma das áreas onde o modelo se destaca é na definição dos gêneros que compõem uma produção musical (Figura 60, Anexo A). Notadamente direcionada para músicas brasileiras, a ontologia visa ilustrar como um gênero pode influenciar e ser influenciado por outros gêneros. Além disso, visa representar se um gênero é composto por subgêneros, se foi concebido em um movimento musical e quais são suas características musicais e sonoras, documentando a origem e as particularidades do gênero (PADRON, 2019).

A modelagem, portanto, adotou um formato semelhante, no qual diversos relacionamentos foram traduzidos em tabelas N:M, visando melhor representar as diversas conexões entre os gêneros. A diferença significativa reside na modelagem das características musicais e sonoras, as quais não foram convertidas para o modelo de banco de dados devido à sua complexidade e à dificuldade de representação nos termos técnicos de um banco de dados. Sendo assim, a modelagem final apresentada na Figura 39.

Embora todos os modelos ontológicos principais já estejam representados em um diagrama de banco de dados, alguns relacionamentos e entidades foram incorporados posteriormente para estabelecer conexões e armazenar informações essenciais para o pleno funcionamento da biblioteca digital. A versão completa desse modelo de dados, acompanhada de uma implementação em SQL para o PostgreSQL, encontra-se disponível no repositório do GitHub¹.

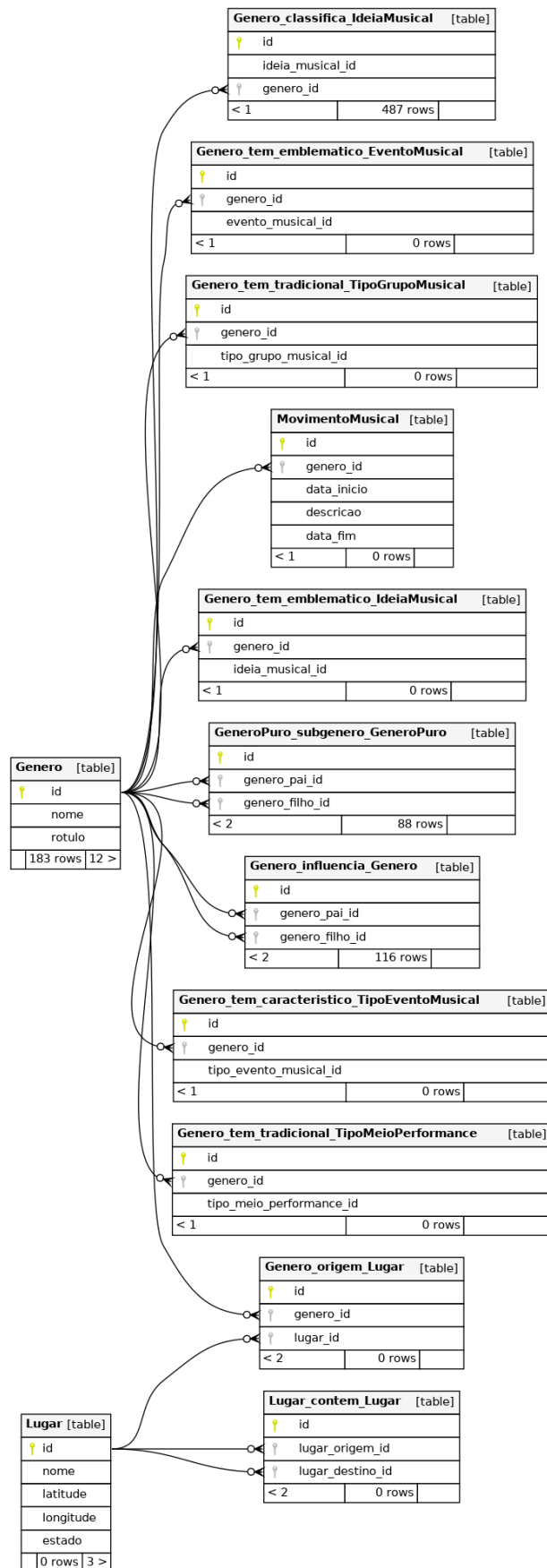
5.2.1 Doremus

De maneira análoga ao modelo proposto por Padron (2019), desenvolveu-se um modelo concebido como uma extensão do FRBRoo, uma ontologia dedicada à representação de informações bibliográficas, com o propósito de modelar objetos culturais relacionados a determinados aspectos e domínios da música. Esse modelo dinâmico abstrai informações relativas ao autor, referido como *Work*, evento e expressão, considerando-os como estruturas fundamentais, e inclui informações complementares como performance e publicação. Ao se fundamentar no FRBRoo, o Doremus torna-se capaz de empregar as mesmas abstrações, enriquecendo assim a modelagem proposta (LISENA; TRONCY, 2017).

Ao integrar a tríade autor, evento e expressão em conjunto com o modelo FRBRoo, o Doremus revela competência em representar de maneira abrangente e detalhada diversas características musicais. Contudo, devido ao seu desenvolvimento específico no contexto da música francesa, suas classes e atributos demonstram uma notável afinidade com composições europeias clássicas. Isso é evidenciado pela modelagem minuciosa de elementos como tempo, estrutura harmônica, escala musical, indicação metronômica, entre outras classes que remetem a características fortemente presentes em orquestras e músicas clássicas.

¹ Repositório do banco de dados: <<https://github.com/TCCJoaoThiago/db>>

Figura 39 – Modelo de banco de dados para gênero.



Este ponto focal acaba por delimitar e especificar consideravelmente o que é viável representar por meio do modelo. Essa restrição dificulta ou até mesmo impede a aplicação do Doremus para representar músicas brasileiras em uma biblioteca digital, sobretudo devido à natureza menos estruturada das composições musicais populares brasileiras em comparação com as músicas clássicas. Além disso, inserido em um contexto majoritariamente composto por músicas clássicas, o modelo tende a simplificar a representação dos gêneros musicais, uma vez que, para essas composições, frequentemente o gênero pode ser considerado puro, desprovido de influências ou subgêneros.

5.2.2 Variations

Diferentemente do Doremus, o projeto Variations, voltado para o desenvolvimento de uma biblioteca digital, optou por uma modelagem mais simplificada, fundamentada no modelo FRBR, com objetos de natureza mais genérica. Apesar dessa abordagem, o Variations incorpora diversos princípios presentes tanto no Doremus quanto no modelo proposto para músicas brasileiras, como exemplos: *work*, *expression*, *manifestation* e *item* (RILEY et al., 2007).

Por se tratar de um modelo mais genérico, composto apenas por quatro entidades (*Contributor*, *Work*, *Container* e *Instantiation*)², é viável sua utilização para representar uma biblioteca digital de músicas brasileiras. No entanto, em contraste com o Doremus, que apresenta informações mais específicas, o Variations abstrai vários desses conceitos em uma das quatro entidades, resultando na perda de diversas informações consideradas importantes na descrição ontológica de músicas brasileiras.

5.2.3 MusicBrainz

Ao contrário dos demais modelos, o MusicBrainz (KAYE, 2000) não foi concebido a partir de uma base ontológica; no entanto, destaca-se como uma das opções mais propícias para o desenvolvimento de uma biblioteca digital de músicas brasileiras em comparação com o Doremus e o Variations. Este modelo não apenas engloba conceitos genéricos, mas também incorpora conceitos bastante específicos, uma vez que sua estrutura é construída em torno da produção musical. Consequentemente, é possível encontrar, por meio de seus relacionamentos, uma ampla gama de informações sobre uma única música nessa base de dados.

O MusicBrainz, ao ser contrastado com o modelo ontológico de músicas brasileiras, evidencia aspectos positivos e negativos quanto à viabilidade de armazenamento e consulta em pesquisas. Destaca-se como ponto positivo primordial a capacidade do MusicBrainz

² O dicionário completo de dados pode ser acessado em: <<https://variations.indiana.edu/pdf/DML-metadata-elements-v1.pdf>>

em documentar todos os participantes de uma produção musical e como suas contribuições influenciam o produto final.

Por outro lado, apesar de apresentar uma modelagem robusta para a questão do gênero, não permite a identificação de movimentos musicais, eventos, artistas, instrumentos e outras características que desempenharam ou estão presentes na criação de um gênero específico. Adicionalmente, não contempla uma modelagem para representar como uma ideia musical pode influenciar e dar origem a novas ideias musicais, características extremamente importantes para as músicas brasileiras.

5.3 Implementação da biblioteca

Como desfecho almejado, delinea-se a criação de um protótipo funcional da biblioteca digital de músicas brasileiras, fundamentada nos conceitos discutidos sobre sistemas paralelos e de recomendação, ontologia e a base de dados modelada. Nesse contexto, também ressalta-se a importância da elaboração de utilitários destinados à coleta de dados, essenciais para o treinamento dos modelos, bem como para o preenchimento da base de dados.

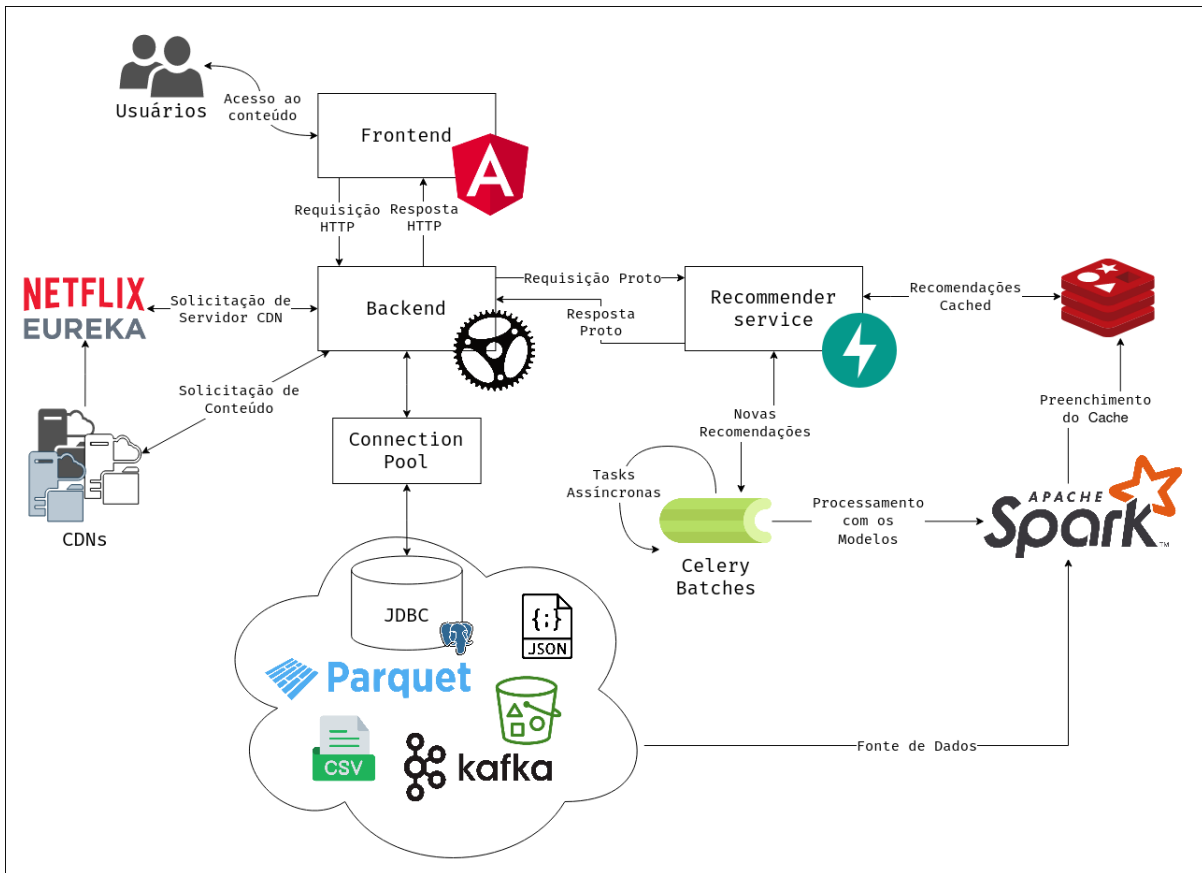
5.3.1 Arquitetura

A arquitetura do sistema, conforme ilustrado na Figura 40, é composta por três camadas distintas: o *front-end*, encarregado da interação com o usuário; o *back-end*, responsável pela comunicação tanto com o banco de dados quanto com o serviço de recomendações; e, por fim, a camada de recomendação. Na totalidade, foram utilizadas sete tecnologias distintas, evidenciando, no contexto da construção da aplicação, a presença de pelo menos três modalidades distintas de comunicação entre os serviços. Essa abordagem visa a edificação de uma biblioteca digital altamente escalável, distribuída e robusta para a geração de recomendações.

Apesar da Figura 40 representar a biblioteca digital, nem todas as técnicas foram implementadas, notadamente três delas: a adoção do gRPC, a integração de múltiplas fontes de dados e a utilização de múltiplas CDNs. A escolha pela utilização do HTTP, em contraposição ao gRPC, na comunicação *back-end-recommender-service* deve-se à facilidade de desenvolvimento proporcionada por essa tecnologia, uma vez que o tempo destinado à implementação do sistema precisou ser distribuído entre várias camadas e a modelagem do banco de dados.

Por sua vez, a decisão de não utilizar múltiplas fontes de dados para alimentar a recomendação no Spark foi motivada pela ausência de dados suficientes que justificassem tal abordagem, optando-se, assim, pelo uso exclusivo do banco de dados. Quanto ao conceito de múltiplas CDNs, sua não implementação baseou-se em dois motivos principais:

Figura 40 – Arquitetura da biblioteca digital.



Fonte: Autores

a limitada quantidade de músicas e imagens disponíveis para armazenamento e a escassez de recursos para a implementação de servidores em diversas partes do globo. Contudo, para suprir a necessidade de espaço de armazenamento para essas mídias, recorreu-se ao Blobstorage da Azure, uma vez que a quantidade de dados existentes permitia acomodação eficiente nessa ferramenta.

Os formatos de comunicação entre as diversas partes do sistema, que merecem destaque, incluem a comunicação HTTP síncrona, o modelo *publish-subscribe* e a piscina de conexões. A escolha da comunicação HTTP baseou-se em sua facilidade em transmitir informações entre as camadas *front-end-back-end* e *back-end-recommender-service*. Nesse formato, as requisições e respostas são conduzidas mediante o uso de JSON, proporcionando facilidade na interpretação dos dados e compreensão das informações transmitidas.

É pertinente salientar que, no contexto da comunicação *front-end-back-end*, foram desenvolvidas duas formas distintas de requisições: requisições simples e *streaming* de dados. As requisições simples consistem em uma solicitação e uma resposta HTTP, sendo este formato empregado para operações como listagem, detalhamento e criação de objetos no banco de dados.

Por outro lado, o *streaming* de dados é utilizado para a transmissão de músicas quando o botão de *play* é acionado. Nesse método de comunicação, o *front-end* efetua múltiplas requisições, solicitando segmentos da mídia e construindo dinamicamente o conteúdo requisitado. Essa abordagem permite que o sistema opere de maneira mais eficiente, uma vez que somente uma parte do conteúdo total necessita ser baixada para disponibilização ao usuário.

A escolha do modelo *publish-subscribe*, por sua vez, possui uma fundamentação estratégica, advinda da necessidade de executar tarefas de maneira assíncrona. Essa decisão é motivada pela construção do *cache* das recomendações, visando reter o resultado do processamento por um intervalo de tempo. Em outras palavras, o modelo treinado para recomendações por conteúdo é acionado periodicamente para avaliar as músicas disponíveis na base de dados e gerar as respectivas recomendações. Com o *cache* estabelecido, o *back-end* consegue realizar requisições quase que instantaneamente.

Outro aspecto que reforça a justificativa para a adoção desse modelo reside na necessidade de realizar tarefas de forma paralelizada. Isso é evidente no caso da recomendação colaborativa, a qual precisa ser processada a cada acesso do usuário à aplicação, uma vez que utiliza informações sobre o que os usuários estão ouvindo para gerar as recomendações. Nesse contexto, a fila de mensageria armazena todas as tarefas que demandam execução por um *worker*, enquanto o Celery coordena como e quando cada tarefa será efetivamente executada. Vale ressaltar que nesse contexto o Redis também foi empregado como *broker* das filas do Celery, assegurando uma eficiente transferência de mensagens.

Finalmente, na camada do *backend*, foi adotada uma piscina de conexões (*connection pool*) com o propósito de assegurar eficiência e desempenho nas consultas ao banco de dados. Essa abordagem foi implementada para mitigar o impacto da criação de uma nova conexão no banco de dados, permitindo que as conexões sejam reutilizadas para a execução de diversas consultas e inserções que ocorrem no *backend*. Cumpre salientar que o Spark, por meio do PySpark, também realiza consultas no banco de dados, contudo, esse processo não utiliza um *pool* de conexões, resultando na criação de uma nova conexão sempre que há necessidade de acessar alguma informação.

5.3.2 Coleta de dados

Com o intuito de proporcionar um protótipo que se assemelhe o máximo possível a uma aplicação real e obter dados apropriados para o treinamento dos modelos de recomendação, decidiu-se realizar a coleta de dados (*web scraping*) de fontes externas. Dentro das várias fontes de informações disponíveis, o MusicBrainz foi selecionado como a fonte principal para a extração de dados, devido ao seu contexto semelhante e à sua abundância de informações.

Assim, foi conduzida uma extensa exploração na aplicação do MusicBrainz visando identificar quais dados poderiam ser extraídos e convertidos para o modelo da biblioteca digital. A análise final resultou na extração de dados relacionados a lugares e seus vínculos com artistas, álbuns, instrumentos, grupos, gêneros, subgêneros, influências e demais inter-relações entre os objetos mencionados anteriormente. Para realizar essa atividade, empregaram-se duas técnicas de extração: *web scraping* de páginas HTML e requisições à API pública. Em ambos os casos, o resultado consistiu em diversos *inserts*³ a serem efetuados na instância em execução do banco de dados.

No total, a etapa de coleta de dados resultou em um *subset* contendo aproximadamente 8800 músicas, 5500 lugares, 183 gêneros, 1400 artistas e 26 grupos musicais. Esses dados foram especialmente empregados no treinamento da recomendação por conteúdos, proporcionando uma variedade de informações relevantes. Adicionalmente à coleta automatizada, um conjunto de informações foi compilado manualmente para destacar as funcionalidades e interfaces presentes no *front-end*. Isso se fez necessário, uma vez que uma parte significativa das informações a serem armazenadas no banco de dados não pôde ser obtida por meio do MusicBrainz.

Finalmente, é relevante salientar que, entre os dados coletados, todos os lugares inseridos correspondem aos municípios brasileiros. No caso dos artistas, a maioria é composta por brasileiros, e aqueles que não são brasileiros foram adicionados por estarem envolvidos na composição, arranjo ou autoria das músicas de artistas brasileiros. Da mesma forma, os gêneros são predominantemente brasileiros, embora seus subgêneros e influências possam ter origem estrangeira.

5.3.3 Recomendação

Para a camada de recomendação, foram elaborados dois fluxos distintos: um voltado para a execução da filtragem colaborativa e outro dedicado à filtragem baseada no conteúdo. É relevante salientar que ambos os métodos visam à sugestão de músicas adicionais com base nos dados coletados. Em ambos os métodos, foram desenvolvidos e treinados modelos específicos, além de visões de bancos de dados (*views*) e fluxos particulares para atender aos requisitos de escalabilidade, eficiência e alta escala.

A filtragem colaborativa, conforme previamente discutido na seção 4.2.1, opera recomendando com base nas atividades dos usuários. Para a implementação desse modelo, tornou-se necessário criar um novo relacionamento entre a entidade "Pessoa" e "Performance" no banco de dados. Essa nova tabela foi designada para armazenar as preferências musicais de um usuário, registrando a quantidade de vezes que uma "Performance" específica foi acessada e escutada por esse usuário. Como mencionado anterior-

³ O conjunto de *inserts* pode ser acessado em: <<https://github.com/TCCJoaoThiago/db/blob/tcc/scrappers/results/results.tar>>

mente, foi criada uma *view*, mostrada na Figura 41, para facilitar o acesso e customização dos dados necessários à execução da recomendação.

Figura 41 – Código-fonte da criação da visão para filtragem colaborativa

```

1
2 CREATE OR REPLACE VIEW public."CollaborativeFilteringView"
3 AS
4 SELECT
5     pessoa.id AS user_id,
6     pessoa.identifier AS pessoa_id,
7     performance.identifier AS performance_id,
8     pessoa_escuta.quantidade
9 FROM "Pessoa_escuta_Performance" pessoa_escuta
10 JOIN "Pessoa" pessoa ON pessoa.id = pessoa_escuta.pessoa_id
11 JOIN "Performance" performance ON performance.id = pessoa_escuta.performance_id;
12

```

Fonte: Autores

O resultado da recomendação é armazenado em cache no Redis para consultas rápidas pelos usuários, conforme documentado na arquitetura do sistema. Se o usuário estiver autenticado, as recomendações são apresentadas na página inicial da interface, conforme exemplificado na Figura 42.

Figura 42 – Apresentação da recomendação por filtragem colaborativa ao usuário



Fonte: Autores, com imagens obtidas a partir do [Cover Art Archive](#)

Já a filtragem baseada em conteúdo sugere músicas por meio da semelhança entre suas características. O desenvolvimento dessa forma de recomendação é viabilizado pela integração entre o modelo de dados e a ontologia de músicas brasileiras, que descreve minuciosamente diversas características relacionadas aos artistas, músicas, gêneros e seus diversos inter-relacionamentos. As características utilizadas são selecionadas por meio da visão mostrada na Figura 43.

Ao aplicar os conceitos abordados na seção 4.2.2 e utilizar as informações disponíveis no banco de dados, obtidas por meio do *web scraping* no MusicBrainz, procedeu-se ao agrupamento dessas características em um vetor, os quais foram processados pelo algoritmo de treinamento. De maneira análoga à recomendação colaborativa, esses dados foram registrados em uma visão (Figura 43), e os resultados da recomendação foram armazenados em cache.

Dessa maneira, o nome, evento, interpretes, grupos musicais, membros do grupo, gênero e as influências do gênero da música são utilizados como característica da recomendação. Diferentemente da recomendação anterior, esta é apresentada no final da tela de detalhes de uma música, uma vez que é inteiramente dependente do conteúdo da mesma. A representação visual dessa recomendação é apresentada de maneira semelhante à apresentada na Figura 42.

5.3.4 Uso do sistema

O uso do sistema pelo usuário acontece por meio do navegador. Na página inicial, é possível visualizar as músicas mais populares, baseada na quantidade de acessos no sistema, e por região, como mostra a Figura 44.

Caso o usuário esteja autenticado, também é mostrado, na página inicial, a recomendação baseada na filtragem colaborativa, como já apresentado na Figura 44. Além disso, na barra de menu superior, é possível observar diversos comandos que facilitam a navegação do usuário, incluindo a opção "Músicas" e o ícone de lupa, que oferecem mecanismo de busca, e o botão "Entrar", que permite o usuário se autenticar.

Ao clicar na aba de "Músicas" no menu superior, o usuário é redirecionado para uma página que permite a realização de diversos filtros para encontrar uma música. Nessa página, é possível encontrar músicas pelo nome, data de realização, gêneros, interpretes e grupos musicais, como mostrado na Figura 45.

Ao clicar em uma música, o usuário é redirecionado para a página de detalhes. Como mostrado na Figura 46, é possível visualizar os dados da performance, ideia musical, autores, gêneros, letras, gravações, as músicas que ela cita, e que a citam, além da recomendação resultante da filtragem baseada em conteúdo.

Caso a gravação tenha um arquivo de áudio registrado, um botão com ícone sonoro é mostrado e, ao ser clicado, o áudio correspondente é tocado. Além disso, também há a função de *playlist*, onde as gravações com áudio são agrupadas sendo reproduzidas sequencialmente.

Além dos detalhes da música, também é possível obter os detalhes do gênero, incluindo suas influências, origens, subgêneros, elementos tradicionais e emblemáticos, músicas emblemáticas e as músicas mais populares deste gênero no sistema. De maneira

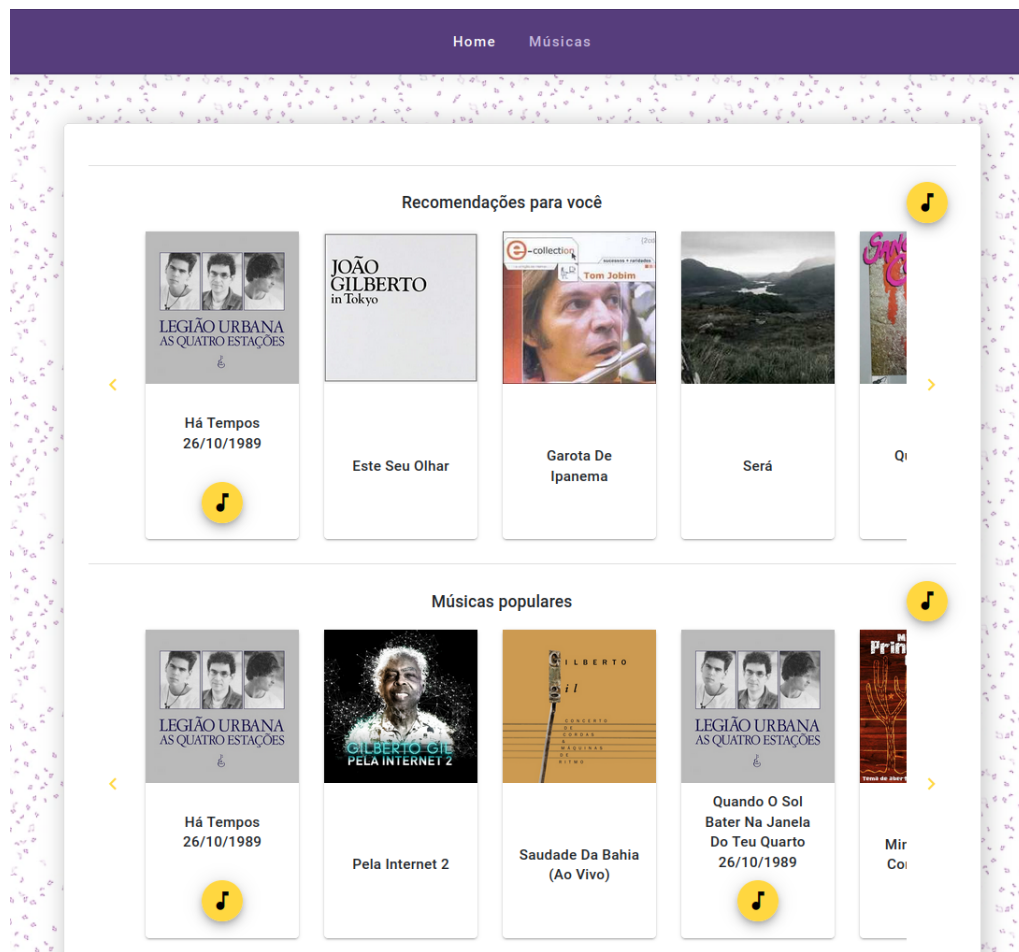
Figura 43 – Código-fonte da criação da visão para filtragem baseada em conteúdo

```

1
2 CREATE OR REPLACE VIEW public."ContentBasedView"
3 AS
4 SELECT
5     perf.id, ideia.nome, evento.nome AS evento, interpretes.nomes AS interpretes,
6     grupos.nomes AS grupos, grupos.membros, generos.nomes AS generos, generos.influencias
7 FROM "Performance" perf
8     JOIN "ExpressaoPerformance" expr ON expr.performance_id = perf.id
9     JOIN "IdeiaMusical" ideia ON ideia.id = expr.ideia_musical_id
10    LEFT JOIN "EventoMusical" evento ON evento.id = perf.evento_id
11    LEFT JOIN ( SELECT inter.performance_id, string_agg(pessoa.nome, ' ') AS nomes
12                FROM "Interprete" inter
13                 JOIN "Pessoa" pessoa ON pessoa.id = inter.pessoa_id
14                GROUP BY inter.performance_id) interpretes
15            ON interpretes.performance_id = perf.id
16    LEFT JOIN ( SELECT grupo_performa.performance_id,
17                string_agg(grupo.nome, ' ') AS nomes,
18                string_agg(membros.nomes, ' ') AS membros
19                FROM "GrupoMusical" grupo
20                 JOIN "GrupoMusical_performa_Performance" grupo_performa
21                  ON grupo_performa.grupo_musical_id = grupo.id
22                 LEFT JOIN ( SELECT membro.grupo_musical_id,
23                             string_agg(pessoa.nome, ' ') AS nomes
24                             FROM "MembroGrupoMusical" membro
25                              JOIN "Pessoa" pessoa
26                               ON pessoa.id = membro.pessoa_id
27                             GROUP BY membro.grupo_musical_id
28                            ) membros ON membros.grupo_musical_id = grupo.id
29                GROUP BY grupo_performa.performance_id) grupos
30            ON grupos.performance_id = perf.id
31    LEFT JOIN ( SELECT genero_classifica.ideia_musical_id,
32                string_agg(genero.nome, ' ') AS nomes,
33                string_agg(genero_influencia.nomes, ' ') AS influencias
34                FROM "Genero" genero
35                 JOIN "Genero_classifica_IdeiaMusical" genero_classifica
36                  ON genero_classifica.genero_id = genero.id
37                 LEFT JOIN ( SELECT genero_influencia.genero_pai_id,
38                             string_agg(genero_filho.nome, ' ') AS nomes
39                             FROM "Genero_influencia_Genero" genero_influencia
40                              JOIN "Genero" genero_filho
41                               ON genero_filho.id = genero_influencia.genero_filho_id
42                             GROUP BY genero_influencia.genero_pai_id) genero_influencia
43                ON genero_influencia.genero_pai_id = genero.id
44                GROUP BY genero_classifica.ideia_musical_id) generos
45            ON generos.ideia_musical_id = ideia.id;

```

Figura 44 – Página inicial



Fonte: Autores, com imagens obtidas a partir do [Cover Art Archive](#)

semelhante, é possível obter os detalhes do artista, grupo musical e do álbum. A visualização dessas interfaces é mostrada no Apêndice A.

Além da visualização, o usuário pode realizar inserções de novos dados. Para isso, é necessário estar autenticado e acessar o menu de cadastro no menu superior. Como mostra a Figura 47, é possível cadastrar artistas, álbuns, gêneros, grupos musicais e músicas.

Acessando o cadastro de gênero, por exemplo, o usuário terá acesso a um formulário, permitindo inserir seus dados e relacionamentos. Assim, de maneira manual, como mostra a Figura 48, é possível adicionar os dados dos subgêneros, influências, rótulo, movimento musical, origem, expoentes, músicas emblemáticas, tipos de evento musical característicos, eventos musicais emblemáticos, meios de performance e tipos de grupo musical tradicionais. Os outros cadastros são apresentados no Apêndice A.

Figura 45 – Busca por músicas

The screenshot displays a web application interface for searching music. At the top, there is a navigation bar with 'Home' and 'Músicas' links, a search icon, and an 'Entrar' button. The main section is titled 'Busca por Músicas' and contains several search filters:

- Nome:** A text input field for searching by name.
- Data:** A date range selector showing '01/01/1992 - 31/12/2023' with a calendar icon.
- Gêneros:** A text input field for filtering by genre.
- Intérpretes:** A text input field for filtering by artist.
- Grupos Musicais:** A filter dropdown menu currently showing 'Angra' with a close icon.


Below the filters is a yellow 'Buscar' button. The search results are displayed in a grid of 10 items, each with a cover image and a title:

- Freedom Call
- Queen Of The Night (Remixed Version)
- Reaching Horizons
- Stand Away (Orchestral Version)
- Painkiller
- Deep Blue (Edit Version)
- Chega De Saudade (Live)
- Never Understand (Live)
- Evil Warning
- Angels Cry

Fonte: Autores, com imagens obtidas a partir do [Cover Art Archive](#)

Figura 46 – Detalhes de uma música

Há Tempos



Data de realização
26/10/1989

Evento Musical
-

Interpretes

- Renato Russo (guitar, electric guitar, lead vocals, bass guitar, keyboard e harmonica)
- Marcelo Bonfá (drums (drum set), percussion, harmonica e bass drum)
- Dado Villa-lobos (guitar, electric guitar, bass guitar, percussion e mandolin)

Grupos Musicais
Legião Urbana


Álbuns
As Quatro Estações

Gênero

- Original
- Alternative Rock
- Folk Rock
- Punk

Letra

Gravações

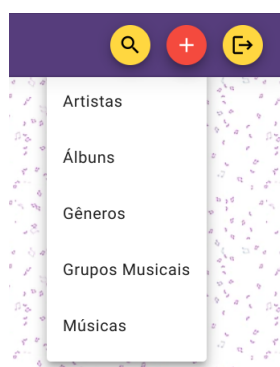


Recomendações

- Será
- Eu Sei
- Metrópole
- Tempo Perdido
- Edua

Fonte: Autores, com imagens obtidas a partir do [Cover Art Archive](#)

Figura 47 – Cadastros



Fonte: Autores

Figura 48 – Cadastro de gênero

Cadastro de Gênero

1 Gênero

Nome*

Subgênero de

Influenciado por

É um rótulo?

É um movimento musical?

2 Origem

3 Expoente do Gênero

4 Músicas Emblemáticas

5 Tipo de Eventos Musicais Característicos

6 Eventos Musicais Emblemáticos

7 Meios de Performance Tradicionais

8 Tipo de Grupo Musical Tradicionais

Salvar

Fonte: Autores

6 Considerações Finais

Como conclusão desta monografia, este capítulo apresentará os objetivos específicos deste trabalho, com o intuito de identificar aqueles que foram alcançados, os que foram parcialmente desenvolvidos e aqueles que não foram iniciados, sumarizado a conclusão do trabalho. Também serão discutidos as contribuições e possíveis trabalhos futuros, conforme os resultados obtidos.

6.1 Resultados alcançados

Diante da conclusão das tarefas propostas, torna-se essencial avaliar o progresso na execução dos objetivos específicos estabelecidos. Essa avaliação é de fundamental importância para mensurar o progresso das atividades planejadas ao longo do projeto. Dessa forma, por meio da Tabela 3, é possível obter uma visão atualizada do estado final de cada item delineado na Seção 1.3.2, indicando claramente se foram concluídos ou não.

Tabela 3 – Objetivos específicos do trabalho

Objetivo	Status	Artefatos Comprobatórios
Explorar o funcionamento de <i>clusters</i> como um requisito para aplicações de larga escala	Concluído	Seção 3.1
Explorar o funcionamento de tecnologias distribuídas como um requisito para a escalabilidade de aplicações de larga escala	Concluído	Seção 3.2
Explorar o funcionamento de orquestradores de software como um requisito para elasticidade de aplicações de larga escala	Concluído	Seção 3.4
Explorar alternativas tecnológicas para transmissão de <i>streaming</i> de áudio	Concluído	Seção 4.1
Explorar alternativas para sistemas de recomendação musical	Concluído	Seção 4.2
Explorar a modelagem de um banco de dados mediante uma ontologia	Concluído	Seção 5.2
Explorar o desenvolvimento de um sistema web segundo os requisitos de escalabilidade, elasticidade e larga escala	Concluído	Seção 5.3
Explorar o desenvolvimento de modelos de recomendação em tempo real	Concluído	Seção 5.3.3

Fonte: Autores

6.2 Contribuições

Este projeto possibilitou a construção de uma infraestrutura escalável, elástica e de larga escala, além da criação de uma biblioteca digital de músicas brasileiras. Essa infraestrutura ofereceu a oportunidade de experimentar vários serviços de processamento, armazenamento, orquestração e monitoramento. Da mesma forma, o desenvolvimento da biblioteca digital envolveu a criação de um *front-end*, *back-end* e, sobretudo, um serviço de recomendação, juntamente com a modelagem do banco de dados.

A experimentação no desenvolvimento de infraestruturas proporcionou uma compreensão mais profunda das vantagens e desvantagens dos diversos tipos de arquiteturas, como *bare-metal*, máquinas virtuais e contêineres. Por meio de testes de estresse e carga, tornou-se possível entender as razões por trás da maior eficiência nos ambientes *bare-metal*, ao mesmo tempo, em que se pôde apreciar as facilidades e a escalabilidade oferecidas pelos sistemas que fazem uso de máquinas virtuais ou contêineres.

Concomitantemente, ao investigar o desenvolvimento das infraestruturas e suas comparações de desempenho, tornou-se viável avaliar a importância de dispor de uma ferramenta de monitoramento capaz de rastrear o estado das máquinas, especialmente durante condições de alta demanda. Essa abordagem revelou-se crucial para assegurar o cumprimento dos requisitos de escalabilidade e a capacidade de atender em larga escala.

É válido observar que grande parte da infraestrutura concebida não foi efetivamente implementada no protótipo da biblioteca digital, devido a limitações de tempo e recursos. Não obstante, esses conceitos fornecem uma base sólida para pesquisas futuras, visando aprimorar a infraestrutura do sistema por meio da adoção de tecnologias de virtualização ou orquestração, tornando-a mais escalável. Da mesma forma, a exploração de CDNs ou até mesmo a utilização de máquinas distribuídas em múltiplos *datacenters* contribuem para o desenvolvimento de uma arquitetura verdadeiramente em larga escala.

De maneira análoga à infraestrutura, com a execução da biblioteca digital, vários resultados merecem destaque. A implementação do *front-end* utilizando a biblioteca Angular proporciona aos usuários a facilidade de acesso a informações abrangentes sobre músicas, artistas e diversos outros elementos, além de receber recomendações personalizadas com base no histórico de audição ou no conteúdo acessado. Essa interface também possibilita a inserção de novas informações, contribuindo para a expansão e enriquecimento contínuo do banco de dados.

No contexto do *back-end*, como resultado, destaca-se a implementação de uma API Rust, que atua como uma camada de acesso tanto ao banco de dados, com comunicação implementada por meio de uma piscina de conexões para garantir maior eficiência, quanto ao serviço de recomendações. Este serviço é responsável pela geração de recomendações por meio dos modelos, de conteúdo e colaborativo, previamente treinados. É relevante

salientar que, nesse serviço, a implementação da arquitetura de filas assíncronas permite que o sistema opere paralelamente na geração de recomendações, além da utilização do Spark para distribuir e executar as tarefas entre o *cluster* de máquinas, alcançando os objetivos de escalabilidade e elasticidade.

Ademais, é de suma importância ressaltar a modelagem do banco de dados a partir da ontologia de músicas brasileiras proposta por [Padron \(2019\)](#). Por meio dos esforços aplicados durante a execução deste trabalho, foi possível desenvolver uma modelagem genérica e rica em informações, capaz de ser reutilizada por sistemas de músicas brasileiras que demandem um banco relacional para armazenar as diversas informações presentes nas músicas

Merece destaque que a escolha de uma ontologia como modelo fundamental para o desenvolvimento, em contraste com técnicas como entrevistas ou abordagens evolutivas, apresentou um desafio significativo. Iniciar o desenvolvimento da aplicação exigiu um entendimento prévio e aprofundado da ontologia de músicas brasileiras. Apesar da complexidade envolvida, esse processo facilitou a compreensão dos requisitos da aplicação, uma vez que o modelo foi construído com base em entrevistas realizadas com músicos e pesquisadores, que são os principais usuários do sistema.

Nesse contexto, a arquitetura desenvolvida revela-se satisfatória, atendendo aos requisitos e objetivos específicos desta monografia. Ela demonstra capacidade para operar em larga escala, de forma elástica e escalável, com alto desempenho e resistência a falhas. Além disso, a arquitetura atua como uma biblioteca digital, conseguindo armazenar informações de maneira ontologicamente bem descrita e gerar recomendações personalizadas aos usuários.

6.3 Trabalhos futuros

Em vista dos resultados alcançados e com o objetivo de aprimorar a infraestrutura, arquitetura, tecnologias empregadas e serviços desenvolvidos, alguns pontos podem ser considerados como direcionamentos para potenciais trabalhos futuros:

- Aprimoramento do sistema de monitoramento da infraestrutura e dos sistemas, visando abranger uma gama mais extensa de métricas e indicadores;
- Implementação da biblioteca digital em arquiteturas orquestradas, incorporando técnicas para escalonamento dinâmico e eficiente distribuição de recursos;
- Investigação de redes de fornecimento de conteúdo (CDNs) para otimização e melhoria na entrega de dados;

-
- Exploração de tecnologias para garantir a distribuição dos serviços em múltiplos *datacenters* visando a alta disponibilidade;
 - Exploração de múltiplas fontes de dados para enriquecer a alimentação do Spark e aprimorar a qualidade das recomendações geradas;
 - Avaliação e adoção de outros protocolos de comunicação, como o gRPC, entre os serviços, buscando alternativas que otimizem a eficiência da comunicação;
 - Aprimoramento das interfaces de interação com os usuários, visando proporcionar uma experiência mais intuitiva e envolvente;
 - Desenvolvimento e treinamento de um modelo capaz de preencher automaticamente as diversas informações presentes no banco de dados, contribuindo para a automação e enriquecimento contínuo do sistema.

Referências

- AHMAD, M. O. et al. Kanban in software engineering: A systematic mapping study. *Journal of Systems and Software*, v. 137, p. 96–113, 2018. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0164121217302820>>. Citado na página 23.
- ALVES, R. de C. Música: patrimônio imaterial do Brasil. 2011. Disponível em: <<https://revistalusofonia.wordpress.com/2011/02/16/musica-patrimonio-imaterial-do-brasil/>>. Citado na página 17.
- APACHE. Apache jmeter. 2022. Disponível em: <<https://jmeter.apache.org/>>. Citado na página 59.
- ASSOCIATION, S. N. I. Common raid disk data format specification. 2009. Disponível em: <https://www.snia.org/sites/default/files/SNIA_DDF_Technical_Position_v2.0.pdf>. Citado na página 29.
- BELL, R.; KOREN, Y.; VOLINSKY, C. Matrix factorization techniques for recommender systems. *Computer*, IEEE Computer Society, Los Alamitos, CA, USA, v. 42, n. 08, p. 30–37, aug 2009. ISSN 1558-0814. Citado na página 66.
- BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The semantic web. *Scientific American*, 2001. Citado 2 vezes nas páginas 74 e 75.
- BURNS, B.; BEDA, J.; HIGHTOWER, K. *Kubernetes Up Running*. 2nd. ed. [S.l.]: O'Reilly Media, Inc., 2019. 1-11 p. Citado na página 44.
- CARDWELL, N. et al. Bbr: Congestion-based congestion control. *ACM Queue*, v. 14, p. 20 – 53, 2016. Citado 2 vezes nas páginas 57 e 58.
- CASALICCHIO, E.; IANNUCCI, S. The state-of-the-art in container technologies: Application, orchestration and security. *Concurrency and Computation: Practice and Experience*, v. 32, 2020. Citado na página 42.
- CHAUDHURI, S.; KÖNIG, A. C.; NARASAYYA, V. Sqlcm: A continuous monitoring framework for relational database engines. *International Conference on Data Engineering*, v. 20, 2004. Citado na página 35.
- CHEN, C.-W. et al. Recsys challenge 2018: Automatic music playlist continuation. In: . Association for Computing Machinery, 2018. p. 527–528. Disponível em: <<https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge>>. Citado na página 67.
- CHEN, P. M. et al. Raid: High-performance, reliable secondary storage. *ACM Computing Surveys*, v. 26, 1994. Citado na página 29.
- CORPORATION, P. S. Progress chef. 2023. Disponível em: <<https://www.chef.io/>>. Citado na página 26.

- CRUZ, F. W. Necessidades de informação musical de usuários não especializados. 2008. Citado 2 vezes nas páginas 18 e 73.
- DANIELS, J. Server virtualization architecture and implementation. *XRDS*, v. 16, p. 8–12, 2009. Citado 2 vezes nas páginas 26 e 27.
- DOCKER. Swarm mode overview. 2023. Disponível em: <<https://docs.docker.com/engine/swarm/>>. Citado na página 43.
- DONGARRA, J. J. Performance of various computers using standard linear equations software. *ACM SIGARCH Computer Architecture News*, v. 20, p. 22–44, 1992. Citado na página 33.
- DONGARRA, J. J.; LUSZCZE, P.; PETITET, A. P. The linpack benchmark: Past, present, and future. 2002. Disponível em: <<https://www.netlib.org/utk/people/JackDongarra/PAPERS/hplpaper.pdf>>. Citado na página 33.
- EISLER, M.; LABIAGA, R.; STERN, H. *Managing NFS and NIS: Help for Unix System Administrators*. O'Reilly Media, 2001. ISBN 9780596551940. Disponível em: <<https://books.google.com.br/books?id=YwsMifqQp70C>>. Citado 2 vezes nas páginas 31 e 32.
- ELASTIC. Elasticsearch. 2023. Disponível em: <<https://www.elastic.co/pt/elasticsearch/>>. Citado na página 36.
- ELASTIC. Kibana. 2023. Disponível em: <<https://www.elastic.co/pt/kibana/>>. Citado na página 36.
- FOUNDATION, A. S. Apache Hadoop. 2023. Disponível em: <<https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html>>. Citado na página 37.
- GABRIEL, E. et al. Open mpi: Goals, concept, and design of a next generation mpi implementation. 2004. Disponível em: <<https://www.open-mpi.org/papers/euro-pvmmpi-2004-overview/euro-pvmmpi-2004-overview.pdf>>. Citado na página 40.
- GERHARDT, T. E.; SILVEIRA, D. T. *Métodos de pesquisa*. 1st. ed. [S.l.]: Ed. da UFRGS, 2009. 211-214 p. ISBN 978-85-386-0071-8. Citado 2 vezes nas páginas 19 e 20.
- GmbH, P. S. S. Proxmox. 2023. Disponível em: <<https://www.proxmox.com/en/>>. Citado na página 28.
- HA, S.; RHEE, I.; XU, L. Cubic: A new TCP-friendly high-speed TCP variant. *SIGOPS Oper. Syst. Rev.*, v. 42, p. 64–74, 2008. Citado na página 57.
- HASAN, M. H.; JAAFAR, J.; HASSAN, M. F. Monitoring web services' quality of service: a literature review. *Springer Science and Business Media LLC*, p. 835–850, 2012. Citado na página 34.
- HASHICORP. Consul. 2023. Disponível em: <<https://developer.hashicorp.com/consul>>. Citado na página 30.
- HAT, R. Ansible. 2023. Disponível em: <<https://www.ansible.com/>>. Citado na página 26.

- HAUSENBLAS, M. et al. Media fragments uri 1.0. *W3C*, 2012. Disponível em: <<https://www.w3.org/TR/media-frags/>>. Citado na página 56.
- HENDAYUN, M.; GINANJAR, A.; IHSAN, Y. Analysis of application performance testing using load testing and stress testing methods in api service. *Jurnal Sisfotek Global*, v. 13, p. 28–34, 2023. Citado na página 32.
- HEYMAN, J. et al. Locust. 2023. Disponível em: <<https://locust.io/>>. Citado na página 59.
- HINDMAN, B. et al. Mesos: A platform for Fine-Grained resource sharing in the data center. In: . USENIX Association, 2011. Disponível em: <<https://www.usenix.org/conference/nsdi11/mesos-platform-fine-grained-resource-sharing-data-center>>. Citado 2 vezes nas páginas 42 e 43.
- IFLA, S. C. of the. *ISBD: International Standard Bibliographic Description*. De Gruyter Saur, Berlin: [s.n.], 2011. Citado na página 75.
- IKHSANUDIN, R.; WINARKO, E. Parallelization of hybrid content based and collaborative filtering method in recommendation system with Apache Spark. *Indonesian Journal of Computing and Cybernetics Systems*, v. 13, n. 02, p. 149–158, 2019. Citado 2 vezes nas páginas 69 e 70.
- KARAU, H. et al. *Advanced Analytics with Spark*. 1st. ed. 1005 Gravenstein Highway North, Sebastopol: O’Reilly Media, Inc., 2015. 230-231 p. Citado 3 vezes nas páginas 17, 65 e 69.
- KAYE, R. Musicbrainz. 2000. Disponível em: <<https://musicbrainz.org/>>. Citado na página 84.
- KOUTSONIKOLA, V.; VAKALI, A. Ldap: framework, practices, and trends. *IEEE Internet Computing*, v. 8, n. 5, p. 66–72, 2004. Citado na página 32.
- KUBERNETES. Kubernetes. 2023. Disponível em: <<https://kubernetes.io/>>. Citado na página 44.
- LABS, G. Grafana. 2023. Disponível em: <<https://grafana.com/grafana/>>. Citado na página 36.
- LABS, G. Grafana k6. 2023. Disponível em: <<https://k6.io/>>. Citado na página 59.
- LABS, G. Prometheus. 2023. Disponível em: <<https://prometheus.io/>>. Citado na página 35.
- LANCKER, W. V. et al. HTTP adaptive streaming with media fragment uris. p. 1–6, 2011. Citado na página 56.
- LESKOVEC, J.; RAJARAMAN, A.; ULLMAN, J. *Mining of Massive Datasets*. 3st. ed. [S.l.]: Cambridge University Press, 2020. 319-321 p. Citado 4 vezes nas páginas 17, 66, 68 e 69.
- LISENA, P.; TRONCY, R. DOing REusable MUSical data (DOREMUS). 2017. Citado na página 82.

- MENG, X. et al. Mllib: Machine learning in apache spark. *CoRR*, 2015. Citado na página 39.
- NAIK, K. *Software Testing and Quality Assurance*. 1st. ed. New Jersey: John Wiley Sons, Inc., 2008. 211-214 p. ISBN 978-0-471-78911-6. Citado 4 vezes nas páginas 18, 33, 58 e 59.
- NETFLIX. Eureka at a glance. 2014. Disponível em: <<https://github.com/Netflix/eureka/wiki/Eureka-at-a-glance>>. Citado na página 30.
- ONGARO, D.; OUSTERHOUT, J. In search of an understandable consensus algorithm. In: . [S.l.: s.n.], 2014. p. 305–320. Citado na página 43.
- OPENMP. *OpenMP Application Programming Interface*. [S.l.: s.n.], 2021. Citado na página 40.
- PADRON, M. F. Uma proposta de modelo conceitual para representação da música popular brasileira. 2019. Citado 4 vezes nas páginas 76, 77, 82 e 98.
- PADRON, M. F.; CRUZ, F. W.; SILVA, J. R. D. F. Extending the ifla library reference model for a brazilian popular music digital library. *International Journal on Digital Libraries*, 2020. Citado 4 vezes nas páginas 17, 69, 76 e 77.
- PADRON, M. F. et al. A proposal of conceptual model for brazilian popular music. *Journal of Documentation*, Institute for Knowledge Organization and Structure, Inc., Lake Oswego, Oregon, USA, v. 79, n. 5, p. 1088–1109, 2023. Disponível em: <<https://doi.org/10.1108/JD-07-2022-0155>>. Citado 8 vezes nas páginas 17, 76, 77, 79, 116, 117, 118 e 119.
- PANTOS, R.; MAY, W. *HTTP Live Streaming*. RFC Editor, 2017. RFC 8216. (Request for Comments, 8216). Disponível em: <<https://www.rfc-editor.org/info/rfc8216>>. Citado na página 56.
- PETITET, A. P. et al. Hpl - a portable implementation of the high-performance linpack benchmark for distributed-memory computers. 2018. Disponível em: <<https://www.netlib.org/benchmark/hpl/index.html>>. Citado na página 33.
- RILEY, J. et al. Definition of a frbr-based metadata model for the indiana university variations3 project. 2007. Disponível em: <<https://dlib.indiana.edu/projects/variatiions3/docs/v3FRBRreport.pdf>>. Citado na página 84.
- RIVA, P. et al. *IFLA Library Reference Model A Conceptual Model for Bibliographic Information*. Netherlands: [s.n.], 2017. Citado na página 75.
- SANDBERG, R. et al. Design and implementation of the sun network filesystem. In: *USENIX Conference Exhibition*. [S.l.: s.n.], 1985. p. 119–130. Citado 2 vezes nas páginas 30 e 31.
- SCHULZRINNE, H. et al. Real-time streaming protocol version 2.0. *IETF*, 2016. Disponível em: <<https://datatracker.ietf.org/doc/html/rfc7826>>. Citado 2 vezes nas páginas 54 e 55.

- SEGUNDO, J. E. S.; CONEGLIAN, C. S.; LUCAS, E. R. de O. Conceitos e tecnologias da web semântica no contexto da colaboração acadêmico-científica: um estudo da plataforma vivo. 2017. Citado na página 74.
- SHVACHKO, K. et al. The Hadoop Distributed File System. In: *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies*. [S.l.: s.n.], 2010. Citado 2 vezes nas páginas 37 e 38.
- SILVA, V. G. da; KIRIKOVA, M.; ALKSNIS, G. Containers for virtualization: An overview. *Applied Computer Systems*, v. 23, p. 21–27, 2018. Citado na página 41.
- SODAGAR, I. The MPEG-DASH standard for multimedia streaming over the internet. *IEEE MultiMedia*, v. 18, n. 4, p. 62–67, 2011. Citado 2 vezes nas páginas 55 e 56.
- SYSTEMS, O. OpenNebula. 2023. Disponível em: <<https://opennebula.io/>>. Citado na página 28.
- TANENBAUM, A. S. *Computer Networks*. 5th. ed. [S.l.]: PRENTICE HALL, 2010. 713-720 p. ISBN 978-0-13-212695-3, 0-13-212695-8. Citado na página 52.
- TANENBAUM, A. S.; BOS, H. *Sistemas Operacionais Modernos*. 4th. ed. [S.l.: s.n.], 2016. 325-356 p. ISBN 978-85-4301-818-8. Citado na página 27.
- TANIAR, D.; RAHAYU, J. W. *Web semantics & ontology*. [S.l.]: Igi Global, 2006. Citado na página 73.
- TSAI, P.-W. et al. Network monitoring in software-defined networking: A review. *IEEE Systems Journal*, p. 3958–3969, 2018. Citado 2 vezes nas páginas 18 e 34.
- VAKALI, A.; PALLIS, G. Content delivery networks: Status and trends. *IEEE Internet Computing*, v. 7, 2003. Citado 2 vezes nas páginas 29 e 30.
- VARIAN, M. Vm and the vm community: Past, present, and future. *Office of Computing and Information Technology*, 1997. Citado na página 26.
- VERMA, A. et al. Large-scale cluster management at Google with borg. In: . [S.l.: s.n.], 2015. Citado na página 44.
- WELLS, D. Extreme programming: A gentle introduction. 2009. Disponível em: <<http://www.extremeprogramming.org/>>. Citado na página 23.
- WU, D. et al. Streaming video over the internet: Approaches and directions. *IEEE Transactions on Circuits and Systems for Video Technology*, v. 11, p. 282–300, 2001. Citado 2 vezes nas páginas 52 e 53.
- ZAHARIA, M. et al. Discretized streams: Fault-tolerant streaming computation at scale. 2013. Disponível em: <http://people.csail.mit.edu/matei/papers/2013/sosp_spark_streaming.pdf>. Citado 2 vezes nas páginas 38 e 39.
- ZHANG, Q. et al. A comparative study of containers and virtual machines in big data environment. In: . [S.l.: s.n.], 2018. p. 178–185. Citado na página 41.

ZHOU, Y. et al. Large-scale parallel collaborative filtering for the Netflix prize. In: FLEISCHER, R.; XU, J. (Ed.). *Algorithmic Aspects in Information and Management*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 337–348. ISBN 978-3-540-68880-8. Citado na página [66](#).

Apêndices

APÊNDICE A – Interfaces

Figura 49 – Detalhes do gênero

Folk Rock

Influenciado por
Rock Folk

Subgênero de
Rock Rural

Meios de performances tradicionais
Electric Guitar

Tipo de eventos musicais emblemáticos
-




Origens
-

Subgêneros
Folk Psicodélico Folk Progressivo


Tipo de grupos musicais tradicionais
Folk Rock

Eventos musicais emblemáticos
-

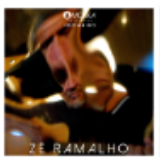
Expoêntes

Nome	Papel	Meios de performance	Parceiros
 Caetano Veloso	Interprete	guitar e lead vocals	-
 Gilberto Gil	Interprete	guitar, electric guitar, lead vocals e accordion	-
 Zé Ramalho	Interprete	-	-


Músicas emblemáticas




Tropicália (342 Amazônia Ao Vivo No Circo Voador)



Chão De Giz




Eternas Ondas




Pela Internet 2


Músicas populares




Há Tempos
26/10/1989




Feedback Song
For A Dying
Friend
26/10/1989



Pais E Filhos
26/10/1989



Quando O Sol
Bater Na Janela
Do Teu Quarto
26/10/1989



Há Tempos
26/10/1989

Fonte: Autores, com imagens de músicas obtidas a partir do [Cover Art Archive](#)

Figura 50 – Detalhes do álbum

As Quatro Estações

Data de lançamento
25/10/1989

Tipo
-

Elaboradores
Renato Russo Marcelo Bonfá Dado Villa-lobos


Músicas

Nome da Música	Data de Lançamento
Há Tempos	26/10/1989
Pais E Filhos	26/10/1989
Feedback Song For A Dying Friend	26/10/1989
Quando O Sol Bater Na Janela Do Teu Quarto	26/10/1989

Fonte: Autores, com imagens de músicas obtidas a partir do [Cover Art Archive](#) e [Legião Urbana](#)

Figura 51 – Detalhes do Artista


Renato Russo




Apelido Renato Russo	Nacionalidade Brasileiro
Data nascimento 27/03/1960	Genero Masculino
Grupos musicais Legião Urbana	Gêneros expoentes -


Músicas

<





Há Tempos







Pais E Filhos







Feedback Song
For A Dying
Friend






Quando O Sol
Bater Na Janela
Do Teu Quarto






E
L

>


Álbuns


<

>


As Quatro
Estações

Fonte: Autores, com imagens de músicas obtidas a partir do [Cover Art Archive](#) e [Legião Urbana](#)

Figura 52 – Detalhes do Grupo Musical

Legião Urbana





LEGIÃO URBANA


Data de criação

01/01/1982

Data de finalização


01/01/1996

Membros




Marcelo Bonfá
(membranophone)

Entrada: 1982
Saída: 1996




Renato Russo
(lead vocals)

Entrada: 1982
Saída: 1996



Dado Villa-Lobos
(guitar)


Entrada: 1983
Saída: 1996




Renato Rocha
(bass guitar)


Entrada: 1984
Saída: 1989

Músicas





Pais E Filhos
26/10/1989







Feedback Song
For A Dying
Friend
26/10/1989






Quando O Sol
Bater Na Janela
Do Teu Quarto
26/10/1989





Eu Era Um
Lobisomem
Juvenil
26/10/1989



Fonte: Autores, com imagens de músicas obtidas a partir do [Cover Art Archive](#) e [Legião Urbana](#)

Figura 53 – Cadastro de Álbum

Cadastro de Álbum

1 Álbum

Nome*
As Quatro Estações

Tipo*
Álbum

Data de publicação*
26/10/1989

Escolha a logo

Legião_Urbana_-_As_Quatro_Estações.jpg

Elaboradores

Elaborador(a) 1

Pessoa*
Renato Russo (Renato Russo) ✓

Elaborador(a) 2

Pessoa*
Marcelo Bonfá (Marcelo Bonfá) ✓

+ Adicionar mais

2 Músicas

Salvar

Fonte: Autores

Figura 54 – Cadastro de Música

Cadastro de Música

1 Performance

Nome* Data da realização*

Gêneros*

Original?
 Com improvisação?

Citações

Interpretes

Grupos Musicais

Evento Musical

Letra

Transcrição do Arranjo

Gravações

Fonte: Autores

APÊNDICE B – Manual de Instalação

Este manual de instalação visa apresentar um passo a passo com os comandos para a execução da aplicação implementada. A aplicação é composta pelo *back-end*, *recommender-service* e o *front-end* e podem ser executadas por meio de contêineres Docker, visando facilitar o processo de instalação.

Para instalar e configurar o *back-end*, juntamente com o banco de dados PostgreSQL, é necessário clonar o repositório do *back-end* utilizando o seguinte comando: “git clone https://github.com/TCCJoaoThiago/backend.git”. Na pasta do repositório, é necessário criar um arquivo chamado “.env”, configurando as variáveis de ambiente conforme o exemplo no arquivo “.env.example” mostrado na Figura 55.

Figura 55 – Variáveis de ambiente do *back-end*

```
1 DATABASE_URL="postgres://postgres:postgres@db:5432/postgres"
2 STORAGE_ACCOUNT="identificador da conta do storage"
3 STORAGE_ACCESS_KEY="chave de acesso do storage"
4 STORAGE_URL="url do storage"
5 RUST_BACKTRACE=1
6 RECOMMENDATION_URL="http://localhost:8000"
7 AUTH_SECRET_KEY="chave secreta para autenticação"
```

Fonte: Autores

Após configurado as variáveis de ambiente, é necessário inicializar o banco de dados, criando as tabelas e aplicando as migrações. Para isso, execute o comando: “docker-compose run --rm --entrypoint='diesel setup' backend”. Assim, é possível executar o *back-end* com o comando: “docker-compose up”.

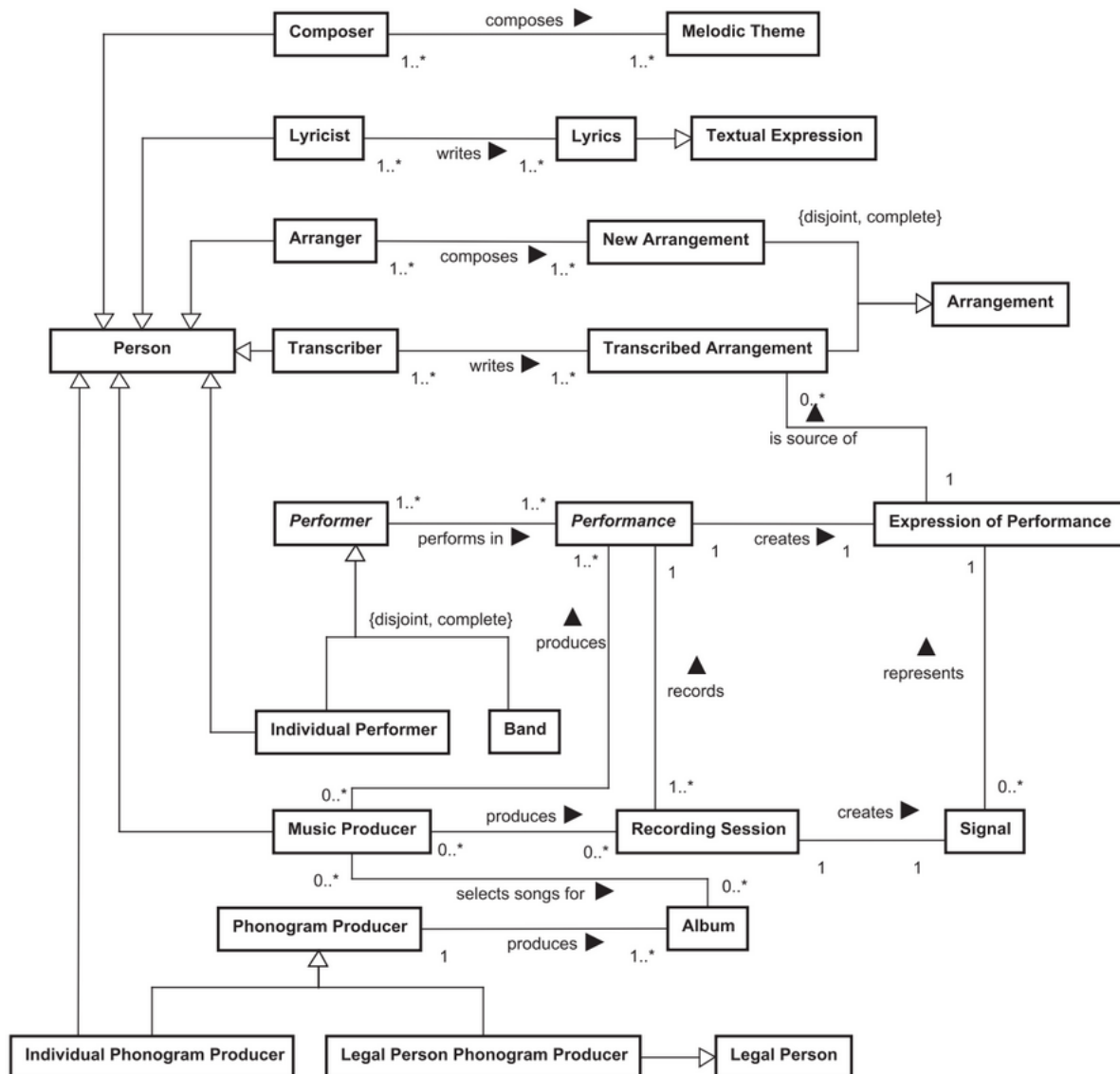
O *recommender-service* é composto por uma aplicação web, em FastApi, o serviço de cache, com o Redis, o serviço de tarefas assíncronas, com o Celery, e o serviço de recomendação, realizado pelo Apache Spark. Primeiramente, é necessário clonar o repositório com o comando “git clone https://github.com/TCCJoaoThiago/recommender-system.git”, e configurar o arquivo “.env”, utilizando o arquivo “.env.example” como exemplo. Por fim, para executar, realize o seguinte comando: “docker-compose up”.

Por fim, o *front-end* implementado com o Angular. Da mesma forma, clone o repositório com o comando “git clone https://github.com/TCCJoaoThiago/front.git”, e configure as variáveis de ambiente no arquivo “src/environments/environment.ts”. Por fim, execute a aplicação com o comando “docker-compose up”. Desta maneira, a aplicação do *front-end* estará disponível no endereço “http://localhost:4200”.

Anexos

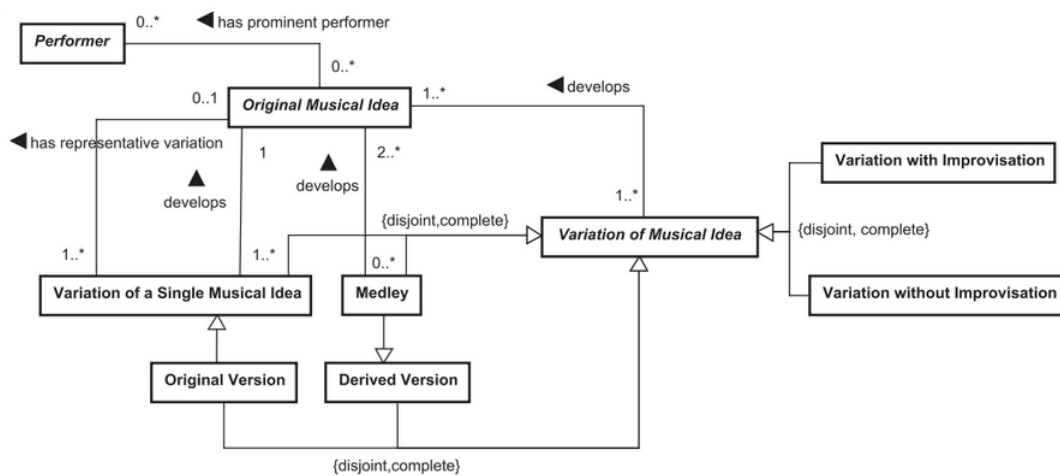
ANEXO A – Diagramas do modelo conceitual de música brasileira

Figura 56 – Diagrama do modelo conceitual para autoria



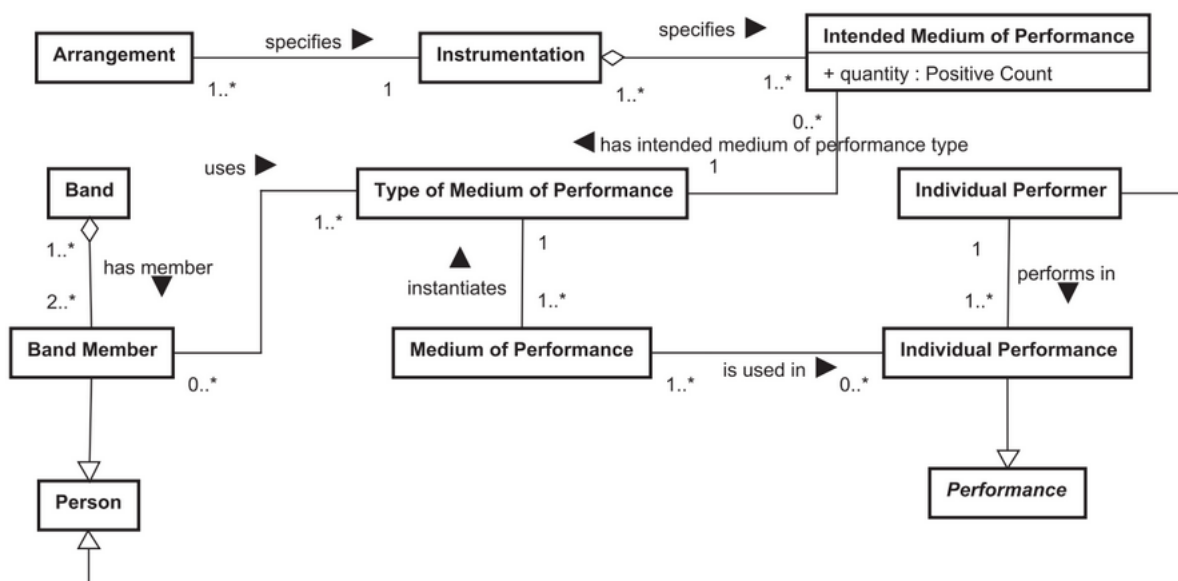
Fonte: Padron et al. (2023)

Figura 57 – Diagrama do modelo conceitual para versões de ideia musical



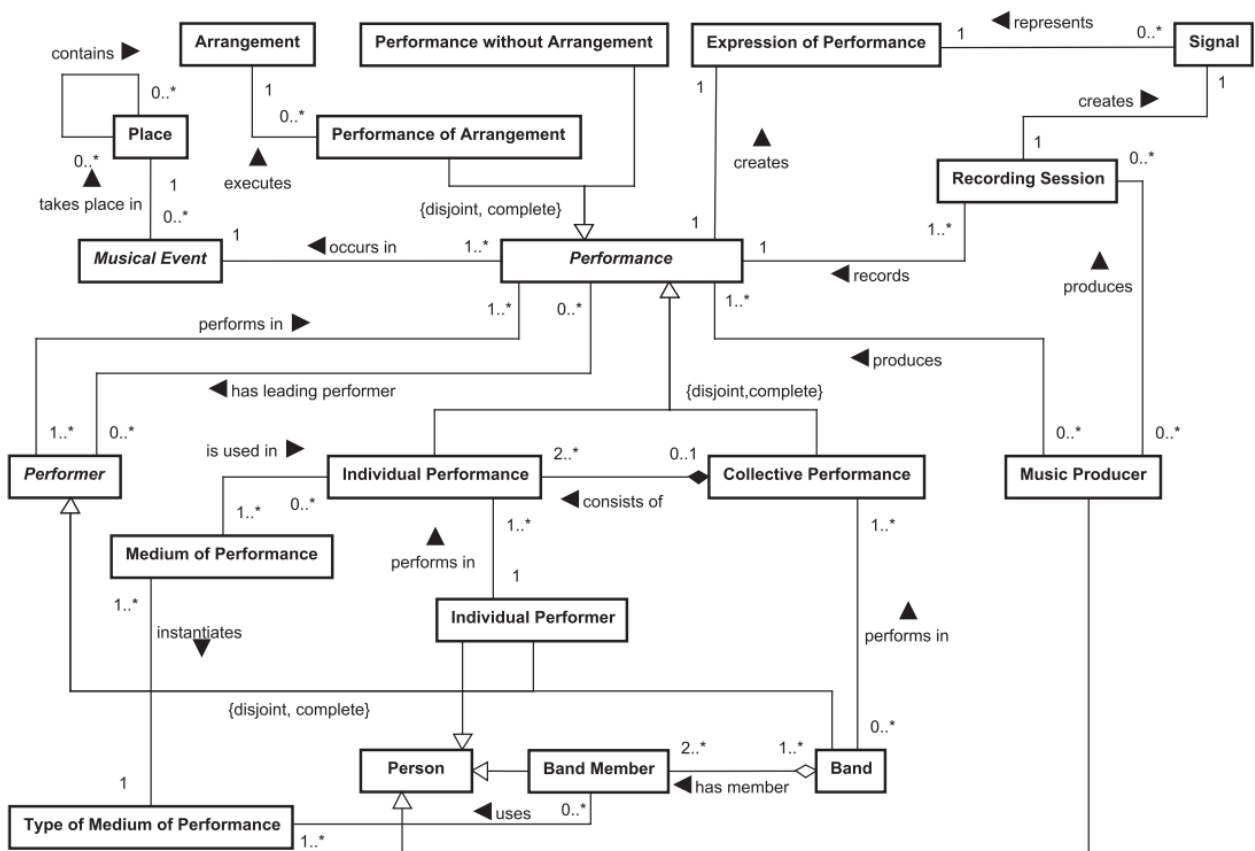
Fonte: Padron et al. (2023)

Figura 58 – Diagrama do modelo conceitual para instrumentação



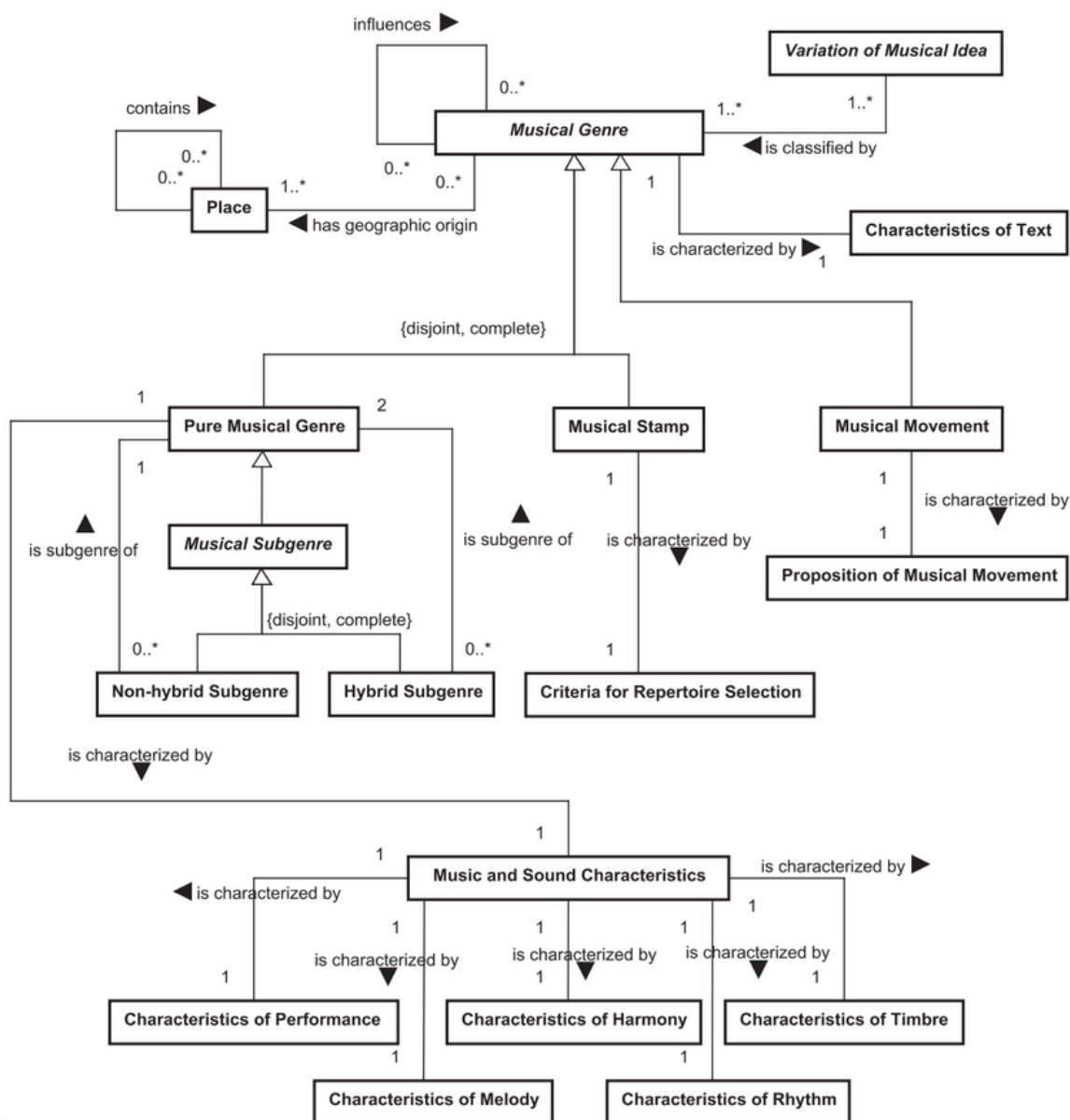
Fonte: Padron et al. (2023)

Figura 59 – Diagrama do modelo conceitual para performance.



Fonte: Padron et al. (2023)

Figura 60 – Diagrama do modelo conceitual para genero



Fonte: Padron et al. (2023)