UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Lucas Miguel Simões Ponce

Escalonamento Multiplataforma de Fluxos de Processamento em Ciência de Dados

Belo Horizonte
2026

Lucas Miguel Simões Ponce

Escalonamento Multiplataforma de Fluxos de Processamento em Ciência de Dados

Versão Final

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de Doutor em Ciência da Computação.

Orientador: Dorgival Olavo Guedes Neto

Belo Horizonte
2026

Ponce, Lucas Miguel Simões.

P792e Escalonamento multiplataforma de fluxos de processamento em ciência de dados [recurso eletrônico] / Lucas Miguel Simões Ponce – 2026.
1 recurso online (158 f. il, color.) : pdf.

Orientador: Dorgival Olavo Guedes Neto.

Tese (Doutorado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciências da Computação.
Referências: f. 118-128

1. Computação – Teses. 2. Computação de alto desempenho – Teses. 3. Sistemas multiplataforma -Teses. 4. Ciência de dados – Teses. I. Guedes Neto, Dorgival Olavo. II. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Computação. III. Título.

CDU 519.6*24(043)

Ficha catalográfica elaborada pela bibliotecária Irenquer Vismeg Lucas Cruz
CRB 6/819 - Universidade Federal de Minas Gerais - ICEx

UNIVERSIDADE FEDERAL DE MINAS GERAIS

ESCALONAMENTO MULTIPLATAFORMA DE FLUXOS DE PROCESSAMENTO EM CIÊNCIA DE DADOS

LUCAS MIGUEL SIMÕES PONCE

Tese defendida e aprovada pela banca examinadora constituída pelos Senhores(a):

Prof. Dorgival Olavo Guedes Neto - Orientador
Departamento de Ciência da Computação - UFMG

Prof. Ítalo Fernando Scotá Cunha
Departamento de Ciência da Computação - UFMG

Prof. Wagner Meira Júnior
Departamento de Ciência da Computação - UFMG

Profa. Jussara Marques de Almeida Gonçalves
Departamento de Ciência da Computação - UFMG

Prof. Philippe Olivier Alexandre Navaux
Instituto de Informática - UFRGS

Profa. Lúcia Maria de Assumpção Drummond
Instituto de Computação - UFF

Belo Horizonte, 26 de fevereiro de 2026.

Documento assinado eletronicamente por **Dorgival Olavo Guedes Neto**, **Professor do**

Magistério Superior, em 26/02/2026, às 18:50, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Italo Fernando Scota Cunha**, **Professor do Magistério Superior**, em 26/02/2026, às 18:56, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Jussara Marques de Almeida Goncalves**, **Professora do Magistério Superior**, em 26/02/2026, às 19:37, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Philippe Olivier Alexandre Navaux**, **Usuário Externo**, em 26/02/2026, às 21:30, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Wagner Meira Junior**, **Professor do Magistério Superior**, em 27/02/2026, às 11:37, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Lucia Maria De Assumpcao Drummond**, **Usuária Externa**, em 11/03/2026, às 14:04, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufmg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **4983117** e o código CRC **0E940EA8**.

Referência: Processo nº 23072.211048/2026-41 SEI nº 4983117

Agradecimentos

Agradeço, primeiramente, aos meus pais e ao meu tio Fernando, que, ainda na minha infância e com muito esforço, me presentearam com o meu primeiro computador. Mal podiam imaginar que aquele gesto despertaria em mim a curiosidade e o interesse pela tecnologia que, anos mais tarde, me trariam até aqui.

Aos meus pais e avós, pelo amor, pelo apoio incondicional e por todos os sacrifícios que fizeram ao longo da minha vida para que eu tivesse oportunidades e pudesse chegar até este momento.

À Julia e ao meu pequeno Miguel, por serem minha fonte diária de motivação, alegria e força. O apoio e o carinho de vocês tornaram essa caminhada muito mais significativa.

Se eu falasse de toda a minha família, certamente não daria espaço para expressar a gratidão que tenho por vocês, Toco, Marina, amo vocês.

Ao meu orientador, professor Dorgival, e ao professor Meira, por me acompanharem durante todos esses longos anos de mestrado e doutorado, pelas orientações, pela confiança e pelos ensinamentos que foram fundamentais para a realização deste e de outros diversos trabalhos.

Aos meus amigos, que estiveram presentes ao longo dessa jornada, em especial ao Walter Santos e ao Tiago, pelo apoio, pelas conversas e pela amizade.

Resumo

Sistemas de processamento de dados multiplataforma são ferramentas que permitem a execução eficiente de fluxos de trabalho complexos entre ferramentas de processamento heterogêneas. No entanto, as soluções existentes enfrentam limitações críticas em implantações reais: (*i*) as interfaces atuais dependem de abstrações de operadores simplificadas, como *map/filter/reduce*, que agregam diversas operações de alto nível como equivalentes, ignorando distinções computacionais cruciais e características dos dados; (*ii*) além disso, a falta de estatísticas prévias sobre os dados de entrada, combinada com a evolução dinâmica das propriedades dos dados ao longo da execução do fluxo de trabalho, leva a decisões subótimas, uma vez que os escalonadores existentes não consideram as mudanças no volume de dados e no esquema ao longo dos estágios do *pipeline*. Nesse contexto, propomos um arcabouço abrangente para escalonamento e execução multiplataforma de tarefas de Ciência de Dados utilizando DataFrame, uma abstração de dados moderna. Nossa proposta aborda essas limitações por meio de uma modelagem estendida baseada em grafos dos fluxos de trabalho. Essa estrutura de grafo captura: (*i*) a evolução dos volumes de dados e das características de esquema ao longo dos estágios do *pipeline*, (*ii*) os custos de execução heterogêneos entre as ferramentas disponíveis, e (*iii*) os custos de migração de dados entre ferramentas distintas. A metodologia proposta integra três contribuições técnicas principais: (*i*) um sistema de transformação baseado em regras para otimização de fluxos de trabalho que opera sobre a representação estendida do grafo; (*ii*) modelos de aprendizado de máquina para estimativa precisa dos tempos de execução; e (*iii*) um mecanismo de recalibração dinâmica que utiliza dados históricos de execução para se adaptar a cargas de trabalho em evolução. Implementamos o nosso modelo como uma extensão de um ambiente de programação visual existente para Ciência de Dados, demonstrando viabilidade prática em três configurações distintas de hardware e múltiplas ferramentas de processamento. A validação experimental demonstra que nossa abordagem possui 93% de acerto na seleção da ferramenta de processamento mais adequada e 75% de acerto na identificação de configurações ideais de hardware. Além disso, o sistema permite melhorias de desempenho de até 3,3 vezes em comparação com a execução tradicional utilizando uma única ferramenta, ao escalonar tarefas entre múltiplas ferramentas. A solução também exibe estabilidade em diversos ambientes de execução e a capacidade de evoluir por meio de aprendizado contínuo a partir de execuções históricas.

Palavras-chave: sistemas multiplataforma; previsor de desempenho; avaliação experimental.

Abstract

Cross-platform data processing systems are tools that can lead to efficient execution of complex workflows across heterogeneous processing tools. However, existing solutions face critical limitations in real-world deployments: (*i*) current interfaces rely on simplified operator abstractions such as *map/filter/reduce* that aggregate diverse high-level operations as equivalent, ignoring crucial computational distinctions and data characteristics; (*ii*) furthermore, the lack of prior statistics regarding input data, combined with the dynamic evolution of data properties throughout workflow execution, leads to suboptimal scheduling decisions, as existing schedulers fail to account for data volume and schema changes across pipeline stages. In this context, we propose a comprehensive framework for cross-platform scheduling and execution of Data Science tasks utilizing DataFrame, a modern data abstraction. Our approach addresses these limitations through an extended graph-based modeling of workflows. This enhanced graph structure explicitly captures: (*i*) the evolution of data volumes and schema characteristics across pipeline stages, (*ii*) heterogeneous execution costs across available tools, and (*iii*) data migration overheads between distinct tools. The proposed methodology integrates three key technical contributions: (*i*) a rule-based transformation system for workflow optimization that operates on the extended graph representation; (*ii*) machine learning models for accurate estimation of execution times; and (*iii*) a dynamic recalibration mechanism that leverages historical execution data to adapt to evolving workloads. We implemented our model as an extension of an existing visual programming environment for Data Science, demonstrating practical viability across three distinct hardware configurations and multiple processing backends. Experimental validation demonstrates that our approach achieves 93% accuracy in selecting the most suitable processing tool and 75% accuracy in identifying optimal hardware configurations. Furthermore, the system enables performance improvements of up to 3.3 times compared to traditional single-tool execution by scheduling tasks across multiple tools. The solution also exhibits robust stability across diverse execution environments and maintains the capability to evolve through continuous learning from historical executions.

Keywords: cross-platform systems; performance predictor; experimental evaluation.

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Sumário

Capítulo 1

Introdução

1.1 Contextualização

Ciência de Dados é um campo interdisciplinar que utiliza preparação de dados, análise algorítmica e técnicas de visualização de dados para analisar e compreender fenômenos reais (Saritha et al., 2021). Um fluxo de trabalho em Ciência de Dados geralmente consiste em várias tarefas de processamento de dados, como coleta, extração/transformação/carregamento (ETL), treinamento de modelos de aprendizado de máquina, etc. (Bisong, 2019). É uma área da ciência cada vez mais importante para os mais diversos campos; por exemplo, tem sido usada por empresários para tomar decisões baseadas em dados (Selinger, 2012), por jornalistas para pesquisar terabytes de documentos vazados por governos e corporações (Hernando, 2017) e por especialistas de marketing que processam extensos registros de usuários para obter informações sobre seu comportamento e preferências (Schrank, 2015).

Com um volume crescente de dados e o aumento da complexidade das análises, diversas abstrações de dados, ferramentas e plataformas têm sido desenvolvidas para atender às necessidades atuais da comunidade, que incluem desde a simplificação do processo de desenvolvimento, como a criação de sistemas de prototipação visual (Santos et al., 2017; Mahapatra et al., 2018), aumento de funcionalidades (McKinney, 2010; Pedregosa et al., 2011) e, principalmente, a melhoria de desempenho (Armbrust et al., 2015; Carbone et al., 2015; Tejedor et al., 2017; NVIDIA RAPIDS, 2025a). Um desses exemplos é o crescente suporte, por parte das ferramentas, às abstrações de dados de alto nível, como a de DataFrame, que provê um formato tabular e conveniente para representar, estruturar e analisar dados a partir de um conjunto de operadores (Armbrust et al., 2015; Petersohn et al., 2020; McKinney, 2022). No entanto, mesmo que existam algumas similaridades entre muitas ferramentas atuais, por causa da diversidade dos cenários, muitas delas possuem requisitos completamente diferentes, por exemplo: consistência versus escalabilidade; escalabilidade versus complexidade algorítmica; processamento em lote versus em *streaming*; ou acesso baseado em chave versus consultas analíticas.

Nesse contexto, vivemos uma época privilegiada com dezenas de ferramentas à nossa disposição. A maioria dessas ferramentas modernas possui uma interface seme-

lhante baseada em linguagem funcional, que é útil para encapsular vários aspectos da programação para o usuário (McKinney, 2010; Armbrust et al., 2015; Carbone et al., 2015). Porém, escolher uma única ferramenta em detrimento de outras nem sempre é fácil, ou mesmo possível. Aspectos como volume dos dados, conjunto de operações disponíveis, integração com outras ferramentas necessárias em um fluxo de dados e desempenho são alguns dos muitos aspectos a serem considerados nessa decisão. Uma abordagem popular utilizada pela comunidade no apoio à tomada de decisão é a utilização de *benchmarks* (Nambiar et al., 2006; Huang, S. et al., 2010; Bajaber et al., 2020; Transaction Processing Performance Council, 2024a). No entanto, diversos trabalhos nesse segmento têm como conclusão a não existência de um vencedor claro na maioria dos casos e, muitas vezes, os resultados diferem entre si (Watson et al., 2017; Dugré et al., 2019; Kassela et al., 2019). Diferentes tipos de dados, detalhes de configuração de operações em um fluxo e, até mesmo, a versão das ferramentas, podem influenciar em um resultado.

Nesse sentido, várias frentes de pesquisa vêm reconhecendo a importância dos sistemas de processamento de dados multiplataforma (do inglês, *cross-platforms*) como uma solução sistemática que permita a execução eficiente de fluxos de análises entre as opções de ferramentas (Gog et al., 2015; Rong et al., 2022; Beedkar et al., 2023; Chen, Q. et al., 2023). O principal objetivo desses sistemas é prover uma interface de prototipação na qual usuários sejam capazes de expressar as tarefas de forma genérica, cabendo ao sistema avaliar o fluxo e decidir a melhor ferramenta de processamento para cada tarefa. Caberia à ferramenta, então, lidar com os aspectos de criação de código-fonte, migração e submissão da execução.

1.2 Desafios

A construção de sistemas multiplataformas apresenta dois desafios principais: como representar o fluxo de tarefas e como implementar a escolha das ferramentas. Em relação ao primeiro desafio, questões como a interface de prototipação, a abstração de dados suportada e a abordagem de conversão de código entre as ferramentas são alguns dos aspectos que os desenvolvedores precisam levar em consideração ao criar esses sistemas. A prototipação de fluxos pode ser realizada por meio de linguagens de programação (The Fugue Development Team, 2022; Apache Software Foundation, 2023a) ou por interfaces visuais (Kranjc et al., 2012; Santos et al., 2017), tendo cada uma delas seus compromissos. As interfaces via linguagens de programação, por exemplo, são mais flexíveis, mas tornam o processo de interpretação e conversão do fluxo mais complexo, uma vez que os usuários podem representar uma determinada atividade de várias formas. Por outro lado, as interfaces visuais, embora forneçam um maior controle sobre a execução para os sistemas multiplataformas e simplifiquem o processo de codificação para os usuários,

possuem menos flexibilidade.

A escolha da interface está diretamente relacionada à abstração de dados e à abordagem de conversão de código. Em sistemas multiplataformas, o mais comum é se limitar a uma abstração de dados baseada em linguagem funcional, como a de DataFrame ou uma abstração no modelo MapReduce (White, 2015), que disponibilizam operadores como `map/filter/reduce`. Pois, suportar e tornar compatível todos os tipos de abstrações que uma ferramenta disponibiliza em um sistema multiplataforma é uma tarefa complexa e que, em muitos casos, exigiria grande esforço para implementar funcionalidades ausentes entre as ferramentas. Abstrações baseadas em linguagem funcional permitem encapsular complexidades da interpretação do fluxo como laços de interação e desvios, a partir de blocos de tarefas, e esconder as diferenças de funcionamento das ferramentas como processamento *single-core* versus *multi-core*, facilitando o processo de conversão de código de uma ferramenta para outra. Sistemas desse tipo podem escolher utilizar uma interface de prototipação genérica como uma interface visual (p.ex., o Lemonade (Santos et al., 2017)), uma linguagem neutra (p.ex., o Apache Beam (Apache Software Foundation, 2023a)) ou uma das linguagens utilizadas pelas ferramentas suportadas, como é o caso do Musketeer (Gog et al., 2015).

O desafio da recomendação da melhor ferramenta para um determinado tipo de tarefa também é complexo. A decisão pode ser a partir de uma limitação prática (p.ex., disponibilidade da operação) ou a partir de uma decisão oportunista, geralmente baseada em uma função de custo mínimo, que pode ser definido pelo tempo de execução, custo monetário para a alocação de recursos em nuvem ou outras métricas como o custo energético. Quando o desempenho é o objetivo principal, os sistemas multiplataforma dependem geralmente de uma função de custo, modelada por previsores de desempenho, ou de um conjunto de regras predefinidas.

Embora populares no passado, o escalonamento das tarefas baseado puramente em um conjunto de regras tende a simplificar as decisões ao utilizar regras criadas a partir de experimentos passados ou decisões manuais, tornando os sistemas menos dinâmicos. Soluções atuais vêm se baseando na criação de um modelo para estimar o custo do melhor plano de execução (isto é, o melhor cenário de execução). E, nesse contexto, a qualidade desses modelos de previsão está diretamente relacionada à capacidade do sistema em representar as características das operações e dos dados relacionados ao fluxo e dos recursos computacionais e ferramentas disponíveis.

Apesar de todos os esforços no desenvolvimento de sistemas multiplataformas, as soluções atuais falham em quatro pontos principais: a inclusão de novas interfaces de prototipação potencialmente limitadas; as soluções usam ferramentas restritas; a manutenção e evolução são complexas; e, principalmente, a qualidade das estimativas é baixa. Devido à complexidade de manter uma compatibilidade entre as linguagens de codificação, as soluções existentes geralmente disponibilizam uma interface própria de codificação, res-

ponsável, internamente, pelo mapeamento dos operadores em código-fonte para as diversas ferramentas suportadas. Mesmo soluções que utilizam uma linguagem neutra, que visa facilitar programas multiplataforma, impõem curvas de aprendizagem elevadas e necessitam conversões dos fluxos originais para essa nova sintaxe proposta. Mais ainda, os operadores fornecidos por interfaces desse tipo geralmente são restritos às ferramentas e operações implementadas pelos criadores da solução. A adição de uma nova ferramenta de processamento ao sistema, ou até mesmo a atualização de uma nova versão, que geralmente possui novas funções e operadores, pode exigir dos usuários um grande conhecimento para adicionar o suporte às novas funcionalidades.

No entanto, em função dos sistemas serem geralmente implementados à parte do processo de codificação e de execução, como uma ferramenta adicional, esses sistemas não têm informações suficientes dos dados de entrada e, por isso, focam suas estimativas em modelos de custo simples baseados em número de linhas, operações e localidade dos dados de entrada. Suas estimativas, por exemplo, não levam em consideração que um conjunto de dados de entrada, inicialmente volumoso, pode ser reduzido drasticamente durante a execução. Com isso, operações subsequentes podem ser executadas em outros tipos de ferramentas de processamento além daquelas focadas em grandes volumes de dados, por exemplo.

1.3 Proposta de tese

Neste trabalho, argumentamos que, em vez de criar um sistema multiplataforma como uma estrutura nova e separada, sistemas desse tipo podem ser adicionados a sistemas de codificação e submissão de tarefas já existentes, como o Lemonade, para criar uma solução melhor, onde os usuários se beneficiam tanto do uso da interface amigável quanto de ganhos de desempenho com o uso de um escalonador de múltiplas ferramentas. Além disso, sistemas de prototipação desse tipo permitem um gerenciamento maior das informações relacionadas à execução, como características dos dados de entrada, recursos computacionais e as operações suportadas. Essas informações muitas vezes não são suportadas pelos sistemas multiplataformas atuais, limitando, assim, suas capacidades de inferência.

Nosso objetivo com este trabalho é mostrar que é possível modelar e implementar um sistema multiplataforma para execução de fluxos de Ciência de Dados. Tais sistemas devem ser capazes de, dado um fluxo de dados com uma ou mais tarefas, recomendar a configuração de cada ferramenta onde cada tarefa deverá ser executada. Para isso, listamos os seguintes objetivos específicos para este trabalho:

1. Propor uma modelagem genérica para estimar o custo de execução de um fluxo de tarefas entre diversas possibilidades de ferramentas de execução e configurações de

recursos computacionais. Essa modelagem deve ser robusta e levar em consideração a evolução da carga ao longo da execução.

2. Apresentar o desenho e a implementação de um sistema multiplataforma que adote a proposta do modelo e que trate dos desafios do sistema relativos à eficiência.

3. Apresentar um amplo estudo de avaliação da qualidade de recomendação das configurações de execução propostas pelo sistema, considerando diversos cenários de dados de entrada e fluxos de tarefas, e comparando nossos resultados com os de outras abordagens.

1.4 Contribuições

Como principais contribuições deste trabalho, destacamos:

1. **Modelo de evolução da carga de entrada**: implementação de uma modelagem que estime, antes do início da execução, as principais estatísticas da carga de entrada ao longo do fluxo de tarefas. Além de poder ser utilizada para auxiliar na estimativa do custo de execução, estimativas sobre o dado, pré-execução, podem ser utilizadas em outros cenários, como na explicabilidade de desempenho.

2. **Modelagem para a estimativa do custo de execução**: implementação de um modelo genérico que estime o custo de execução de operações de alto nível de Ciência de Dados, a partir de informações de parâmetros da tarefa e estimativas dos dados de entrada. O modelo de custo busca estimar o desempenho de um plano de execução em relação aos demais planos, levando em consideração as previsões do tempo de execução de cada tarefa. Para estimar o tempo de execução, consideramos informações sobre a configuração de hardware disponível, os dados de entrada e os parâmetros de execução definidos pela operação.

3. **Metodologia de recomendação de planos de execução**: implementação de sistema de recomendação híbrido, que se baseia em um modelo de custo aliado a um conjunto de regras. Essas regras podem ser utilizadas para limitar os cenários de busca (p.ex., ferramentas que precisam carregar os dados completos na memória principal pode ter problemas para executar cenários onde a memória disponível em uma máquina seja menor que o tamanho esperado do dado) ou para forçar um comportamento específico para um usuário (p.ex., se o usuário quer explicitamente executar um trecho em uma ferramenta devido a diferenças de implementação de algum algoritmo). O sistema será responsável por enumerar planos de forma eficiente, aplicar regras, agregar custos estimados e ranquear os melhores planos;

4. **Extensão do Lemonade para Interface única de prototipação**: um dos desafios de criar fluxos de trabalho para diferentes ferramentas é o suporte de geração de código. Para o objetivo proposto, desenvolvemos uma extensão da ferramenta Lemonade para permitir o suporte a execução em sistemas multiplataformas.[1] Dessa forma, fluxos de tarefas criados pela ferramenta podem ser executados de forma mais eficiente, utilizando uma ou mais ferramentas de execução.

1.5 Organização

Para esse objetivo, o restante deste texto está organizado da forma que segue. O capítulo 2 apresenta os principais conceitos necessários ao desenvolvimento de sistemas multiplataformas como as abordagens para estimativa do desempenho dos planos de execução, os compromissos na escolha das abstrações de dados e das interfaces de prototipação. A solução proposta é apresentada no capítulo 3 e os resultados experimentais são avaliados no capítulo 4. Por fim, no capítulo 5, apresentamos as conclusões e discutimos os trabalhos futuros.

[1] Disponível em: https://github.com/lucasmsp/lemonade-cross-platform

Capítulo 2

Conceitos e trabalhos relacionados

Neste capítulo, abordamos alguns dos principais conceitos e trabalhos relacionados à criação de sistemas multiplataformas. Na seção 2.1, discutimos com mais detalhes algumas abordagens utilizadas no escalonamento das tarefas, bem como as abordagens para a previsão de desempenho. A definição formal da abstração de DataFrame e as diferenças de sua implementação entre diversas ferramentas são apresentadas na seção 2.2. Tais diferenças influenciam a complexidade de criação de sistemas multiplataformas genéricos. Por fim, na seção 2.3, apresentamos algumas abordagens na criação de sistema de interfaces de prototipação relacionando seus compromissos.

2.1 Sistemas de processamento multiplataforma

Atualmente, dados são processados e gerenciados em uma ampla variedade de contextos, muitas vezes com requisitos muitas vezes completamente diferentes (por exemplo, consistência versus escalabilidade, processamento em tempo real versus em lote, cargas de trabalho intensivas em escrita versus leitura, ou ainda acesso baseado em chave versus consultas analíticas). Como consequência, nas últimas duas décadas, diversos novos sistemas de gestão e processamento de dados foram desenvolvidos, cada um projetado com diferentes cenários de aplicação em mente. Neste cenário, cabe ao usuário a responsabilidade de escolher qual sistema melhor atende às necessidades específicas de seu caso de uso.

Diante dessa heterogeneidade, a integração entre esses sistemas tornou-se um tema recorrente de pesquisa, sendo explorado por meio de diferentes abordagens. A tabela 2.1 apresenta um resumo das principais terminologias e conceitos relacionados a esse tema. Um exemplo clássico de abordagem mais restrita de integração é a técnica de federação de sistemas, muito utilizada na conexão entre ferramentas de repositório de dados (como o Apache Hive (Camacho-Rodríguez et al., 2019) e o Apache Druid (Yang et al., 2014)) e motores de processamento (como o Apache Flink (Carbone et al., 2015) ou o Apache Spark (Armbrust et al., 2015)). Nesse modelo, a federação permite que uma única consulta seja capaz de acessar múltiplas fontes de dados, mesmo que esses dados estejam armazenados em locais, formatos ou tecnologias distintas, e retorne um resultado único e

consolidado. Em geral, cada parte da consulta federada é executada localmente em seu sistema de origem, minimizando o custo de movimentação dos dados para um repositório centralizado. Cabe ao motor de processamento federado apenas a responsabilidade de orquestrar a execução das diferentes partes da consulta. No entanto, o principal desafio dessa abordagem é o nível limitado de integração entre os sistemas, uma vez que essa integração se dá apenas entre o repositório de dados e o motor de processamento escolhido pelo usuário.

Tabela 2.1: Diferenças entre abordagens na integração de sistemas.

Abordagem	Foco	Objetivo
Execução federada	Execução (Técnica)	Executar uma única consulta sobre múltiplas fontes
Persistência poliglota	Repositórios (Técnica)	Escolher o melhor repositório para cada tipo de consulta
Polystore	Repositórios (Sistema)	Integrar vários bancos diferentes como um único
Multiplataforma	Execução (Sistema)	Suportar aplicações que executam em várias ferramentas ou hardwares

Uma outra abordagem bastante explorada na literatura é a dos chamados sistemas de persistência poliglota (do inglês, *Polyglot Persistence* ou *Polyglot Data Store*). O termo "poliglota", originalmente utilizado na área de compiladores desde a década de 90, referia-se à capacidade dos sistemas de suportarem múltiplas linguagens de programação (DeMichiel et al., 1993; Nystrom et al., 2003). Mais recentemente, esse conceito foi incorporado ao contexto de sistemas gerenciadores de dados, passando a designar sistemas capazes de suportar múltiplos tipos de repositórios de dados em um mesmo sistema de aplicação (Sadalage et al., 2012; Kiehn et al., 2022). Nessa abordagem, a principal ideia é utilizar diferentes bancos ou repositórios de dados de acordo com as características específicas dos dados e das operações a serem realizadas, em vez de forçar que todos os dados se adequem a um único modelo ou tecnologia. Internamente, cada repositório é utilizado de forma independente, sendo responsável por uma parte específica da aplicação ou do sistema. Assim, a cada requisição do usuário, o sistema avalia qual repositório é o mais adequado para armazenar ou processar determinada informação. Dessa forma, diferentes bancos podem ser especializados, por exemplo, no armazenamento de dados estruturados, semi-estruturados ou em grafos, permitindo uma maior flexibilidade e otimização no processamento e gerenciamento dos dados.

Uma evolução das abordagens anteriores é representada pelos sistemas *polystore*. Esses sistemas têm como objetivo integrar múltiplos bancos de dados e repositórios heterogêneos, provendo uma camada de abstração única que permite ao usuário acessar e consultar dados armazenados em diferentes tecnologias de forma transparente (Khine et

al., 2019). Em geral, sistemas desse tipo combinam técnicas de execução federada com estratégias típicas dos sistemas de persistência poliglota, buscando tirar proveito das vantagens de cada abordagem. Entre os principais exemplos de sistemas *polystore* destacam-se o Apache Drill (Apache Software Foundation, 2023b) e o BigDAWG (Duggan et al., 2015). No Apache Drill, o foco está em oferecer uma interface unificada baseada em SQL, por meio da qual os usuários podem realizar consultas sobre dados distribuídos em diferentes fontes e formatos. Já o BigDAWG propõe um modelo mais sofisticado, combinando a interface unificada com mecanismos internos capazes de decidir, de forma automática, pela movimentação ou replicação de dados entre os diferentes repositórios, com o objetivo de otimizar o desempenho das consultas. Apesar dessas inovações, na prática, o desempenho e as estratégias de execução desses sistemas ainda estão fortemente condicionados às características dos próprios repositórios de dados utilizados, limitando, em parte, o potencial de otimizações dos planos de execução.

Diferentemente dos sistemas *polystore*, cujo principal objetivo é integrar múltiplos repositórios de dados por meio de uma camada de abstração única de consulta, os sistemas de processamento de dados multiplataforma (também chamados de plataformas unificadas de análises de dados) têm como foco a integração de diferentes ferramentas durante de processamento dos dados. O propósito central desses sistemas é permitir que um mesmo código ou consulta possa ser executado, de maneira transparente, sobre diferentes mecanismos de processamento.

Enquanto os sistemas *polystore* concentram-se na unificação do acesso aos dados, os sistemas multiplataforma buscam oferecer suporte a um espectro mais amplo de aplicações, incluindo não apenas Sistemas Gerenciadores de Bancos de Dados (SGBDs), mas também sistemas analíticos de propósito geral, como motores de processamento distribuído. Nessa abordagem, o usuário desenvolve suas consultas de forma independente da ferramenta subjacente, cabendo ao sistema decidir, em tempo de execução, qual tecnologia ou mecanismo utilizar para executar cada parte da consulta.

Essas decisões podem ser guiadas por restrições práticas, por exemplo, a disponibilidade de uma determinada operação em uma ferramenta específica, ou por decisões oportunistas, baseadas na otimização de algum critério de custo. Entre os critérios frequentemente utilizados destacam-se o tempo de execução, o custo monetário associado ao uso de recursos em ambientes de nuvem ou métricas alternativas, como o consumo energético. Nos casos em que o desempenho é o principal objetivo, os sistemas multiplataforma geralmente fazem uso de funções de custo, modeladas a partir de preditores de desempenho, ou ainda de regras heurísticas predefinidas.

Embora as diferentes abordagens de integração de sistemas apresentadas nesta seção possam, em alguns momentos, se sobrepor ou até mesmo se confundir, o foco deste trabalho está direcionado para um escopo mais amplo de integração: a construção de sistemas multiplataforma voltados, especificamente, para a execução de fluxos de trabalho

(ou *workflows*) em cenários modernos de Ciência de Dados. Nesse contexto, a definição 1 apresenta e formaliza as principais terminologias que serão adotadas ao longo deste trabalho.

Definição 1. (Sistema multiplataforma) Um **sistema multiplataforma** (do inglês, *multiplatform system*, *cross-plataform system* ou *unified data processing system*) é um sistema capaz de utilizar, de maneira integrada, diferentes **ferramentas de processamento de dados**[1] para a composição e execução de um único fluxo de dados. Considerando a complexidade inerente às operações e algoritmos oferecidos por essas ferramentas, os sistemas multiplataforma assumem que cada tarefa de um fluxo de dados é indivisível no contexto de execução, ou seja, cada tarefa deve ser atribuída integralmente a uma única ferramenta de processamento, sem divisão ou particionamento entre múltiplas ferramentas. Dessa forma, o principal objetivo de um sistema multiplataforma é realizar o escalonamento adequado das tarefas de um fluxo de dados, definindo qual ferramenta será responsável pela execução de cada tarefa. Essa execução pode ocorrer em uma ou mais **configurações de hardware**, a depender da disponibilidade e da heterogeneidade dos recursos computacionais existentes.

O problema modelado por um sistema multiplataforma é reconhecidamente *NP-difícil* (Gog et al., 2015; Kruse et al., 2019). Nessas condições, isto significa que encontrar uma solução ótima é computacionalmente inviável para instâncias grandes, e métodos heurísticos ou de aproximação são geralmente empregados. Além disso, se assemelha em muitos aspectos com os problemas tradicionais de escalonamento em computação de alto desempenho (sigla do inglês, HPC):

1. Otimização baseada em custo: otimização da alocação de tarefas para recursos heterogêneos, usando modelos de custo para avaliar alternativas, adaptando-se ao hardware disponível e ao perfil de carga de trabalho;

2. Gerenciamento de heterogeneidade: enquanto sistemas multiplataformas lidam com múltiplas ferramentas de processamento de dados e executa tarefas distribuídas de forma transparente, escalonadores HPC lidam com recursos físicos heterogêneos e tipos variados de jobs, buscando maximizar desempenho e utilização.

3. Abstração e flexibilidade: sistemas multiplataformas oferecem abstração para esconder a complexidade das ferramentas de execução subjacentes, facilitando o desenvolvimento. Da mesma forma, sistemas HPC abstraem detalhes complexos dos recursos para usuários e gestores, focando no uso eficiente e balanceado da infraestrutura.

[1] Neste contexto, é comum na literatura o uso do termo **plataformas** como sinônimo de ferramentas de processamento de dados (Lucas et al., 2018; Beedkar et al., 2023; Chen, Q. et al., 2023).

2.1.1 Representação do problema

No contexto de sistemas de processamento multiplataforma, um aspecto fundamental da metodologia é a definição de como as tarefas de um fluxo de dados serão representadas no componente responsável por recomendar as ferramentas a serem utilizadas em cada etapa. A forma como o problema é modelado influencia diretamente a metodologia sistemática empregada na avaliação das alternativas de solução, afetando tanto a eficiência da busca quanto a qualidade dos resultados obtidos.

Embora o mapeamento do problema possa ser representado de diferentes maneiras, como no CLIC (Chen, Q. et al., 2023) e no Robopt (Kaoudi et al., 2020), que utilizam uma matriz onde cada linha representa uma operação e cada coluna descreve uma propriedade associada, a maioria das abordagens opta por uma modelagem baseada em grafos. Essa preferência se deve principalmente à capacidade dos grafos de representar de forma mais natural as relações de precedência entre tarefas. Ainda assim, existem diversas formas de modelar o problema utilizando grafos, variando de acordo com os objetivos do sistema e o grau de detalhamento necessário.

Tradicionalmente, na área de escalonamento de tarefas, a forma mais comum de representar um fluxo de dados é por meio de um grafo de tarefas (Kwok et al., 1998; Coleman et al., 2024). Nessa modelagem, utiliza-se um grafo direcionado e acíclico (DAG, do inglês Directed Acyclic Graph), em que os vértices representam as operações do fluxo e as arestas indicam as dependências entre essas operações. No exemplo ilustrado na figura 2.1, o fluxo compreende a leitura de duas fontes de dados distintas (L_1 e L_2), seguidas de operações de filtragem independentes para cada fonte (F_1 e F_2). Em seguida, os dados resultantes são combinados por meio de uma operação de junção (J), submetidos a uma nova filtragem (F_3) e, por fim, agrupados com uma função de agregação (G). A exigência de que o grafo seja direcionado garante que a ordem de execução respeite as dependências entre operações. Além disso, a restrição de aciclicidade assegura que o fluxo tenha um início e um fim bem definidos. No exemplo apresentado, a altura em que cada vértice foi posicionado no grafo não necessariamente reflete a ordem de execução real. Operações sem dependências, como as leituras iniciais (L_1 e L_2), podem ser executadas em ordens distintas, dependendo da ferramenta utilizada.

Essa modelagem permite representar fluxos compatíveis com diversos tipos de ferramentas, independentemente de estas suportarem paralelismo de tarefas ou de dados. Por exemplo, caso o fluxo anterior fosse executado em uma ferramenta que suporte paralelismo de tarefas, isto é, a execução simultânea de operações em núcleos de processamento distintos, como em Tejedor et al. (2017), os grupos de tarefas (L_1; F_1) e (L_2; F_2) poderiam ser executados em paralelo. No entanto, devido à dependência entre L_1 e F_1, bem como entre L_2 e F_2, as operações de filtragem só poderiam ser iniciadas após a conclusão das respectivas leituras. O mesmo raciocínio se aplica às demais etapas do fluxo. Neste

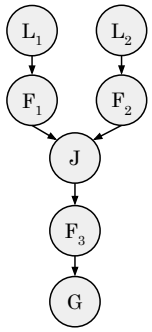


Figura 2.1: Exemplo de um fluxo de dados como um grafo de tarefas.

Onde: L - Leitura de dados, F - Filtragem de dados, J - Junção e G - Agrupamento.

exemplo, não haveria oportunidades adicionais de paralelismo após a etapa de filtragens iniciais, uma vez que as demais operações apresentam dependências sequenciais.

Embora essa representação seja amplamente utilizada, ela apresenta algumas limitações. No contexto de sistemas multiplataforma, por exemplo, além de mapear cada tarefa como um nó do grafo, é desejável representar também o conjunto de ferramentas disponíveis para a execução de cada tarefa. Sem esse mapeamento explícito, características e restrições importantes precisam ser tratadas em etapas adicionais à modelagem, como nas chamadas decisões de controle de fluxo (Huang, T. W. et al., 2022). Essa etapa extra torna o processo de decisão mais complexo e impõe custos adicionais ao modelo, dificultando tanto a análise quanto a otimização do escalonamento.

Nesse contexto, o Musketeer (Gog et al., 2015) é um dos primeiros sistemas multiplataforma baseados em ferramentas modernas de *big data* que propõe uma representação alternativa ao tradicional grafo de tarefas. Nesse sistema, o fluxo de dados é inicialmente modelado como um grafo de tarefas convencional, mas a resolução do problema de alocação das tarefas entre as ferramentas é tratada como um problema de particionamento em k-partições disjuntas, com o objetivo de minimizar o número de arestas cortadas entre as partições, ou seja, minimizar a comunicação entre diferentes ferramentas. Cada partição representa, portanto, uma ferramenta responsável por executar um subconjunto das tarefas. Mais detalhes sobre esse sistema serão discutidos na seção 2.1.3. No entanto, uma das limitações dessa abordagem é a necessidade de realizar uma ordenação topológica do grafo para linearizar o problema de particionamento, o que pode impedir que operadores compatíveis sejam alocados na mesma partição, mesmo quando isso resultaria em um menor custo de execução.

Em problemas nos quais uma tarefa pode ser executada por meio de diferentes variações, como é o caso dos sistemas de processamento multiplataforma, é comum interpretar o fluxo de dados como um grafo multivariante (do inglês, *multi-variate graph*) (Holzer et al., 2007; Graf et al., 2014; Beedkar et al., 2023), como exemplificado na figura 2.2. Um grafo multivariante é um grafo direcionado em que cada nó representa uma tarefa e está associado a um conjunto de variantes de implementação possíveis, cada uma com

características específicas e custos potenciais distintos, como, por exemplo, o tempo de execução estimado para cada opção de ferramenta.

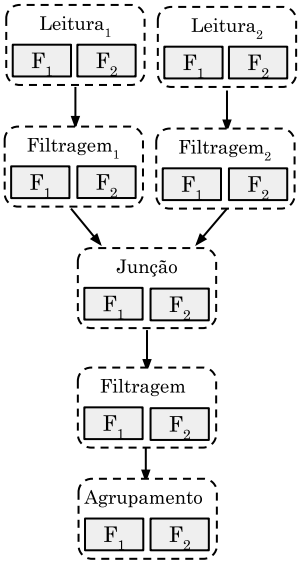


Figura 2.2: Exemplo de representação do fluxo da fig. 2.1 como um grafo multivariante. Onde: F_1 e F_2 representam possibilidades de ferramentas de processamento.

O Apache Wayang (Beedkar et al., 2023), por exemplo, é um sistema multiplataforma que adota a abordagem baseada em grafos multivariantes, como o da figura 2.2. Na definição do plano de execução no Wayang, cada tarefa é representada por um operador que se assemelha ao conceito de bloco. Internamente, esse bloco pode conter diversas opções de ferramentas capazes de implementar a tarefa desejada. Nesse contexto, o processo de busca pela melhor configuração de execução consiste em selecionar, para cada bloco básico, a variante (ou implementação) mais adequada, de forma a minimizar o custo total de execução da aplicação. Essa mesma abordagem também é empregada no Octopus-DF (Rong et al., 2022).

No entanto, existem dois problemas principais que limitam a abordagem de modelagem utilizada pelo Wayang e pelo Octopus. O primeiro é que os nós variantes ficam encapsulados nos blocos, o que dificulta a representação explícita de relações e regras entre variantes como, por exemplo, “se a tarefa A utilizar a ferramenta X, então a tarefa B também deve utilizar X” ou “evitar combinar operações entre as ferramentas X e Y devido a problemas de compatibilidade”. O segundo problema é que essa modelagem se concentra em representar apenas as relações entre tarefas, sem tratar das relações entre os dados. Dessa forma, operações mais complexas, como a divisão de dados em múltiplos tipos de resultados, têm representação limitada, já que não é possível modelar explicitamente a movimentação dos dados entre tarefas. Devido a essas limitações, etapas adicionais de processamento podem ser necessárias para avaliar restrições ou ajustar as propriedades do grafo.

No capítulo 3, apresentaremos a nossa proposta de modelagem baseada em grafos multivariantes, como uma alternativa à utilizada por sistemas existentes. A nossa representação permite modelar diversos aspectos relevantes para a tomada de decisão em sistemas multiplataforma, como as opções de ferramentas e configurações de hardware, as variações no custo de execução de cada tarefa em diferentes ferramentas, os estágios intermediários dos dados, bem como os custos associados à transmissão de dados entre ferramentas.

2.1.2 Abordagens de previsão de desempenho

Embora o conceito de Modelos de Previsão de Desempenho tenha sido estabelecido na década de 1980 (Devarakonda et al., 1989), sua aplicação a sistemas de Ciência de Dados vem ganhando popularidade apenas nos últimos anos, com o aumento da pesquisa em áreas como ferramentas de *big data* e execuções em nuvem. Hoje em dia, existem previsores de desempenho em vários domínios, preenchendo a lacuna entre diferentes comunidades de pesquisa, que vão desde computação distribuída em ambientes de nuvem (Venkataraman et al., 2016; Alipourfard et al., 2017; Maros et al., 2019), operação de *data centers* (Delimitrou et al., 2014) até a computação de alto desempenho(Pfeiffer et al., 2008). Dentre essas diversas pesquisas, as mais populares são aquelas baseadas em prever o tempo de execução de um fluxo de dados (Vianna et al., 2013; Venkataraman et al., 2016; Shen et al., 2018; Maros et al., 2019). No entanto, existem outras soluções que buscam prever os melhores parâmetros de configuração para minimizar o tempo de execução (Dias et al., 2018; Kroß et al., 2019), e também existem outras linhas de pesquisa que buscam utilizá-los para tratar outros aspectos de uma execução, como quantidade de recursos utilizados (Matsunaga et al., 2010), tráfego de rede (Ghoshal et al., 2019), especialmente importante para execuções em nuvem, ou até o gasto energético (Shapi et al., 2021).

Além dos diferentes domínios, os modelos de previsão também podem ser criados usando diferentes abordagens. Os exemplos mais populares são aqueles baseados em modelos analíticos (Vianna et al., 2013; Baldacci et al., 2019), simulação (Zheng et al., 2015; Venkataraman et al., 2016; Kroß et al., 2019; Cheng et al., 2021) e, mais recentemente, em aprendizado de máquina (Justus et al., 2018; Lattuada et al., 2019; Shapi et al., 2021; Yu, X. et al., 2022). No primeiro caso, a ideia é criar um conjunto de equações matemáticas que definam o comportamento de uma execução. O trabalho de Baldacci et al. (2019) é um dos exemplos mais relevantes para a modelagem de custo analítico em Spark SQL. Segundo os autores, o modelo é capaz de estimar, com menos de 20% de erro, o custo de uma aplicação Spark envolvendo um subconjunto de operadores SQL mapeados. Além das informações da aplicação e dos dados, a equação resultante leva em consideração os recursos computacionais consumidos pela aplicação e outras métricas provenientes do

hardware, como taxas de leitura e gravação e taxas de transferência de rede.

Embora uma abordagem analítica proporcione maior transparência aos aspectos envolvidos na execução, muitas vezes é considerado inviável construir um conjunto gerenciável de equações devido à complexidade das aplicações e das ferramentas modernas. Por exemplo, muitas ferramentas modernas, buscando simplificar a programação para o usuário, optam por uma abstração de dados que oculta aspectos de execução que podem influenciar no desempenho (Yu, Y. et al., 2008; Zhao, B. et al., 2015).

As abordagens baseadas em simulação e aprendizado de máquina tendem a ser similares, uma vez que ambas costumam usar algum tipo de regressão; a principal ideia diferenciadora é que abordagens por simulação requerem geralmente a execução de uma aplicação específica, geralmente com um cenário de amostra, para então extrapolar seus resultados. Por outro lado, as abordagens de aprendizado de máquina utilizam geralmente dados históricos de execuções ou a partir de uma calibragem inicial, que podem ser correspondências exatas do mesmo problema ou depender de métricas de similaridade para identificar cenários próximos para criar o modelo por regressão, classificação ou até mesmo um algoritmo de aprendizado profundo.

Como exemplo de abordagens baseadas em simulação, também no Spark, o trabalho de Venkataraman et al. (2016) é um estado da arte na previsão de desempenho de aplicações iterativas em Spark para ambientes de nuvem. A solução, também conhecida como modelo de Ernest, é baseada em simulações a partir de um cenário pequeno, utilizando uma amostra da entrada e um menor número de iterações. O modelo utiliza a simulação e os padrões de comunicação observados para prever o tempo de execução do cenário real. Embora o modelo seja fundamentado em princípios simples e funcione bem em algoritmos iterativos devido à sua regularidade, outros autores mostraram que erros de até 187% podem ser encontrados em fluxos de trabalho mais complexos (Alipourfard et al., 2017; Lattuada et al., 2019).

Modelos que usam aprendizado de máquina são abordagens que vêm se tornando mais promissoras. Nessa abordagem, a solução mais comum é transformar o problema em um problema de regressão. A regressão define uma ampla classe de métodos que mapeiam observações em um espaço numérico e, em seguida, ajustam uma função a partir de um conjunto de características que se aproximam de uma determinada variável alvo.

Além da definição do método utilizado, uma etapa importante para a criação do modelo é a definição de seus parâmetros de entrada (isto é, seleção de características). Nessa lógica, também existem diferentes abordagens para a seleção de características dos modelos, algumas obtidas *a priori* da execução (também chamadas de soluções *black-box*) e outras com informações obtidas durante ou após a execução (*gray-box*) (Lattuada et al., 2019). No entanto, enquanto modelos *gray-box* permitem obter mais informações sobre o comportamento da ferramenta de execução ao longo das execuções e, com isso, possam gerar modelos de custo mais robustos, geralmente esse tipo de abordagem pode ser

utilizada apenas em sistemas especializados em uma ferramenta de execução. Diferentes ferramentas de execução têm arquiteturas diferentes e, com isso, é esperado que os fatores importantes para se descrever o comportamento de cada ferramenta sejam diferentes. Abordagens *a priori/black-box* são um meio de permitir a criação de modelos de custo que sejam genéricos para múltiplas ferramentas.

2.1.3 Sistemas multiplataformas modernos

Neste trabalho, focaremos nossos estudos em sistemas multiplataformas capazes de executar fluxos de dados em ferramentas adaptadas aos cenários atuais de *big data* e que utilizam abstrações modernas de alto nível. A tabela 2.2 apresenta alguns dos trabalhos relacionados ao nosso contexto. Esta seção discute os sistemas apresentados na tabela e suas limitações.

Tabela 2.2: Exemplos de sistemas multiplataformas.

Nome	Abstração	Estimativa
Musketeer (Gog et al., 2015)	SQL, grafo, `map/filter/reduce`	taxas de vazão
Apache Wayang (Beedkar et al., 2023)	`map/filter/reduce`	modelo analítico
Robopt (Kaoudi et al., 2020)	`map/filter/reduce`	modelo de regressão
Octopus-DF (Rong et al., 2022)	DataFrame	modelo de regressão
CLIC (Chen, Q. et al., 2023)	`map/filter/reduce`	modelo de classificação

O Musketeer, já mencionado na seção 2.1.1, é uma ferramenta capaz de interpretar códigos dos *front-ends* suportados, como HiveQL, Pig e Spark SQL, convertê-los em uma representação intermediária e, a partir dela, definir a melhor escolha de *back-ends* para a execução, como Hadoop ou Spark. Após a criação do grafo de tarefas, o sistema adota uma abordagem mista entre regras e modelo de custo para o particionamento das tarefas entre as ferramentas. As regras são utilizadas para limitar e direcionar algumas opções de escolha. Por exemplo, se o fluxo a ser executado foi escrito em uma linguagem baseada em SQL, apenas um subconjunto de back-ends será suportado.

Em cenários nos quais a execução pode ser dividida entre múltiplas ferramentas, a solução adota um modelo de custo simples, baseado em quatro métricas de vazão obtidas durante etapas de calibração: a taxa de leitura dos dados na fonte; o custo de carregar ou transformar dados para o formato interno do sistema; a taxa de processamento real dos dados; e a taxa de gravação dos resultados.

Essa abordagem, no entanto, apresenta limitações. A primeira, já discutida na seção 2.1.1, está relacionada ao uso da heurística de particionamento linear do grafo, que

pode deixar de explorar oportunidades de otimização. Outra limitação é a ausência de representação dos dados, que não modela nem estima sua evolução ao longo da execução. Assim, o modelo de custo torna-se impreciso em cenários com dados muito variados ou com operadores que dependem de parametrizações fornecidas pelo usuário.

Outro sistema já mencionado é o Apache Wayang, também conhecido como Rheemix. Trata-se do primeiro sistema de processamento de dados multiplataforma, na era do *big data*, a ultrapassar o escopo puramente acadêmico[2]. A ferramenta baseia-se em modelos de custo e expõe sua própria API de linguagem agnóstica, que suporta a geração de código para diferentes ferramentas como Java Streams, Spark e Flink, entre outras.

No Wayang, as tarefas são codificadas por meio de operadores de baixo nível, como `map`, `filter` e `reduce`. Para cada operador, é fornecido um modelo de função de custo baseado em constantes, cardinalidade dos dados de entrada/saída e estimadores de perfil de carga (que estimam aspectos como carga de CPU, uso de disco, entre outros) que devem ser calibrados em cada configuração de hardware. Embora o Wayang ofereça um utilitário para aprendizado desses estimadores a partir de execuções reais, os custos base da função são definidos manualmente. Nesse contexto, adicionar suporte a uma função definida pelo usuário (do inglês, User Defined Function ou UDF) ou a um novo operador pode representar um desafio considerável. Devido à dependência desses estimadores de perfil de carga, o sistema não oferece suporte a múltiplas configurações de recursos, assumindo sempre um hardware homogêneo de execução. Além das limitações já discutidas na seção 2.1.1 relacionadas à sua representação interna, outro fator que dificulta sua adoção prática é a exigência de uma codificação baseada em operadores de baixo nível, o que limita o aproveitamento direto de aplicações desenvolvidas com códigos modernos de Ciência de Dados, exigindo dos usuários uma carga considerável de recodificação.

Métodos baseados em aprendizado de máquina (Marcus et al., 2019; Kaoudi et al., 2020; Zhou et al., 2020) vêm sendo propostos para simplificar as abordagens de estimativa de custo. Em Robopt, por exemplo, o fluxo de trabalho é codificado como um vetor de recursos, enumerado para todas as combinações possíveis de operadores entre ferramentas. Em seguida, os vetores são inseridos no modelo de ML para estimar o plano com custo mínimo. Um problema dessa abordagem é a possibilidade de cálculos desnecessários, uma vez que para um fluxo de trabalho com n operadores lógicos, cada um com k ferramentas de execução, haveria um total de n^k planos de execução a serem computados. Além disso, pelo fato da ferramenta ter sido implementada a partir do Rheemix, sua interface possui os mesmos problemas que o Apache Wayang, ou seja, uma interface de prototipação a partir de operadores simples.

Já o Octopus-DataFrame é um projeto baseado na compatibilidade entre o Pandas (McKinney, 2010), Spark e Dask (Dask Development Team, 2016). Sua ideia se baseia em adaptar uma funcionalidade comum entre as três interfaces. Por exemplo, um

[2]É um projeto em estágio de incubação no momento em que este trabalho está sendo escrito.

dos principais esforços no Octopus é suportar operações baseadas em índices do Pandas, como `loc` e `iloc`, que permitem manipular células de uma tabela individualmente, nos outros dois sistemas. Atualmente, a ferramenta suporta aproximadamente vinte operadores variando entre operadores de leitura e gravação, seleções como `head`, `tail`, `filter` e operadores complexos como `join`, `merge` e `sort_values`. Octopus usa uma abordagem baseada na execução do fluxo original a partir de uma amostra dos arquivos de entrada para estimar como os dados vão evoluir ao longo da execução. A partir cada estimativa é então aplicado modelos de regressão linear para construir seu plano de custos.

Sem atualizações de maio de 2021, a ferramenta cria amostras, de até 8 mil linhas, dos arquivos de entrada para as estimativas. Sendo assim, a partir das execuções sobre esses dados, as informações do número de linhas e colunas são então interpoladas a partir da razão da amostragem. Nessa abordagem, embora o tempo de execução dos fluxos em dados desse tamanho tenham custos de execução baixos, o custo inicial da leitura desses arquivos para a realização da amostragem pode ser significativo.

No Octopus, o modelo de estimativa de tempo é composto tanto pelo tempo decorrido em cada tarefa quanto pelo tempo necessário para a migração de dados.[3] A calibração desse modelo depende da execução de diversos arquivos de teste, específicos para cada combinação de ferramenta, operador e tamanho de linha. Essas execuções são utilizadas para construir uma função de regressão, definida por seus coeficientes e constantes, para cada operador, a fim de prever o tempo de execução das operações.

Embora o Octopus seja o primeiro esforço para criar um sistema multiplataforma baseado nas operações de alto nível de DataFrame, ele ainda apresenta outras limitações além das já discutidas. Em primeiro lugar, embora a tentativa de adicionar suporte de índices globais e outros elementos internos específicos do Pandas ajude na compatibilidade de sintaxe entre as outras ferramentas, ela também pode impactar o desempenho geral ao adicionar sobrecarga e novas dependências ao processo. Além disso, embora Pandas, Dask e Spark compartilhem um grande número de operadores, eles geralmente diferem na quantidade de parâmetros utilizados e outros detalhes. O Spark, por exemplo, tende a ter menos opções de parâmetros devido às complexidades de paralelização, limitando o suporte total à compatibilidade. Por fim, a premissa de se utilizar regressões lineares simples pode não representar bem todos os tipos de tarefas.

O CLIC é um dos trabalhos relacionados mais recentes. Diferentemente das abordagens anteriores, que estimam os custos de possíveis planos de execução, a seleção da ferramenta é partir da modelagem como um problema de classificação. Nessa modelagem, o vetor de entrada de dados representa uma operação, onde cada dimensão é uma característica da operação para as ferramentas suportadas, ou seja, quanto maior o número de ferramentas suportadas, maior a dimensionalidade. Já a classe associada a esse vetor

[3] A migração de dados no Octopus é realizada por meio da escrita dos dados em disco, em um formato binário, para posterior leitura pela ferramenta de destino.

representa a melhor ferramenta para o operador, informação obtida a partir de execuções de calibração. O sistema de classificação utiliza uma Rede Neural de Grafos Convolucional (do inglês, Graph Convolutional Network ou GCN), onde as operações de um fluxo de dados são codificadas a partir uma função de *embedding*, semelhante ao processo de *word embedding* usado em tarefas de linguagem natural. Nessa lógica, operações relacionadas são tratadas como representações próximas. Além das informações das operações, a GCN criada leva em consideração informações do hardware como memória, rede e GPU.

Apesar dessa abordagem apresentar bons resultados, os testes consideram apenas operadores de baixo nível como `map/filter/reduce`. Em nosso entendimento, mudar o nível de abstração dos operadores para um nível mais alto, como o de DataFrames, pode influenciar na geração dos *embeddings* da rede neural, uma vez que o modelo assume operações simples, enquanto representações mais modernas como a de DataFrame geralmente possuem operadores mais complexos. Como as ferramentas suportadas são codificadas como dimensões do modelo, adicionar uma nova ferramenta exige um novo treinamento do modelo. Além disso, a interpretabilidade de um resultado gerado por rede neural costuma ser difícil de se alcançar, dificultando a auditoria da solução. No entanto, o ponto mais complicado, que também é compartilhado pelas outras soluções aqui discutidas, como o Octopus, Robopt, Wayang e o Musketeer, é a ausência dessas soluções em considerar a mutabilidade dos dados de entrada em cada estágio de processamento durante a execução de um fluxo de trabalho. Dados de entrada podem, durante o fluxo, aumentar ou reduzir de tamanho drasticamente, e, pelo fato de todas essas soluções funcionarem apenas como um ponto de submissão às ferramentas de processamento, elas não possuem informações suficientes sobre os dados de entrada para realizar inferências sobre como os dados de entrada evoluem ao longo do fluxo.

Nesse contexto, a proposta deste trabalho é modelar um sistema multiplataforma voltado para cenários de Ciência de Dados, com suporte a uma abstração popular e de alto nível, como a de DataFrame (discutida com mais detalhes na próxima seção). Acreditamos que uma abstração desse tipo não só simplificará o processo de criação de fluxos para os usuários como também ajudará na qualidade das recomendações, uma vez que a utilização de operadores de baixo nível, como `map/filter/reduce`, em cenários complexos, pode não considerar a relação entre os operadores e as tarefas no fluxo. Por exemplo, embora operações como a remoção de linhas duplicadas e a agregação em DataFrames possam ser expressas como um operador de `group-by` em uma interface de baixo nível, o comportamento dessas duas operações claramente não será igual. Trabalhar com abstrações como a de DataFrame aumenta a granularidade das tarefas, auxiliando na modelagem dos custos. Além disso, devido à velocidade com que novas ferramentas são criadas e evoluem, a solução precisa suportar ferramentas independentemente de suas diferenças de execução, respeitando suas singularidades. Por exemplo, nenhuma das soluções anteriores leva em consideração o tamanho dos dados como um fator limitante de uma ferramenta.

Todos os cenários avaliados consideram que todas as ferramentas possuem a capacidade de processar os mesmos dados, uma premissa que nem sempre é verdadeira. A modelagem do sistema de previsão de desempenho também precisa levar em consideração a diversidade dos recursos computacionais, não se limitando a apenas uma configuração de sistema, e, principalmente, ser capaz de estimar a evolução dos dados durante uma execução. A maioria das soluções apresentadas anteriormente falham em não considerar que os dados durante uma execução podem ser reduzidos ou mesclados, influenciando o tempo de processamento.

2.2 Abstração de DataFrame

A abstração de dados de DataFrame tem se tornado cada vez mais popular e conveniente para representar, estruturar, limpar e analisar dados durante as tarefas de Ciência de Dados. Um DataFrame, exemplificado na figura 2.3, é uma tabela ou uma estrutura bidimensional semelhante a um *array*. Na matemática, pode ser interpretado como uma matriz regular. Suas colunas são nomeadas e podem ter tipos heterogêneos, exigindo, no entanto, um comprimento idêntico entre todas as colunas. Os DataFrames são semelhantes às tabelas de um banco de dados relacional tradicional, mas são projetados para fornecer APIs de alto nível, permitindo que especialistas do domínio, como cientistas de dados e estatísticos, os utilizem facilmente em programas procedimentais. Esses DataFrames oferecem uma variedade de operações, incluindo filtragem de linhas com base em condições, união de dados e agregação de valores de coluna com base em uma coluna-chave. Além disso, eles proporcionam operações analíticas avançadas, como médias móveis, e estão completamente integrados ao sistema de *array* subjacente para suportar cálculos de *array*.

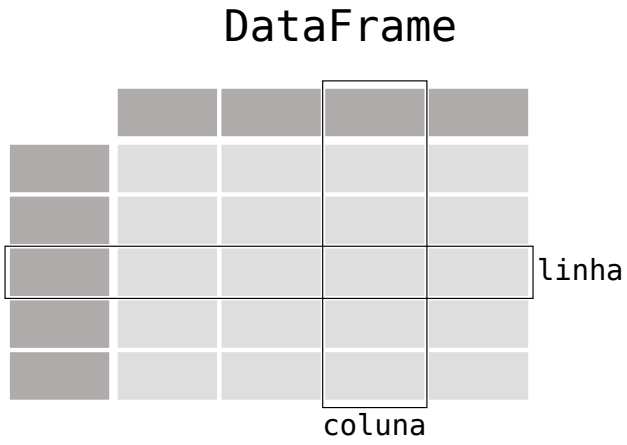


Figura 2.3: Exemplificação da abstração de DataFrame.

Fonte: Adaptado de Pandas Development Team (2024b).

Diferentemente das abstrações clássicas baseadas em operadores de baixo nível, que geralmente exigem modelar uma atividade em abordagens baseadas em listas ou como chave-valor, os DataFrames possuem diversas características que os tornam uma escolha atraente para exploração de dados: um modelo de dados intuitivo que abrange uma ordenação implícita em linhas e colunas e as trata simetricamente; uma linguagem de consulta que une uma variedade de modalidades de análise de dados, incluindo operadores relacionais (por exemplo, filtrar, juntar), álgebra linear (por exemplo, transpor) e do tipo planilha (por exemplo, pivô); e, uma sintaxe de consulta combinável de forma incremental que incentiva a validação fácil e rápida de expressões simples e seu refinamento iterativo e composição em consultas complexas (McKinney, 2022).

Embora a idealização dessa abstração seja antiga, introduzida pela primeira vez na linguagem S em 1990 (Chambers et al., 1990), só na linguagem R (The R Foundation, 2024), uma implementação de código aberto de S, lançada em 1995, foi que os DataFrames começaram a ganhar popularidade na comunidade de estatísticos. Hoje em dia, diversas ferramentas, em várias linguagens de programação, implementam DataFrames, conforme exemplificado na tabela 2.3. Essa abstração é tão popular que alguns pesquisadores frequentemente citam ferramentas como o Pandas como sendo a razão da popularidade da linguagem Python (Claburn, 2017; Robinson, 2017).

Tabela 2.3: Abstração de DataFrame em diversas ferramentas.

Modelo de avaliação	Ferramenta	Linguagem	Multi-núcleos	Multi-nós	Referência
Ansiosa (*eager*)	DataFrame	R	Não	Não	(The R Foundation, 2024)
	Pandas	Python	Não	Não	(McKinney, 2010)
	DataFrames.jl	Julia	Sim	Não	(Bouchet-Valat et al., 2023)
	cuDF	Python	Sim	Sim	(NVIDIA RAPIDS, 2025a)
Preguiçosa (*lazy*)	Dask	Python	Sim	Sim	(Dask Development Team, 2016)
	Apache Spark	Python, Scala, Java	Sim	Sim	(Armbrust et al., 2015)
	DDF/COMPSs	Python	Sim	Sim	(Ponce et al., 2021)
	HiFrames	MPI/C++	Sim	Sim	(Totoni et al., 2017)
Ambos	Polars	Rust, Python	Sim	Não	(The Polars Development Team, 2023)
	Modin	Python	Sim	Sim	(Petersohn et al., 2020)

Além das diferenças de linguagens e de operadores suportados, as implementações de DataFrames podem variar no suporte ao paralelismo de dados e a distribuição de tarefas, além do modelo de avaliação de execução. Existem dois modos de avaliação de consultas em sistemas de processamento de dados: preguiçoso (*lazy-evaluation*) e ansioso (*eager-evaluation*). No paradigma ansioso, adotado por sistemas como Pandas, o controle do programa não é devolvido ao usuário até que a instrução tenha sido completamente

avaliada, forçando o usuário a ficar ocioso durante a computação. Nesse contexto, a otimização da consulta não é possível em um sistema ansioso porque cada operador dentro da instrução deve ser completamente executado à medida que é instanciado pelo usuário.

Por outro lado, no paradigma preguiçoso, que vem se tornando mais popular na última década (Armbrust et al., 2015; Dask Development Team, 2016), o controle é devolvido ao usuário imediatamente após a chamada do operador, e cabe ao sistema decidir quando executar a computação, talvez até que o usuário solicite o seu resultado. Ao agendar a computação, o sistema pode esperar que subexpressões de consulta maiores sejam montadas, levando a maiores oportunidades de otimização, como é feito em sistemas como o Spark. A desvantagem da avaliação preguiçosa é que a computação só começa quando o usuário solicita o resultado de uma consulta. Além disso, durante o tempo em que um usuário está pensando ou digitando, o sistema poderia estar trabalhando em direção a um resultado, porém, no paradigma preguiçoso, o sistema fica ocioso durante o tempo de reflexão. Para conjuntos de dados menores ou em consultas simples, o tempo ocioso pode potencialmente levar a um tempo de execução geral mais longo se houver períodos em que a reflexão exceda o tempo de execução das instruções na consulta.

2.2.1 Pandas

O Pandas, atualmente na versão 2.3.3,[4] talvez seja uma das ferramentas de Ciência de Dados mais populares hoje em dia. Uma pesquisa realizada em 2018 (Rule et al., 2018) apontou que cerca de 40% de uma amostra de 1 milhão de notebooks Jupyter disponíveis publicamente no GitHub utilizavam o Pandas. O sistema foi lançado originalmente em 2008 por Wes McKinney e seu nome deriva do termo Dados em Painel (do inglês, *Panel Data*), que se refere a dados obtidos da observação de vários indivíduos ao longo do tempo.

Até a versão 1.5.3 (lançada em 19/01/2023), a ferramenta utilizava internamente o NumPy (nome derivado do termo Numerical Python), uma biblioteca Python popular para computação numérica e científica, que funciona como um *wrapper* em linguagem C. Por causa disso, embora o Pandas seja uma solução Python, grande parte de suas operações são executadas de forma mais eficiente em C, sendo utilizado por muitos, não só para ganhos em abstração de dados, como também para eficiência de execução. Atualmente, o sistema fornece mais de 200 operadores distintos, permitindo que muitos fluxos de trabalho possam ser implementados a partir de abordagens diferentes.

Desde a versão 2.0, o núcleo do Pandas vem sendo reconstruído para o Apache Arrow (Apache Software Foundation, 2024), uma ferramenta com suporte a várias linguagens para o desenvolvimento de ferramentas de análise de dados colunares. A ferramenta padroniza um formato de dados em memória, orientado a colunas, para operações analíticas eficientes em hardware moderno de CPU e GPU. Atualmente, embora o NumPy continue

[4] Até a data de 20/11/2025.

sendo o *back-end* padrão, os usuários podem especificar a nova versão durante a leitura de novos dados. Somente na versão 3.0, ainda em desenvolvimento, o Pandas passará a exigir execuções via Apache Arrow (Pandas Development Team, 2024a). Essa migração é uma das principais tentativas de corrigir problemas conhecidos do Pandas, como o consumo excessivo de memória RAM. Por exemplo, uma prática popular, inclusive divulgada pelo criador da ferramenta, é trabalhar com recursos com 5 a 10 vezes mais RAM do que o tamanho do conjunto de dados (McKinney, 2017). Além disso, essa migração prevê simplificação da manutenção do núcleo da ferramenta e melhorias de desempenho em operações como ingestão e exportação de dados.

Para a ferramenta adequar às complexidades atuais de Ciência de Dados, frequentemente é executada em conjunto com outras bibliotecas de algoritmos, como o Scikit-Learn (Pedregosa et al., 2011), para complementar as execuções com algoritmos de estatísticas e de aprendizado de máquina. O Scikit-Learn, também chamado de Sklearn, é a biblioteca de algoritmos de aprendizado mais popular em Python (Stančin et al., 2019), possui um grande número de algoritmos que variam desde operações de Engenharia de Características (*Feature engineering*) até a criação de modelos de classificação, agrupamento de dados e regressão.

2.2.2 Apache Spark

O Apache Spark, atualmente em sua versão 4.0.1,[5] é um arcabouço para processamento de grandes volumes de dados que tinha como uma de suas premissas originais conseguir processar os dados de forma mais eficiente do que o Apache Hadoop (White, 2015). A arquitetura do Spark, exemplificada na figura 2.4, é baseada em três componentes principais: o `Driver`, responsável por iniciar a aplicação e comandar a execução das tarefas; o Gerenciador de recursos, encarregado de distribuir os recursos computacionais; e os `Workers`, responsáveis pela execução, de fato, das tarefas.

O Spark realiza suas execuções baseado em avaliação preguiçosa, onde tarefas são empilhadas até que se decida o momento do início da execução. Para isso, o Spark cria um grafo direcionado acíclico (DAG) que descreve as dependências e a ordem de execução das tarefas. Essas tarefas podem ser operações de transformação ou de ação. Transformações, como um `map` ou um `reduce`, como o nome diz, transformam dados de entrada em outros de saída. Já as ações, como uma tarefa de salvamento de dados em disco, são operações que exigem um retorno de um resultado para o usuário e, por isso, forçam o início das atividades em sua arquitetura preguiçosa.

Da sua criação até a versão 1.6, o Spark possuía apenas a abstração de dados baseada no modelo de Resilient Distributed Datasets (RDDs). O RDD se baseia em operadores `map/filter/reduce`, similares ao modelo MapReduce do Hadoop. Por causa

[5] Até a data de 20/11/2025.

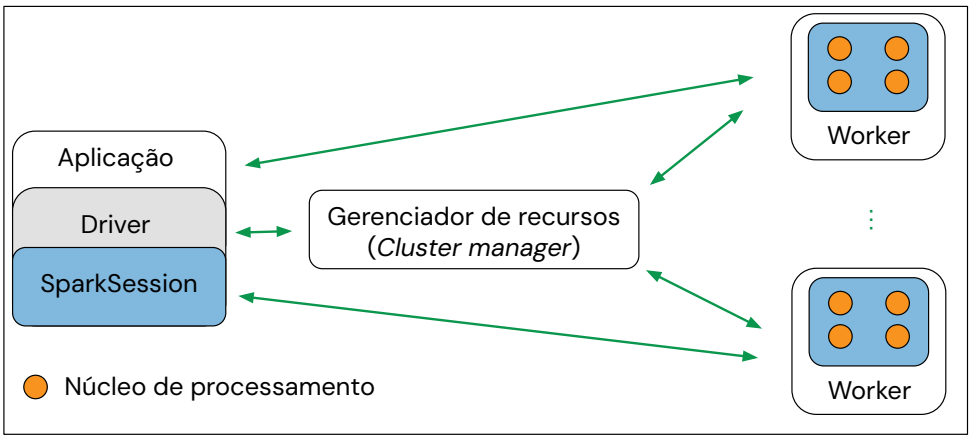


Figura 2.4: Componentes e arquitetura do Spark.

Fonte: Adaptado de Karau et al. (2020, p. 10).

disso, muitas ferramentas de previsões de desempenho (Kroß et al., 2019; Zhao, H. et al., 2019) e até mesmo sistemas multiplataformas mais recentes (Kaoudi et al., 2020; Beedkar et al., 2023), que suportam o Spark, foram desenvolvidos considerando essa abstração. Internamente, um RDD permite criar uma memória distribuída entre os trabalhadores, para distribuir as atividades e viabilizar o reuso de dados, tornando-o tolerante a falhas. Um RDD pode ser interpretado como dados particionados em blocos, divididos entre os trabalhadores, de forma que, durante uma execução, cada processo será responsável por executar a operação programada em seus blocos respectivos.

Hoje em dia, o Spark disponibiliza também a abstração de DataFrame, uma abstração em cima do RDD, que busca ser uma interface similar à do Pandas, mas para cenários de grandes volumes de dados (Armbrust et al., 2018). Internamente, DataFrames em Spark podem ser interpretados como RDDs com uma estrutura fixa de dados, definidos por um esquema. O fato da tipagem dos dados ser previamente descrita possibilita uma série de otimizações a partir do seu otimizador, Catalyst (Armbrust et al., 2015). Por exemplo, como as operações a serem realizadas possuem um tipo de dados pré-definido, o Catalyst consegue realizar reorganizações internas do plano de execução para encontrar um mais eficiente, mantendo a lógica de execução.

Inicialmente, como mostra a figura 2.5, as transformações aplicadas a um DataFrame são convertidas em um plano lógico. Este plano é otimizado por meio de melhorias lógicas baseadas em SQL. Posteriormente, ocorre a conversão desse plano em diversos planos físicos, os quais correspondem às operações internas do Spark (mais próxima do RDD). A escolha do plano físico é determinada por um modelo de custo, que busca selecionar o plano que fornece uma estimativa mais rápida para a redução do número de linhas ao início da execução. Após a definição do plano físico escolhido, ele é, enfim, convertido em código-fonte para ser executado.

Similar à arquitetura Pandas/Scikit-Learn, o Spark possui uma biblioteca dedicada

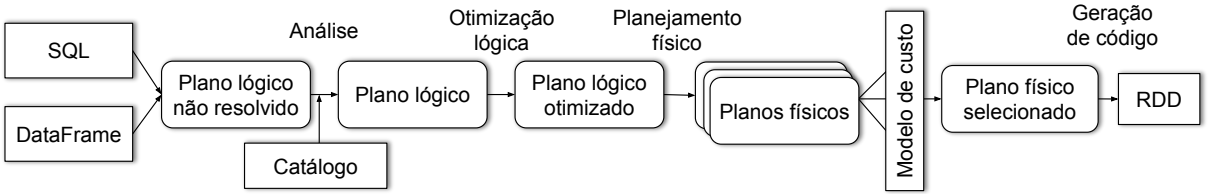


Figura 2.5: Exemplificação do funcionamento do Catalyst no Spark.

Fonte: Adaptado de Armbrust et al. (2015)

para aplicações de aprendizado de máquina, denominada Spark MLlib (Meng et al., 2016). A biblioteca conta com a implementação de diversos estimadores e algoritmos para a criação de modelos de aprendizado de máquina. Essas implementações usam estruturas nativas do Spark para armazenar os dados em memória, garantindo alta escalabilidade de processamento das tarefas.

2.2.3 cuDF

A biblioteca cuDF (NVIDIA RAPIDS, 2025a) é uma tecnologia desenvolvida pela NVIDIA como parte do ecossistema RAPIDS, com o objetivo de acelerar o processamento de dados tabulares a partir da utilização de GPUs. Trata-se de uma alternativa eficiente às bibliotecas tradicionais baseadas em CPU, permitindo a manipulação de grandes volumes de dados com uma abstração de alto nível diretamente na GPU. A biblioteca oferece uma API similar à do Pandas, o que facilita a transição para usuários já familiarizados com essa ferramenta. Atualmente, em sua versão 25.10, a cuDF é baseada em CUDA (Compute Unified Device Architecture), tecnologia proprietária da NVIDIA, o que implica que apenas placas gráficas da marca NVIDIA são compatíveis com essa biblioteca.

A figura 2.6 exemplifica o funcionamento do cuDF. Quando a biblioteca é ativada, um *wrapper* para o Pandas é automaticamente carregado. As operações realizadas a partir desse *wrapper* são executadas na GPU sempre que possível; caso contrário, os dados são copiados da memória da GPU para a memória RAM, para que a operação seja executada diretamente pelo Pandas. Ao final desse processo, os dados são novamente transferidos para a GPU, permitindo a continuidade das demais operações.

Atualmente, cerca de 60% das operações e funções implementadas no Pandas estão disponíveis no cuDF, cobrindo a maior parte das operações mais comuns (Cotton, 2025). No entanto, para tarefas menos usuais, é necessário recorrer à compatibilidade com o Pandas para que sua execução possa ocorrer na GPU. Uma lista detalhada da compatibilidade entre Pandas e cuDF está disponível na documentação oficial (NVIDIA RAPIDS, 2025b). Mesmo para métodos já suportados pelo cuDF, não necessariamente existe compatibilidade completa com o Pandas. Por exemplo, a operação replace no Pandas, usada para substituir valores de uma coluna com base em uma lógica, possui três parâmetros que ainda não são suportados no cuDF: `regex`, que permite substituições por expressões

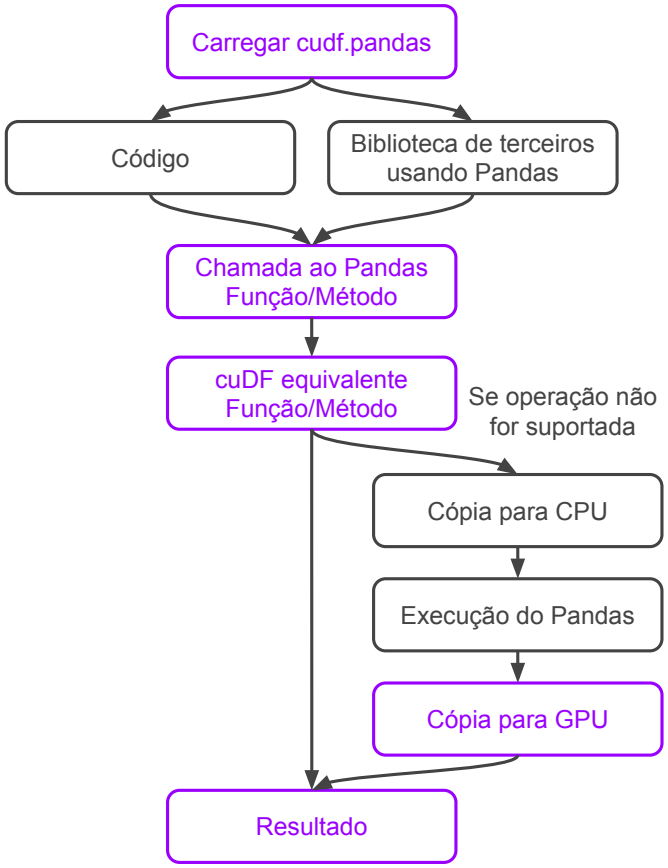


Figura 2.6: Exemplificação do funcionamento do cuDF.

Fonte: Adaptado de NVIDIA RAPIDS (2025a)

regulares; `method`, que especifica a abordagem de substituição para valores escalares; e `limit`, parâmetro que limita o número de substituições quando o `method` é definido.

O ecossistema RAPIDS também inclui uma biblioteca chamada cuML (NVIDIA RAPIDS, 2025c), dedicada a aplicações de aprendizado de máquina, de forma análoga ao Scikit-Learn no ecossistema Pandas, oferecendo suporte a cerca de 27 algoritmos. Assim como o cuDF, o cuML (atualmente na versão 25.10) executa seus algoritmos diretamente na GPU sempre que possível. Caso contrário, o *wrapper* do cuML direciona a execução para o Scikit-Learn. No entanto, nem todos os algoritmos implementados são totalmente executados na GPU. Em alguns casos, determinadas partes do processo ainda podem depender de computação na CPU.

2.3 Interfaces de prototipação em sistemas multiplataformas

Uma parte importante na definição de um sistema multiplataforma é o projeto da sua interface de prototipação. A escolha da abordagem, seja a partir de uma linguagem de

programação ou por uma interface visual, afetará não só a usabilidade da ferramenta mas também a complexidade e a qualidade da solução. Nesse contexto, pesquisas envolvendo ferramentas e linguagens unificadas de programação são temas antigos (Bukhres et al., 1993; Carey et al., 1995). Quando se trata de linguagens, sistemas podem ser criados de três formas: (*i*) permitindo a codificação a partir de todas as sintaxes suportadas; (*ii*) utilizando como padrão uma das linguagens suportadas pelo sistema; (*iii*) ou a partir de uma linguagem própria, independente das ferramentas suportadas.

A primeira abordagem é mais complexa, pois exige grande esforço em compatibilizar as ferramentas. As soluções mais comuns são aquelas em que a sintaxe de todas as ferramentas é restrita ao SQL (Gog et al., 2015). Já a segunda abordagem simplifica o processo ao limitar o escopo de compatibilidade, mas ainda exige esforços quando ferramentas possuem grandes diferenças em suas implementações (Rong et al., 2022). A terceira abordagem é a mais tradicional; nesse contexto, a construção dos sistemas é baseada em uma linguagem funcional de operadores genéricos, podendo se encaixar em diversas ferramentas (The Fugue Development Team, 2022; Apache Software Foundation, 2023a; Beedkar et al., 2023). Embora a prototipação via linguagens seja mais predominante, recentemente, a utilização de sistemas de codificação visuais vem ganhando atenção na comunidade, principalmente em ambientes de Ciência de Dados (Hall et al., 2009; Podpečan et al., 2011; Prekopcsak et al., 2011; Kranjc et al., 2012; Mund, 2015; Santos et al., 2017; Lucas et al., 2018; Dataiku Team, 2024). Nessa abordagem, além de prover uma interface visual, geralmente onde usuários podem clicar e arrastar operações, também lidam com o escalonamento e execução das atividades.

2.3.1 Sistemas baseados em linguagens de descrição

O Apache Beam é talvez um dos arcabouços atuais mais conhecidos com foco na unificação de diferentes ferramentas de processamento de dados. Ele é um projeto baseado na experiência do Google com o DataFlow (Akidau et al., 2015) para fornecer um modelo de programação unificado para representar e transformar dados em processamentos em lote ou em dados contínuos. Similar ao DataFlow, o Apache Beam define seu fluxo de dados como um *pipeline* de tarefas definidas a partir de seus operadores, que são descritos seguindo uma lógica `map/filter/reduce`. Mais recentemente, junto com o suporte à linguagem Python, o Apache Beam implementou o Beam DataFrames, uma abstração para permitir a interação com o DataFrame do Pandas, sendo um primeiro avanço nessa nova linha de programação baseada em operações de alto nível. O recurso permite interpretar dados do Apache Beam como um DataFrame, que pode ser invocado a partir de operadores Beam ou convertidos em DataFrames pela API do Pandas para a execução de comandos ainda não suportados. A grande diferença entre Beam DataFrames e Pandas DataFrames é que as operações são via API do Beam, são empilhadas para oferecer

suporte ao modelo de processamento paralelo do Beam, enquanto operações diretamente em Pandas seguem o seu modelo de avaliação ansiosa.

Fugue (The Fugue Development Team, 2022) é outro projeto recente nessa mesma direção, mas já focado na abstração de dados tabulares como o DataFrame. A ideia do Fugue é permitir a execução de *workflows* em motores de processamento mais adaptados de *big data* como Spark, Dask, Ray e DuckDB a partir de códigos FugueSQL (linguagem derivada do SQL), Pandas ou Polars. Sua linguagem FugueSQL, por exemplo, permite a conversão de códigos SQL em códigos em qualquer um dos *backends* suportados. É um projeto promissor, no entanto, aspectos de desempenho ainda precisam ser avaliados, uma vez que muitos dos códigos gerados são similares à ideia de portabilidade de um processamento sequencial em sistemas distribuídos, ao invés de uma adaptação da lógica de programação para a realidade da ferramenta alvo. Por exemplo, conforme exemplificado na documentação oficial do sistema (The Fugue Development Team, 2023), a conversão do Fugue para um código Python onde é realizada a criação de uma nova coluna a partir de predições de um algoritmo de regressão linear do Scikit-Learn para a linguagem Spark envolve encapsular o modelo criado originalmente no Scikit-Learn como uma UDF em Spark. Embora o código execute com sucesso, uma abordagem mais natural seria uma adaptação do código completo, por exemplo, criando o modelo utilizando o algoritmo de regressão linear disponível na MLLib do Spark.

De fato, cada projeto tem sua visão para permitir uma interface única. Como cada ferramenta de processamento tem suas particularidades, alguns projetos como Apache Beam buscam prover uma interface de desenvolvimento mais simples para generalidade das execuções entre diversos modelos. Já projetos como Fugue buscam manter o mesmo nível de abstração a partir de uma compatibilidade de código, no entanto, nessa tentativa de deixar os códigos compatíveis, impactos no desempenho podem acontecer. Por causa desses problemas, sistemas de execução e de codificação visuais vêm sendo utilizados como alternativa para abstrair as complexidades de se criar uma linguagem universal.

2.3.2 Sistemas baseados em interfaces visuais

Sistemas visuais, como ClowdFlows (Kranjc et al., 2012), Lemonade (Santos et al., 2017), Microsoft Azure Machine Learning (ML) Studio (Mund, 2015) e o Orange4WS (Podpečan et al., 2011), permitem expressar atividades de um fluxo de dados por meio de caixas de tarefas visuais. A principal vantagem dessa abordagem em relação às interfaces baseadas em linguagens de programação é possibilitar a criação de diversos tipos de fluxos de trabalho, utilizando operações bem definidas para o sistema. Por exemplo, uma simples operação de substituição de *tokens* por valores numéricos pode ser realizada de pelo menos duas formas em Pandas: via operador de transformação (`apply`) ou pelo operador `replace`. Em uma abordagem por linguagem de programação, o sistema deverá

mapear cada um dos operadores para garantir total compatibilidade com as ferramentas suportadas. Além disso, do ponto de vista de análise de desempenho, diferentes abordagens podem ter desempenhos distintos. No exemplo anterior, a operação de substituição via `replace` pode ser ordens de magnitude mais rápida do que a primeira abordagem. Dessa forma, os sistemas baseados em linguagens, por oferecerem flexibilidade ao usuário para expressar suas aplicações de diversas formas, possibilitam a execução de planos não otimizados.

Já as abordagens visuais geralmente se baseiam na implementação de um conversor das dependências expressas visualmente, bem como de seus parâmetros, em códigos na linguagem alvo. Para cada operação suportada, é necessário criar um *template* de código correspondente para cada linguagem. Cada tarefa visual pode corresponder a mais de um comando ou operação na linguagem alvo, permitindo que operações complexas sejam expressas em tarefas de alto nível. Além disso, devido ao fato da interface gráfica limitar as escolhas de parâmetros e operações possíveis às fornecidas visualmente para o usuário, é possível garantir uma execução mais bem-planejada e otimizada, seguindo as boas práticas de cada ferramenta.

Lemonade é um desses sistemas visuais focadas em Ciência de Dados, voltada para usuários que não conhecem uma linguagem de programação ou que precisam desenvolver fluxos de trabalho usando o conjunto de ferramentas existente (Santos et al., 2017). O sistema é dedicada à criação de fluxos de análise e mineração de dados em nuvem, com garantias de autenticação, autorização e contabilização de acesso. Em sua versão atual, o Lemonade possui diretrizes capazes de gerar e executar códigos em Apache Spark, Pandas (Python com Pandas, Scikit-Learn e outras bibliotecas relacionadas), COMPSs (Tejedor et al., 2017) e Keras+Tensorflow (Chollet et al., 2015). As três primeiras são motores de processamento geral, enquanto a ferramenta Keras é focada em redes neurais.[6] Internamente, o Lemonade possui um conjunto de serviços responsáveis por manter informações sobre as operações disponíveis em cada opção de ferramenta, informações sobre os conjuntos de dados e os recursos computacionais disponíveis para cada usuário, permitindo que a ferramenta converta o fluxo visual em código-fonte e o execute de maneira transparente.

Usando uma interface para a construção visual de fluxos, o Lemonade permite a criação e execução de fluxos, encapsulando os detalhes de armazenamento, codificação e processamento distribuído. Isso possibilita o uso em hardwares locais ou em nuvem por especialistas no domínio dos dados. Para alcançar esse propósito, o sistema oferece ao usuário um conjunto de caixas de operações que representam funções e algoritmos previamente implementados em uma linguagem de programação. A partir disso, o usuário é capaz de criar um fluxo lógico estabelecendo conexões entre as caixas por meio de pontos de comunicação direcionais (entrada ou saída). Cada caixa possui um número específico de entradas e saídas, determinando o tipo de resultado que será produzido, bem como o

[6]Uma nova ferramenta baseada em Polars está em fase de desenvolvimento.

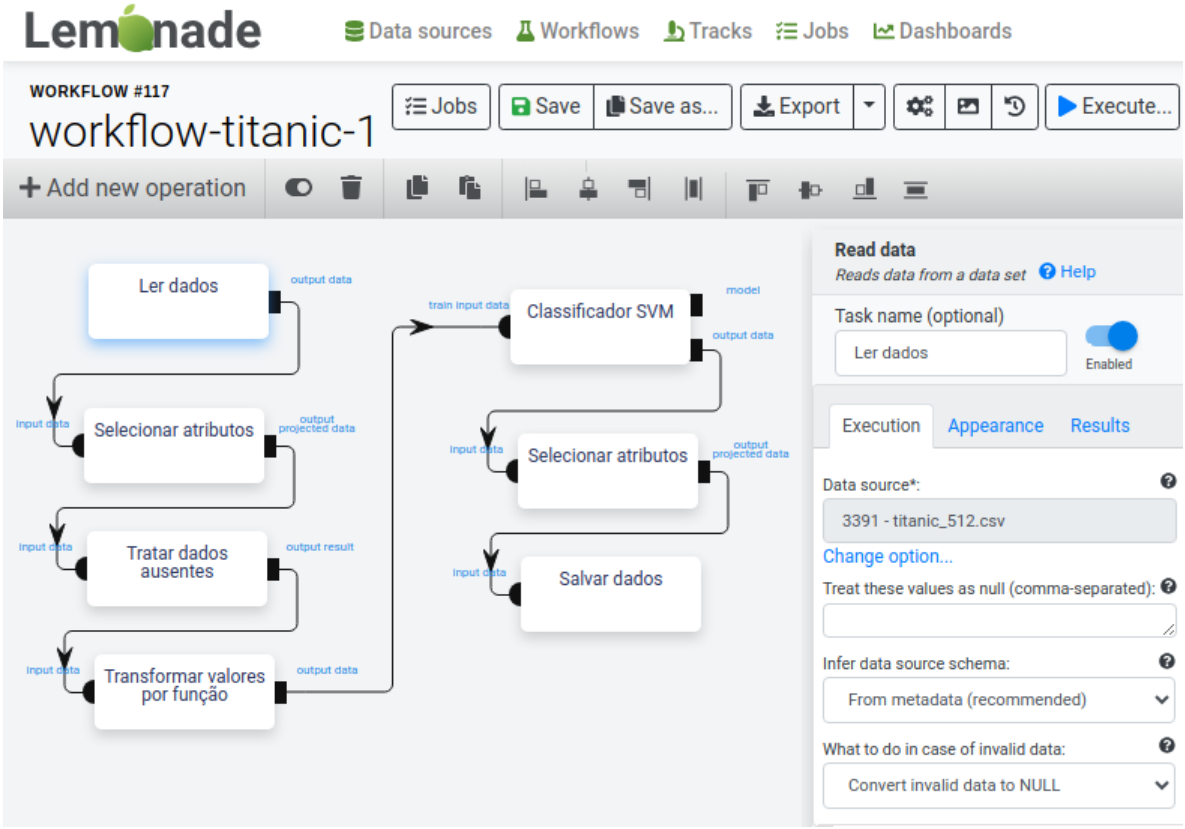


Figura 2.7: Exemplo de uma aplicação criada a partir do Lemonade

conjunto de parâmetros que o usuário pode especificar durante a execução. Existem caixas no Lemonade nas quais a especificação de determinado parâmetro é obrigatória, como, por exemplo, o nome do arquivo de entrada a ser lido. Além disso, existem parâmetros opcionais, como o número máximo de iterações de um algoritmo, para os quais são fornecidos valores padrão caso o usuário opte por não especificar. A Figura 2.7 exemplifica o tipo de aplicação que o Lemonade é capaz de criar. Tal aplicação consiste em: uma etapa de pré-processamento dos dados representada pelas caixas de leitura, remoção de algumas colunas, o tratamento de elementos ausentes e uma transformação a partir de um mapeamento definido por uma função do usuário; uma outra etapa utilizada na criação de um modelo de classificador e, por fim, uma última etapa de persistência do resultado.

Capítulo 3

Abordagem proposta

O presente capítulo discute nossa abordagem para a criação de um sistema multiplataforma eficiente. Para auxiliar nesse processo, a figura 3.1 apresenta uma visão geral do esquema de recomendação das melhores opções de ferramentas para cada tipo de tarefa em um fluxo de dados.

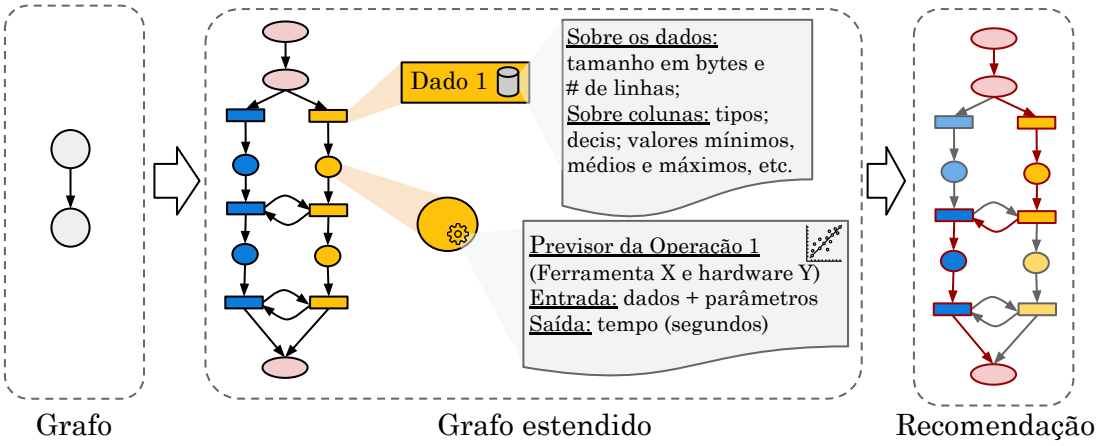


Figura 3.1: Esquema simplificado da abordagem para a recomendação de ferramentas.

Em linhas gerais, a partir da modelagem inicial do fluxo de dados como grafos, discutida inicialmente na seção 2.1.1, criamos uma nova representação estendida do grafo original para modelar diversos aspectos de uma execução, conforme será abordado na seção 3.1. Nessa representação, além das operações serem diferenciadas entre as ferramentas, ela permite modelar informações sobre os dados de entrada de cada operação. Essas informações servem como parâmetros de entrada para modelos de regressão estimarem o tempo de execução de cada operação em cada ferramenta. A partir desse enriquecimento, o processo de recomendação (seção 3.2) se resume a escolher o caminho mais curto para percorrer todos os tipos de operações presentes no fluxo. A seção 3.3 discute como modelos de previsão de tempo de operações podem ser criados a partir de um conjunto de treinamento inicial e como esses mesmos modelos podem ser enriquecidos com o passar do tempo. Por fim, a seção 3.4 apresenta o protótipo da implementação de nossa abordagem no Lemonade.

3.1 Representação estendida do grafo de tarefas

Uma parte importante de um sistema multiplataforma é a abordagem utilizada para representar os fluxos de dados no algoritmo de recomendação, pois essa definição impacta diretamente a forma como o espaço de busca da solução será percorrido.[1] Como discutido no capítulo 2.1.1, essa representação geralmente é feita por meio de grafos multivariantes, um tipo de grafo no qual os nós estão associados a conjuntos de variantes. No entanto, os sistemas multiplataforma atuais tendem a subutilizar as capacidades dessa forma de representação.

Sistemas como o Wayang buscam simplificar a representação do problema ao modelar as variações de execução como um nó "bloco" (isto é, um nó associado a diversas variações de implementação), convertendo o grafo multivariante em um grafo de tarefas simples. Dessa forma, a busca pela solução resume-se em considerar cada variante de um bloco individualmente. Todos os aspectos além das relações de dependência entre as tarefas são armazenados e tratados separadamente do grafo. Assim, as soluções atuais, além de não representarem explicitamente outros tipos de relacionamentos e informações relevantes, como os dados intermediários e os estados de execução, acabam por limitar o nível de sofisticação das recomendações. Além disso, tornam o processo de recomendação mais complexo, uma vez que exigem etapas adicionais de processamento para refinar a solução com base em informações que não estão presentes diretamente no modelo do fluxo.

Por exemplo, além do custo de tempo de execução, uma operação pode gerar múltiplos resultados; esses resultados podem variar conforme a ferramenta utilizada; dados intermediários podem ser transferidos entre ferramentas distintas; e configurações de hardwares distintos podem possuir tempos de inicialização específicos. Uma boa modelagem de fluxo de dados é aquela capaz de capturar a maior quantidade possível de aspectos envolvidos na execução. Isso não apenas permite uma representação mais natural das variáveis do problema, como também viabiliza que o processo de recomendação se resuma a uma caminhada no grafo modelado.

Nesse contexto, nosso trabalho propõe uma representação que permite estimar o custo de execução de um fluxo considerando diversos fatores, como os discutidos a seguir:

Dados de entrada. Um conjunto de informações importantes para prever o custo de execução de uma operação são características dos dados de entrada. Abordagens como as citadas no capítulo 2.1.3, por não terem mecanismos de gerenciamento de bases de dados, não possuem informações previamente calculadas para estimar como dados de entrada serão processados em cada operação ou algoritmo. Mais ainda, mesmo aquelas soluções

[1] Nesse contexto, o termo Design Space Exploration (exploração do espaço de projeto) é comumente utilizado para definir o objetivo e o conjunto de possibilidades que podem ser avaliadas em problemas desse tipo (Cardoso et al., 2017).

que levam em consideração o número de linhas inicial (Armbrust et al., 2015; Rong et al., 2022) apresentam erros elevados de previsão do tempo à medida que o fluxo de dados fica mais complexo. Por exemplo, uma operação de filtragem de dados a partir de um valor poderá reduzir drasticamente o número de linhas dos dados de entrada. Essa redução não será bem estimada por aquelas soluções mais simples. Mesmo otimizadores lógicos SQL, como o Catalyst (otimizador do Spark) ou o Calcite (Begoli et al., 2018), que possuem mecanismos para computar algumas características dos dados de entrada tabulares, o objetivo do sistema não é estimar o tempo de execução, mas sim o tamanho dos dados (geralmente expresso em número de linhas). Tanto no Catalyst quanto no Calcite, o melhor plano de execução (o de menor custo) será aquele que reduz o número de linhas mais rapidamente durante o fluxo. Nem o número de linhas é esperado ser preciso, nem o custo é comparável com outras soluções ou ferramentas.

Parâmetros das operações. Determinados parâmetros em operações e algoritmos podem influenciar o tempo de execução. O número máximo de iterações em um algoritmo interativo como o K-Means é um dos exemplos mais claros de como um parâmetro pode influenciar na execução, onde valores muito altos ou baixos podem aumentar ou reduzir o tempo em que o algoritmo será executado. No entanto, outros tipos de parâmetros, como os categóricos, podem influenciar uma execução. Por exemplo, em uma operação de remoção de elementos ausentes, o parâmetro que especifica a ação a ser aplicada, como remover a linha ou a coluna inteira, substituir por um valor específico ou por valores de média, podem resultar em tempos de processamento diferentes. Outro exemplo é a leitura de dados a partir de fontes diferentes, como dados armazenados no HDFS ou na máquina local. Ou seja, os parâmetros escolhidos pelos usuários durante a criação do fluxo de dados também devem ser levados em consideração na composição do conjunto de características (*features*) de entrada para os modelos de regressão do tempo de execução. Isso é uma das complexidades a mais ao se trabalhar com abstrações de alto nível como a de DataFrame. Soluções de previsões baseadas em `map/filter/reduce` geralmente adotam um comportamento constante para os operadores.

Opções de ferramentas. Como visto no capítulo 2.2, cada ferramenta de execução atua de forma diferente, algumas, como o Pandas, possuem processamento em *single-core*; já outras, como o Spark, além de serem *multi-cores*, também podem suportar a divisão de tarefas em múltiplas máquinas (*multi-nodes*). Devido a essas diversas possíveis diferenças, tentar representar o custo intrínseco de cada ferramenta como coeficientes em uma expressão pode não ser a melhor abordagem. Determinadas operações, por exemplo, podem ser mais rápidas em uma ferramenta do que em outra.

Opções de migração de dados. Hoje em dia, as ferramentas estão implementando mecanismos, expressos como operadores, para importar e exportar dados para outras

ferramentas populares. Nesse contexto, o projeto Apache Arrow vem se consolidando como uma representação intermediária para esses DataFrames, podendo ser utilizado para facilitar a migração de DataFrames de uma implementação para outra. Ferramentas como Pandas e Spark adotam essa tecnologia, e por isso, as representações de ambas podem ser migradas facilmente entre si. Outra abordagem, principalmente em ferramentas que não possuem opções de migrações diretas, é a persistência em disco, para leitura na ferramenta seguinte, como adotado pelo Octopus-DF.

Configurações de hardware. Enquanto algumas soluções buscam generalizar o custo de execução envolvendo fatores como número de núcleos de processamento, rede e memória, nossa ideia busca simplificar esse processo (Baldacci et al., 2019; Beedkar et al., 2023). Acreditamos que essa abordagem, contraintuitivamente, tem o potencial de reduzir a qualidade das previsões por tornar a modelagem mais complexa. Em muitos casos, ferramentas complexas como o Apache Spark possuem uma série de parâmetros internos que podem influenciar no desempenho de uma execução. Considerar apenas os recursos computacionais pode levar a uma previsão imprecisa. Diferentemente dessas soluções, nossa abordagem permite a execução de fluxos em diversas configurações de recursos computacionais. No entanto, em vez de permitir a execução em qualquer configuração, os hardwares suportados serão limitados a um conjunto de opções pré-definidas pelos administradores, similar ao que ferramentas visuais como o Lemonade geralmente expõem aos seus usuários.

Para incluir todos esses aspectos, criamos uma representação estendida, conforme ilustrado na figura 3.2. Nessa modelagem, existem três tipos de vértices: (*i*) estados de execução, que mapeiam os eventos de criação do ambiente, início e término da execução; (*ii*) operações a serem executadas em uma determinada ferramenta, representadas por círculos, onde cada cor e tipo de borda representa uma ferramenta possível de ser utilizada; e (*iii*) nós, desenhados como retângulos, que representam o estado dos dados em uma ferramenta após a execução de uma operação. Todos esses nós são relacionados por dois tipos de arestas direcionadas que representam a ordem dos eventos. Algumas arestas possuem pesos, representando custos de processamento (medidos em segundos) durante a execução de alguma atividade: do tempo gasto entre a solicitação de recursos para inicializar uma execução em uma configuração de hardware (custo C); do tempo gasto para a execução de uma operação (custo T); e do custo de migração de dados entre ferramentas (custo M). Na figura, as arestas que possuem custos são representadas como linhas contínuas, enquanto as sem custo são tracejadas. No grafo, as arestas não ponderadas especificam as dependências de dados para uma operação ou as relações de início e término de execução, sem custos envolvidos.

No grafo, a execução de uma operação (vértices circulares) sempre produzirá um novo estágio de dados (vértices retangulares) e essa execução sempre estará atrelada a

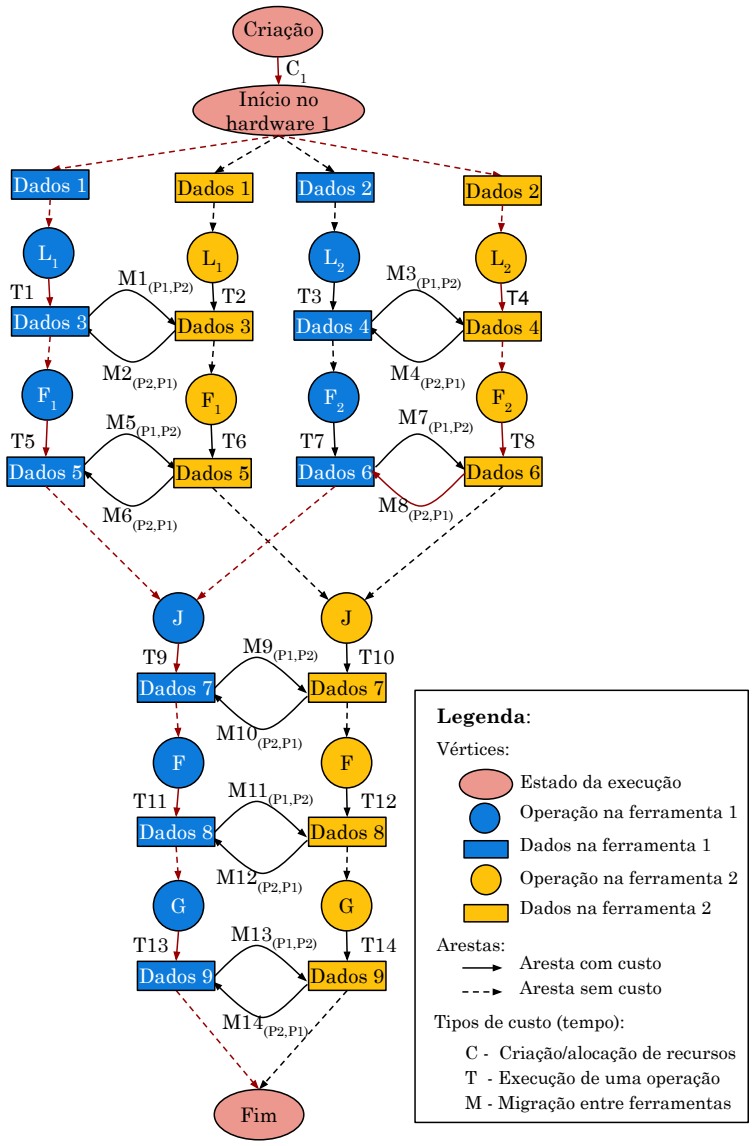


Figura 3.2: Representação estendida do grafo da fig. 2.1.

um custo T (uma aresta ponderada que representa a linhagem desse dado). Embora não seja uma regra, uma operação quase sempre terá, como entrada, um estágio de dados que representa um arquivo de entrada ou o resultado de um estágio do processamento realizado por outra operação anterior. Nesse caso, essas arestas que apontam para uma operação não terão custos, uma vez que relacionam apenas as dependências das operações. Como seria inviável separar os tempos de leitura de dados, processamento e escrita de dados, optamos por representar todo o tempo de processamento de uma operação, envolvendo a leitura de dados de entrada e a escrita dos dados de saída, como um único peso associado à aresta que liga a operação ao conjunto de dados de saída.

Os custos de configuração derivam do fato de que é esperado, por exemplo, que um tempo gasto de alocação dos recursos computacionais em uma máquina local seja diferente de um custo envolvendo um cluster de várias máquinas. Por isso é importante

discretizar o custo C em nossa modelagem, representando os tempos de ativação de cada ambiente.

Finalmente, consideramos que cada ferramenta atualmente possui sua própria representação interna de dados em memória, sendo potencialmente diferente da de outras ferramentas. Sendo assim, na maioria dos casos a migração de dados entre ferramentas exige a execução de uma etapa conversão de dados entre as ferramentas, que consumirá algum tempo M.

Com base nessa discussão, o passo-a-passo para a criação do grafo estendido é descrito a seguir:

Primeiro, estenda a representação tradicional adicionando vértices que representem os conjuntos de dados manipulados pelo fluxo de trabalho:

1. para cada vértice de entrada v (um vértice sem arestas de entrada), adicione um vértice de conjunto de dados d_i e uma aresta (d_i, v), onde i deve ser diferente para todos os vértices;

2. para cada aresta (v, w) na representação tradicional, crie um vértice de conjunto de dados d_i, onde i deve ser diferente para todos os vértices, e substitua a aresta (v, w) pelas arestas (v, d_i) e (d_i, w);

3. para cada vértice de saída v na representação tradicional (um vértice sem arestas de saída), crie um vértice de conjunto de dados d_i e uma aresta (v, d_i), onde i deve ser diferente para todos os vértices.

Com essa representação intermediária, devemos agora criar um grafo separado para representar sua execução em uma determinada ferramenta, associada a uma certa arquitetura. Para isso, vamos supor que temos os seguintes conjuntos: H, contendo todas as configurações de hardware disponíveis, e, para cada configuração de hardware $h \in H$, E_h, representando os mecanismos que podem ser executados nessa configuração.

4. a partir de cada vértice v na representação intermediária (tanto vértices de operação quanto de conjunto de dados), para todo $h \in H$, crie vértices $v_e \ \forall e \in E_h$;

5. para cada aresta (v, w) no grafo intermediário, para todo $h \in H$, crie arestas $(v_e, w_e) \ \forall e \in E_h$;

6. para cada vértice de conjunto de dados d no grafo intermediário, para cada arquitetura de hardware h, para todos $i, j \,|\, i, j \in EnginesInHardware_h \wedge i \neq j$, crie arestas (d_i, d_j) para representar a migração de um conjunto de dados de um mecanismo para outro (inicialmente, as arestas existem em ambas as direções para todos os casos).

Os passos 4 e 5 acima criam uma cópia da representação intermediária para cada mecanismo de execução disponível. O passo 6 cria arestas para representar a possibilidade de migração de um conjunto de dados para sua representação em um mecanismo de execução diferente na mesma configuração de hardware.

O resultado das ações anteriores é um conjunto de grafos disjuntos, um para cada mecanismo de execução. O passo final é uni-los em uma única estrutura:

7. crie um vértice $Creation$;
8. para cada arquitetura de hardware h, crie um vértice $StartHardware_h$ para representar a instanciação dessa configuração de hardware e adicione uma aresta $(Creation, StartHardware_h)$;
9. para cada mecanismo e associado à configuração h, para todos os vértices de entrada in_e, crie uma aresta $(StartHardware_h, in_e)$;
10. crie um vértice End e conecte todos os vértices de saída (de conjuntos de dados) a ele.

3.1.1 Transformações na representação estendida

A representação estendida da figura 3.2 permite mapear diversas características da execução, como o tempo de inicialização de um ambiente de execução (isto é, alocação de recursos em um hardware), os parâmetros utilizados na execução de uma operação, a entrada e saída de dados, além de diferenciar o comportamento de cada ferramenta envolvida. A partir dessa representação base, outros aspectos da execução ou comportamentos mais complexos podem ser modelados por meio de transformações no grafo. A seguir, discutimos alguns exemplos dessas transformações.

O primeiro tipo de transformação, ilustrado na figura 3.3a, consiste na remoção de nós de operação associados a uma determinada ferramenta, seja por ausência de implementação ou por limitações relacionadas ao tamanho dos dados que a ferramenta suporta. Nesses casos, a operação realizada sobre o grafo consiste na remoção do nó correspondente à operação em questão. Além disso, quando duas ou mais operações consecutivas de uma mesma ferramenta são removidas, os nós intermediários que representam os dados resultantes dessas operações podem também ser eliminados, com o objetivo de otimizar a busca por recomendações. Por exemplo, além da operação de remoção de elementos duplicados (representada no grafo pelo nó R), caso a operação F também fosse removida, o estágio intermediário "Dados 2" poderia ser eliminado sem comprometer a validade da solução final. Esse tipo de transformação também é útil em cenários em que ferramentas, como o Pandas, requerem que os dados caibam integralmente na memória RAM, mas o hardware

disponível não oferece recursos suficientes para isso. A ausência desse tipo de mapeamento, como será discutido posteriormente na seção 4.4.3, pode levar à recomendação de soluções inviáveis.

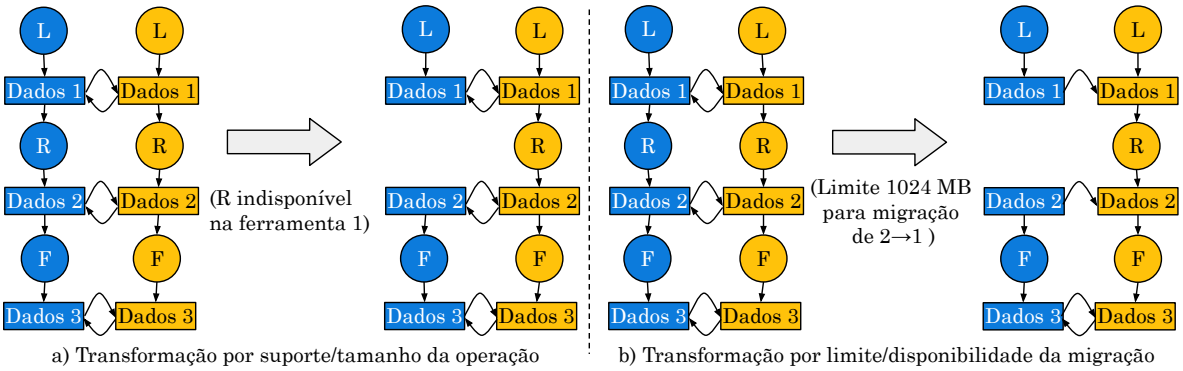


Figura 3.3: Exemplo de transformações em nós e arestas devido a limites e suportes das ferramentas.

Nota: Onde as operações: L - Leitura de dados, R - Remoção de elementos duplicados, e F - Filtragem.

O segundo tipo de transformação, ilustrado na figura 3.3b, consiste na remoção de arestas de migração em casos em que não é possível realizar a transferência de dados entre ferramentas. Esse cenário é relevante quando se deseja restringir explicitamente a possibilidade de migração entre determinados pares de ferramentas, seja por limitações práticas, seja por restrições específicas do hardware ou das ferramentas envolvidas. Como exemplo, podemos citar situações em que determinadas ferramentas impõem limites internos para evitar degradações de desempenho. Um caso comum ocorre no Spark, cuja configuração padrão limita o tamanho máximo de dados que podem ser coletados e centralizados no Driver a 1 GB. Essa limitação implica em uma restrição à migração de dados entre o Spark e outras ferramentas, como o Pandas ou o cuDF, que dependem do carregamento total dos dados na memória centralizada para executar suas operações. Assim, a remoção dessas arestas no grafo garante que a recomendação de caminhos desconsidere soluções inviáveis do ponto de vista técnico.

Esse tipo de transformação também pode ser explorado como uma estratégia de otimização em determinados cenários. Por exemplo, em fluxos que contêm a tarefa de projeção (ou seleção de colunas), é possível otimizar o processo de recomendação removendo as arestas de migração existentes a partir dos dados de entrada dessa operação. A motivação para essa otimização reside no fato de que a seleção de colunas é, em geral, uma operação de baixo custo computacional, e que o volume de dados resultante tende a ser sempre menor ou igual ao volume de dados de entrada. Dessa forma, caso uma migração de dados seja necessária nesse trecho do fluxo, é preferível que ela ocorra após a execução da projeção, e não antes, a fim de reduzir o volume de dados transferidos.

Embora a nossa solução seja capaz de identificar automaticamente os melhores pontos de migração com base nos custos estimados, esse tipo de transformação permite reduzir o espaço de busca durante o processo de recomendação e forçar, de maneira controlada, comportamentos considerados desejáveis em determinados cenários.

Além dos aspectos relacionados ao desempenho, outros fatores também podem motivar a restrição de migração de dados entre ferramentas. Um exemplo comum está relacionado a requisitos de segurança: uma organização pode impor que as etapas iniciais de um fluxo de trabalho sejam executadas exclusivamente em uma ferramenta específica, capaz de operar em um hardware considerado mais seguro. Após a realização de etapas de tratamento ou anonimização dos dados, as demais atividades do fluxo podem, então, ser delegadas a outras ferramentas, com foco em desempenho.

Por fim, o terceiro tipo de transformação, ilustrado na figura 3.4, busca contornar as limitações associadas a ferramentas baseadas em avaliação preguiçosa, como o Spark, que não armazenam temporariamente os resultados intermediários dos estágios de processamento. Nesse contexto, em cenários com bifurcações no fluxo de execução, como representado na figura após a operação de remoção de elementos duplicados (R), em que dois caminhos distintos são originados a partir de filtragens diferentes (F_1 e F_2), a execução no Spark ocorre da seguinte maneira: inicialmente, são computadas as operações L, R, F_1 e G_1; em seguida, o Spark executa novamente as operações L, R, F_2 e G_2. Dessa forma, as operações L e R são processadas duas vezes pela ferramenta. Nesse cenário, para que a previsão do tempo de execução seja mais aderente ao comportamento real do sistema, o processo de estimativa deve considerar essas particularidades.

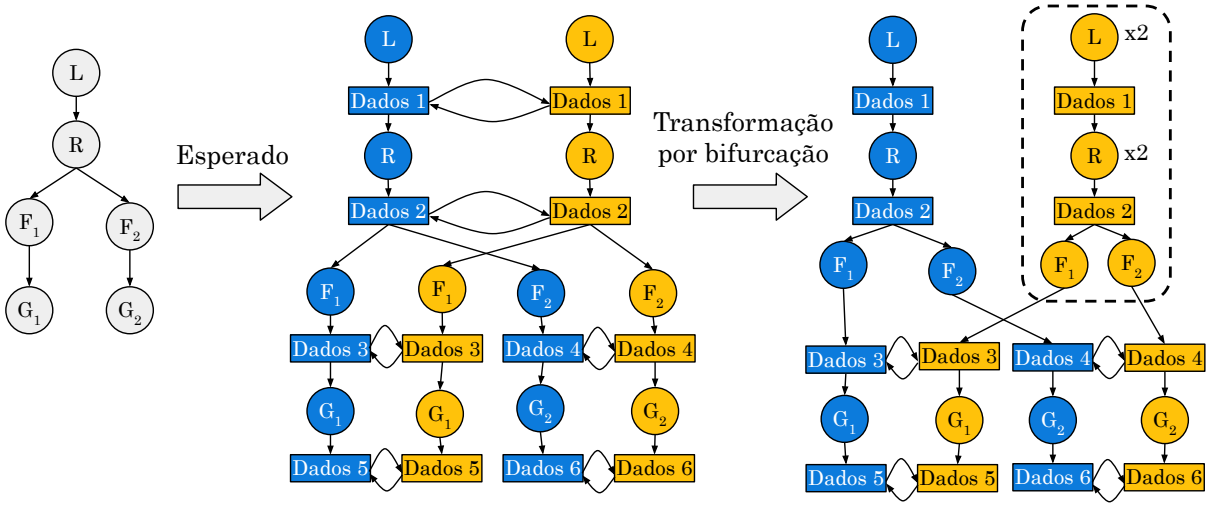


Figura 3.4: Exemplo de transformação devido à presença de bifurcação.

Para mapear esse comportamento, é criado um nó adicional, denominado `meta-nó`, representado pela área tracejada na figura 3.4, o qual encapsula internamente as operações que fazem parte do início da bifurcação (F_1 e F_2), bem como seus nós predecessores (L e

R). O custo estimado de execução dos nós predecessores dentro do `meta-nó` é multiplicado pelo número de ramificações existentes, no exemplo da figura, esse fator é igual a dois.

A principal ideia desse agrupamento é garantir que, em ferramentas baseadas em avaliação preguiçosa, esse conjunto de operações seja tratado como uma única atividade, impossibilitando a migração de dados antes da execução de F_1 e F_2. Assim, durante a busca pelo melhor caminho de execução, o modelo de recomendação poderá decidir se o processamento será realizado na ferramenta 1 (em azul) ou na ferramenta 2 (em amarelo). Caso a execução seja iniciada diretamente pela ferramenta 2, o custo de processamento das operações F_1 e F_2 já estará incorporado na estimativa.

Uma abordagem alternativa seria eliminar a bifurcação do grafo por meio da clonagem dos nós predecessores (L e R), criando ramificações independentes para F_1 e F_2. Embora essa estratégia torne a representação do grafo mais intuitiva, ela também aumenta consideravelmente a complexidade do processo de recomendação. Por esse motivo, optou-se pela utilização do `meta-nó`, conforme ilustrado na figura 3.4.

3.1.2 Estimativas dos dados

Uma etapa importante na criação do grafo estendido é a estimativa do volume de dados como uma propriedade dos vértices de estados de dados. Essa informação será utilizada pelos modelos de regressão (seção 3.1.3), a fim de ajudar na previsão do tempo de execução. A ideia é ser capaz de estimar como os dados de entrada ficarão imediatamente após a execução de uma tarefa, a fim de ajudar os modelos de regressão a estimar o tempo de execução que a operação subsequente levará para processar esses dados recebidos. Nesse contexto, a qualidade da estimativa do volume de dados, seja em bytes ou em número de linhas, está diretamente relacionada ao conjunto de informações disponíveis sobre os dados. Considerando dados tabulares, como os manipulados pela abstração de DataFrame, estatísticas sobre as colunas dos dados, como valores mínimos, médios, desvio padrão e máximos, quantidade de elementos distintos e elementos nulos, bem como a quantidade de elementos em cada um dos decis[2], podem ajudar a estimar a evolução dos dados após a execução de uma operação. O processo de estimativa é realizado sequencialmente, seguindo as etapas:

1. Faça uma ordenação topológica sobre os vértices de dados do grafo estendido;[3]

2. Solicite ao sistema as estatísticas dos n primeiros vértices que representam os arquivos de entrada presentes no fluxo. Como é esperado que um arquivo de dados seja

[2]Decis representam os dez valores que dividem os dados ordenados de uma variável em dez partes iguais, de modo que cada parte representa 1/10 da amostra ou população.

[3]Em teoria dos grafos, uma ordenação topológica de um grafo direcionado e acíclico é uma ordem linear de seus nós em que cada nó vem antes de todos os nós para os quais este tenha arestas de saída.

utilizado em mais de uma execução, é importante ter essas informações precomputadas para reduzir o tempo gasto durante toda a etapa de recomendação de configuração. Essas informações podem ser computadas manualmente e armazenadas pelo sistema, podem ser disponibilizadas a partir de uma integração com sistemas de gerenciamento de dados de propósitos gerais, como o Apache Atlas (Apache Software Foundation, 2015) ou o OpenMetadata (OpenMetadata Development Team, 2024), ou, no caso da utilização de um sistema de criação de fluxos visuais, como o Lemonade, podem ser obtidas diretamente pelo sistema;

3. Para cada novo vértice percorrido, estime como os dados serão atualizados após a execução da operação que produziu os dados em questão. Essa estimativa exige uma lógica de mapeamento para cada tipo de operação que, a partir dos parâmetros utilizados na operação e do estado dos dados anterior, estime como as estatísticas sobre os dados irão evoluir. Essa etapa será explicada com mais detalhes a seguir.

Para uma estimativa dos dados eficiente, cada operação suportada pelo sistema multiplataforma terá que implementar uma lógica capaz de estimar os dados de saída a partir da entrada e dos parâmetros utilizados. A cada operação, as estatísticas dos dados de entrada, como número de linhas, elementos distintos e ausentes, valores mínimos e máximos, além dos decis, são utilizadas para gerar as estatísticas dos dados de saída, sendo algumas operações mais precisas que outras, dada a complexidade das operações. Operações que possuem um alto grau de precisão de estimativa são operações como Ordenação, Adição de linhas ou de colunas, e Seleção de colunas. Operações desse tipo envolvem basicamente uma reorganização das estatísticas de entrada. Outras operações, como Filtragem, Agrupamento de linhas ou Junção, são menos regulares, casos em que heurísticas precisam ser utilizadas.

A figura 3.5 apresenta dois exemplos de como o número de linhas de saída pode ser estimado a partir de informações dos dados de entrada: o primeiro considerando uma operação de filtragem de linhas onde os valores da coluna alvo sejam superiores a 32, e o segundo agrupando linhas similares. Na figura, linhas coloridas representam o que seria os resultados reais, já as linhas marcadas por um círculo vermelho representam as informações estimadas. Na Filtragem (figura 3.5-a), primeiramente encontramos qual o decil que engloba o limiar da condição de filtragem; depois disso, contabilizamos as quantidades de elementos presentes nos decis que satisfazem a condição. No exemplo, para a filtragem a partir de valores maiores que 32 em uma coluna com valores de 1 a 100, selecionamos os decis entre 41 e 91, cuja soma resultante de elementos nesse intervalo é 60. Já no agrupamento de linhas a partir de duas colunas, o cálculo da estimativa envolve a análise da quantidade de elementos distintos. Não temos como pré-avaliar a correlação dos elementos entre as colunas, pois para isso, seria necessário processar todos os dados durante a estimativa. Ao invés disso, nossa heurística considera a quantidade de

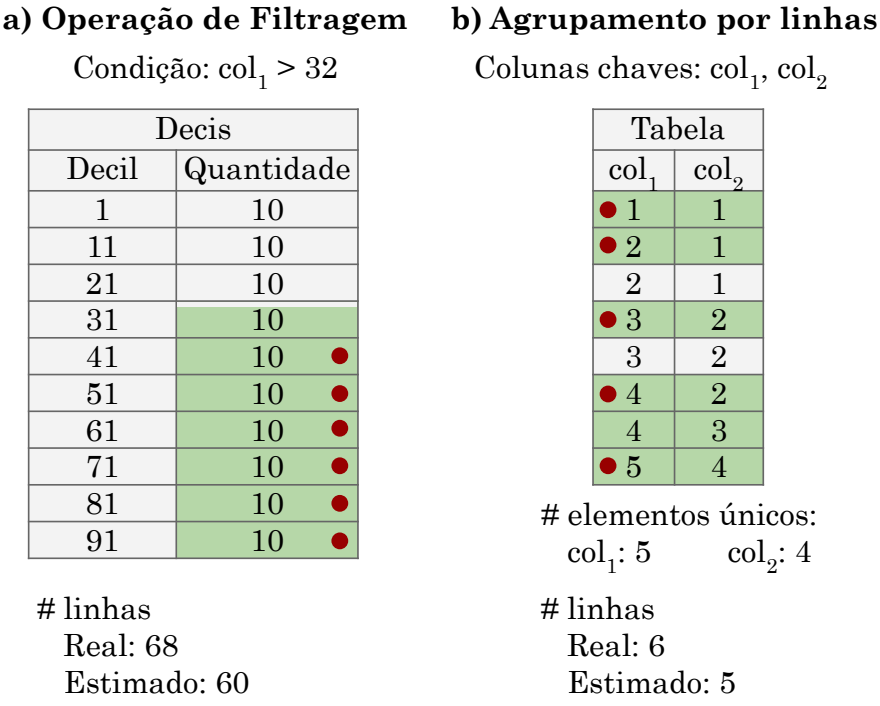


Figura 3.5: Estimativas do número de linhas de saída a partir de informações da entrada.

elementos distintos da coluna de maior diversidade. No exemplo da figura 3.5-b, o número de linhas resultantes é estimado como sendo igual ao número de valores na "col_1", ou seja, o agrupamento de linhas a partir das colunas "col_1" e "col_2" será estimado como 5 linhas resultantes.

Embora as operações sobre DataFrames possuam um alto nível de abstração e, consequentemente, uma grande diversidade, internamente o comportamento das operações pode ser modelado a partir de uma composição de operações mais simples. Por exemplo, operações de transformações por funções de mapeamento possuem um comportamento similar à operação de substituir elementos por valor, de aplicar modelos de aprendizado de máquina, substituir elementos ausentes por um determinado valor, entre outros. Todas essas operações têm em comum a criação de uma nova coluna ou a alteração de uma já existente a partir de uma função. Outro exemplo é a similaridade entre as operações de tratar dados ausentes e a de agrupamento de linhas, ambas envolvem avaliar o número de elementos duplicados em cada coluna para estimar o número de linhas resultantes.

A implementação dessa lógica, que permite estimar como os dados vão sendo transformados ao longo das atividades, sem a necessidade de execução do fluxo de dados, pode ser utilizada em outros contextos além de auxiliar na previsão do tempo de execução. Informações desse tipo são úteis, por exemplo, para ajudar ao usuário a entender melhor o fluxo de dados proposto, identificando, possivelmente, algum gargalo ou anomalia.

3.1.3 Modelagem das características de uma operação

A etapa de modelagem de características se resume a definir qual tipo de informação sobre uma operação deverá ser passada ao modelo de regressão para descrever o tempo de execução. Esta atividade nem sempre é trivial; no entanto, a ideia é escolher, para cada tipo de operação, os principais parâmetros que podem justificar uma variação do tempo de execução. Exemplos, já mencionados anteriormente, são o número máximo de interações para um algoritmo como o K-Means ou parâmetros categóricos, que especificam uma opção de comportamento, como o tipo de ação a ser realizada em caso de elementos ausentes ou o método de inicialização de um algoritmo.

Infelizmente, criar um sistema de definição automática das características seria uma tarefa complexa. Por exemplo, especificar todos os parâmetros que foram definidos durante a criação do fluxo cria modelos com alta dimensionalidade e diversidade de cenários, impactando diretamente, não apenas no tempo gasto para a criação de uma amostra de treinamento, como também na qualidade das previsões. Em geral, nossa metodologia para a seleção manual das características pode ser esquematizada pelas seguintes etapas:

1. A partir da estimativa dos dados de entrada, discutida na seção anterior, adicione as informações do número de linhas de entrada e de saída de dados (se o número de linhas for constante, adicione apenas o primeiro);

2. Selecione os parâmetros categóricos que especificam um tipo de ação em uma operação ou um método interno de um algoritmo;

3. Para parâmetros numéricos, selecione apenas os casos que especificam um comportamento de execução (por exemplo, número de *clusters* em algoritmo de agrupamento ou número máximo de interações);

4. Cada operação tem suas particularidades, podendo, inclusive, não ter nenhum parâmetro definido pelas etapas anteriores. Por causa disso, adicione outros tipos de informações que relacionam a evolução dos dados e os parâmetros utilizados na execução.

Uma modelagem que permita especificar todos os fatores influentes no tempo de execução exige, por parte do desenvolvedor, um certo conhecimento sobre o comportamento interno da operação para as diversas ferramentas de execução. No entanto, pela nossa experiência, as etapas descritas acima servem como um ponto de partida promissor. Por exemplo, como a ideia é construir um modelo para cada operação em uma ferramenta e um tipo de hardware, não precisamos considerar as diferenças de funcionamento interno das ferramentas; tudo isso é abstraído pelo modelo. Mesmo que uma operação tenha diversas possibilidades de parâmetros, não precisamos especificar todos, uma vez que a influência que esses parâmetros possuem na execução já está sendo modelada a partir da

estimativa da evolução do dado. A tabela 3.1 apresenta três exemplos para a modelagem de características em operações diferentes.

Tabela 3.1: Exemplo de seleção de características para três operações.

Operação	Parâmetro
Junção	Soma do número de linhas dos dois dados de entrada Número de linhas estimado na saída Tipo de junção
SVM	Número de linhas dos dados de entrada Número máximo de interações Dimensionalidade do vetor de entrada
Transformação	Número de linhas dos dados de entrada Número de expressões de transformação Quantidade de colunas utilizadas nas expressões Número de colunas sobrescritas

No exemplo da tabela 3.1, a modelagem do custo da operação de junção ocorre pela relação entre a quantidade de dados na entrada e na saída, bem como o tipo de junção (se é uma junção natural, junções à esquerda ou à direita, entre outros). À medida que as ferramentas disponibilizarem outras opções de parametrização de execução, essa evolução pode ser incluída na modelagem. Outro exemplo é a operação de transformação, que suporta uma ou mais expressões de transformação de dados, podendo criar novas colunas ou alterar as já existentes. Essas diferenças de execução são mapeadas pelos quatro parâmetros especificados na tabela.

A partir dessa seleção de características inicial, os desenvolvedores podem atualizá-la em etapas futuras caso queiram tentar melhorar a qualidade das previsões durante a etapa de treinamento dos modelos, discutida na seção 3.3. Aliado a isso, algoritmos de redução de dimensionalidade, algoritmos e métricas de seleção e avaliação de características podem ser utilizados para otimizar os resultados.

3.2 Recomendação da solução

Uma vez criada a estrutura do grafo estendido, o problema pode ser modelado como o de encontrar o conjunto de caminhos de menor custo, da origem (vértice “Criação”) ao destino (vértice “Fim”), englobando todos os tipos de operações e estágios de dados. Podemos utilizar essa modelagem de diferentes formas: identificar a melhor ferramenta de execução (seguindo uma abordagem tradicional); identificar a melhor configuração de hardware; ou identificar a melhor configuração oportunista de ferramentas.[4] A seguir, discutiremos com mais detalhes esse processo.

[4]O termo oportunista vem sendo utilizado na área de sistemas multiplataformas, como no Wayang (Beedkar et al., 2023), para definir abordagens onde o sistema é capaz de identificar uma con-

Caso o grafo estendido não possua bifurcações (por exemplo, pontos onde mais de um fluxo é criado a partir de um estágio de processamento) ou junções (por exemplo, operações de junção, união de DataFrames ou qualquer outra operação que necessite de mais de uma entrada), o problema proposto é similar ao de encontrar os caminhos de custo mínimo. Para esses casos, a execução de algoritmos como o de Dijkstra é suficiente para encontrar o melhor plano de execução. No entanto, quando o grafo não satisfaz essa condição, como na figura 3.2, a resolução do problema é mais complexa uma vez que, nesse caso, o resultado seria incompleto, pois não leva em consideração a necessidade de percorrer todos os tipos de operações definidos no fluxo de dados. Por exemplo, um resultado válido pelo algoritmo é o caminho $\mathit{Criação} \rightarrow \mathit{Início} \rightarrow L_1 \rightarrow F_1 \rightarrow J \rightarrow F \rightarrow G \rightarrow \mathit{Fim}$, e para simplificar a explicação, com todas as operações sendo executadas em apenas uma ferramenta. O problema desse resultado é que esta solução não compreende as operações L_2 e F_2, significando que uma parte do fluxo não seria executado.

Uma forma de lidar com esse problema é dividir o grafo estendido em subgrafos, sem bifurcações ou junções. Por exemplo, uma abordagem para o grafo da figura 3.2 seria a criação de dois subgrafos: (L_1, F_1,J, F, G) e (L_2, F_2,J, F, G). No entanto, não adianta computar o caminho mais curto em cada subgrafo individualmente, pois uma solução encontrada para um subgrafo pode causar conflito para os outros subgrafos. Usar uma abordagem de fixar as ferramentas nas operações comuns entre os subgrafos já percorridos, sem nenhum outro critério, também não dará certo, pois é possível que exista uma operação desconhecida para os subgrafos avaliados, que possua um alto custo de processamento ou que os dados tenham um alto custo de migração, e isso exigirá uma reorganização na solução geral. Para a resolução correta desse problema, uma abordagem seria a enumeração de todas as possibilidades, ou seja, envolveria, no pior caso, uma ordem de complexidade igual a $\mathcal{O}(T^E)$, onde T corresponde ao número de tarefas e E o número de ferramentas de processamento disponíveis. Como apresentado na definição 1, na seção 2.1, o problema modelado é, de fato, reconhecido como NP-difícil.

No entanto, como trabalhamos com grafos direcionados e acíclicos, com pesos positivos, podemos simplificar o processo de enumeração para apenas alguns pontos de interesse. Primeiramente, precisamos dividir o grafo de uma forma diferente do exemplo anterior. Nossa abordagem se baseia nos métodos de blocagem utilizados na área de análise de fluxo de controle em Compiladores.[5] O processo da divisão do grafo em partes, que chamaremos de blocos, é o seguinte: caso o grafo possua mais de um componente conexo (isto é, dois ou mais fluxos de dados com operações não dependentes entre si), separe cada componente em um bloco. Se o grafo original possuir uma operação que promova bifurcação ou junção de caminhos, divida o grafo nesse ponto, mas mantendo uma

figuração envolvendo múltiplas ferramentas de execução, de modo a minimizar o tempo de execução global.

[5] A análise de fluxo de controle é uma técnica de análise de código estático para determinar o fluxo de controle de um programa (Allen, 1970).

réplica da operação em todos os subgrafos criados por essa divisão. Repita o processo até que não exista nenhuma operação com mais de uma aresta de entrada ou de saída. O resultado de nossa abordagem de blocagem para o grafo da figura 3.2 é exemplificado na figura 3.6. No exemplo, três blocos tiveram que ser criados pois o grafo original possui uma junção (J) que opera sobre dois caminhos distintos (a partir da leitura de dados L_1 e L_2). Nesse sentido, o Bloco A é definido pela entrada L_1 e saída J; Bloco B pela entrada L_2 e saída J; e o Bloco C, pela entrada J e saída G.

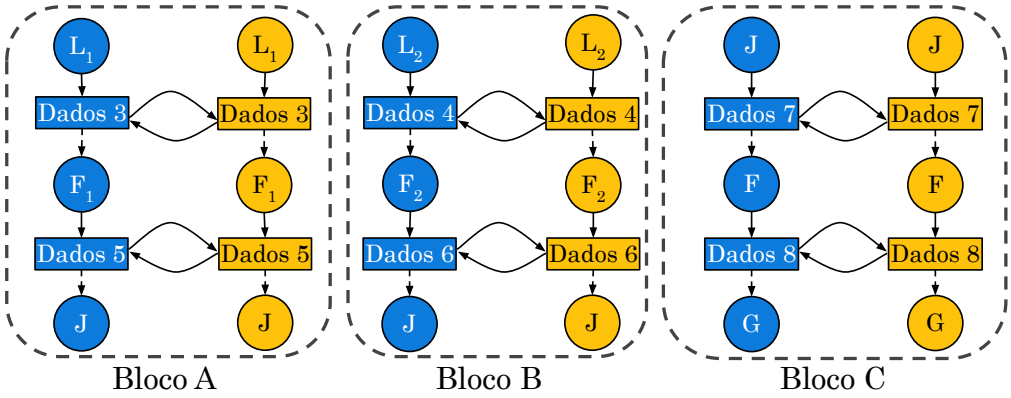


Figura 3.6: Divisão do grafo estendido em subgrafos.

Onde: L - Leitura de dados; F - Filtragem de dados; J - Junção; e G - Agrupamento.

Seguindo esse processo, cada bloco criado possui apenas um tipo de operação de entrada e de saída. Dessa forma, cada bloco funciona como uma caixa preta, na qual a configuração interna do fluxo de dados não interessa para os outros blocos; no entanto, as entradas e saídas de cada bloco exigem uma concordância entre todas as partes. A ideia da nossa abordagem é que a enumeração dos caminhos é necessária apenas entre as operações de entrada e de saída de cada bloco, que são os únicos pontos onde poderão existir conflitos de definições de ferramentas entre os blocos. Uma vez estabelecidas as ferramentas para as operações de entrada e de saída de um bloco, uma execução de um algoritmo de caminho mais curto será capaz de prover a melhor solução para o fluxo interno do bloco, sem que existam penalidades para os demais blocos. O processo completo é descrito no algoritmo 1.

Um exemplo de execução do algoritmo 1 é ilustrado na figura 3.7, a partir dos blocos criados na figura 3.6. Na figura, nós coloridos são aqueles com a ferramenta definida, enquanto que os brancos são nós ainda não percorridos. Na primeira enumeração, fixamos as operações L_1, L_2, J e G na ferramenta Spark e, para cada bloco, executamos o algoritmo de caminho mais curto individualmente. Após essa iteração, o custo da soma dos caminhos de cada bloco é somado e armazenado. Esse processo é realizado para cada enumeração, escolhendo, ao final, aquele de menor custo. No exemplo, a segunda enumeração foi a melhor configuração de ferramenta estimada. Como a enumeração é realizada apenas para as M operações de interesse, isto é, para primeiras e últimas operações de fluxo além das operações com bifurcações ou convergências de dados, conseguimos reduzir o

Algoritmo 1: Computação do plano de execução.

Dados: Blocos e Ferramentas
Resultado: $caminho[i]$ para um i que minimiza o valor em $custo[i]$
início
  M = criação da lista unificada das operações de entrada e de saída dos blocos;
  $matriz$ = criação da matriz de enumeração onde cada operação em M representa uma coluna, e cada linha corresponde a uma combinação de ferramenta;
  $custo$ = criação de uma lista para mapear o custo de cada combinação descrita em $matriz$;
  $caminho$ = criação de lista para mapear as configurações de ferramentas (nós percorridos);
  para $i \in Matriz$ **faça**
    custo[i] = 0;
    caminho[i] = [ ];
    para $bloco \in Blocos$ **faça**
      $tarefas\_novas$ = remova os nós do bloco que não foram percorridos (ausentes em $caminho[i]$);
      $tarefas\_novas\_enumerada$ = força a ferramenta em $tarefas\_novas$ presentes em $matriz[i]$;
      $novo\_caminho$ = execute o algoritmo de caminho mais curto para o trecho $tarefas\_novas\_enumerada$;
      $caminho[i] = caminho[i] + novo\_caminho$;
      $custo[i] = custo[i]$ + custo de $novo\_caminho$;
    fim
  fim
fim

espaço de enumeração de $\mathcal{O}(T^E)$, onde T representa todas as operações de um fluxo e E o número de ferramentas disponíveis, para $\mathcal{O}(M^E)$ uma vez que o conjunto M é menor que o número de tarefas em um fluxo. Na prática, nossa enumeração é rápida, pois o custo de inferência de cada tarefa é computado apenas uma vez a partir de uma referência ao grafo estendido original. Dessa forma, durante cada enumeração, é necessário apenas computar o custo total do caminho. Além disso, à medida que o custo total é computado, caso um custo supere uma solução já armazenada, podemos pular essa iteração. Na prática, grandes fluxos de trabalho geralmente são divididos em etapas bem definidas, como pré-processamento, transformações e análises. Assim, cada etapa tende a conter um conjunto de tarefas relativamente autocontido, o que limita a complexidade de cada subfluxo. Além disso, como o conjunto M é menor do que o número de tarefas, reduz-se o risco de explosão computacional em cenários reais.

A solução recomendada pode envolver uma combinação de ferramentas de execução ou apenas uma ferramenta. No entanto, caso seja uma combinação, antes de retornar o plano de execução recomendado pela modelagem, avaliamos o resultado proposto com as

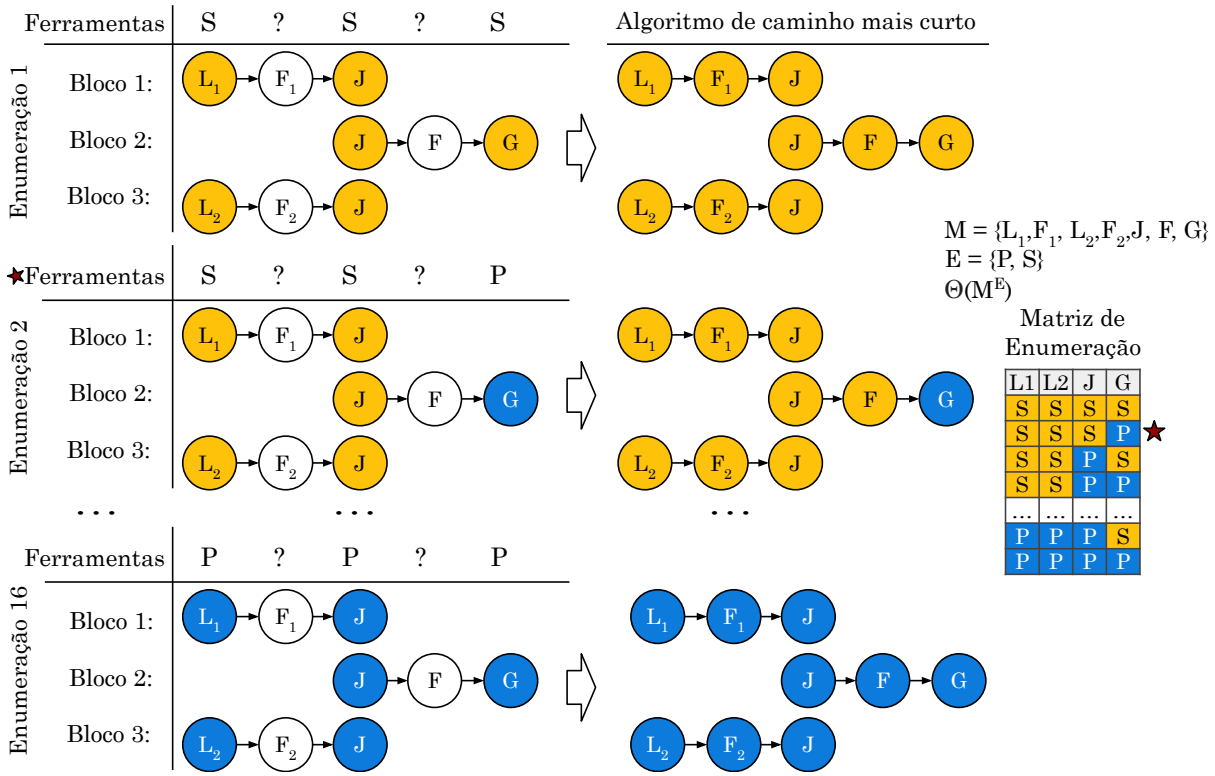


Figura 3.7: Exemplificação da enumeração de caminhos.

Onde: L - Leitura de dados; F - Filtragem de dados; J - Junção; e G - Agrupamento.

outras opções de execução (utilizando apenas uma ferramenta) para flexibilizar a solução. Nessa avaliação, confrontamos o Speedup previsto da solução em relação ao *baseline* (definição 2) para avaliar o quanto é esperado da solução recomendada ser melhor que uma abordagem tradicional. Embora nossa abordagem de escolha seja exclusivamente a partir do desempenho, outras abordagens poderiam ser incluídas a partir de uma extensão do conjunto de regras. Por exemplo, poderíamos relacionar o ganho da solução com o número de nós para obter uma solução baseada no custo-benefício entre desempenho e recursos alocados.

Definição 2. (Speedup previsto) O Speedup é uma métrica que representa o quanto a nossa solução, que possui um tempo de execução estimado como ($Tempo_m$), é esperada de ser mais eficiente em comparação à melhor abordagem tradicional, que referenciaremos como o nosso *baseline* ($Tempo_{baseline}$). Seguindo essa lógica, Speedup pode ser computado como $Speedup_m = Tempo_{baseline}/Tempo_m$. Quanto maior o seu valor, mais rápida é a configuração de ferramentas recomendadas para a execução do cenário avaliado. Por definição do problema, a configuração proposta pela nossa solução terá o menor tempo previsto ou, pelo menos, um tempo igual às execuções envolvendo apenas uma ferramenta. Em outras palavras, nossa solução será aquela com maior Speedup.

Ao trabalhar com o Speedup, ao contrário de trabalhar diretamente com o tempo previsto de execução, conseguimos um valor que relaciona as ferramentas. Além disso, o objetivo de nossa solução é recomendar a melhor configuração de ferramenta, e embora utilizemos indiretamente o tempo de execução, ele não é o nosso resultado. O tempo de predição, como discutido na seção 3.1, depende de uma série de outros fatores como a estimativa dos dados ou da modelagem do custo das operações. Nossa ideia é que embora fatores como a estimativa dos dados possam promover um erro na previsão do tempo de execução, a influência desse erro é compartilhada para todas as configurações de ferramentas avaliadas. Com isso, ao trabalhar com essa métrica, afrouxamos a condição de qualidade da solução ao fornecer não apenas o tempo de predição mais correto, mas o ganho de utilizar a ferramenta definida em contraste com um *baseline*.

3.3 Processo de criação dos modelos de regressão

A previsão do tempo de execução de cada parte de um fluxo de dados se dá a partir de diversos modelos de regressão, como já mencionado na seção 2.1.1, um para cada combinação de operação, ferramenta e configuração de hardware. A regressão é uma técnica de análise de dados que prevê o valor de dados desconhecidos usando referências de dados relacionados e conhecidos. Ela modela matematicamente a variável desconhecida (p.ex., tempo de execução) a partir de um conjunto de variáveis conhecidas (p.ex., parâmetros e carga de trabalho da operação) como uma equação ou transformação que dependerá da técnica de regressão utilizada. No contexto de Aprendizado de Máquina, modelos de regressão precisam ser treinados a partir de uma amostra inicial. Relacionado ao nosso problema, tal amostra deverá conter um conjunto de execuções que relacione as operações, seus parâmetros e dados de entrada ao tempo gasto de cada operação.

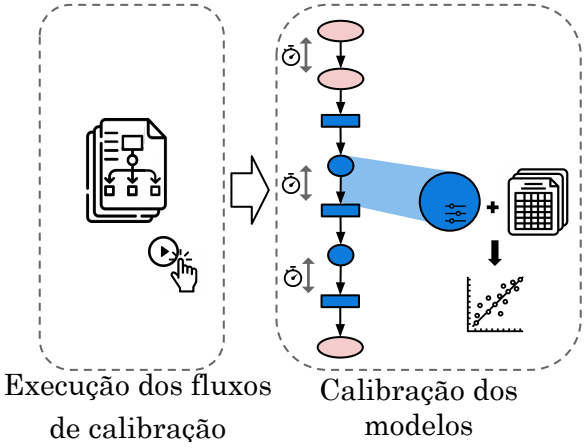


Figura 3.8: Esquema simplificado do processo de calibração inicial dos modelos.

Nossa abordagem realiza o treinamento desses modelos nas etapas exemplificadas

na figura 3.8. Primeiramente, é necessária a criação e a execução de um conjunto de fluxos para servir como fonte de dados para a calibração dos modelos. A execução desses fluxos é necessária para a coleta dos tempos de execução medidos para servir como referência ao modelo (seção 3.3.1). A ideia é criar uma amostra de dados inicial com as informações das operações presentes nos grafos estendidos, modelados a partir dos fluxos executados, informações essas que foram definidas para representar o comportamento de uma execução, vinculando ao tempo de execução coletado. Por fim, uma vez criada a amostra de dados inicial, cada modelo é treinado a partir dos registros referentes à mesma combinação de operação (isto é, operação, ferramenta e hardware). Este processo é descrito na seção 3.3.2.

3.3.1 Criação dos fluxos de treinamento inicial

Uma das complexidades da nossa tarefa é como realizar a medição do tempo gasto por operação, uma vez que ferramentas de execução podem ter comportamentos diferentes. Avaliar o tempo de execução de uma operação em uma ferramenta que possui um modelo de avaliação ansiosa como o Pandas é mais simples, pois basta computar a diferença entre o tempo logo antes e depois de uma operação. Por exemplo, na figura 3.9, o tempo gasto na operação de Filtragem é definido por $T3 - T2$. No entanto, em ferramentas de avaliação preguiçosa, como o Spark, a submissão de um operador não implica na computação da operação. No mesmo exemplo, o valor de $T3 - T2$ será próximo de zero em uma ferramenta como o Spark, uma vez que, durante a execução do operador Filtragem, apenas o empilhamento da tarefa é feito. Nessa ferramenta, o empilhamento de tarefas é realizado até que um operador de ação, ou seja, até que um operador que exija o retorno de um resultado para o usuário seja submetido, como o operador de Escrita. Dessa forma, a medição do $T4 - T3$ inclui o tempo gasto para a execução da leitura, filtragem e escrita de dados.

Uma prática bastante comum para estimar o tempo de execução de um conjunto de operações, sem recorrer explicitamente a escrita dos dados em disco, em ferramentas baseadas em avaliação preguiçosa, como o Spark, é a utilização do operador `count()` em cada estágio do fluxo de execução. Esse operador é responsável por computar o número de linhas dos dados até o estado atual, forçando, assim, a execução das operações anteriores. No entanto, embora a utilização do operador `count()` seja uma prática recorrente, essa abordagem não é totalmente confiável em contextos que utilizam DataFrames. Isso ocorre porque o otimizador SQL do Spark pode eliminar transformações desnecessárias para o cálculo da contagem de linhas. Em alguns casos, por exemplo, o Spark consegue computar o resultado da operação `count()` sem a necessidade de aplicar todas as transformações presentes no fluxo, desconsiderando colunas ou operações que não influenciam o número final de registros.

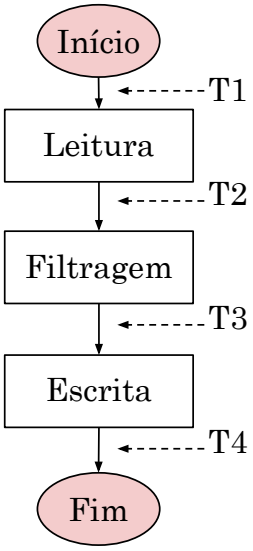


Figura 3.9: Exemplo de medição do tempo de execução de um fluxo de dados.

Uma abordagem mais interessante, mas menos conhecida, para forçar a execução de um conjunto de transformações no Spark consiste na escrita dos dados em formato "noop" (do inglês, *no operation*). Esse formato é, na verdade, uma anotação disponível no Spark a partir da versão 3.0 que orienta o Spark a realizar a computação de todas as tarefas até o ponto atual sem a necessidade de escrever os resultados em um sistema de armazenamento (Karau et al., 2024).

Ainda que existam mecanismos nativos, como o formato "noop", capazes de forçar a execução das transformações em sistemas como o Spark, o processo de medição do tempo de execução de etapas de um fluxo de dados pode apresentar desafios adicionais. Isso ocorre, principalmente, devido ao próprio modelo de execução adotado por sistemas projetados para processar grandes volumes de dados, os quais assumem que o estado completo dos dados dificilmente será mantido em memória. Dessa forma, a aplicação de operadores de ação resulta no descarte dos dados intermediários processados, a menos que esses dados tenham sido explicitamente persistidos. Como exemplo, considere o fluxo ilustrado na Figura 3.9, no qual um operador de escrita "noop" é inserido entre cada operação. Nesse cenário, o tempo medido no intervalo $T2 - T1$ corresponderá, predominantemente, ao tempo necessário para leitura da fonte de dados. Já o intervalo $T3 - T2$ incluirá o tempo de uma nova leitura (uma vez que os dados intermediários foram descartados) acrescido do tempo de aplicação da operação de filtragem. Por consequência, o intervalo $T4 - T3$ englobará o tempo de leitura, filtragem e escrita dos dados.

Devido a complexidade para medir o tempo gasto por uma tarefa em modelos de avaliação preguiçosa, precisamos seguir uma abordagem alternativa e genérica para a computação desses valores, que considera modelos tanto de avaliação preguiçosa quanto de avaliação ansiosa. Nossa abordagem consiste em criar fluxos de trabalho simples, que

chamaremos de fluxos de calibração, que contêm o menor número de operações para a execução de uma tarefa. Por exemplo, para medir o tempo de execução gasto por uma operação de leitura de dados, o seu fluxo de calibração respectivo conterá apenas essa operação. Já o fluxo para avaliar a escrita de dados apresentará tanto a leitura quanto a operação de escrita. Para a operação de filtragem, duas operações são necessárias: a de leitura de dados e a operação alvo (filtragem). Além disso, internamente, ao executar um fluxo de calibração para uma ferramenta como o Spark, tanto a operação de leitura de dados quanto a operação alvo são acompanhadas por operadores escrita "noop". Dessa forma, temos como estimar o tempo total da execução T e medir o tempo gasto entre cada operação (no exemplo anterior, leitura L e filtragem F). Sendo assim, podemos definir o tempo gasto pela operação de filtragem F, em um ambiente preguiçoso, como sendo $T - 2 \times L$. A lógica é que logo após a medição da operação de Leitura, o modelo de avaliação preguiçosa precisará ler novamente os dados para executar a operação alvo.

A modelagem do custo de migração é realizada como uma operação padrão, assim como as demais. Nesse caso, o fluxo de calibração criado envolve a leitura de dados em uma ferramenta e a operação de escrita em disco, executada em outra ferramenta suportada. Internamente, o sistema reconhece que se trata de um fluxo criado para a calibração do tempo de migração a partir de metadados e força a migração para alguma ferramenta especificada anteriormente, que possua integração a partir do ambiente de origem. A principal especificidade da coleta do tempo de migração é a sua forma de cálculo. Quando exportamos dados, mesmo os provenientes de sistemas de avaliação preguiçosos, realizam a ação naquele instante para a disponibilidade dos dados em outro sistema. Por causa disso, sua medição pode ser feita como $M - L$, onde M é o tempo coletado até a migração e L é o custo de leitura, como nas abordagens anteriores.

3.3.2 Treinamento dos modelos

Após a criação da amostra de treinamento, os tempos de execução coletados de cada fluxo são particionados com base na ferramenta, configuração de hardware e na operação alvo. Cada conjunto é treinado separadamente, considerando diversos algoritmos de regressão. Não nos limitamos apenas a modelos lineares, como Ordinary Least Squares ou Elastic-Net (Scikit-learn developers, 2024c), mas também exploramos Ensemble Methods, métodos que usam vários algoritmos de aprendizado para obter um melhor desempenho preditivo, como GradientBoosting (Scikit-learn developers, 2024a) ou XGBoost (XGBoost developers, 2022). Nossa premissa é que cada operação possui um comportamento de execução único, e forçá-las em uma adequação linear poderia resultar em uma baixa qualidade de predição. Em vez disso, optamos por escolher o algoritmo mais adequado para descrever o comportamento de cada operação separadamente.

Para cada algoritmo de regressão disponível, realizamos uma otimização dos hiper-

parâmetros associados, com o objetivo de encontrar a configuração mais adequada para cada caso. Essa otimização é feita utilizando técnicas como GridSearchCV (Scikit-learn developers, 2024b). Cada um dos modelos treinados é avaliado seguindo três métricas complementares:

1. **Coeficiente de determinação ($\mathbf{R^2}$)**: representa a proporção da variância da variável dependente que é explicada pelo modelo de regressão. A melhor pontuação possível é 1,0 e pode ser negativa (porque o modelo pode ser arbitrariamente pior). É uma métrica mais adaptada a modelos de regressão linear.

2. **Erro médio absoluto (MAE)**: representa a média da diferença absoluta entre os valores reais e os previstos no conjunto de dados. O valor calculado possui a mesma unidade da variável (tempo em segundos), tornando fácil sua análise. No entanto, é suscetível a *outliers*.

3. **Erro percentual médio absoluto (MAPE)**: é o equivalente percentual da métrica MAE e mede a magnitude média do erro produzido por um modelo, ou o quão distantes estão as previsões. Por exemplo, um valor MAPE de 20% significa que a diferença percentual absoluta média entre as previsões e os valores reais é de 20%. No entanto, a métrica não é indicada com muitos valores próximos de zero.

Na metodologia proposta, os modelos de regressão são ordenados com base no coeficiente de determinação (R^2) discretizado em quatro faixas: "<0,4", "<0,6", "<0,8" e "≥0,8". A ordenação considera, sequencialmente: (*i*) o valor discretizado de R^2; (*ii*) o tipo de modelo de regressão (1 para modelos lineares, 2 para baseados em *boosting* e 3 para os demais); (*iii*) se o erro absoluto médio (MAE) é inferior a 5 segundos; e (*iv*) o valor do erro percentual absoluto médio (MAPE). O objetivo é priorizar modelos com comportamento mais próximo ao linear, dada sua maior capacidade de extrapolação. Em casos de empate ou comportamento quase constante da variável dependente, métricas adicionais como MAE e MAPE são utilizadas para melhor avaliação do modelo. Todas as métricas e parâmetros utilizados são armazenados com os modelos para possibilitar auditorias futuras.

3.3.3 Processo de recalibração dos modelos

Uma das principais vantagens da utilização de modelos de aprendizado de máquina é a sua capacidade de evolução contínua à medida que novos dados são incorporados ao seu processo de treinamento. Quanto maior o volume de dados disponibilizado, ou seja, quanto mais cenários de execução forem fornecidos ao sistema, maior será a probabilidade do modelo capturar melhor as particularidades e variações dos diferentes contextos de execução, aprimorando, assim, sua capacidade preditiva. Apesar da criação de novos

cenários de calibração e suas respectivas execuções representar uma etapa custosa, a metodologia proposta nesta seção contempla uma abordagem que visa explorar o histórico de execuções já realizadas, após uma etapa inicial de calibração, como mecanismo de refinamento dos modelos de previsão.

Considerando cenários de execução em ferramentas baseadas na avaliação ansiosa, as medições realizadas nativamente pelo Lemonade, que registra o início e o término de cada operação via mensagens de notificação, podem ser consideradas adequadas para estimar o tempo de execução de cada tarefa. Nesses casos, o processamento dos logs de execução armazenados no sistema é suficiente para alimentar o processo de recalibração. Entretanto, em ferramentas baseadas na avaliação preguiçosa, como o Spark, a coleta desse tipo de informação exigiria adaptações no código de execução, o que, conforme discutido na seção 3.3.1, pode tornar o processo significativamente mais custoso. Sendo assim, para esse tipo de ferramenta, apenas métricas globais, como o tempo gasto na inicialização da execução e o tempo total de processamento, podem ser consideradas confiáveis. Nesse contexto, a figura 3.10 apresenta o esquema simplificado da abordagem proposta para o aproveitamento desses dados no processo de refinamento dos modelos preditivos.

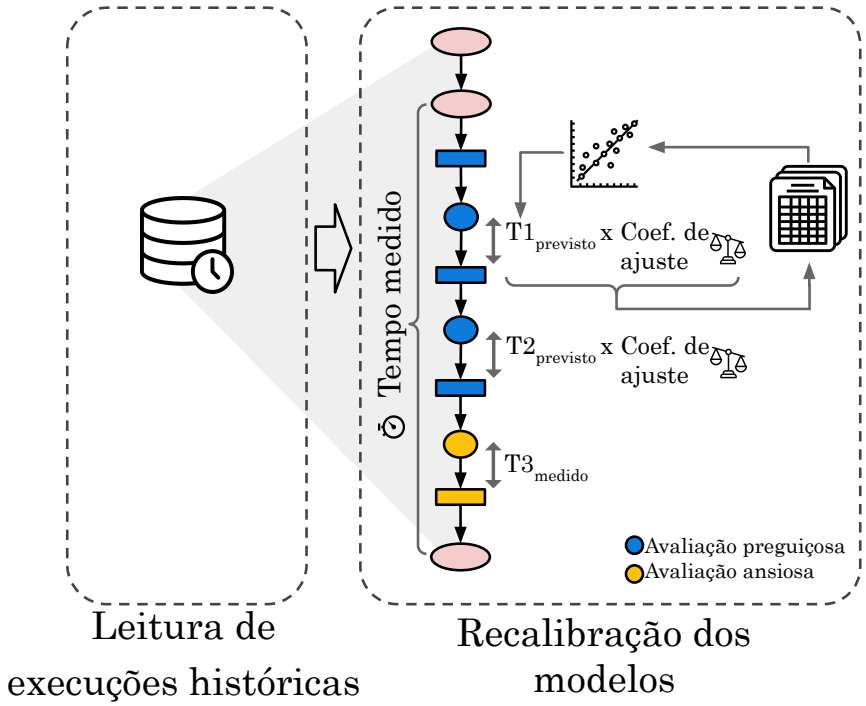


Figura 3.10: Esquema do processo de recalibração dos modelos a partir do histórico das execuções.

O esquema ilustrado na figura 3.10 apresenta a heurística proposta para estimar o tempo de execução de cada fluxo de dados previamente executado. A partir do histórico de execuções, o grafo estendido correspondente ao fluxo original é reconstruído, considerando apenas os nós e arestas efetivamente percorridos durante a execução real, ou seja, eliminando as porções do grafo que não foram utilizadas. Nesse grafo resultante,

as operações executadas em ferramentas baseadas na avaliação ansiosa, como o Pandas ou cuDF, terão seus tempos de execução diretamente associados aos valores medidos durante a execução, representados na figura como $T3_{medido}$. Por outro lado, para operações realizadas em ferramentas baseadas na avaliação preguiçosa, como o Spark, o tempo de execução é aproximado. Para isso, considera-se o tempo originalmente previsto pelo modelo de recomendação (representado na figura como $T1_{previsto}$ e $T2_{previsto}$), ajustado por um coeficiente de correção que será detalhado na sequência.

Idealmente, a soma dos tempos de execução previstos para cada operação individual de um fluxo deveria se aproximar do tempo total de execução medido. No entanto, dada a complexidade dessa tarefa, é esperado que existam divergências entre os tempos previstos e os tempos reais observados. Para mitigar esses erros ao longo do tempo, a metodologia proposta adota uma estratégia de ajuste proporcional dos tempos estimados das operações. Esse ajuste é realizado com base na comparação entre o tempo total medido da execução e a soma dos tempos originalmente previstos para as operações.

Dessa forma, o fator de ajuste de cada operação é calculado de modo a distribuir proporcionalmente o tempo excedente ou deficitário, considerando a contribuição relativa de cada operação na predição inicial. Por exemplo, considere o grafo ilustrado na figura 3.11, que representa um cenário similar ao apresentado anteriormente na figura 3.10. Esse grafo corresponde à execução de um fluxo de dados composto por três operações: leitura (L), filtragem (F) e agregação (G). A partir do histórico de execução disponível, observa-se que o tempo total medido para a execução completa do fluxo foi de 120 segundos. Adicionalmente, sabe-se que as operações L e F foram executadas utilizando o Spark, enquanto a operação G foi realizada após a migração dos dados para o Pandas.

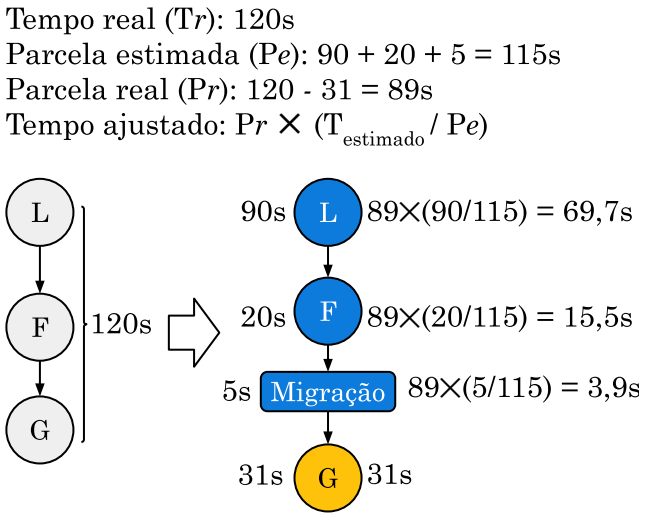


Figura 3.11: Exemplo de refinamento das previsões a partir do tempo de execução total. No exemplo, operações em azul representam execuções em Spark e amarelo em Pandas.

Com base nessas informações, é possível determinar que o tempo de execução da

operação G foi de 31 segundos. Esse valor pode ser considerado confiável, uma vez que a operação foi executada em uma ferramenta baseada em avaliação ansiosa, cuja medição é mais precisa. Por outro lado, os tempos individuais das demais operações e da etapa de migração de dados não estão disponíveis, restando apenas a informação do tempo total dessas etapas em conjunto.

Sabe-se, contudo, que o tempo previsto inicialmente para esse conjunto de operações (denotado como Pe) foi de 115 segundos. No entanto, a execução real indicou que esse trecho consumiu, na prática, apenas 89 segundos (denotado como Pr). Ou seja, o modelo superestimou o tempo necessário para essas operações. Para ajustar essa diferença e redistribuir o tempo real entre as operações de acordo com a proporcionalidade prevista, propomos a utilização da equação 3.1. Essa equação permite recalibrar o tempo previsto de cada operação individual, produzindo um novo tempo ajustado, a partir da razão entre a parcela de tempo real medida (Pr) e a parcela de tempo estimada inicialmente (Pe). Assim, as proporções entre as operações são preservadas, mas os tempos são ajustados para refletir melhor o comportamento real da execução.

$$\frac{Tempo\_operação}{Tempo\_total} = \frac{Tempo\_operação_{previsto}}{Tempo\_total_{previsto}} \tag{3.1}$$

Aplicando essa abordagem no exemplo da figura 3.11, obtém-se os novos tempos aproximados para as operações de leitura, filtragem e migração: 69,7 segundos; 15,5 segundos e 3,9 segundos. O tempo da operação de agregação (G) permanece inalterado, por ter sido obtido diretamente por medição confiável.

Embora essa estratégia de recalibração se baseie em uma heurística, ela se mostra eficaz na obtenção de estimativas mais realistas em cenários históricos onde o detalhamento dos tempos de cada operação não está disponível. Além disso, espera-se que, com o avanço do uso do sistema e o acúmulo de novos dados de execução, os ajustes realizados pelo modelo se tornem progressivamente mais precisos, capturando com maior fidelidade os diferentes padrões e comportamentos dos fluxos de dados.

Uma vez finalizada a etapa de definição dos tempos de execução dos fluxos presentes no histórico, os dados são então mesclados com os dados utilizados na etapa de calibração anterior. Esses dados servirão de entrada para o treinamento de novos modelos, seguindo as mesmas diretrizes discutidas na seção anterior (3.3.2).

3.4 Implementação da proposta no Lemonade

Para validar o processo discutido nas seções anteriores, implementamos um protótipo de nossa abordagem como uma extensão ao Lemonade, o que traz benefícios para ambos os sistemas. Por um lado, o Lemonade torna-se capaz de executar fluxos de trabalho com

maior desempenho, permitindo execuções multiplataforma. Por outro lado, a construção desse sistema multiplataforma torna-se mais fácil uma vez que temos ao nosso dispor: uma interface amigável, já existente, provida pelo Lemonade; um sistema de gerenciamento de fontes de dados; e um sistema de controle e monitoramento de execução. Caso outro sistema fosse utilizado, pelo menos esses três pontos precisariam ser levados em consideração.

3.4.1 Princípios de desenvolvimento no Lemonade

Adicionar o suporte a uma execução multiplataforma em um sistema de execução visual envolve etapas iguais ao processo de suportar uma ferramenta tradicional. No Lemonade, todo esse processo de suporte a uma nova ferramenta é discutido com maiores detalhes em um trabalho passado (Ponce, 2018). Basicamente, o processo envolve quatro etapas principais, como ilustrado na figura 3.12: (*i*) implementar as operações e algoritmos que serão suportados; (*ii*) cadastrar as operações e seus parâmetros; (*iii*) criar o *transpiler*[6], responsável por interpretar as operações e parâmetros configurados visualmente pela interface gráfica em código-fonte da ferramenta alvo; e por fim, (*iv*) criar o componente responsável por mesclar o código de cada operação em um único código-fonte, iniciar e gerenciar a execução da ferramenta alvo.

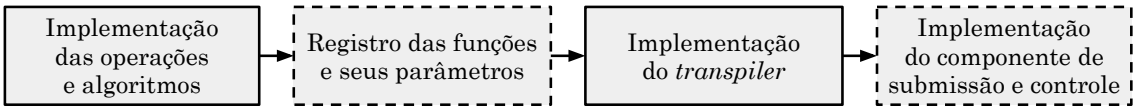


Figura 3.12: Etapas do processo de adição de uma nova ferramenta no Lemonade.

A extensão do Lemonade para a geração de execuções multiplataformas pode ser interpretada como o suporte a uma nova ferramenta qualquer. Durante a criação de um novo fluxo de dados, os usuários podem selecionar a ferramenta desejada (execução tradicional), assim como podem optar pela opção de um sistema multiplataforma. Essa abordagem permite que os usuários tenham maior controle sobre como desejam executar seus fluxos, além de possibilitar uma maior modularidade dos componentes do Lemonade. No entanto, durante o desenvolvimento do suporte ao sistema multiplataforma, algumas etapas realizadas durante o suporte individual de cada ferramenta podem ser aproveitadas, reduzindo o esforço de desenvolvimento, como ilustrado na figura 3.12.

As etapas representadas com bordas sólidas na figura 3.12 correspondem a processos já implementados nas ferramentas individuais atualmente suportadas pelo Lemonade. Dessa forma, apenas as etapas com bordas tracejadas exigem desenvolvimento adicional para integração com o sistema multiplataforma, no caso das ferramentas já compatíveis,

[6]Conversor de código-fonte para código-fonte

como Pandas e Spark. A ideia é que as operações já implementadas anteriormente mantenham seus respectivos métodos de conversão da representação visual para o código-fonte. Assim, o esforço de adaptação se concentra na seleção das operações a serem disponibilizadas nesse novo contexto, na seleção das características importantes para os modelos de regressão e na geração do código-fonte correspondente. Para o suporte à ferramenta cuDF, por outro lado, todas as etapas do fluxo foram implementadas do zero. No entanto, em muitos casos, a seleção das características para a modelagem podem ser reaproveitadas de outras ferramentas entre operações iguais.

Desta forma, do ponto de vista do Lemonade, quando um fluxo de dados é submetido para execução, o componente responsável por gerar o código-fonte e iniciar a execução é inicializado. Durante esse processo, o fluxo de dados, representado inicialmente pela interface, é convertido no grafo estendido, onde validações, como análise de dependências, e o enriquecimento das informações são realizados. Uma vez definido o melhor plano de execução, cada nó de operação presente do plano de execução recomendado é convertido em código-fonte, com a inclusão dos devidos trechos de código para as migrações de dados necessárias. Para o usuário ou para qualquer outro aspecto, a execução multiplataforma é similar a uma execução tradicional. Essa abordagem está alinhada com os princípios de desenvolvimento de novas ferramentas no Lemonade, que buscam manter uma interface comum entre as ferramentas para maximizar a similaridade de usabilidade.

3.4.2 Seleção de ferramentas e operações

Em nossa implementação do protótipo do sistema multiplataforma de proposta geral, optamos por oferecer suporte a três ferramentas de processamento de dados: Spark (versão 3.5.2), Pandas (versão 2.2.3) e cuDF (versão 24.12). A escolha das duas primeiras ferramentas se justifica por serem amplamente utilizadas no contexto do Lemonade, tanto em ambientes didáticos, como nas disciplinas de Processamento de Dados Massivos em Nuvem e Mineração de Dados, ofertadas na Universidade Federal de Minas Gerais, quanto em projetos desenvolvidos em parceria com empresas. Por outro lado, o cuDF é uma contribuição do nosso projeto, incorporado ao protótipo com o objetivo de avaliar o suporte a ferramentas emergentes que exploram arquiteturas baseadas em GPU para acelerar o processamento de dados. A ferramenta vem ganhando relevância na comunidade científica e industrial, sobretudo em aplicações que demandam alto desempenho.

Essas três ferramentas representam diferentes abordagens de motores de processamento de dados, como discutido no capítulo 2.2. O cuDF caracteriza-se como uma ferramenta de execução *multi-core* baseada em avaliação ansiosa em GPU; o Pandas, por sua vez, corresponde a um motor *single-core* de avaliação ansiosa em CPU; enquanto o Spark se destaca como um motor de processamento distribuído *multi-core/multi-node*, com execução preguiçosa baseada em CPU.

Uma vez definidas as ferramentas de execução, a seleção das operações e algoritmos suportados em cada uma delas foi realizada buscando manter, sempre que possível, uma similaridade de parâmetros e funcionalidades entre as ferramentas, conforme sumarizado na tabela 3.2 para o Spark e Pandas. Essa preocupação facilita a portabilidade e interoperabilidade dos fluxos de dados entre os diferentes motores de execução. Como exemplo, dentre as 110 operações atualmente suportadas, no Lemonade, em Spark e as 78 operações disponíveis no Pandas, identificamos que 47 operações são idênticas entre as duas ferramentas. Ou seja, compartilham exatamente os mesmos parâmetros de configuração. Outras 15 operações foram classificadas como similares, dado que, embora tenham finalidades equivalentes, apresentam diferenças em um ou mais parâmetros, específicos de cada ferramenta.

Tabela 3.2: Correspondência entre operações em diferentes ferramentas no Lemonade.

Correspondência	Número de operações	
	Spark	Pandas
Exclusiva	48	16
Igual	47	
Similar	15	

Observa-se ainda que o Spark, por ter sido a ferramenta inicialmente suportada no Lemonade e por ser mais adequado ao processamento de grandes volumes de dados em ambientes distribuídos, concentra o maior número de operações exclusivas (48). Por outro lado, o Pandas, por seu ecossistema mais voltado a análises em ambientes locais e pelo suporte a um maior número de bibliotecas auxiliares, apresenta 16 operações exclusivas voltadas, principalmente, a algoritmos de aprendizado de máquina mais complexos. Muitos desses algoritmos não possuem implementações equivalentes no Spark ou em cuDF, em função dos desafios técnicos relacionados ao seu suporte em ambientes paralelos e distribuídos. Essa limitação, inclusive, reforça uma das principais motivações para a construção de sistemas multiplataforma, como o proposto neste trabalho: ao integrar ferramentas com características complementares, é possível ampliar o conjunto de operações disponíveis para a construção de fluxos de dados mais sofisticados e adequados a diferentes cenários de execução.

A definição das operações que compõem o sistema multiplataforma pode ser realizada a partir de diferentes abordagens, dependendo do nível de correspondência existente entre as ferramentas envolvidas. A abordagem mais simples consiste em considerar apenas as operações com correspondência exata entre as ferramentas. Apesar de facilitar a implementação e reduzir a complexidade, essa estratégia resulta em um conjunto bastante restrito de operações disponíveis ao usuário.

Considerando, no entanto, que o sistema proposto possui uma interface visual para a construção dos fluxos de dados, uma alternativa viável é incluir também as operações

com correspondência similar. Nesse caso, uma estratégia natural é permitir ao usuário configurar apenas os parâmetros comuns entre as ferramentas, enquanto os parâmetros específicos, presentes em apenas uma das ferramentas, podem ser automaticamente preenchidos com valores padrão, definidos pelo sistema. Essa abordagem possui ainda a vantagem de facilitar a evolução do sistema: operações inicialmente classificadas como exclusivas de uma ferramenta podem, futuramente, ser estendidas para outras ferramentas à medida que novas funcionalidades forem implementadas.

Por fim, uma terceira abordagem possível é suportar todas as operações disponíveis, incluindo aquelas exclusivas de uma única ferramenta. Nessa estratégia, o mecanismo de geração do plano de execução deve ser capaz de identificar tais operações e garantir que elas sejam executadas exclusivamente na ferramenta em que estão disponíveis, respeitando as limitações de cada ferramenta.

Em nosso protótipo, selecionamos um conjunto inicial de operações contemplando as três abordagens discutidas anteriormente. Dessa forma, foram incluídas operações que possuem exatamente o mesmo conjunto de parâmetros de entrada nas três ferramentas; operações com correspondência similar, ou seja, que apresentam pequenas variações nos parâmetros suportados; e, por fim, operações que estão disponíveis apenas em duas das três ferramentas analisadas. Essa escolha permite que o protótipo desenvolvido englobe todas as possibilidades de correspondência entre operações, servindo como base para explorar cenários distintos e validar a flexibilidade do sistema multiplataforma proposto.

A tabela 3.3 apresenta as 23 operações selecionadas para o protótipo desenvolvido. Dentre elas, todas estão disponíveis nas três ferramentas consideradas (Spark, Pandas e cuDF), com exceção do algoritmo de Regressão Isotônica, que atualmente não possui suporte na ferramenta cuDF. As operações escolhidas contemplam diferentes categorias definidas pelo Lemonade, abrangendo tanto operações de manipulação e transformação de dados quanto algoritmos de aprendizado de máquina. Adicionalmente, a tabela apresenta o *ranking* de frequência de uso dessas operações, considerando dados históricos de fluxos de dados criados por alunos de disciplinas ministradas no Departamento de Ciência da Computação da UFMG. Observa-se que todas as operações selecionadas apresentam utilização significativa, reforçando a importância do conjunto de operações escolhido para o desenvolvimento do protótipo.

Embora o conjunto inicial de operações selecionadas para o protótipo seja relativamente pequeno, trata-se de uma lista bastante diversificada, contemplando operações com diferentes comportamentos de execução, característica evidenciada, por exemplo, pela variação no número de linhas de entrada e saída observada na tabela 3.3. No futuro, essa lista de operações pode ser facilmente estendida, de modo a incorporar novos operadores e ferramentas conforme a evolução do sistema.

Tabela 3.3: Lista de operações implementadas

Categoria		Operação	Ranking	Número de linhas	
				Entrada	Saída
Entrada e saída		Leitura de dados	1	0	N
		Escrita de dados	21	N	0
Manipulação de dados	Geral	Filtrar por função	7	N	$\leq$ N
		Junção	6	N,M	$\leq$ NM
		Seleção de colunas	11	N	N
		Tratar dados ausentes	10	N	$\leq$ N
	Por linha	Adição de linhas	52	N,M	N+M
		Agrupar linhas por função	4	N	$\leq$ N
		Remover linhas duplicadas	16	N	$\leq$ N
	Por coluna	Adição de colunas	35	N,M	max(N,M)
		Ordenação	5	N	N
		Substituição de valores	40	N	N
		Transformação	2	N	N
Pré-processamento de dados	Amostragem	Amostrar exemplos	18	N	$\leq$ N
	Redefinir escala	Máximo-Absoluto	89	N	$\leq$ N
		Mínimo-Máximo	29	N	$\leq$ N
		Padrão (Z-Score)	49	N	$\leq$ N
	Representação de atributos	Converter categórico para numérico	15	N	$\leq$ N
Aprendizado de Máquina	Agrupamento	K-Means	24	N	N
	Classificação	SVM	47	N	N
		Regressão Logística	44	N	N
	Regressão	Regressão Linear	26	N	N
		Regressão Isotônica	80	N	N

Capítulo 4

Avaliação experimental

Neste capítulo, avaliamos a qualidade da solução para recomendar o melhor plano de ferramenta de execução para um fluxo de dados. Na seção 4.1, descrevemos os resultados da calibração inicial e o ambiente de testes utilizado nos experimentos. Já a seção 4.2 descreve o planejamento dos experimentos, bem como a definição das métricas utilizadas para avaliação da qualidade da solução. Os experimentos realizados se dividem em dois eixos: um relacionado à estimativa dos volumes de dados após a execução de cada tarefa (seção 4.3) e outro à recomendação de escalonamento da ferramenta (seção 4.4).

4.1 Calibração inicial e ambiente de testes

Em aprendizado de máquina, é esperado que os modelos de regressão sejam treinados com dados próximos aos cenários reais, como conjuntos de dados cujos tamanhos estejam dentro do intervalo esperado. Com o tempo, à medida que os cenários se tornem mais complexos, com o aumento do volume de dados ou a adição de mais operações e de opções de configuração de hardwares, outras etapas de calibração podem ser realizadas para atualizar os modelos. Pensando nisso, utilizamos duas bases de dados sintéticas, com diferentes configurações de tamanho e número de colunas, para serem utilizadas nos fluxos de dados de calibração criados conforme a metodologia discutida na seção 3.3:

1. **Sintético 1**: conjunto de dados sintético composto por 11 colunas que incluem dados inteiros, datas, pontos flutuantes e campos textuais. As colunas são: `id`, um número inteiro e sequencial; `date`, representa uma data; `name`, nome gerado artificialmente usando a biblioteca Faker; `age`, número inteiro gerado aleatoriamente entre 1 e 80; `city`, valor textual escolhido entre uma lista de 50 possibilidades; `state`, valor textual escolhido entre uma lista de 50 possibilidades; `fare`, ponto flutuante gerado aleatoriamente entre 10 e 100, com possibilidade de 10% dos valores serem nulos; `gender`, escolha aleatória entre Male e Female, com possibilidade de 5% dos valores serem nulos; `occupation`, campo textual escolhido entre uma lista de 20 opções, com possibilidade de 15% dos valores serem nulos; `int1`, escolha entre três números inteiros; e por fim, `label`, valor gerado a partir da escolha entre 0 e 1;

2. **Sintético 2**: conjunto de dados sintético semelhante ao anterior, porém com três colunas adicionais: `date2`, uma nova coluna do tipo data; `float1`, ponto flutuante gerado aleatoriamente entre 1 e 1000; e `float2`, número gerado aleatoriamente entre -1000 e 1000.

O objetivo dos fluxos de calibração não é necessariamente realizar uma análise específica, mas sim medir o tempo gasto por cada operação em diversas configurações de parâmetros e cargas de trabalho. Por isso, a utilização de dados sintéticos foi pensada para maximizar a variação no número de colunas e em suas tipagens entre arquivos, minimizando a quantidade de arquivos necessária. Uma premissa que adotamos é que um dos principais custos durante a leitura de dados, além do volume de dados, é a inferência de tipos das colunas. Nesse sentido, colunas do tipo data tendem a ser as mais custosas de inferir, pois é preciso identificar o formato das datas para, em seguida, convertê-las. Assim, utilizamos dois arquivos: o Sintético 1, com uma coluna do tipo data, e o Sintético 2, com duas colunas do tipo data. Outro exemplo é que alguns algoritmos, como os de aprendizado de máquina, podem ser influenciados pelo número de dimensões do vetor de características. Por essa razão, o Sintético 2 possui duas colunas adicionais que podem ser utilizadas para avaliarmos a relação entre dimensionalidade e tempo de execução.

Embora a utilização de dados sintéticos possa não representar bem dados reais, que apresentam distribuições variadas e outras particularidades não discretizadas, no nosso problema não estamos preocupados com o conteúdo dos dados em si, mas com a relação entre seu tamanho e o custo de processamento das operações em diferentes volumes de dados. Ainda assim, todo modelo de aprendizado depende de calibração com características próximas aos cenários em que será avaliado. À medida que os cenários de teste forem muito diferentes daqueles usados na calibração inicial, as previsões tendem a piorar, como será visto mais adiante na seção 4.4.2, ao testarmos nossa solução com um arquivo de 110 colunas. A tabela 4.1 apresenta as configurações das bases utilizadas nessa etapa.

Tabela 4.1: Relação das fontes de dados utilizadas na calibração.

Conjunto de dados	Pacote	Número de linhas	Tamanho (MB)
Sintético 1	1	1×10^3	$< 0{,}1$
	2	1×10^5	8,1
	3	1×10^6	81,4
	4	1×10^7	814,1
	5	1×10^8	8.152,0
Sintético 2	1	1×10^6	104,4
	2	5×10^6	526,4
	3	1×10^7	1.053,8

Para algumas operações, como a adição de linhas, em que não existem parâmetros, a variabilidade das execuções nos experimentos é definida pela variação dos dados de en-

trada. No entanto, para outras operações, como a filtragem, existem parâmetros, como a expressão de filtragem, que podem influenciar diretamente na execução. Para esses casos, variações adicionais de execução foram criadas a partir de um template para contemplar alguns cenários de parâmetros. Ainda no exemplo da operação de filtragem, variações podem ser feitas especificando limiares de valores diferentes ou até mesmo adicionando mais condições a expressão. No geral, foram utilizadas entre 1 e 6 variações de parâmetros por operação avaliada, totalizando 63 variações para o conjunto de 20 dentre 23 operações utilizadas nos experimentos.[1] Cada uma dessas variações foi executada três vezes, considerando cada tamanho dos dados de entrada e ferramenta de execução, além de três opções de configuração de hardware descritas a seguir:

1. **Local[16]**: configuração com apenas uma máquina onde a execução do *workflow* ocorre localmente. A máquina possui 62 GB de memória RAM e 32 núcleos de processamento e uma GPU da NVIDIA GeForce RTX 3090 de 24 GB; no entanto, em execuções Spark, apenas 16 núcleos de processamento são liberados;

2. **Local[30]**: mesma configuração de máquina Local[16]; no entanto, em execuções Spark, são liberados 30 núcleos de processamento;

3. **YARN**: configuração envolvendo duas máquinas idênticas às utilizadas na configuração Local[30]. Em cada execução, acessível pelo escalonador YARN, cada máquina disponibiliza 28 núcleos de processamento e 60 GB de memória RAM e execuções em GPU são realizadas apenas na máquina principal.

Utilizando essas configurações, a execução das tarefas para a criação inicial dessa amostra de calibração, para todas as variações mencionadas, demorou 97 horas. Conforme é apresentado na tabela 4.2, em média, uma execução durou 72 segundos, com mediana de 6 segundos e máximo de 63 minutos. Embora o tempo total para a criação dessa amostra seja elevado, trata-se de uma operação que não precisará ser repetida constantemente e poderá ser incremental. Por exemplo, ao adicionar uma nova operação ou dados de entrada maiores, não será necessário repetir as execuções já realizadas anteriormente. Além disso, nos testes realizados, como o nosso objetivo é compreender o comportamento da execução de cada operação, utilizamos um número elevado de variações nas bases. Por exemplo, em fluxos utilizando o conjunto de dados Sintético 1, o tamanho da base foi variado cinco vezes.

A etapa de treinamento e seleção de todos os modelos durou aproximadamente 25 minutos, realizada na mesma máquina referente ao hardware Local[16]. A qualidade dos modelos é apresentada na tabela 4.3. Cerca de 97,2% dos modelos selecionados possuem um coeficiente de determinação maior ou igual a 0,8, com a maioria sendo superior a 0,95.

[1] Embora tenhamos suportados 23 operações, não foi criado fluxos envolvendo as operações Máximo-Absoluto, Z-score e Regressão Logística para as avaliações.

Tabela 4.2: Estatísticas sobre o tempo de execução (em segundos) dos cenários avaliados.

Métrica	Valor
Média	72,2
Desvio Padrão	212,9
Mínimo	0,1
Q1 (25%)	2,1
Q2 (50%)	6,0
Q3 (75%)	34,0
Máximo	3775,8

O menor valor medido de R^2 foi 0, representando, em nosso caso, operações com valores próximos a constantes em relação às variações executadas.[2] Mesmo nesses cenários com R^2 inferiores a 0,8, os modelos ainda assim apresentam um MAE (em mediana) próximo de 0 segundos, o que pode indicar uma boa capacidade de previsão.

Tabela 4.3: Avaliação dos modelos de regressão criados a partir da calibração inicial.

Intervalo do R^2	Quantidade de modelos (%)	Mediana do MAE
$[0,0; 0,1)$	2,6	0,0
$[0,1; 0,8)$	0,0	0,0
$[0,8; 0,9)$	8,5	7,3
$[0,9; 0,95)$	8,5	3,2
$[0,95; 1,0)$	80,4	0,0

4.2 Planejamento dos experimentos de avaliação

Embora existam *benchmarks* conhecidos com foco em Ciência de Dados, geralmente eles falham na diversidade de operações por consulta ou no tamanho dos dados de entrada. Por exemplo, no Sanzu (Watson et al., 2017), consultas com mais de uma operação utilizam conjuntos de dados reais de até 2 megabytes. Outro exemplo é o MLPack (Edel et al., 2014), que, embora seja uma coleção de *benchmarks* para várias bibliotecas de aprendizado de máquina, possui conjuntos limitados a poucos megabytes. O M2Bench (Kim et al., 2022), frequentemente citado como *benchmark* para Ciência de Dados em múltiplos modelos de fluxos de dados, apresenta tipos de consultas geralmente restritos a operações relacionais como projeção, agregação e junção. Talvez o *benchmark* mais relacionado ao nosso problema seja o TPCx-BB (Transaction Processing Performance Council, 2024b); no entanto, ele está desatualizado e exige grande esforço para compilar e configurar ferramentas necessárias à geração de dados sintéticos.

[2] Como é o caso da operação de amostragem de registros, em que o custo de execução está diretamente relacionado ao valor escolhido para amostragem. Nesse contexto, se forem selecionados valores pequenos na escala de dezenas ou centenas, o custo de execução será aproximadamente constante.

Para garantir maior diversidade de cenários, utilizamos 15 fluxos de trabalho distintos para avaliar a capacidade de recomendação da solução. Cada fluxo implementado responde a um problema específico associado a um dos seis conjuntos de dados selecionados para avaliação, descritos a seguir:

1. **Titanic**: conjunto de dados dos passageiros do Titanic (Cukierski, 2012). É uma base amplamente utilizada no aprendizado de Ciência de Dados, contendo 12 colunas com informações dos passageiros e de seu embarque, como nome, idade, classificação de sobrevivência, código da passagem e classe do assento. Embora o arquivo original tenha apenas 891 registros, é possível expandi-lo replicando linhas até um limite pré-definido;

2. **TPC-H**: conjunto de dados sintético utilizado no Benchmark TPC-H (Transaction Processing Performance Council, 2024a). A base compreende, ao todo, oito tabelas que descrevem a relação de produtos, histórico de vendas, histórico de transporte, entre outras informações relacionadas ao gerenciamento de vendas de uma empresa fictícia. Para os cenários implementados, utilizamos três bases: *orders*, que descreve cada item vendido em um pedido, apresentando informações como o estado do pedido, prioridade de envio e preço; *lineitem*, que descreve dados gerais sobre uma venda, contendo campos como identificação do pedido, valor total, valor de taxa e de desconto, datas da compra, envio e modalidade de transporte; e *customer*, que descreve informações sobre os clientes como nome, endereço e telefone;

3. **Last.fm**: dados do *Million Song Dataset Challenge* (Bertin-Mahieux et al., 2011), com as preferências musicais de aproximadamente 1 mil usuários até maio de 2009. O conjunto é dividido em dois arquivos: *users*, que contém informações sobre os usuários, como idade, gênero e nacionalidade; e *tracks*, que contém informações sobre as músicas escutadas, por usuário, como nome da música, do artista e sua duração;

4. **Spotify**: conjunto de dados utilizado no *Spotify Million Playlist Challenge* (Chen, C.-W. et al., 2018). O conjunto de dados contém informações sobre *playlists* do Spotify, criadas durante o período de janeiro de 2010 a outubro de 2017. O conjunto foi dividido em duas partes: *playlists*, que contêm informações gerais sobre as *playlists* como nome, número de usuários que a escutaram e informações da edição, como a data da última edição; e *tracks*, que contêm informações sobre as músicas de cada playlist, como o nome, artista e ordem da música na lista;

5. **Taxi-Rome**: conjunto de dados de traços de mobilidade de táxis em Roma, Itália (Bracciale et al., 2022). Esse conjunto de dados contém coordenadas GPS de aproximadamente 320 táxis coletados durante o mês de fevereiro de 2014, com um inter-

valo de 7 segundos entre coletas. Junto com as coordenadas, cada registro possui o *timestamp* e o identificador do táxi;

6. **US-Flights**: conjunto de dados disponibilizado no curso Machine Learning Foundations da AWS Academy (Academy, 2024). Este conjunto de dados contém horários de partida e chegada programados e reais relatados por transportadoras aéreas certificadas dos EUA para voos entre 2013 e 2018. No total, o conjunto de dados possui 110 colunas contendo informações como data, hora, origem, destino, companhia aérea, distância e status dos voos.

Os fluxos de trabalho estão detalhados na tabela 4.4, que relaciona o conjunto de dados utilizado em cada fluxo e o número de operações correspondente, variando de 4 a 17. Além da quantidade de operações, os fluxos diferenciam-se pela organização das tarefas e pelo número de arquivos de entrada. Sete dos quinze fluxos (1, 3, 4, 6, 7, 8, 13, 14 e 15) são compostos por operações consecutivas, formando um caminho único de execução a partir de uma única fonte de dados. Os demais fluxos (2, 5, 9, 10, 11 e 12) apresentam bifurcações ou junções variadas, originadas de uma ou mais fontes de entrada.

Os conjuntos de dados utilizados variam de aproximadamente 1 mil a 125 milhões de registros, com tamanhos entre menos de 1 MB e até 13,6 GB. A tabela 4.5 sumariza o tamanho e o número de linhas em cada configuração de execução (identificada por um id de carga). Para o Titanic e o TPC-H (dados sintéticos), cada configuração corresponde a um fator de escala na geração dos dados, enquanto nos demais conjuntos representam amostras de um conjunto maior. Fluxos com múltiplas fontes de dados (indicadas na terceira coluna da tabela 4.5), como TPC-H, Last.fm e Spotify, são executados a partir dos arquivos definidos em cada id de carga.

4.2.1 Métricas para avaliação da qualidade da solução

Avaliamos a qualidade da solução a partir de três perspectivas: (*i*) como problema de classificação, cujo objetivo é recomendar configurações de ferramentas que minimizem o tempo de execução; (*ii*) como problema de minimização, visando reduzir o tempo de execução de fluxos de trabalho; e (*iii*), como análise da precisão das estimativas de dados durante a execução de um fluxo de trabalho. No primeiro tipo de avaliação, consideramos que o modelo acerta se a configuração recomendada apresentar um tempo inferior ou no máximo 5% pior que a configuração mais rápida entre as ferramentas tradicionais (`cuDF`, `Pandas` ou `Spark`). Nesse caso, as seguintes métricas foram utilizadas nesse tipo de avaliação:

1. **Taxa de acerto**: representa a porcentagem de quantos cenários a configuração recomendada foi realmente a com menor tempo de execução, ou no máximo 5% pior que a melhor opção;

Tabela 4.4: Relação dos fluxos de dados utilizados na avaliação.

ID	Conjunto de dados	Número de operações	Descrição
1	Titanic	7	Realiza a classificação de sobrevivência do passageiro a partir de campos numéricos, pré-processados, como idade, valor do ticket e número de parentes embarcados.
2		10	Geração de uma tabela sumarizada, relacionando o número de crianças, mulheres e homens que sobreviveram ou não ao acidente.
3		5	Geração de uma lista ordenada da média de idades dos passageiros por classe social.
4	TPC-H	6	Query 1 do *Benchmark*. Esta consulta sumariza informações sobre negócios faturados, enviados e devolvidos em uma janela de dias que compreende a data de maior envio contínuo do banco de dados.
5		11	Query 3 do *Benchmark*. Esta consulta recupera os 10 pedidos não enviados com o valor mais alto de um determinado intervalo de tempo.
6		5	Calcula a média dos valores de venda, por dia, para transações abaixo de um valor limiar.
7		9	Executa o algoritmo K-Means para agrupar transações semelhantes a partir dos valores de vendas e taxas. Posteriormente são contabilizadas quantas transações semelhantes foram inferidas a partir dos *centroids*.
8	Spotify	4	Contabiliza o número médio de músicas, artistas, álbuns e duração entre todas as playlists.
9		15	Criação de um classificador que prevê se uma playlist terá ao menos uma música do Drake, com base em metadados da lista como número de seguidores, número de músicas e artistas.
10		12	Geração de uma tabela relacionando a popularidade de um artista em playlists e a sua média do número de músicas.
11	Last.fm	17	Computa os top 10 artistas mais escutados pelos usuários, bem como o top 10 das faixas de idades, nacionalidades e a representatividade de cada sexo na plataforma.
12		11	Relaciona a idade do usuário com a diversidade de artistas, computando estatísticas como número mínimo, médio e máximo de artistas diversos por idade.
13	Taxi-Rome	5	Utiliza o K-Means para identificar as zonas das cidades com maiores concentrações de fluxo.
14	US-Flights	11	Cria uma regressão linear para prever o atraso da partida de um voo a partir de diversas informações.
15		6	Cria uma regressão isotônica para prever o atraso da chegada de um voo a partir do atraso na partida.

Tabela 4.5: Relação das fontes de dados utilizadas na avaliação.

Conjunto de dados	ID da carga	Arquivos	Número de linhas	Tamanho (MB)
Titanic	1	titanic	3.916.836	264,7
	2		7.833.672	529,5
	3		15.667.344	1.058,9
	4		62.669.377	4.235,7
	5		125.338.753	8.471,5
TPC-H	1	lineitem	6.001.215	718,9
		orders	1.500.000	162,6
		customer	150.000	23,1
	2	lineitem	29.999.795	3.686,4
		orders	7.500.000	862,7
		customer	750.000	121,4
	3	lineitem	59.986.052	7.372,8
		orders	15.000.000	1.638,4
		customer	1.500.000	232,1
Last.fm	1	users	992	$< 0,1$
		tracks	18.876.957	2.457,6
Spotify	1	playlists	161.000	9,1
		tracks	10.760.544	1.900,2
	2	playlists	725.000	41,1
		tracks	48.034.802	8.505,1
Taxi-Rome	1	input_rome	21.817.850	1.536,0
US-Flights	1	us-flights	4.750.567	2.141,6
	2		11.638.891	5.246,4
	3		30.144.616	13.594,8

2. **Precisão**: métrica utilizada em aprendizado de máquina para medir a proporção de identificações positivas que estavam corretas. Em nosso cenário, a métrica mede a relação entre todos os dados classificados para uma configuração de ferramenta e quantos deles são realmente recomendados para aquela configuração (menor tempo de execução);

3. **Revocação (Recall)**: mede a proporção de positivos verdadeiros identificados corretamente. Em nosso cenário, mede qual a porcentagem de dados classificados para execução em uma ferramenta comparado com a quantidade de casos que foram executados em menor tempo na ferramenta alvo;

4. **F1**: métrica une a Precisão e a Revocação a partir de uma média harmônica para trazer um número único que determina a qualidade geral do modelo. Essa métrica é complementar à taxa de acertos por reduzir os efeitos do desbalanceamento de classes;

5. **AUC**: outra métrica complementar para reduzir os efeitos do desbalanceamento

de classes. A área embaixo da curva (sigla do inglês, *area under the curve*, AUC) mede a qualidade da separabilidade de um modelo de classificação, ou seja, o quão bom o modelo é em distinguir valores de diferentes classes. Uma AUC próxima de 1 significa que o modelo possui tanto alta precisão quanto revocação, ou seja, o modelo possui baixas taxas de ambos falsos positivos e falsos negativos.

As cinco métricas acima são comuns em avaliações de modelos de classificação. No nosso problema, elas permitem avaliar a capacidade do modelo em recomendar uma solução em que sua execução seja tão rápida quanto ou mais rápida do que a melhor configuração de execução utilizando apenas uma ferramenta. No entanto, para avaliar o impacto das recomendações sobre uma perspectiva de desempenho, a segunda perspectiva, que avalia o problema como um desafio de minimização, se faz necessária, e por isso, as seguintes métricas precisaram ser incluídas:

1. **Erro absoluto**: diferença do tempo previsto e o tempo real, em segundos;
2. **Erro relativo**: diferença do tempo previsto e o tempo real, sobre o tempo real;
3. **Speedup da recomendação**: métrica que relaciona o Speedup da solução recomendada em função das abordagens tradicionais (cuDF, Pandas ou Spark). Embora o cálculo seja similar ao Speedup previsto (definição 2), utilizamos essas duas terminologias para diferenciar entre os valores previstos e medidos.

Este segundo tipo de avaliação, a partir de métricas como o Erro Absoluto, Erro Relativo e Speedup, complementa a avaliação anterior, quantificando quão mais rápida é a solução prevista ou, em casos de erro, quão mais lenta é a solução recomendada. Por fim, o terceiro tipo de avaliação analisa a precisão da evolução das estimativas dos dados durante a execução, com base na diferença absoluta e relativa entre o número de linhas previsto e o real em cada estágio de processamento de um fluxo de dados. Esse tipo de análise ajuda a entender como as estimativas influenciam a previsão do tempo de execução de cada operação.

Em todos os cenários de avaliação, cada fluxo de dados foi executado diretamente pelo Lemonade, tanto na configuração recomendada quanto nas tradicionais, coletando-se o tempo total de execução de cada abordagem em pelo menos três repetições. Especificamente para as avaliações das estimativas da evolução dos dados, foi executada apenas uma configuração de ferramenta por fluxo, pois essas estatísticas são independentes da ferramenta escolhida. Entretanto, modificações no código-fonte do fluxo são necessárias para incluir a coleta de métricas, como o número de linhas processadas e outros parâmetros relevantes. Para isso, implementamos mecanismos de personalização de submissão de tarefas no Lemonade por meio de nossa extensão. Tais mecanismos não foram integrados ao código-fonte padrão, pois afetariam o tempo de execução final.

4.2.2 Recalibração

Após a validação inicial da ferramenta, avaliaremos também como a qualidade da solução se comportará à medida que os modelos forem recalibrados com execuções históricas. A tabela 4.6 apresenta a seleção de seis fluxos (aproximadamente 40% do total) para utilização no processo de recalibração descrito no capítulo 3.3.3. Buscamos escolher os fluxos de forma a maximizar o número de operações e minimizar a quantidade de conjuntos de dados distintos, visando reduzir o efeito de sobreajuste na predição de novas execuções nos demais fluxos que utilizam outras fontes de dados.[3] Para isso, selecionamos fluxos que envolvem os conjuntos de dados com maior diversidade de operações: fluxos 1, 4, 7 e 14 possuem sequências únicas de tarefas que maximizam o número de operações distintas; já os fluxos 2 e 5 incluem múltiplas fontes de dados e estruturas complexas, com junções ou bifurcações.

Tabela 4.6: Fluxos utilizados na etapa de recalibração

Conjunto de dados	Fluxos
Titanic	1, 2
TPC-H	4, 5
Spotify	7
US-Flights	14

4.3 Avaliação da previsão do tamanho dos dados de saída de um operador

Como o modelo de decisão depende da avaliação das estatísticas de cada tarefa (parâmetros e dados de entrada), a inferência das estatísticas dos dados de saída é um aspecto importante para avaliar o comportamento geral da solução. Para operações como *Seleção de Colunas* ou *Ordenação*, é possível estimar com precisão as características dos dados de saída, pois seus comportamentos são previsíveis. Entretanto, em tarefas como *Junções* ou *Agrupamento por Função* (que operam sobre múltiplas colunas), as distribuições das colunas de entrada impactam significativamente a previsibilidade. Ainda assim, espera-se que as previsões mantenham boa precisão, considerando as decisões tomadas em cada estágio do fluxo de dados.

A avaliação foi conduzida sobre os dois tipos de fluxos de trabalho: (*i*) fluxos de calibração, compostos apenas por leitura e pela operação alvo; e (*ii*) fluxos complexos definidos na seção 4.2, com múltiplas operações interligadas. O primeiro tipo, por

[3]Sobreajuste (do inglês, *overfitting*) é um termo estatístico que descreve quando um modelo se ajusta muito bem aos dados observados, mas se mostra ineficaz para prever novos resultados.

ser simples, permite avaliar a precisão na estimativa do número de linhas dos dados de saída após cada operação. Já no segundo tipo, analisa-se, além do número de linhas, a propagação e inferência das estatísticas em geral em um cenário real.

4.3.1 Estimativa em um operador

De modo geral, na maioria das operações analisadas, a inferência do número de linhas se mostrou eficaz. Informações como o número de linhas de entrada, quantidade de elementos duplicados e ausentes de cada coluna, além da computação dos decis, nos ajudam a inferir o número de linhas de saída. Dentre as 20 operações utilizadas na avaliação, apenas três apresentaram erros relativos superiores a 2% nos fluxos de calibração: *agrupar linhas por função*, *filtrar por função* e *junção*. A figura 4.1 exibe o gráfico de violino com a distribuição dos erros relativos para essas tarefas. Nessa representação, as curvas associadas ao boxplot ilustram as densidades das distribuições dos valores medidos.

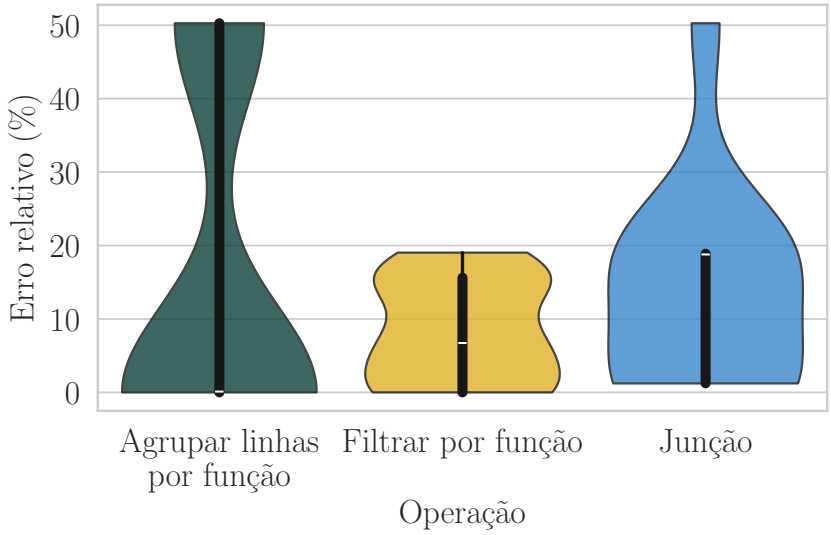


Figura 4.1: Gráfico de violino do erro relativo do número de linhas inferidas nos fluxos de calibração.

Para a operação de *Agrupar linhas por função*, embora a mediana dos erros tenha sido de 0% (linha branca) e grande parte dos pontos medidos tenha se concentrado abaixo de 20% (conforme a densidade das curvas), o erro relativo máximo chegou a 50%. A dificuldade de inferência nesse tipo de operação ocorre quando utilizamos mais de uma coluna como chave de agregação. Com apenas uma coluna, por exemplo, podemos prever que o número de linhas na saída será igual ao número de elementos distintos dessa coluna. No entanto, ao utilizarmos múltiplas colunas, a ausência de uma visão geral dos dados antes da execução faz com que o resultado seja fortemente influenciado pela especificidade do conjunto de dados.

A qualidade da previsão na operação de *Filtrar por função* também depende significativamente dos dados de entrada. Nesse caso, utilizamos principalmente os decis para calcular o número de linhas. Para cada operação de filtragem, avaliamos em qual decil o valor de filtragem se encontra e selecionamos os demais decis que satisfazem a condição. Nesse caso, o erro da predição está relacionado à quantidade de elementos presentes na mesma faixa de decil em que o valor limiar se encontra. Em nosso experimento, a média do erro relativo foi de 8%, com valores máximos de 20%.

Já a operação de *Junção*, como esperado, é provavelmente a mais difícil de inferir. Nesse caso, a escolha das chaves de junção e a distribuição de seus valores impactam significativamente o resultado. Por exemplo, a junção interna de duas bases por um ID sequencial e único resultará em um número de registros próximo ao total de IDs. Já uma junção baseada em uma coluna como uma idade pode ocasionar uma explosão de dados devido aos múltiplos candidatos à mesclagem (pessoas com idades iguais). Para a inferência, utilizamos principalmente os decis, o número de elementos distintos e o total de linhas não nulas. Para essa operação, a mediana do erro relativo foi de 20%, mas, no pior caso, chegou a 50%.

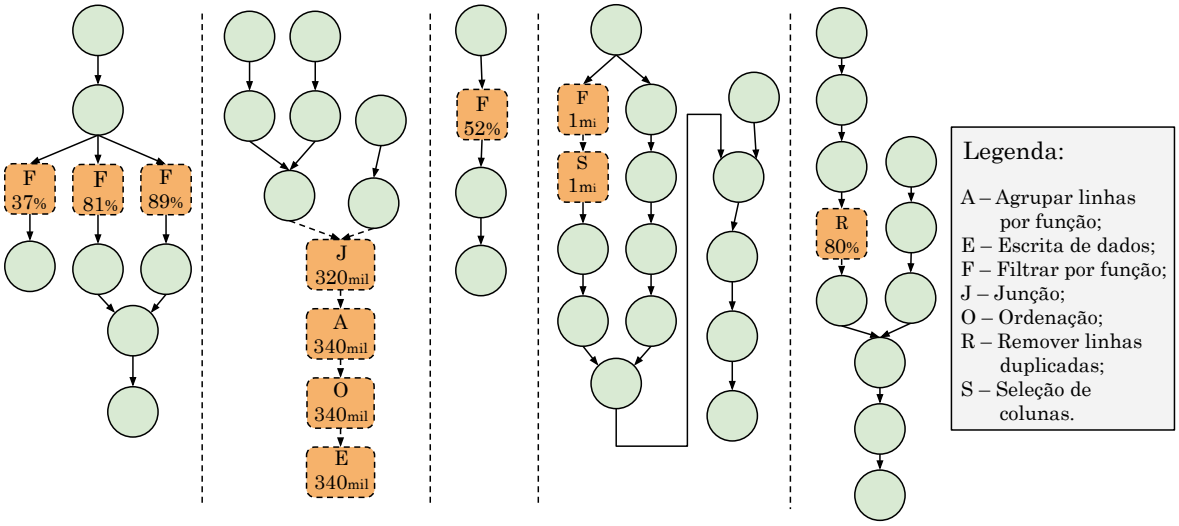
Discussão. Os resultados aqui apresentados servem como uma referência para avaliar o quão bem podemos estimar o comportamento da saída para as operações mapeadas. Em geral, grande parte das operações pode ter a sua saída de dados facilmente mapeada a partir das estatísticas dos dados de entrada. No entanto, em alguns casos, as estatísticas coletadas podem não ser suficientes para estimar com exatidão os dados de saída, pois as operações como *Junção* e *Filtrar por função* dependem das colunas e parâmetros utilizados como chave. Na próxima seção, é avaliado o impacto do erro de inferência na propagação das estimativas entre operações sucessivas. Já o impacto que esses erros provocam na qualidade das recomendações dos planos de execução é discutido na seção 4.4.

4.3.2 Estimativa para um fluxo inteiro

Em cenários reais, é comum que um fluxo de dados envolva múltiplas operações. Assim, não apenas o erro na estimativa do número de linhas produzidas por uma tarefa pode ser propagado para a operação seguinte, mas também a precisão de outras estatísticas, como decis e o número de elementos ausentes ou distintos, pode afetar os mapeamentos subsequentes. Esta seção tem como objetivo avaliar a precisão da estimativa do número de linhas em cenários complexos, considerando todos os 15 fluxos de avaliação (tabela 4.4).

Na presente avaliação, consideramos uma estimativa satisfatória quando o erro relativo é menor que 20% ou o erro absoluto é inferior a 200 mil linhas, ambos definidos empiricamente. Utilizamos esses dois limiares porque, embora a delimitação pelo erro relativo seja mais adequada na maioria dos casos, há situações em que uma operação pode produzir poucas linhas, inflacionando as medições sem necessariamente ser uma

diferença que impactará no tempo da estimativa. No contexto de nossa análise, onde buscamos utilizar essas estimativas para prever o tempo de execução, pequenas variações no número de linhas não impactam significativamente o tempo de processamento de uma operação, mesmo que o erro relativo seja elevado. Por exemplo, considere uma operação que, a partir de 1 mil linhas, produza 10 linhas como saída, enquanto a previsão indicava 20. Nesse caso, o erro relativo seria de 100%, mas a estimativa ainda seria aceitável. O limiar de 200 mil linhas visa flexibilizar a avaliação da qualidade da estimativa em cenários onde o erro na previsão não causaria um aumento expressivo no tempo de execução.



a) Fluxo 2 **b) Fluxo 5** **c) Fluxo 6** **d) Fluxo 9** **e) Fluxo 12**

Figura 4.2: Workflows com erros de previsão do número de linhas acima de 20%.

Nota: Nós com valores absolutos ou em porcentagem representam operações com erros maiores que o limiar definido de 20%.

Segundo esses critérios, 92% das operações executadas apresentaram estimativas satisfatórias. As onze operações com erros elevados pertencem a cinco fluxos de dados, conforme ilustrado na figura 4.2. Na figura, os nós retangulares representam operações que não atenderam aos critérios definidos. Assim como nos resultados da seção anterior, as operações de *Junção*, *Filtrar por função* e *Agrupar linhas por função* foram as que apresentaram erros mais consistentes. Alguns desses erros, de fato, foram propagados para operações subsequentes, como nos casos das figuras 4.2 (b) e (d). No entanto, é interessante observar que essa propagação nem sempre ocorre, o que sugere que, em alguns casos, as estimativas das operações seguintes conseguem se ajustar às novas configurações, reduzindo o erro anterior ou gerando previsões mais precisas.

Discussão. Estimar os dados de saída a partir de um conjunto limitado de operações é uma tarefa complexa. No entanto, o objetivo dessas estimativas é auxiliar na definição

do plano de execução mais eficiente entre as ferramentas disponíveis. Em alguns casos, erros nessas estimativas não representam um problema crítico, pois os resultados servem como referência para a recomendação da melhor configuração de ferramenta. Apesar das imprecisões observadas, os resultados avaliados mostraram-se promissores: a maioria dos fluxos de dados atendeu ao critério de qualidade estabelecido e, mesmo nos fluxos que apresentaram erros, esses erros ocorreram apenas em partes do processo.

4.4 Avaliação da capacidade de recomendação

Um sistema multiplataforma pode adotar duas políticas de execução: a primeira consiste em executar todas as atividades na ferramenta mais indicada, no geral, para sua realização; a segunda segue uma abordagem oportunista, na qual as tarefas podem ser distribuídas entre diversas ferramentas para maximizar o desempenho. Nossa solução suporta ambas as abordagens, permitindo ao usuário escolher entre a estratégia oportunista ou a execução em uma única ferramenta ao longo do fluxo de trabalho. Os experimentos a seguir buscam analisar a qualidade da recomendação da solução com base nos resultados obtidos a partir dos 15 fluxos de dados descritos na seção 4.2 sob os seguintes aspectos:

1. capacidade de recomendar a melhor ferramenta individualmente (seção 4.4.1);
2. eficácia da recomendação oportunista, envolvendo o uso de múltiplas ferramentas (seção 4.4.2);
3. desempenho da recomendação em comparação com outras abordagens (seção 4.4.3);
4. capacidade de recomendar a melhor configuração de hardware para a execução (seção 4.4.4);
5. impacto da recalibração dos modelos utilizando execuções históricas na qualidade do sistema (seção 4.4.5).

Cada um desses aspectos foi analisado considerando duas configurações distintas de ferramentas: uma que permite execuções apenas em Pandas e Spark, e outra que inclui a possibilidade de execuções em cuDF. Essa diferenciação se justifica pelo fato de o cuDF, ferramenta baseada em execução via GPU, geralmente apresentar um desempenho superior às demais ferramentas, influenciando diretamente as recomendações. Ao dividir as avaliações entre essas duas configurações, buscamos entender melhor a capacidade do modelo de recomendação em lidar com cenários mais complexos de decisão e a sua capacidade de operar em diversas configurações. Os resultados são apresentados a partir de uma média de três execuções para cada fluxo de dados computado, em muitos casos, apresentando um coeficiente de variação menor que 5%.

4.4.1 Recomendação da melhor ferramenta individual

A presente seção analisa a capacidade do escalonador de tarefas de identificar a ferramenta ideal para uma aplicação, considerando um hardware específico. Mesmo nesse caso de uso mais rígido, esse tipo de tarefa já representa um desafio, visto que ferramentas podem ter desempenhos bastante diferentes em determinados tipos de operações. Como será mostrado adiante nesta seção, nossos experimentos apontam que uma recomendação incorreta pode resultar em execuções até nove vezes mais lentas que uma solução ótima.

A tabela 4.7 apresenta a distribuição de acertos na recomendação da melhor ferramenta de execução, considerando dois cenários: um com apenas Pandas e Spark (referenciado na tabela como PS) e outro incluindo também o cuDF (CPS). As taxas de acerto variam de 82,5% a 100%, dependendo do hardware e das ferramentas disponíveis. Essa variação pode estar relacionada com a quantidade de recursos computacionais disponíveis: quanto mais núcleos de processamento o hardware oferece, maior é a diferença de desempenho entre Spark e Pandas (que não aproveita o paralelismo em múltiplos núcleos), facilitando a escolha da ferramenta mais eficiente.

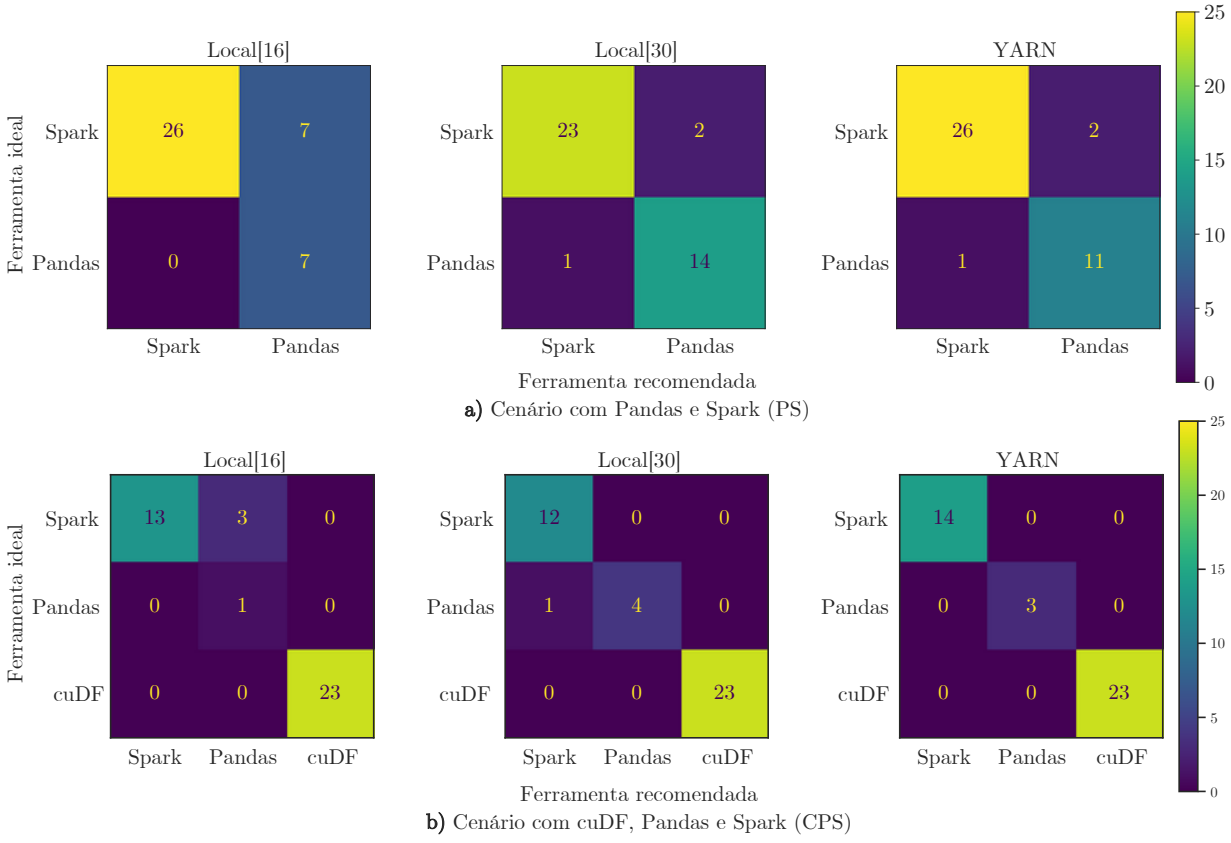
Tabela 4.7: Distribuição dos acertos para a recomendação da ferramenta de processamento individual.

Cenário	Ferramenta	Hardware		
		Local[16]	Local[30]	YARN
PS	Pandas	7 (17,5%)	14 (35,0%)	11 (27,5%)
	Spark	26 (65,0%)	23 (57,5%)	26 (65,0%)
	Taxa de acerto =	82,5%	92,5%	92,5%
CPS	cuDF	23 (57,5%)	23 (57,5%)	23 (57,5%)
	Pandas	1 (2,5%)	4 (10,0%)	3 (7,5%)
	Spark	13 (32,5%)	12 (30,0%)	14 (35,0%)
	Taxa de acerto =	92,5%	97,5%	100,0%

Além disso, como esperado, a inclusão do cuDF como opção de execução melhora a qualidade da recomendação, aumentando a taxa de acerto em cerca de 10% em relação ao mesmo cenário sem essa ferramenta. No geral em cenários CPS, observa-se que o sistema recomenda execuções no cuDF na maior parte dos casos (cerca de 58%) quando o volume do conjunto de dados e o conjunto de operações estão disponíveis para a ferramenta, enquanto o Spark aparece como a segunda opção mais recomendada (de 30% a 65%, dependendo do cenário avaliado).

Percebe-se também um melhor desempenho do Spark em relação ao Pandas. Esse comportamento é esperado, uma vez que as configurações de hardware testadas possuem 16, 30 e até 56 núcleos de processamento disponíveis, permitindo ao Spark explorar um alto nível de paralelismo. No entanto, é interessante notar que, mesmo com a capacidade do Spark de explorar uma quantidade maior de recursos, existem cenários em que pelo

menos 35,0% das execuções são mais eficientes em Pandas. Devido ao desbalanceamento das recomendações, foi criado a figura 4.3 e a tabela 4.8 para complementar a avaliação com informações sobre a Matriz de Confusão e outras métricas como F1, Precisão e Revocação.



a) Cenário com Pandas e Spark (PS)

b) Cenário com cuDF, Pandas e Spark (CPS)

Figura 4.3: Matriz de confusão da recomendação de ferramenta individual.

Tabela 4.8: Estatísticas das métricas de avaliação das ferramentas individuais.

Métrica (%)	Cenário PS			Cenário CPS		
	Local[16]	Local[30]	YARN	Local[16]	Local[30]	YARN
Taxa de acerto	82,5	92,5	92,5	92,5	97,5	100,0
Precisão	91,3	93,1	92,8	98,1	97,7	100,0
Revocação	82,5	90,3	92,5	92,5	97,5	100,0
F1	84,4	91,6	92,6	94,4	97,4	100,0
AUC	89,4	94,4	92,3	94,4	98,1	100,0

Como apresentado na figura 4.3, em grande parte das execuções, nossa solução é capaz de recomendar a melhor ferramenta. Os maiores erros geralmente ocorrem nas recomendações entre Pandas e Spark, especialmente em hardwares com menos recursos, como o Local[16]. No entanto, à medida que mais recursos são adicionados, a taxa de acertos aumenta. Complementando essa análise, os resultados da tabela 4.8 também indicam um

desempenho satisfatório, com todas as métricas apresentando valores superiores a 82%. Os valores elevados medidos tanto para F1 quanto para AUC demonstram a estabilidade do modelo na recomendação das ferramentas. Isso indica que, embora tenha ocorrido uma predominância de recomendações para o Spark no cenário PS e para o cuDF no cenário CPS, o modelo não apresenta um viés excessivo, garantindo uma distribuição equilibrada das recomendações e mantendo a confiabilidade das decisões tomadas.

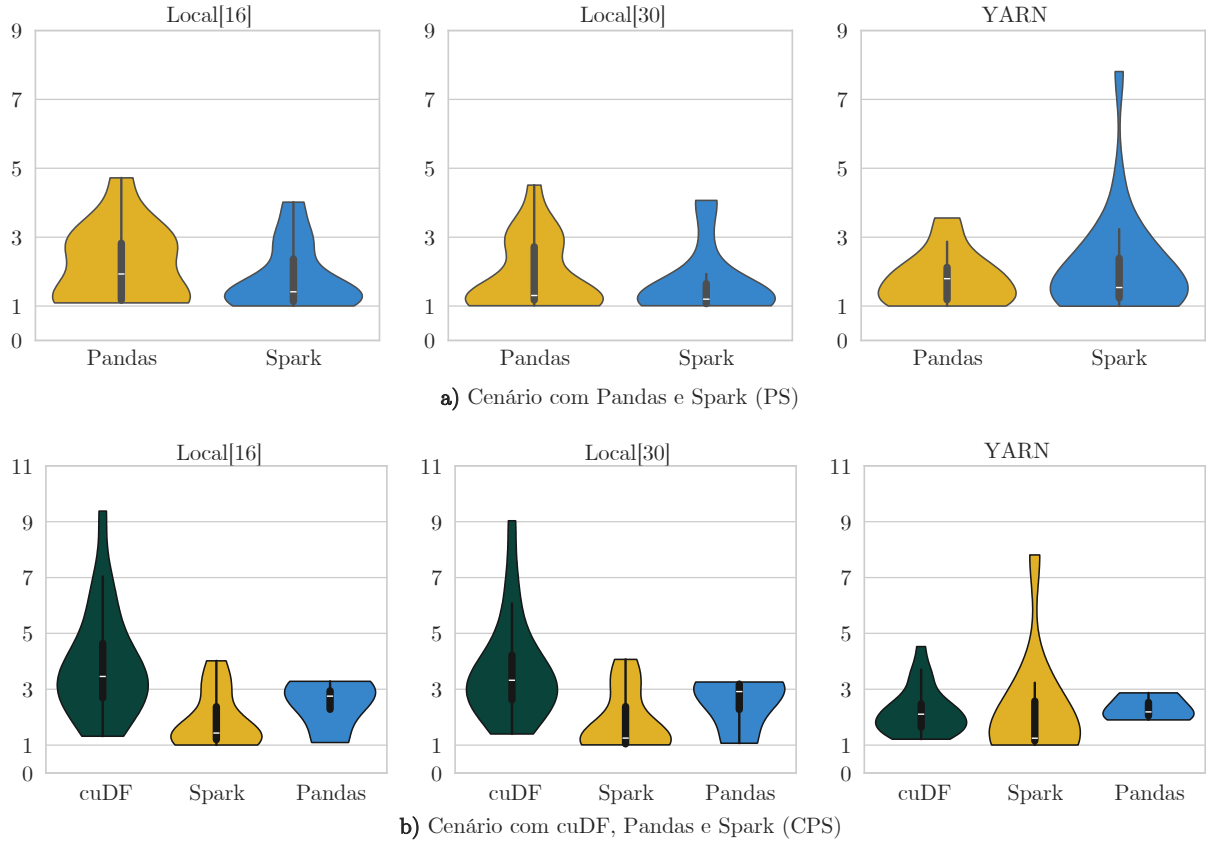


Figura 4.4: Gráfico de Violino que representa o Speedup da melhor opção de ferramenta em relação à segunda melhor.

Outra análise relevante ainda nesse contexto é a avaliação do ganho ao utilizar a melhor opção de ferramenta em relação às outras. O gráfico da figura 4.4 apresenta esse resultado a partir da distribuição do Speedup da melhor opção de ferramenta em relação à segunda melhor para aquela mesma configuração, na forma de um gráfico de violino. Nesse gráfico, valores maiores que 1 representam o quanto a melhor ferramenta para um determinado fluxo é superior às demais opções.

Avaliando os cenários que envolvem apenas Pandas e Spark (fig. 4.4a), podemos perceber que, embora na mediana dos casos as duas ferramentas possam promover ganhos próximos de 1,2 a 2 vezes em relação à outra, existem diversos casos em que os ganhos medidos podem ultrapassar 3 vezes, com casos extremos chegando a 5 e até 7,8 vezes. Essa avaliação reitera o impacto que a escolha errada de uma ferramenta pode provocar

no desempenho de uma aplicação, além de evidenciar que não existe uma única ferramenta ideal para todos os cenários.

Mesmo nos casos com a possibilidade de execução em cuDF (fig. 4.4b), que tende a apresentar maior desempenho médio, os ganhos variam. Nesse cenário, embora o cuDF possa ser superior ao Pandas e ao Spark entre 2 a 3,5 vezes na mediana dos casos, os ganhos máximos medidos foram próximos de 9 vezes. Observa-se também que, quanto maior o número de núcleos de processamento disponíveis, menor tende a ser esse ganho, uma vez que o desempenho do Spark aumenta, reduzindo a margem de superioridade do cuDF. Além disso, o número de cenários em que o Spark se destaca cresce à medida que seus recursos computacionais aumentam, ampliando também seu speedup (visto a partir do aumento de densidade no eixo do Spark).

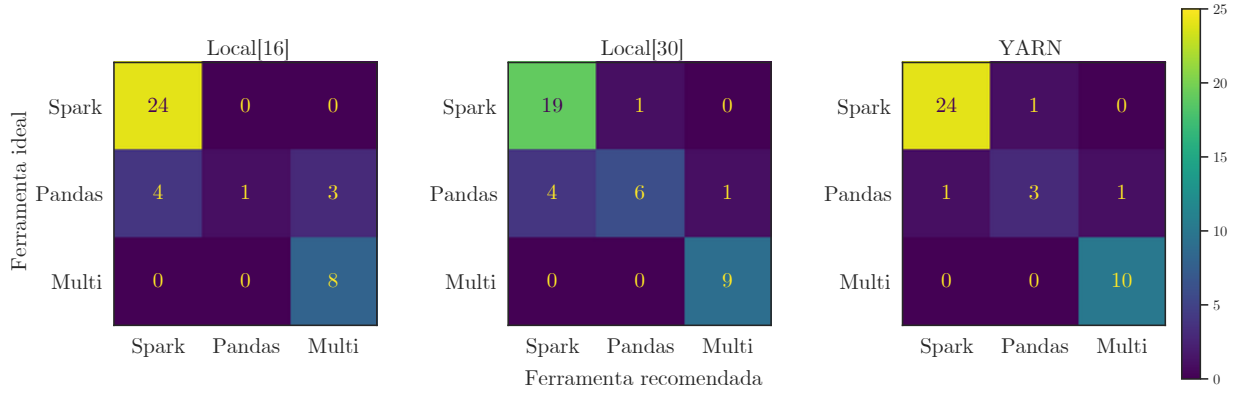
4.4.2 Recomendação oportunista

Enquanto na seção 4.4.1 avaliamos a assertividade da solução em recomendar apenas a ferramenta mais adequada para um determinado cenário, nesta seção flexibilizamos a resposta para permitir também a recomendação de combinações de ferramentas. Esse tipo de recomendação é mais difícil que o anterior, pois com a possibilidade de migração entre ferramentas, o espaço de busca da solução aumenta significativamente. Nesse contexto, os resultados do novo quadro de recomendações é ilustrado na figura 4.5. Em geral, as estimativas realizadas apontam que nem todos os fluxos se beneficiam dessa possibilidade de combinação. Nos experimentos, esse número variou de 8 a 10 cenários, representando de 20% a 25% do total.

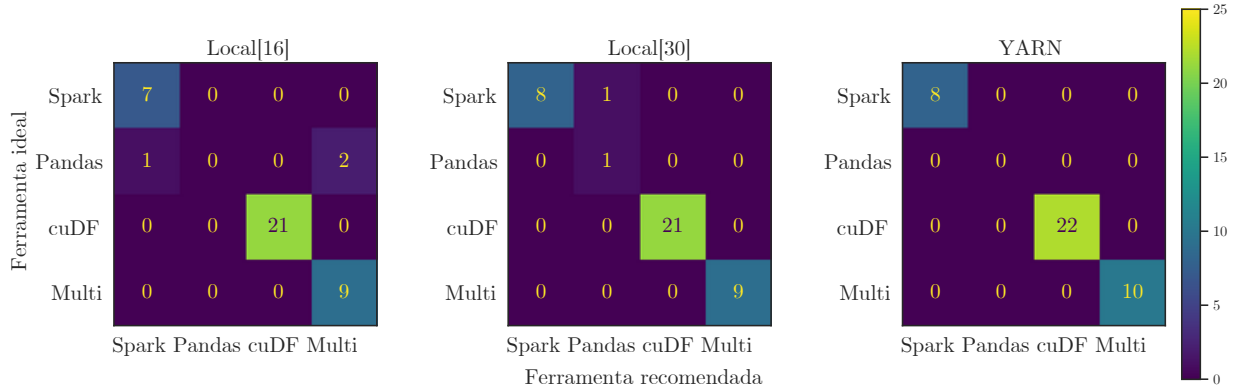
A maioria dos cenários multi-ferramentas envolvendo apenas o Pandas e o Spark (figura 4.5a) está relacionada a configurações que, anteriormente (figura 4.3a), eram atribuídas exclusivamente ao Pandas. Isso ocorre, por exemplo, devido à presença de algoritmos de aprendizado de máquina mais otimizados do que os disponíveis no Spark. No entanto, ao flexibilizar a solução para permitir a combinação de ferramentas, a maior parte desses fluxos apresentou ganhos ao executar algumas tarefas também no Spark.

Ao considerar cenários em que o cuDF está disponível (figura 4.5b), observa-se que a maioria das recomendações envolvendo múltiplas ferramentas corresponde a casos que, anteriormente, estavam vinculados ao Spark (figura 4.3b). Isso ocorre tanto pela indisponibilidade de determinadas operações no cuDF quanto pelo tamanho do conjunto de dados de entrada, que pode exceder a capacidade de memória da GPU. Nesses casos, com a possibilidade de recomendações envolvendo múltiplas ferramentas, torna-se viável a combinação de Pandas ou Spark com cuDF para maximizar o desempenho.

Complementar ao gráfico anterior, a tabela 4.9 sumariza as métricas de avaliação computadas. Os detalhes de cada fluxo de dados estão disponíveis no apêndice A. A taxa de acerto, assim como outras métricas, permanece próxima aos resultados da reco-



a) Cenário com Pandas e Spark (PS)



b) Cenário com cuDF, Pandas e Spark (CPS)

Figura 4.5: Matriz de confusão da recomendação de envolvendo opção de multiplas ferramentas.

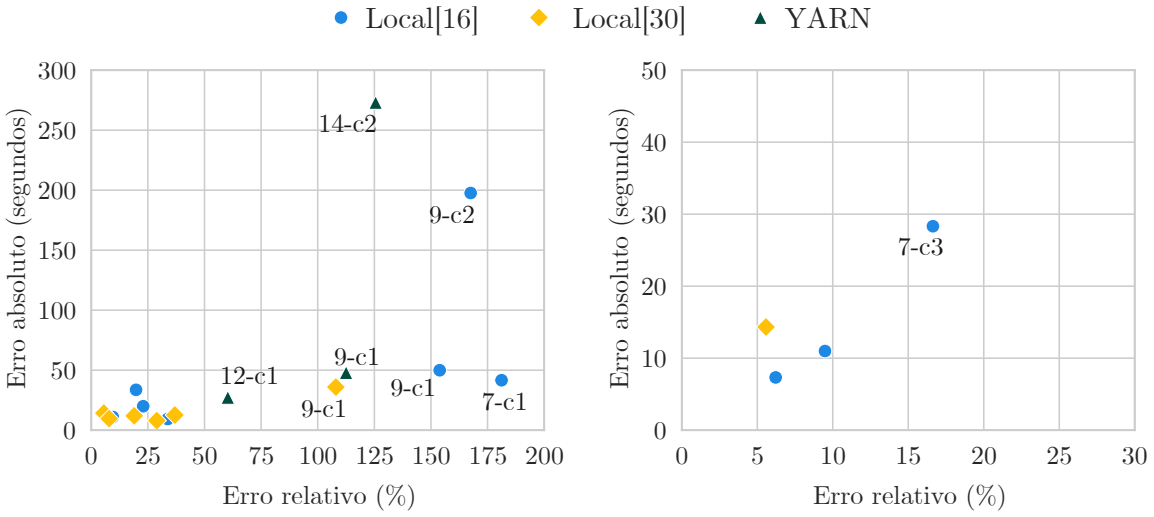
mendação de apenas uma ferramenta. Os cenários onde houve a maior redução nos valores das métricas correspondem ao hardware Local[30]. Nesse caso, o F1 diminuiu de 91,6% para 80,7%, enquanto o AUC reduziu de 94,4% para 88,6%. Como já discutido, uma possível causa para esse comportamento é à maior dificuldade de avaliar o desempenho entre o Pandas e o Spark para esse hardware com recursos intermediários. No geral, os resultados são satisfatórios, e todas as métricas indicam que o sistema possui uma boa capacidade de recomendar configurações de execução, sejam elas baseadas em uma única ferramenta ou na combinação de múltiplas ferramentas disponíveis.

Tabela 4.9: Estatísticas das métricas de avaliação das recomendações gerais.

Métrica (%)	Cenário PS			Cenário CPS		
	Local[16]	Local[30]	YARN	Local[16]	Local[30]	YARN
Taxa de acerto	82,5	85,0	92,5	92,5	97,5	100,0
Precisão	89,3	83,7	92,5	87,5	98,4	100,0
Revocação	78,1	80,6	90,0	90,3	96,8	100,0
F1	74,8	80,7	91,1	88,8	97,2	100,0
AUC	86,9	88,8	94,4	95,0	98,3	100,0

Grande parte dos cenários em que a solução erra ao indicar uma ferramenta, con-

forme apresentado na figura 4.6, são casos onde o erro absoluto é inferior a 50 segundos. Já em termos relativos, os sete cenários com erros superiores a 50% estão associados a apenas quatro fluxos: 7, 9, 12 e 14.



a) Cenário com Pandas e Spark (PS) **b)** Cenário com cuDF, Pandas e Spark (CPS)

Figura 4.6: Erros relativos versus erros absolutos para os cenários onde a solução não recomendou a melhor configuração de ferramenta(s).

Nota: Anotações no gráfico, no formato, fluxo-carga, identificam execuções com erros superiores ao limiar definido.

Nos cenários envolvendo Pandas e Spark, o fluxo 14, com carga de ID 2, é o com as piores estimativas. No entanto, esse erro é compreensível devido a uma particularidade desse fluxo: o conjunto de dados utilizado nesse fluxo possui 110 colunas, um número significativamente superior ao empregado nas etapas de calibração (no máximo 14). Dessa forma, o modelo não foi capaz de estimar o impacto desse aspecto no tempo de execução das operações com a precisão necessária, principalmente na operação de leitura de dados. No entanto, como será visto na seção 4.4.5, que aborda as avaliações após a recalibração dos modelos, esses casos extremos podem ser reduzidos ao longo do tempo. Já nos cenários que incluem a possibilidade de execução no cuDF (figura 4.6b), os erros observados foram inferiores a 30 segundos, o que representa um bom resultado, considerando a complexidade da tarefa.

Nesta seção, avaliamos a capacidade da solução na recomendação de uma configuração de ferramenta(s) visando o ganho de desempenho. A seguir, complementaremos essa análise, discutindo em maior detalhe quais cenários envolvendo múltiplas ferramentas foram identificados pela solução, bem como o impacto no desempenho quando mais de uma ferramenta pode ser utilizada. Para melhor estruturação, dividimos as avaliações em duas seções: na seção 4.4.2.1, contendo execuções apenas entre Pandas e Spark; e na seção 4.4.2.2, envolvendo execuções com todas as possibilidades de ferramentas.

4.4.2.1 Ganho de desempenho de ferramentas mistas em cenários com Pandas e Spark

A tabela 4.10 apresenta detalhes sobre os casos em que a utilização de múltiplas ferramentas foi, de fato, mais vantajosa do que a escolha de uma única ferramenta. Nessa tabela, são exibidos os tempos médios de execução de cada fluxo ao utilizar exclusivamente Spark (S), Pandas (P) ou uma combinação de múltiplas ferramentas (M).

Tabela 4.10: Tempo de execução, em segundos, dos cenários PS em que a combinação de ferramentas foi benéfica para a execução.

Hardware	Fluxo	Carga	S	P	M	S / M	P / M
Local[16]	10	2	86,3	107,7	84,7	1,02	1,27
	14	2	627,3	221,7	170,3	3,68	1,30
	15	1	65,7	69,3	49,3	1,33	1,41
		2	155,3	161,0	103,0	1,51	1,56
		3	430,0	-	249,0	1,72	-
Local[30]	14	2	640,3	202,7	145,0	4,42	1,39
YARN	7	1	76	42,3	38,3	1,98	1,11
		2	274,7	152,3	110,7	2,48	1,38
		3	578,3	303,3	205,0	2,82	1,48
	9	2	305,7	139,7	124,7	2,45	1,12
	15	1	78,7	78,3	67,3	1,17	1,16
		2	155,3	193,3	118,3	1,31	1,63
		3	394	-	273,7	1,44	-

No total, foram recomendados 13 cenários, distribuídos em cinco fluxos de dados distintos, nos quais a adoção de múltiplas ferramentas resulta em benefícios para a execução: cinco no hardware Local[16], um no Local[30] e sete na configuração YARN. Na tabela, os dois casos representados pelo símbolo "-", ambos para o fluxo 15 com a carga de entrada de ID 3, correspondem a cenários cuja execução completa não foi possível na ferramenta avaliada (Pandas), devido à falta de memória RAM. Nos demais casos, os cenários apresentados ilustram situações em que uma tarefa é mais eficiente em uma ferramenta do que em outra, seja pelo custo inerente da operação, seja pelo volume de dados no estado executado. Os ganhos obtidos nesse tipo de cenário variam de valores próximos a 1,02 até 4,42 vezes.

De forma mais objetiva, ao considerar os ganhos apenas em relação à melhor ferramenta individual, como ilustrados no gráfico da figura 4.7, os valores variam entre 1,02 e 1,7 vezes. Nesse caso, os principais cenários encontrados estão relacionados aos fluxos 7, 14 e 15. Em média, esses fluxos apresentaram ganhos próximos de 1,37 vezes em relação à melhor opção de ferramenta individual (informação que não possuímos a priori). Em termos absolutos, a abordagem proposta conseguiu reduzir o tempo de execução de um único fluxo de dados em até 181 segundos (fluxo 14 para o hardware Local[30]), o que

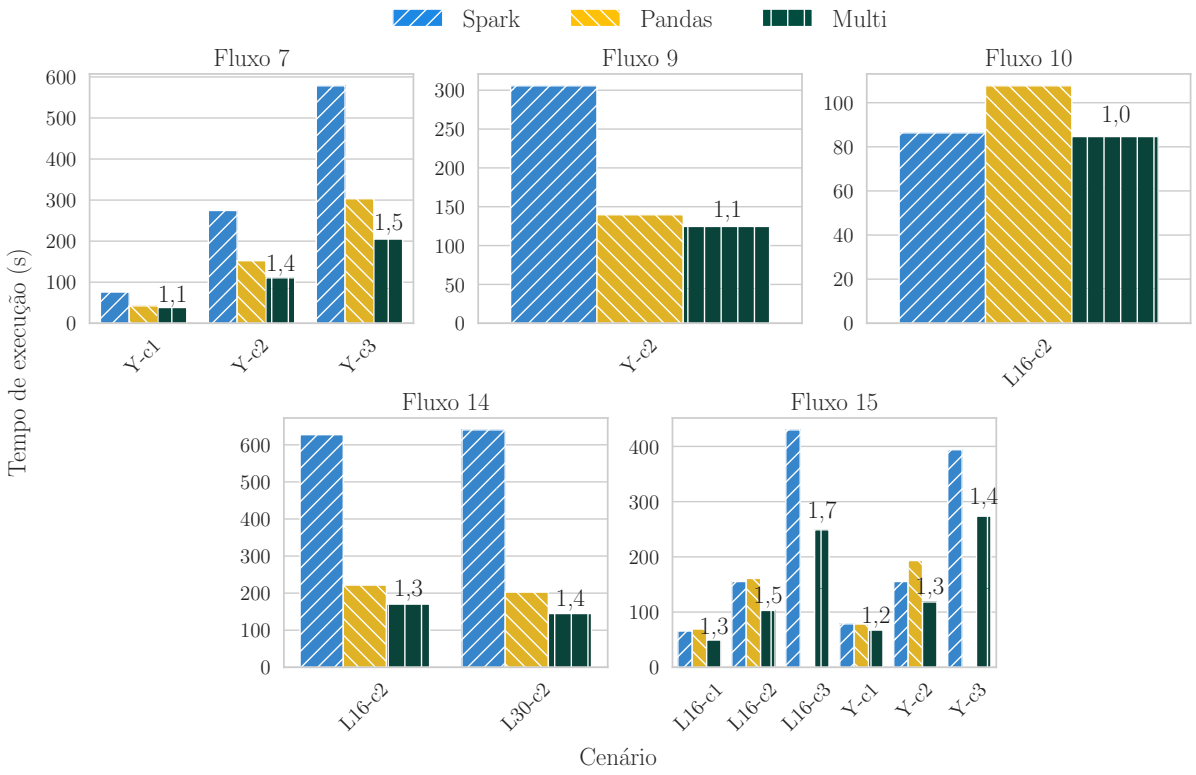


Figura 4.7: Ganho de desempenho em cenários oportunistas, considerando execuções em Pandas e Spark.

Nota: Anotações no gráfico correspondem ao speedup da recomendação da nossa abordagem em relação à melhor opção de ferramenta.

representou um ganho de até 1,7×.

4.4.2.2 Ganho de desempenho de ferramentas mistas em cenários com cuDF, Pandas e Spark

Pelo fato do cuDF ser geralmente mais rápido que o Pandas e o Spark, a maioria dos fluxos recomendados para execuções envolvendo as três possibilidades de ferramentas (tabela 4.11) ocorre em execuções onde o conjunto de dados ultrapassa a capacidade da memória da GPU. A exceção foi o fluxo 15, que exigia a execução do algoritmo Regressão Isotônica, indisponível no cuDF.

No total, 17 cenários mistos foram identificados, com as seguintes combinações: seis cenários do fluxo 15 envolveram a combinação entre Pandas e Spark, incluindo três no hardware Local[16] e três no YARN; seis cenários do fluxo 1 utilizaram a combinação entre Pandas e cuDF; e cinco cenários restantes envolveram a combinação entre Spark e cuDF.

A preferência por Pandas e cuDF, em vez de Spark e cuDF, pode ser explicada

Tabela 4.11: Tempo de execução, em segundos, dos cenários CPS em que a combinação de ferramentas foi benéfica para a execução.

Hardware	Fluxo	Carga	S	P	C	M	S / M	P / M	C / M
Local[16]	1	4	468,0	1437,3	-	150,7	3,1	9,5	-
		5	930,7	2834,3	-	297,3	3,1	9,5	-
	14	2	627,3	221,7	-	96,3	6,5	2,3	-
	15	1	66,7	71,7	-	48,7	1,4	1,5	-
		2	156,7	179,3	-	101,3	1,6	1,8	-
		3	429,0	-	-	246,7	1,7	-	-
Local[30]	1	4	434,3	1428,5	-	130,7	3,3	10,9	-
		5	929,3	2843,8	-	281,7	3,3	10,1	-
	14	2	640,3	202,7	-	97,0	6,6	2,1	-
YARN	1	4	454,0	1419,7	-	155,3	2,9	9,1	-
		5	882,3	2853,3	-	332,0	2,7	8,6	-
	7	3	578,3	303,3	-	196,3	2,9	1,5	-
	9	2	305,7	139,7	-	131,3	2,3	1,1	-
	14	2	624,0	217,6	-	141,0	4,4	1,5	-
	15	1	75,3	88,0	-	59,7	1,3	1,5	-
		2	163,3	177,0	-	113,0	1,5	1,6	-
		3	435,0	-	-	268,3	1,6	-	-

pelo custo adicional da migração entre ferramentas. Como a conversão entre cuDF e Spark exige uma etapa intermediária para a migração dos dados para o Pandas, essa etapa adicional pode representar um custo significativo, reduzindo o benefício da execução híbrida.

O gráfico da figura 4.8 representa visualmente os dados da tabela 4.11, destacando o ganho das configurações mistas em relação à melhor ferramenta individual. Comparando com o gráfico anterior (fig. 4.7), a inclusão do cuDF não apenas possibilitou novas recomendações, mas também melhorou o desempenho de cenários já presentes, por meio de novas combinações otimizadas.

Um exemplo de uma nova recomendação são os cenários do fluxo 1, onde o tempo de execução do Pandas é mais de 3 vezes maior que o Spark. Isso significa que, em um cenário onde apenas Pandas e Spark estão disponíveis, não seria vantajoso combiná-los. No entanto, ao incluir o cuDF, a combinação entre cuDF e Spark possibilitou um aumento de desempenho de até 3,3 vezes em relação ao uso exclusivo do Spark. Já considerando os demais fluxos, a média do ganho para os cenários mistos envolvendo as três possibilidades de ferramentas foi de aproximadamente 2,2 vezes, evidenciando que a combinação otimizada de ferramentas pode trazer ganhos significativos de desempenho.

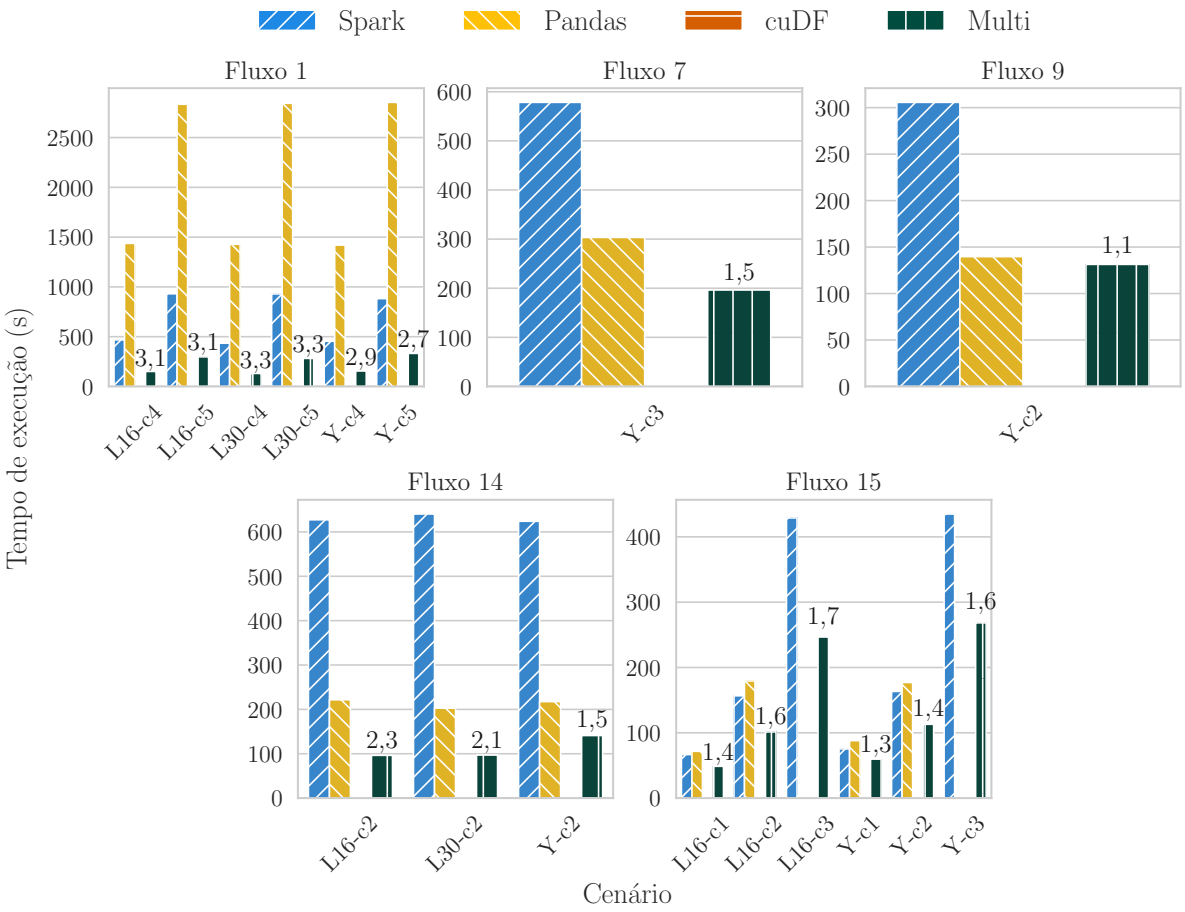


Figura 4.8: Ganho de desempenho em cenários oportunistas, considerando execuções em cuDF, Pandas e Spark.

Nota: Anotações no gráfico correspondem ao speedup da recomendação da nossa abordagem em relação à melhor opção de ferramenta.

4.4.3 Comparação com outras abordagens

Na seção 4.4.2, discutimos a capacidade do sistema de recomendar as ferramentas mais adequadas para a execução de um determinado fluxo, analisamos os ganhos de desempenho obtidos e exploramos os cenários nos quais a recomendação envolveu mais de uma ferramenta. No entanto, é importante reforçar que o *baseline* utilizado para comparação é baseado no tempo de execução da melhor opção de ferramenta individual. Esse tipo de informação não é conhecida *a priori*, ou seja, não sabemos previamente qual ferramenta isolada teria o melhor desempenho para cada fluxo. Nesse contexto, consideramos uma boa resposta se a configuração sugerida pelo sistema apresentar um tempo de execução melhor ou, no máximo, 5% pior que o *baseline*.

Nesta seção, expandimos as análises anteriores para avaliar a capacidade do sistema em identificar as melhores configurações dentre todas as possibilidades para cada fluxo, ou

seja, comparamos as recomendações sugeridas com todas as possibilidades de migrações entre as ferramentas disponíveis. Por exemplo, para um fluxo de dados com 5 operações, que podem ser executadas em duas opções de ferramentas, existiriam 2^5 combinações de execuções. Nosso experimento, consiste em executar todas essas combinações para encontrar qual combinação, de fato, produz a solução ótima.

Devido à quantidade exponencial de combinações possíveis que precisariam de serem executadas, selecionamos cinco fluxos de trabalho diferentes com um pequeno número de operações (entre 4 e 6 operações, e apenas um com 11). E, ainda para simplificar o experimento, foram realizadas apenas execuções envolvendo Pandas e Spark, todas utilizando a configuração de hardware Local[16]:

1. **Fluxo 3 - carga 4 (4,2 GB)**: composto por cinco operações e a sua recomendação foi a execução utilizando apenas Spark;
2. **Fluxo 8 - carga 2 (8,5 GB)**: composto por quatro operações e a sua recomendação também envolveu execuções apenas via Spark;
3. **Fluxo 13 - carga 1 (1,5 GB)**: composto por cinco operações e a sua recomendação foi a execução utilizando apenas Pandas;
4. **Fluxo 14 - carga 2 (5,2 GB)**: composto por onze operações e sua recomendação envolveu ferramentas mistas (1º a 2º operações em Spark, 3º a 9ª em Pandas, e o restante em Spark);
5. **Fluxo 15 - carga 2 (5,2 GB)**: composto por seis operações e sua recomendação envolveu ferramentas mistas (1º a 3º operações em Spark, 4ª em Pandas, e o restante em Spark);

Ao executar todo o conjunto de possibilidades, podemos comparar o quão longe nossa recomendação está da solução ótima. Além disso, também incluímos uma comparação das recomendações geradas por outras abordagens existentes, como o Octopus (discutido na seção 2.1.3) e modelos de linguagem de grande escala (sigla do inglês, LLM) populares, como ChatGPT e Perplexity.

A escolha desses fluxos permite avaliar a precisão das recomendações do sistema em diferentes contextos, além de viabilizar uma comparação direta entre as abordagens. Para isso, executamos todas as combinações possíveis de ferramentas para cada operação usando GridSearch, considerando Pandas e Spark no hardware Local[16]. Consideramos apenas as duas ferramentas pois, além de aumentar o número de execuções exponencialmente ao adicionar uma nova ferramenta, a presença de apenas essas duas ferramentas tornam os cenários mais complexos. No caso do Fluxo 14, devido ao número elevado de operações, seria necessário avaliar 2^{11} cenários, por causa disso, nos limitamos a avaliação aleatória de 30% das combinações (RandomGrid).

A tabela 4.12 apresenta os tempos ótimos e piores de execução, além do número de combinações possíveis e executáveis (que não geraram erros). Nessa tabela, observamos que a diferença entre a melhor e a pior configuração foi de até 21,3 vezes (Fluxo 8). Em cenários como Fluxo 14, 69% das combinações testadas foram inválidas devido a restrições de migração entre Spark e Pandas (excedendo 1 GB, limite padrão do Spark). Esses resultados destacam um outro requisito importante para um sistema multiplataforma: além de recomendar uma solução eficiente, o sistema precisa garantir que a recomendação seja executável, evitando falhas por restrições de memória ou outras limitações como migração de dados.

Tabela 4.12: Comparação do tempo de recomendação e de execução, em segundos, entre diversas abordagens considerando as ferramentas Pandas e Spark (PS). Primeiras três linhas da tabela representam respectivamente, a solução ótima, a pior combinação de ferramentas, e a relação entre as combinações executáveis e as totais.

Fluxo/carga	3/c4	8/c2	13/c1	14/c2	15/c2
Melhor	82,5	71,3	57,0	128,0	77,3
Pior	163,5	1508,1	803,7	704,3	238,5
#	20 (32)	10 (16)	32 (32)	189 (608)	48 (64)
Proposta	0,1+86,0	0,1+**71,3**	0,2+**57,0**	0,3+**139,1**	0,2+101,5
Octopus	32,7 †	60,4+**71,3**	11,9+596,7	83,7 †	54,8+**79,0**
GPT-4o v1	25,4+**82,5**	29,6+146,3	28,9+**57,0**	26,0+522,0	20,1+162,2
GPT-4o v2	76,2 †	27,1+**71,3**	72,1+782,7	25,2+522,0	50,1 †
GPT-4.1 v1	31,7+86,0	38,6+**71,3**	24,1+**57,0**	43,7+627,3	37,6+157,0
GPT-4.1 v2	49,5+86,0	39,9+**71,3**	66,7+255,7	46,6+627,3	51,4+101,5

†, cenário onde a configuração de ferramenta(s) recomendada apresentou erros de execução.

A segunda parte da tabela 4.12 apresenta o resultado da comparação do sistema proposto com as recomendações do Octopus e com abordagens baseadas em LLM. No caso do Octopus, pelo fato da ferramenta estar desatualizada há quatro anos e utilizar versões obsoletas do Spark e do Pandas, nossa abordagem se limitou a usá-lo apenas para recomendar a configuração de ferramenta(s), uma vez que todas as combinações possíveis já foram executadas na etapa anterior. Outro problema identificado na utilização do Octoputs foi a ausência de muitas das operações utilizadas nesses fluxos. Para contornar essa limitação, adicionamos as operações necessárias em uma versão atualizada, seguindo a metodologia empregada nas operações originais do Octopus.

As comparações envolvendo motores baseados em LLMs foram realizadas com dois modelo da OpenAI: o `GPT-4o`, acessado via ChatGPT gratuito; e o modelo `GPT-4.1`, acessado por meio do plano pago da plataforma Perplexity. Os experimentos foram conduzidos em duas versões distintas na data de 23/04/2025. A primeira abordagem, referenciada na tabela como "v1", foi aplicada tanto no ChatGPT quanto no Perplexity. Nessa versão, utilizou-se um *prompt* contendo as informações do conjunto de dados (como tamanho do arquivo e nomes das colunas), seguido da enumeração das operações a serem executadas

e dos detalhes da configuração do hardware. Ao final do *prompt*, solicita-se que o modelo avalie o fluxo completo e decida qual ferramenta deve ser utilizada em cada etapa, considerando os possíveis custos de migração entre ferramentas.

Na segunda versão dos experimentos (referenciada como "v2"), foi adotada uma abordagem baseada em dados históricos de execução. Nela, carregou-se uma planilha contendo os tempos médios de execução de cada operação nas diferentes ferramentas, considerando uma entrada de dados específica, caracterizada pelo número de linhas e colunas. A partir dessas informações, solicitou-se ao modelo LLM que recomendasse quais ferramentas utilizar em cada etapa, levando em conta os tempos médios apresentados. Exemplos dos *prompts* utilizados em cada abordagem estão disponíveis no apêndice B.

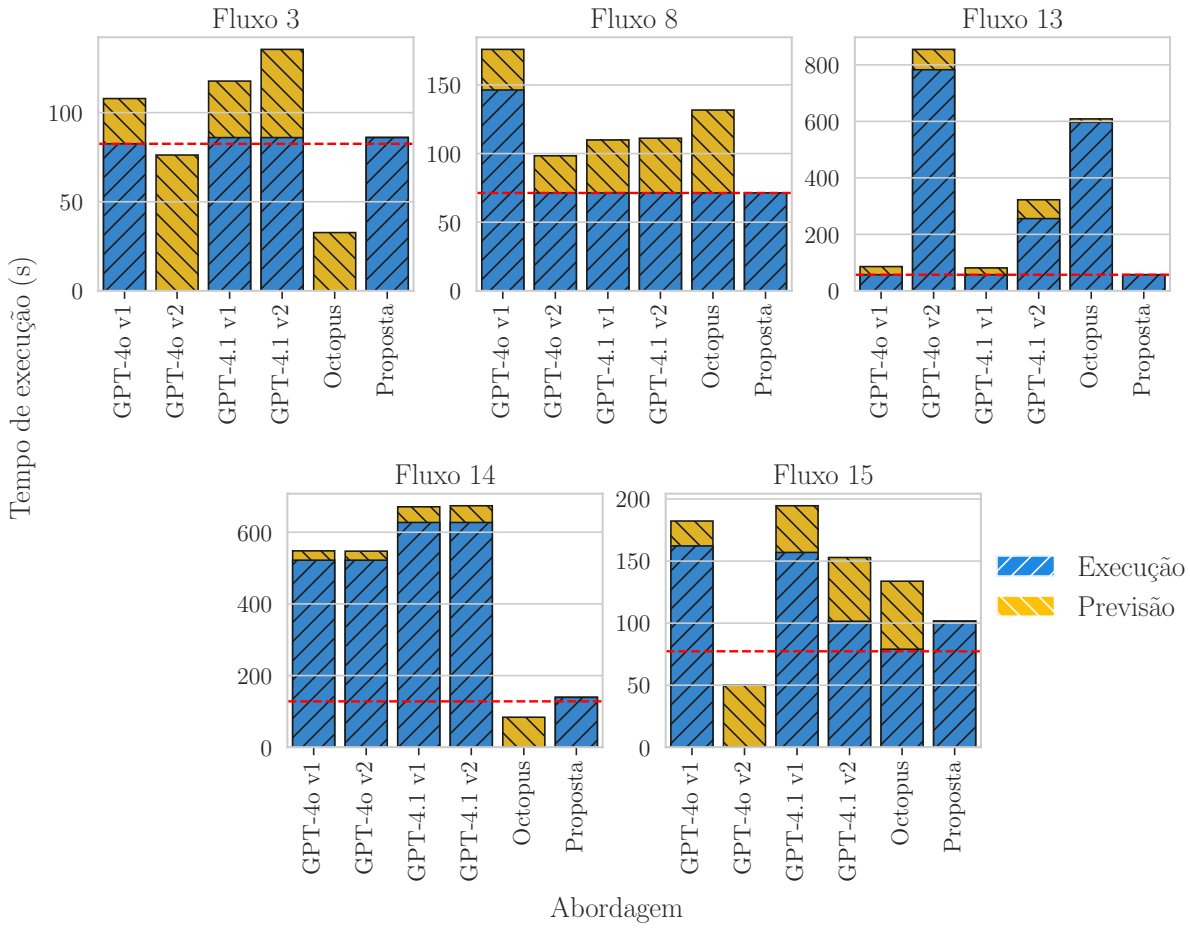


Figura 4.9: Comparação das recomendações entre diversas abordagens considerando Pandas e o Spark.

Na tabela 4.12, os resultados de cada abordagem são apresentados considerando, inicialmente, o tempo gasto para a geração da recomendação e, posteriormente, o tempo de execução para o cenário recomendado. Essa avaliação também é representada graficamente na figura 4.9. Nossa abordagem apresenta o menor tempo de recomendação, levando menos de 1 segundo para todos os cenários avaliados. Já o Octopus, por precisar ler o arquivo de entrada para gerar a amostra dos dados utilizada nas avaliações, pode

ter um tempo de predição bastante elevado. Por exemplo, no Fluxo 14, que envolvia um arquivo de entrada de 4,9 GB, o custo médio da previsão foi de 83,7 segundos. Por causa disso, na maioria dos cenários do Octopus, o tempo de previsão foi próximo do tempo de execução, o que impacta negativamente os ganhos em utilizar a ferramenta, tornando sua aplicação menos vantajosa em comparação ao ganho de desempenho que ela fornece.

Similar ao Octopus, o custo de previsão ao utilizar modelos LLM também foi elevado. O tempo de recomendação para consultas do tipo "v1" no ChatGPT permaneceu entre 20 e 30 segundos. Já nas consultas executadas via Perplexity, bem como na versão "v2" do ChatGPT, o tempo de resposta foi mais alto, variando entre 25 e 72 segundos. No caso da segunda abordagem no ChatGPT, esse tempo adicional pode estar relacionado ao custo computacional de processar a planilha anexada à consulta, necessária para inferir as melhores opções de ferramentas com base nas operações descritas. Já no Perplexity, o tempo maior pode ser atribuído ao custo adicional de realizar buscas em fontes externas, incluindo a web e bases acadêmicas. Talvez, em experimentos utilizando modelos LLM configurados localmente, o tempo de previsão pudesse ser reduzido, pois haveria menos latência de comunicação e a eliminação da fila de requisições que serviços como OpenAI e Perplexity impõem internamente. No entanto, manter uma GPU robusta e dedicada a essa tarefa pode gerar outros custos, o que pode tornar essa abordagem inviável em algumas situações.

O gráfico da figura 4.9 mostra que, no geral, nosso sistema produziu as melhores recomendações em três dos cinco cenários avaliados. Em segundo lugar, destaca-se o modelo `GPT-4.1`, na consulta do tipo "v1", com três recomendações bem-sucedidas. Observa-se também que as consultas do tipo 'v2", apesar de fornecerem mais detalhes sobre o contexto de execução, não resultaram em melhorias nas recomendações e, em alguns casos, como no Fluxo 13, chegaram a prejudicá-las. Em nossa interpretação, isso indica que esse tipo de modelo atua como um especialista generalista, oferecendo conselhos amplos, mas com dificuldade de adaptação às particularidades de cada cenário. Como consequência, fluxos como o 14 e o 15 apresentaram recomendações que levaram a execuções significativamente lentas ou até mesmo a configurações inviáveis.

Por outro lado, a abordagem proposta por nosso sistema foi, na maioria dos casos, capaz de recomendar soluções próximas da ótima. Por exemplo, no Fluxo 3, embora a recomendação final tenha sido a segunda melhor, o sistema identificou corretamente que a configuração ideal envolvia a execução das quatro primeiras operações em Spark e da última em Pandas. No entanto, como a diferença de desempenho entre essa configuração e a execução inteiramente em Spark era inferior a 5%, o sistema optou por manter todas as operações em Spark, priorizando simplicidade e robustez. Caso contrário, a solução ótima teria sido corretamente selecionada.

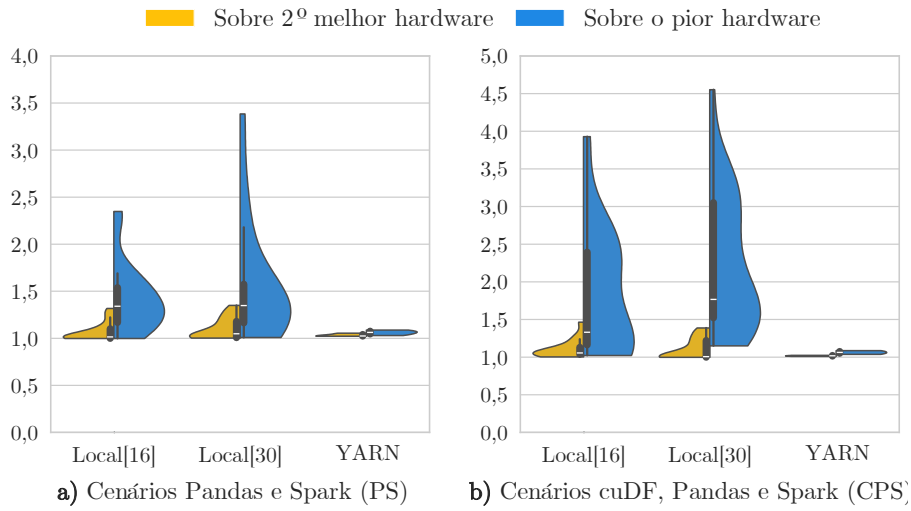
Ao considerar o tempo total, isto é, o somatório entre o tempo de recomendação e o tempo de execução, nossa abordagem se mostra mais eficiente em todos os cenários

avaliados, inclusive naqueles em que outras abordagens recomendaram a configuração ótima, como nos Fluxos 3 e 15. No caso específico do Fluxo 15, por exemplo, embora a recomendação gerada pelo nosso sistema tenha resultado em um desempenho 30% inferior ao da configuração ótima, o tempo total de processamento foi, ainda assim, 31% mais rápido do que o obtido com o Octopus nesse mesmo cenário.

4.4.4 Recomendação da configuração de hardware

Nas seções anteriores, avaliamos nosso agendador na recomendação das ferramentas mais adequadas para uma configuração de hardware fixa. Nesta seção, incluímos o hardware como mais uma variável de recomendação, permitindo que o próprio sistema determine a melhor opção. Embora o senso comum leve muitos usuários a optarem pelo hardware com mais recursos, esperando maior desempenho, os resultados apresentados aqui mostram que essa abordagem intuitiva nem sempre é a mais eficiente.

Em cenários reais, um usuário pode ter acesso a múltiplas opções de hardwares, cada uma com diferentes configurações de recursos. No entanto, ferramentas como o Lemonade podem impor restrições de uso, limitando o acesso a determinados hardwares conforme o perfil do usuário. Diante disso, um sistema multiplataforma eficiente deve ser capaz de se adaptar às configurações disponíveis, levando em conta restrições e possibilidades. Caso haja mais de um hardware compatível, o sistema deve avaliar e selecionar automaticamente a opção mais adequada para a execução do fluxo, garantindo desempenho e melhor aproveitamento dos recursos computacionais.



a) Cenários Pandas e Spark (PS) b) Cenários cuDF, Pandas e Spark (CPS)

Figura 4.10: Gráfico de violino da distribuição do speedup da melhor configuração de hardware.

O impacto da escolha do hardware no tempo de execução é ilustrado no gráfico da figura 4.10. Embora a mediana dos cenários indique que selecionar o hardware ideal pode resultar em um ganho de desempenho entre 1,1× e 1,4×, já justificando a importância

dessa análise, casos extremos apresentaram ganhos de até 4,5×, motivando ainda mais relevância dessa decisão para a otimização do fluxo de trabalho. Em geral, os hardwares Local[16] e Local[30] apresentaram um comportamento similar: em diversos cenários, esses hardwares foram até 1,4 vezes mais rápidos que a segunda melhor opção disponível. Por outro lado, houve poucos casos em que a execução no YARN foi superior e, mesmo nesses casos, o ganho obtido foi pouco expressivo. Mesmo em cenários onde o desempenho de um fluxo é similar em duas ou mais configurações de hardwares, ser capaz de estimar esse desempenho permite otimizar o uso dos recursos computacionais, escolhendo o hardware com menos recursos, evitando ociosidade e maximizando a eficiência da infraestrutura.

A tabela 4.13 apresenta os resultados experimentais sobre a precisão da ferramenta na recomendação do melhor hardware. Na avaliação, em casos de empate no tempo de execução entre os três hardwares, a resposta correta considerada é aquela que utiliza menos recursos computacionais. Seguindo esse critério, nossa solução foi capaz de recomendar corretamente em até 80,0% dos cenários avaliados.

Tabela 4.13: Avaliação da recomendação de hardwares

Métrica (%)	PS	CPS
Taxa de acerto	70,0	80,0
Precisão	78,5	91,8
Revocação	70,0	80,0
F1	67,2	83,2
AUC	77,5	85,0

Como discutido anteriormente, cenários envolvendo apenas Pandas e Spark são mais difíceis de avaliar devido ao desempenho semelhante dessas ferramentas na maioria dos casos. No entanto, mesmo com uma taxa de acerto de 70,0%, os resultados são promissores. A área abaixo da curva (AUC) de 77,5% indica que a qualidade do modelo é moderada e seu desempenho é superior ao de um classificador aleatório, que teria um AUC de 50%. Ainda assim, há espaço para melhorias, que podem ser alcançadas, por exemplo, por meio de recalibrações do modelo. Por outro lado, a taxa de acerto de 85,0% dos cenários CPS, é geralmente interpretada como um bom valor para um modelo preditivo. Complementando esses resultados, a matriz de confusão da figura 4.11 apresenta a relação entre as recomendações da ferramenta e o hardware ideal para cada cenário.

Tanto nos cenários PS (fig. 4.11a) quanto em CPS (fig.4.11b), o hardware Local[16] é o mais indicado para a maioria das execuções. Esse resultado contraria a percepção tradicional de que o hardware com mais recursos é sempre a melhor escolha. Na fig. 4.11b, essa predominância do Local[16] é ainda mais evidente, provavelmente porque o cuDF executa paralelismo via GPU, e, portanto, um maior número de núcleos de CPU não impacta diretamente o desempenho.

Ao considerar apenas execuções em Spark e Pandas, o cenário ideal teria 21

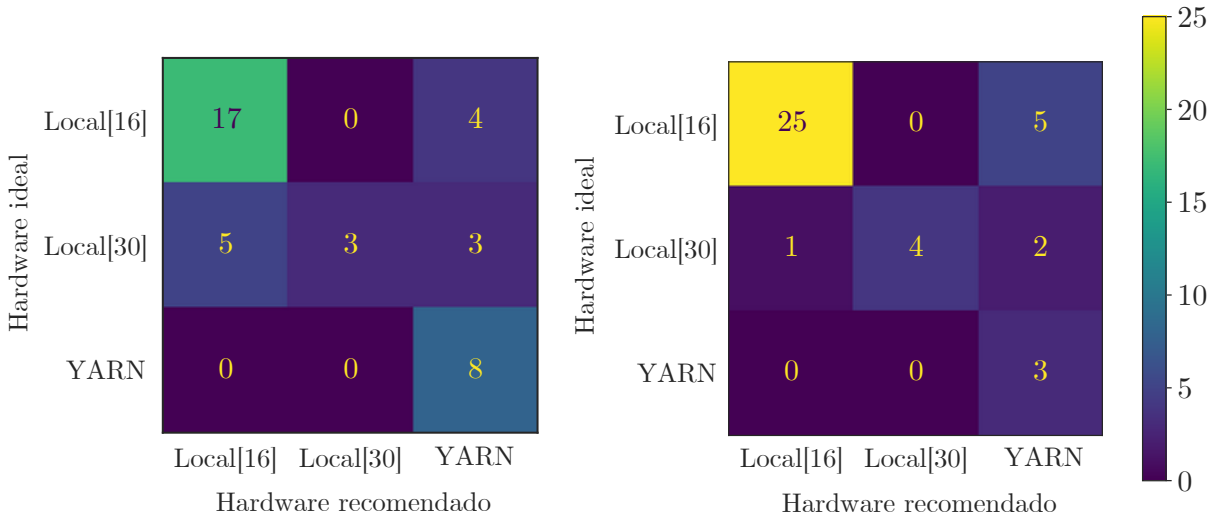


Figura 4.11: Matriz de confusão da recomendação da configuração de hardware.

execuções no Local[16], 11 no Local[30] e 8 em YARN. No entanto, observa-se que o Local[30] foi o hardware mais difícil de distinguir dos demais, provocando mais erros. Esse resultado faz sentido, pois o Local[30] está situado entre dois extremos, o que pode gerar sobreposição de desempenho com as outras opções de hardware.

Ao considerar o decaimento da quantidade de cenários para cada configuração de hardware, uma análise preliminar poderia sugerir que a decisão do hardware de execução depende exclusivamente do tamanho do conjunto de dados de entrada. Embora esse fator seja importante, o gráfico da figura 4.12 indica que outros aspectos também influenciam essa decisão.

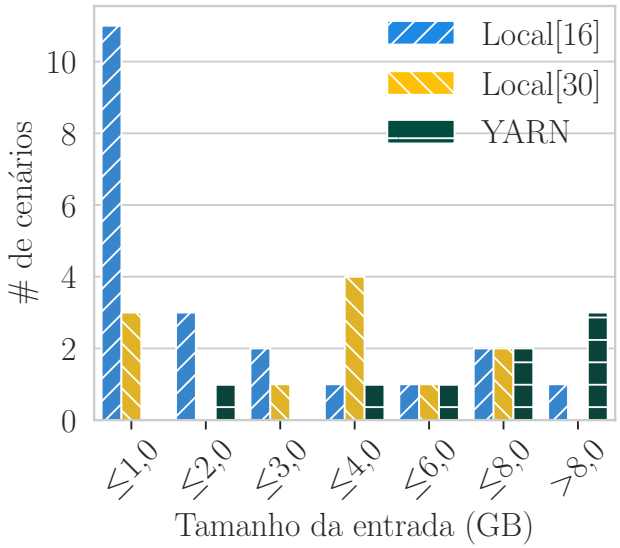


Figura 4.12: Quantidade de cenários por configuração de hardware em execuções Spark.

O gráfico em questão mostra a quantidade de cenários Spark em função do tamanho

dos dados de entrada para cada configuração de hardware. Nota-se uma tendência geral: (*i*) conjuntos de dados menores são mais frequentemente executados no Local[16]; (*ii*) cenários intermediários tendem a ser processados no Local[30]; e (*iii*) já bases maiores são preferencialmente executadas no YARN. No entanto, há exceções a esse padrão, por exemplo: existem dois cenários com dados de entrada entre 6 e 8 GB que o Local[16] ainda é a melhor opção, além de um outro cenário em que o arquivo de entrada foi de 13 GB. Também existem cenários com arquivos menores ou iguais a 1 GB, em que Local[30] já seria a melhor escolha. Isso indica que, além do tamanho dos dados de entrada, outros fatores impactam a escolha do hardware, como: (*i*) o conjunto de operações aplicadas sobre os dados; (*ii*) a organização das tarefas dentro do fluxo de execução; e (*iii*) o nível de paralelismo suportado por cada configuração. Portanto, uma decisão baseada apenas no tamanho dos dados pode ser limitada, e uma análise mais abrangente é essencial para maximizar o desempenho e a eficiência da execução.

4.4.5 Impacto da recalibração no aumento de qualidade

Nosso modelo de abordagem também inclui a capacidade de aprender com execuções anteriores. Nesta seção, reavaliamos a capacidade da solução em recomendar a melhor configuração de ferramenta após recalibrar o modelo usando dados históricos semelhantes aos utilizados em nossos experimentos. Conforme detalhado no planejamento experimental (seção 4.2.2), a recalibração incorporou os históricos de execução dos fluxos 1, 2, 4, 5, 7 e 14. Os demais fluxos permaneceram restritos às avaliações. Como explicado no capítulo 3.3.3, os novos dados incluídos nas calibrações dos modelos são a partir das estimativas das estatísticas dos dados em cada estágio de processamento, bem como os parâmetros da operação e o tempo estimado (para execuções preguiçosas em Spark) ou o tempo real (para execuções ansiosas como o cuDF e Pandas) gasto em cada tarefa executada de um fluxo.

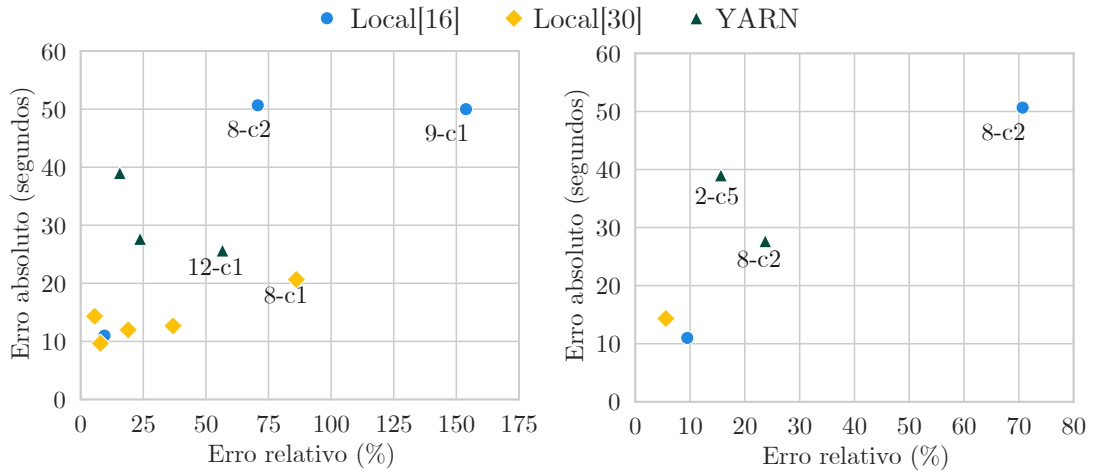
Conforme resumido na tabela 4.14, a recalibração dos modelos com dados históricos de fluxos mais complexos melhorou significativamente a qualidade da recomendação oportunista. Na tabela, a maioria das métricas apresenta valores superiores aos da calibração original (tabela 4.9), exceto os elementos destacados em negrito. A maioria desses casos está associada a execuções no hardware YARN. Acreditamos que esse comportamento seja decorrente do fato de que, como demonstrado nos gráficos das figuras 4.10 e 4.11, poucos cenários foram favorecidos pelo hardware YARN, além de apresentar um ganho inferior aos demais hardwares.

Em termos práticos, no cenário YARN em CPS, a redução das métricas deve-se à introdução de apenas dois erros entre 40 cenários avaliados. Essa informação é ilustrada na figura 4.13, que mapeia a relação entre erro absoluto (em segundos) e erro relativo nos cenários com recomendações não ideais. Esses dois casos de erro impactaram, no

Tabela 4.14: Estatísticas das métricas de avaliação da recomendação oportunista após recalibração. Setas relacionam os valores antes e depois da recalibração.

Métrica (%)	Pandas e Spark (PS)			cuDF, Pandas e Spark (CPS)		
	Local[16]	Local[30]	YARN	Local[16]	Local[30]	YARN
Tax. acerto	82,5 → 92,5	85,0 → 87,5	92,5 → 92,5	92,5 → 95,0	97,5 → 97,5	100,0 → **95,0**
Precisão	89,3 → **84,9**	83,7 → 86,1	92,5 → 92,1	87,5 → **92,0**	98,4 → 98,1	100,0 → **95,7**
Revocação	78,1 → 85,0	80,6 → 82,8	90,0 → 83,3	90,3 → 92,0	96,8 → **96,3**	100,0 → **95,0**
F1	74,8 → 84,7	80,7 → 84,1	91,1 → **85,9**	88,8 → 92,0	97,2 → 96,7	100,0 → **94,5**
AUC	86,9 → 94,4	88,8 → 90,6	94,4 → 94,4	95,0 → 96,6	98,3 → 98,3	100,0 → **96,2**

máximo, 25% do tempo de execução, um efeito relativamente controlado no contexto geral da avaliação.



a) Cenário com Pandas e Spark (PS)

b) Cenário com cuDF, Pandas e Spark (CPS)

Figura 4.13: Erros relativos versus erros absolutos para os cenários onde a solução, após recalibração, não recomendou a melhor configuração de ferramenta(s).

Nota: Anotações no gráfico, no formato, fluxo-carga, identificam execuções com erros superiores ao limiar definido.

Pelo gráfico da figura 4.13, podemos observar que o erro absoluto máximo reduziu significativamente de 273,1 segundos (figura 4.6) para apenas 50,7 segundos. Além disso, os dois piores cenários, `9-c2` e `14-c2`, que anteriormente apresentavam recomendações pelo menos 200 segundos mais lentas do que a melhor ferramenta individual, foram eliminados após a recalibração.

A melhora no cenário `14-c2` é esperada, pois fluxos históricos desse cenário foram incluídos na recalibração. Como já mencionado anteriormente, esse fluxo continha um grande número de colunas (110), uma característica que não havia sido considerada adequadamente na calibração original. De forma que, com uma nova calibração, o modelo conseguiu aprender melhor a avaliar o custo do número de colunas na leitura de dados. Por outro lado, o cenário `9-c2` não fez parte da recalibração, o que indica que o modelo foi

capaz de generalizar melhor as informações aprendidas e aplicá-las corretamente a novos cenários, conforme previsto.

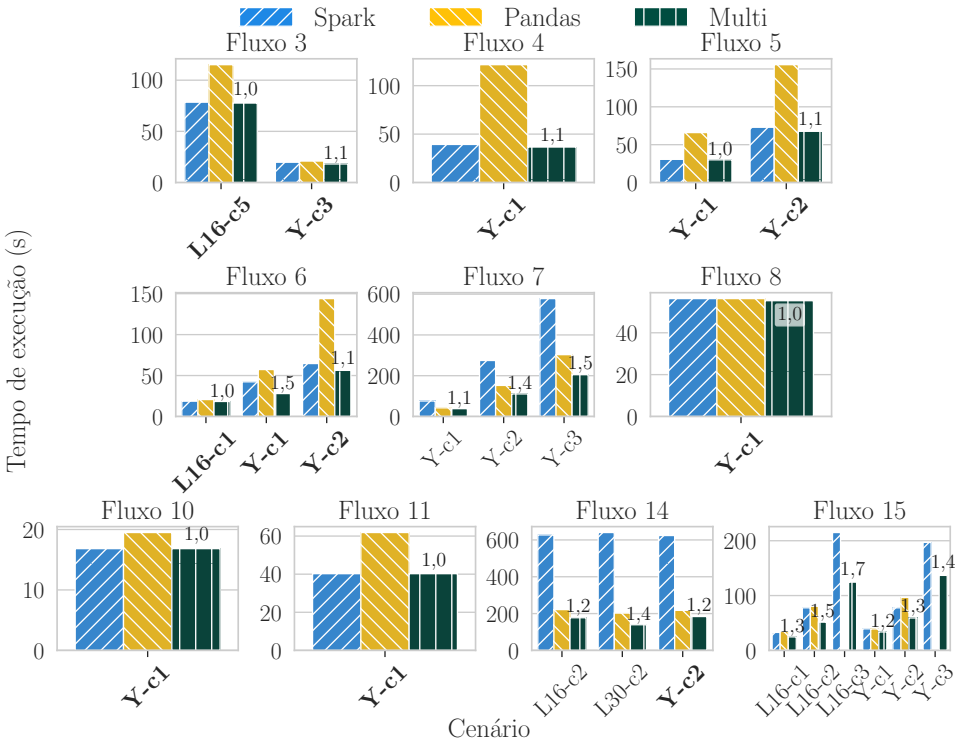


Figura 4.14: Ganho de desempenho em cenários oportunistas após recalibração (considerando Pandas e Spark).

Considerando apenas o gráfico da figura 4.13b, percebe-se também que alguns erros anteriores, como o `7-c3`, foram eliminados. No entanto, três novos cenários de erro surgiram, sendo dois deles pertencentes ao mesmo Fluxo 8. Apesar disso, acreditamos que erros abaixo de 50 segundos sejam aceitáveis, dada a alta taxa de recomendações corretas e o impacto reduzido dessas pequenas imprecisões no tempo total de execução.

Com a recalibração, além da redução dos erros de recomendação e melhoria das métricas discutidas na tabela 4.14, também observamos um aumento na identificação de cenários multi-ferramentas, nos quais a combinação de diferentes ferramentas resultou em desempenhos superiores às execuções iniciais. As figuras 4.14 e 4.15 ilustram essa evolução, respectivamente para cenários PS e CPS. Os nomes em negrito representam novos cenários multi-ferramentas identificados após a recalibração.

Comparando esses gráficos com os das figuras 4.7 e 4.8, podemos destacar três pontos principais: (*i*) a quantidade de fluxos recomendados com múltiplas ferramentas aumentou de 5 para 10 em execuções envolvendo Pandas e Spark; (*ii*) novos cenários de carga/hardware multi-ferramentas foram encontrados para os fluxos 7 e 15 em CPS; (*iii*) em alguns casos, o ganho de cenários já existentes aumentou após a recalibração, como observado no cenário `Y-c2` do fluxo 14 (CPS). Esses resultados demonstram que

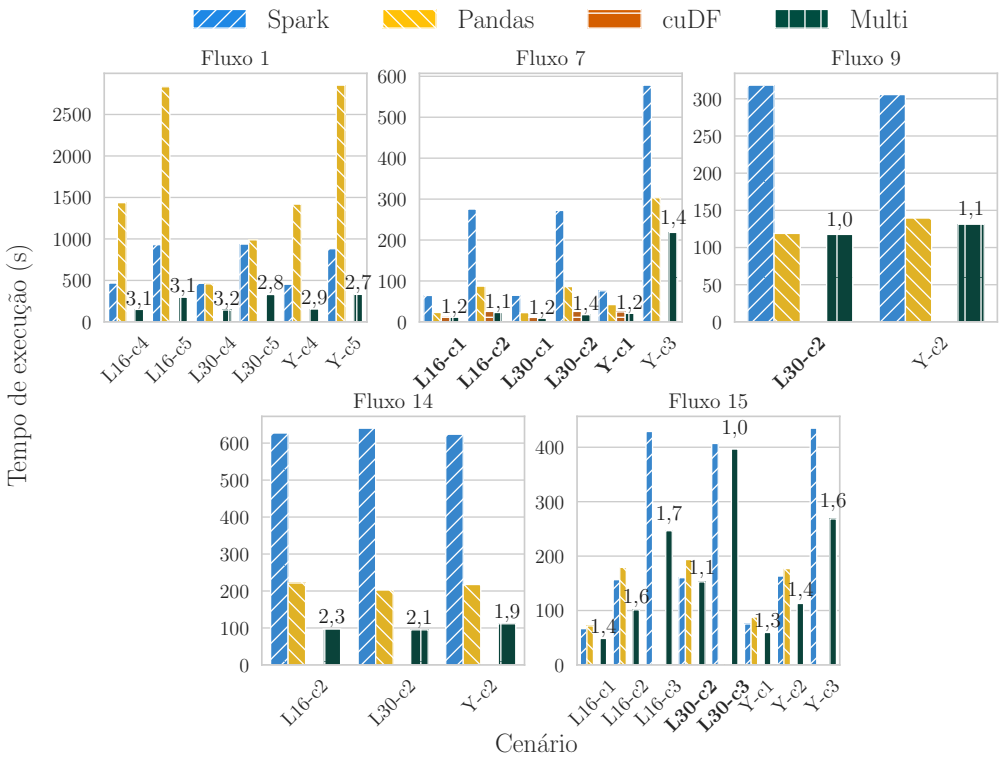


Figura 4.15: Ganho de desempenho em cenários oportunistas após recalibração (considerando cuDF, Pandas e Spark).

a recalibração não apenas reduziu erros de recomendação, mas também tornou o modelo mais eficiente na identificação de combinações de ferramentas ideais, maximizando o desempenho dos fluxos de execução.

Discussão. Na seção 4.4, analisamos a qualidade das recomendações de configuração de execução com base em métricas como taxa de acerto, ganho relativo e erros de previsão. Em geral, a solução demonstrou ser eficaz na identificação da(s) ferramenta(s) mais adequadas para cada cenário de execução. Nossos experimentos evidenciaram casos em que a escolha de uma ferramenta menos eficiente resultou em uma diferença de até 9,4× no tempo de execução, reforçando a importância de um sistema automatizado de recomendação para evitar perdas de desempenho significativas. Considerando abordagens oportunistas, o sistema foi capaz de recomendar cenários híbridos envolvendo duas ferramentas, alcançando ganhos de até 3,3× em comparação à melhor abordagem individual. Além disso, os resultados da seção 4.4.5 demonstram que o modelo evolui à medida que mais dados históricos são incorporados, tornando-se progressivamente mais preciso na recomendação das configurações ideais de execução.

Capítulo 5

Conclusão

A área de Ciência de Dados tem ganhado destaque nos últimos anos, possibilitando a resolução de uma variedade de problemas em contextos empresariais e acadêmicos. Essa disciplina oferece abordagens que, em épocas anteriores, seriam consideradas impraticáveis. Contudo, a rapidez com que novas técnicas e ferramentas surgem para atender a novos requisitos torna desafiador acompanhar as inovações. A decisão sobre qual ferramenta utilizar, seja por requisitos de funcionalidades, interoperabilidade ou desempenho, torna-se cada vez mais complexa diante da diversidade de opções disponíveis. Assim, destaca-se a importância de um sistema capaz de simplificar essa tomada de decisão.

Neste trabalho, propomos um sistema multiplataforma para a execução de fluxos de Ciência de Dados baseados em DataFrame. Esse sistema é capaz de converter um fluxo de dados criado visualmente, em uma interface genérica e amigável, em código-fonte para diversas ferramentas, eliminando a necessidade do usuário especificar onde cada atividade deverá ser executada. Demonstramos que não apenas o sistema possui uma boa taxa de acerto na recomendação das melhores configurações de ferramentas para o problema, mas também que existem casos em que a melhor solução é aquela que envolve a interoperabilidade entre ferramentas, aproveitando o melhor de cada uma.

Do ponto de vista da implementação, ressaltamos que sistemas multiplataformas desenvolvidos sem a capacidade de inferir a evolução dos dados ao longo do fluxo de atividades terão resultados limitados. Os fluxos de Ciência de Dados frequentemente envolvem etapas extensas, como pré-processamento, criação de modelos de aprendizado de máquina e aplicação de modelos. Assim, é esperada uma variação no tamanho dos dados ao longo do fluxo e, consequentemente, uma variação no tempo de execução em relação a uma abordagem estática. A avaliação de nosso protótipo demonstrou que a integração de um sistema multiplataforma a um sistema de codificação e execução visual, como o Lemonade, oferece benefícios para ambas as partes. Para os sistemas multiplataformas, a integração se beneficia de uma interface visual agnóstica e de um sistema de gerenciamento de bases para permitir inferências mais precisas sobre os dados de entrada. Por outro lado, os sistemas visuais se beneficiam ao incluir a capacidade de geração de código mais eficiente, ampliando a funcionalidade oferecida aos usuários. A próxima seção revisita os sistemas multiplataformas discutidos na seção 2.1.3, contextualizando nossas contribuições.

5.1 Contribuições em relação aos sistemas existentes

Como mencionado anteriormente, a construção de sistemas multiplataformas envolve dois eixos principais de desafios: (*i*) como fornecer uma interface na qual usuários possam criar seus fluxos de tarefas; e (*ii*) como o sistema irá avaliar o fluxo e decidir onde cada tarefa deverá ser executada. Quanto ao primeiro desafio, os sistemas podem oferecer suporte às linguagens e sintaxes já existentes em uma ou mais ferramentas ou, alternativamente, adotar uma sintaxe genérica. Sistemas como o Musketeer e o Octopus seguem a primeira estratégia, enquanto o CLIC e o Wayang utilizam a segunda abordagem.

A estratégia de suportar operações e funcionalidades específicas de uma ferramenta em outras, como nos casos de Musketeer e Octopus, busca simplificar a portabilidade de códigos existentes, mas pode introduzir limitações decorrentes de diferenças de design e de implementação entre ferramentas. Na prática, essa tentativa de compatibilidade pode: (a) restringir as conversões de código a subconjuntos de ferramentas, como observado no Musketeer; ou (b) demandar a implementação de funcionalidades inexistentes em algumas ferramentas para alinhar comportamentos, como no Octopus. Por outro lado, a adoção de sintaxes genéricas pode reduzir o leque de operadores disponíveis, como ocorre no Wayang e no CLIC. Por exemplo, nesses sistemas, ao se oferecer um conjunto reduzido de operadores como `map/filter/reduce`, semelhante à ideia de programação por chaves-valores (KV), os códigos gerados podem se afastar da forma como fluxos de Ciência de Dados costumam ser construídos na prática.

Sintaxes desse tipo, embora simplifiquem o desenvolvimento de sistemas multiplataformas, podem apresentar limitações quando aplicadas ao contexto de Ciência de Dados. Em geral, elas não modelam bem relacionamentos e, principalmente, não disponibilizam diretamente operações de alto nível frequentemente utilizadas por usuários, como correlações, remoção de elementos duplicados, normalizações de dados, entre outras. Em abordagens baseadas em KV, por exemplo, o usuário tende a implementar parte dessas tarefas usando apenas os operadores disponíveis, além de gerenciar requisitos de integridade dos dados ao longo do processo.

Por isso, um aspecto fundamental da nossa abordagem é disponibilizar uma abstração de alto nível que, ao mesmo tempo, seja genérica o suficiente para acomodar diferenças entre as ferramentas suportadas. Em nosso protótipo, utilizamos o Lemonade para esse fim, mas outra solução que ofereça uma abstração tabular, como a de DataFrames, também poderia ser utilizada.

Do ponto de vista da modelagem do problema de escalonamento, nossa abordagem possui três contribuições principais: (*i*) estimativas da evolução dos dados ao longo do fluxo, de modo a melhorar as previsões de tempos de execução; (*ii*) suporte a múltiplas configurações de hardware durante a submissão de um fluxo de dados; e (*iii*) capacidade

de evolução contínua, à medida que novos fluxos de dados forem executados.

Como nosso sistema trabalha com dados tabulares e está integrado a um sistema que fornece catálogos de metadados com informações sobre os dados de entrada, nossa capacidade de previsão pode ser favorecida em relação a sistemas que oferecem interfaces mais simples. Em interfaces como DataFrame, que possuem um esquema de dados mais explícito, acompanhar como as operações transformam os dados tende a ser mais direto do que em ferramentas baseadas em KV, onde a estrutura frequentemente permanece implícita dentro do valor.

Atualmente, com virtualização e computação em nuvem, é comum que usuários tenham acesso a recursos computacionais diferentes, e um sistema multiplataforma moderno precisa se adequar a essa realidade. No entanto, sistemas baseados em modelos de custo analíticos, como Wayang, podem ter dificuldades nesse aspecto, dada a complexidade de incorporar características de diferentes hardwares em um conjunto manejável de variáveis. Em comparação com a literatura que revisamos, nossa abordagem considera múltiplas configurações de hardware. Para isso, ela se baseia em uma calibração inicial, a partir de um conjunto de execuções para cada configuração, de modo a aprender o comportamento de cada operação/ferramenta para cada hardware. Uma vez treinado, o conhecimento de cada ambiente é armazenado em conjunto com o de outros ambientes, sem sobrescrever os anteriores.

Por fim, a capacidade de evolução dos modelos de previsão é outro aspecto relevante em um sistema de processamento de dados multiplataforma, especialmente no contexto de Ciência de Dados, no qual se utilizam dados com características diversas em fluxos igualmente variados. Como nosso sistema utiliza modelos de regressão e armazena o histórico de execuções, podemos explorar esse histórico para evolução contínua. Em contrapartida, abordagens baseadas exclusivamente em modelos analíticos tendem a enfrentar barreiras de manutenção e adaptação em cenários reais. Embora a abordagem de regressão utilizada no Octopus também permita um comportamento semelhante ao nosso, por ser um sistema fracamente acoplado ao processo de execução, ela apresenta a limitação de não possuir informações prévias sobre a fonte de dados (o que eleva o custo de criação de uma amostra inicial) e de não armazenar o histórico de execuções para evoluir seus modelos.

5.2 Limitações e problemas não abordados

O problema tratado no presente trabalho envolve uma série de desafios complexos. No entanto, foram estabelecidas algumas delimitações de escopo para restringir o espaço de atuação. A seguir, enumeramos os principais pontos:

1. **Abstração de dados**: como a abordagem se baseia em uma abstração de Data-

Frame, a cobertura das tarefas tende a ficar restrita a um subconjunto de operadores e comportamentos suportados pelas ferramentas-alvo;

2. **Otimizações internas das ferramentas**: atualmente, ferramentas baseadas em avaliação preguiçosa vêm implementando técnicas de otimização relacional. Neste trabalho, não consideramos esses aspectos. Ao representar o fluxo de dados como um grafo, consideramos as dependências lógicas entre operações conforme foram codificadas. Incorporar tais otimizações adicionaria camadas de complexidade, como (*i*) a necessidade de obter o plano de execução final da ferramenta (que, em muitos casos, pode não ser disponibilizado ou só estar disponível no momento da execução) e (*ii*) a necessidade de considerar múltiplas configurações de planos de execução para um mesmo fluxo;

3. **Configurações de hardware**: além de não suportarmos migração de execuções entre diferentes configurações de hardware (como será discutido na próxima seção, em propostas de trabalhos futuros), nossa solução exige um novo treinamento para cada nova configuração de hardware. Uma possível alternativa para esse ponto seria aproveitar modelos de outra configuração próxima como ponto de partida e, ao longo do tempo, evoluí-los. No entanto, esse estudo ainda não foi avaliado.

4. **Paralelismo de tarefas entre ferramentas**: fluxos de dados podem ser compostos por mais de um pipeline de tarefas e, em algumas situações, há independência de execução em pelo menos uma parte do código. Neste trabalho, não exploramos as possibilidades de paralelismo de tarefas entre diferentes ferramentas; por exemplo, cada ferramenta poderia ficar responsável pelo processamento de uma entrada de dados independente. No cenário atual, uma ou mais ferramentas podem compartilhar o mesmo hardware e, com isso, alocar recursos que seriam importantes para as demais. Para viabilizar o paralelismo de tarefas em nosso sistema, uma etapa importante seria oferecer suporte à migração de execução entre diferentes configurações de hardware, permitindo combinar recursos de ambientes distintos em uma única execução.

5. **Dependência de metadados/estatísticas e custo de obtenção**: neste trabalho, utilizamos catálogos/metadados para as previsões. Sistemas de codificação e de execução de tarefas geralmente disponibilizam esses metadados em sua lógica; contudo, a inferência de esquema e outras informações, assim como a computação de estatísticas, podem envolver custos computacionais que não foram considerados aqui, pois assumimos que se trata de uma tarefa que a plataforma na qual o escalonador será integrado já realizaria independentemente da integração.

6. **Intervalo entre recalibrações**: nossa abordagem suporta que, periodicamente, os modelos sejam recalibrados utilizando o histórico de execuções. Atualmente, o inter-

valo de tempo entre recalibrações pode ser parametrizado pelo administrador (por exemplo, via expressão *cron*). No entanto, não foram realizados estudos para determinar qual seria o melhor intervalo de recalibração (p.ex., diário ou mais espaçado). Uma alternativa seria definir a recalibração a partir do coeficiente de ajuste discutido na etapa de recalibração: caso o coeficiente ultrapasse um limiar, isso indicaria degradação na estimativa e, potencialmente, a necessidade de recalibrar os modelos.

5.3 Trabalhos futuros

Os trabalhos futuros podem ser desenvolvidos em diferentes frentes. Em uma perspectiva mais direta, a expansão do suporte a novas operações e ferramentas representa um avanço interessante. Neste estudo, além do Pandas e Spark, ferramentas inicialmente suportadas pelo Lemonade, estendemos a plataforma para incluir suporte ao cuDF, um motor de execução baseado em GPU, visando validar a generalidade da solução. No entanto, outras ferramentas, como Polars, que já está em desenvolvimento no Lemonade, e Dask, cuja API se assemelha à do Pandas, mas com funcionalidades de avaliação preguiçosa, podem ser incorporadas para ampliar ainda mais o escopo da solução.

Do ponto de vista da modelagem do problema, é possível explorar a generalização do modelo de custo para funções definidas pelo usuário. Atualmente, o foco está no desempenho, mas uma abordagem mais abrangente poderia incluir equilíbrio entre desempenho e consumo de recursos, custo monetário de alocação em nuvem ou eficiência energética, dependendo das necessidades do usuário.

No modelo atual, a execução de um fluxo é limitada a uma única configuração de hardware, selecionada como a mais adequada. No entanto, investigar a migração de dados entre hardwares pode representar um avanço significativo. Essa abordagem não apenas pode melhorar o desempenho da solução, mas também reduz a ociosidade de recursos durante a execução. A migração de dados entre hardwares é um tema de crescente interesse da comunidade, especialmente no contexto de computação em nuvem. Entre os desafios dessa área, destacam-se estimativas de tempo de alocação de recursos, transferência de dados e contexto de execução.

Por fim, uma continuidade nos experimentos envolvendo motores LLM pode ser explorada para aprimorar diferentes etapas do processo de recomendação. No presente trabalho, avaliamos a utilização de LLMs como alternativa à modelagem de grafos para seleção de ferramentas, mas outras perspectivas podem ser investigadas. Por exemplo, LLMs poderiam ser utilizados para melhorar a estimativa de características dos dados ao longo do fluxo. Entretanto, um desafio crucial seria avaliar a viabilidade desses modelos em instâncias locais, visando garantir privacidade e segurança, além de minimizar custos e latências relacionadas às requisições.

Referências

ACADEMY, AWS. **US Domestic Flights Delay Prediction (2013 - 2018)**. [S.l.]: Kaggle, 2024. Disponível em: https://www.kaggle.com/dsv/7423794.

AKIDAU, Tyler; BRADSHAW, Robert; CHAMBERS, Craig et al. The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, out-of-Order Data Processing. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 8, p. 1792–1803, ago. 2015.

ALIPOURFARD, Omid; LIU, Hongqiang Harry; CHEN, Jianshu et al. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In: PROCEEDINGS of the 14th USENIX Conference on Networked Systems Design and Implementation. Boston, EUA: USENIX Association, 2017. (NSDI'17), p. 469–482.

ALLEN, Frances E. Control Flow Analysis. In: PROCEEDINGS of a Symposium on Compiler Optimization. Urbana-Champaign, USA: Association for Computing Machinery, 1970. P. 1–19. DOI: 10.1145/800028.808479.

APACHE SOFTWARE FOUNDATION. **Apache Atlas - Data Governance and Metadata framework for Hadoop**. 2015. Disponível em: https://atlas.apache.org. Acesso em: 9 abr. 2025.

APACHE SOFTWARE FOUNDATION. **Apache Beam**. 2023. Disponível em: https://beam.apache.org. Acesso em: 9 abr. 2025.

APACHE SOFTWARE FOUNDATION. **Apache Drill - Schema-free SQL Query Engine for Hadoop, NoSQL and Cloud Storage**. 2023. Disponível em: https://drill.apache.org. Acesso em: 9 abr. 2025.

APACHE SOFTWARE FOUNDATION. **Apache Arrow**. 2024. Disponível em: https://arrow.apache.org. Acesso em: 9 abr. 2025.

ARMBRUST, Michael; DAS, Tathagata; TORRES, Joseph et al. Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark. In: PROCEEDINGS of the 2018 International Conference on Management of Data. Houston, EUA: Association for Computing Machinery, 2018. (SIGMOD'18), p. 601–613.

ARMBRUST, Michael; XIN, Reynold S.; LIAN, Cheng et al. Spark SQL: Relational Data Processing in Spark. In: PROCEEDINGS of the 2015 ACM SIGMOD International

Conference on Management of Data. Melbourne, Austrália: Association for Computing Machinery, 2015. (SIGMOD'15), p. 1383–1394.

BAJABER, Fuad; SAKR, Sherif; BATARFI, Omar et al. Benchmarking big data systems: A survey. **Computer Communications**, v. 149, p. 241–251, jan. 2020.

BALDACCI, Lorenzo; GOLFARELLI, Matteo. A cost model for SPARK SQL. **IEEE Transactions on Knowledge and Data Engineering**, IEEE Computer Society, v. 31, n. 5, p. 819–832, 2019.

BEEDKAR, Kaustubh; CONTRERAS-ROJAS, Bertty; GAVRIILIDIS, Haralampos et al. Apache Wayang: A Unified Data Analytics Framework. **ACM SIGMOD Record**, Association for Computing Machinery, New York, USA, v. 52, n. 3, p. 30–35, nov. 2023.

BEGOLI, Edmon; RODRÍGUEZ, Jesús Camacho; HYDE, Julian et al. Apache Calcite: A Foundational Framework for Optimized Query Processing Over Heterogeneous Data Sources. In: SIGMOD '18: Proceedings of the 2018 International Conference on Management of Data. Houston, EUA: Association for Computing Machinery, 2018. (SIGMOD'18), p. 221–230.

BERTIN-MAHIEUX, Thierry; ELLIS, Daniel P.W.; WHITMAN, Brian et al. The Million Song Dataset. In: PROCEEDINGS OF THE 12TH INTERNATIONAL CONFERENCE ON MUSIC INFORMATION RETRIEVAL (ISMIR 2011). Miami, EUA: University of Miami, 2011. P. 591–596.

BISONG, Ekaba. Kubeflow and Kubeflow Pipelines. In: BUILDING MACHINE LEARNING AND DEEP LEARNING MODELS ON GOOGLE CLOUD PLATFORM: A COMPREHENSIVE GUIDE FOR BEGINNERS. 1. ed. [S.l.]: Apress Berkeley, CA, 2019. P. 671–685.

BOUCHET-VALAT, Milan; KAMIŃSKI, Bogumił. DataFrames.jl: Flexible and Fast Tabular Data in Julia. **Journal of Statistical Software**, Journal of Statistical Software, v. 107, n. 4, p. 1–32, 2023.

BRACCIALE, Lorenzo; BONOLA, Marco; LORETI, Pierpaolo et al. **CRAWDAD roma/taxi**. [S.l.]: IEEE Dataport, 2022. Disponível em: https://dx.doi.org/10.15783/C7QC7M.

BUKHRES, Omran A. et al. InterBase: An Execution Environment for Heterogeneous Software Systems. **Computer**, IEEE Computer Society, Washington, EUA, v. 26, n. 8, p. 57–69, ago. 1993.

CAMACHO-RODRÍGUEZ, Jesús; CHAUHAN, Ashutosh; GATES, Alan et al. Apache Hive: From MapReduce to Enterprise-Grade Big Data Warehousing. In: PROCEE-

DINGS of the 2019 International Conference on Management of Data. Amsterdam, Netherlands: Association for Computing Machinery, 2019. (SIGMOD'19), p. 1773–1786.

CARBONE, Paris; KATSIFODIMOS, Asterios; EWEN, Stephan et al. Apache Flink™: Stream and Batch Processing in a Single Engine. **The Bulletin of the Technical Committee on Data Engineering**, IEEE Computer Society, v. 38, n. 4, p. 28–38, 2015.

CARDOSO, João M.P.; COUTINHO, José Gabriel F.; DINIZ, Pedro C. Chapter 8 - Additional topics. In: CARDOSO, João M.P.; COUTINHO, José Gabriel F.; DINIZ, Pedro C. (Ed.). **Embedded Computing for High Performance**. Boston: Morgan Kaufmann, 2017. P. 255–280.

CAREY, M.J.; HAAS, L.M.; SCHWARZ, P.M. et al. Towards heterogeneous multimedia information systems: the Garlic approach. In: PROCEEDINGS RIDE-DOM'95. Fifth International Workshop on Research Issues in Data Engineering-Distributed Object Management. Taipei, Taiwan: IEEE Computer Society, 1995. P. 124–131.

CHAMBERS, J.; HASTIE, T.; PREGIBON, D. Statistical Models in S. In: PROCEEDINGS in Computational Statistics, 9th Symposium. Dubrovnik, Yugoslavia: Physica-Verlag HD, 1990. P. 317–321.

CHEN, Ching-Wei; LAMERE, Paul; SCHEDL, Markus et al. Recsys Challenge 2018: Automatic Music Playlist Continuation. In: PROCEEDINGS of the 12th ACM Conference on Recommender Systems. Vancouver, Canadá: Association for Computing Machinery, 2018. (RecSys'18), p. 527–528.

CHEN, Qixiang; CHEN, Zhijun; ZHANG, Kai et al. CLIC: An Extensible and Efficient Cross-Platform Data Analytics System. **IEEE Transactions on Parallel and Distributed Systems**, IEEE Computer Society, v. 35, n. 1, p. 34–45, jan. 2023.

CHENG, Guoli; YING, Shi; WANG, Bingming et al. Efficient Performance Prediction for Apache Spark. **Journal of Parallel and Distributed Computing**, Elsevier Inc., v. 149, p. 40–51, 2021.

CHOLLET, François et al. **Keras**. [S.l.]: GitHub, 2015. Disponível em: https://github.com/fchollet/keras. Acesso em: 9 abr. 2025.

CLABURN, Thomas. **Python explosion blamed on pandas**. The Register. 2017. Disponível em: https://t.ly/U9rIr. Acesso em: 9 abr. 2025.

COLEMAN, Jared; KRISHNAMACHARI, Bhaskar. **Comparing Task Graph Scheduling Algorithms: An Adversarial Approach**. [S.l.: s.n.], 2024. arXiv: 2403.07120 [cs.DC]. Disponível em: https://arxiv.org/abs/2403.07120.

COTTON, Richie. **NVIDIA Announces cuDF Pandas Accelerator Mode**. 2025. Disponível em: https://www.datacamp.com/blog/nvidia-announces-cudf-pandas-accelerator-mode. Acesso em: 9 abr. 2025.

CUKIERSKI, Will. **Titanic - Machine Learning from Disaster. Kaggle**. 2012. Disponível em: https://kaggle.com/competitions/titanic. Acesso em: 9 abr. 2025.

DASK DEVELOPMENT TEAM. **Dask: Library for dynamic task scheduling**. 2016. Disponível em: https://dask.org. Acesso em: 9 abr. 2025.

DATAIKU TEAM. **Dataiku - Everyday AI, Extraordinary People**. 2024. Disponível em: https://www.dataiku.com. Acesso em: 9 abr. 2025.

DELIMITROU, Christina; KOZYRAKIS, Christos. Quasar: Resource-Efficient and QoS-Aware Cluster Management. In: ASPLOS '14: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems. Salt Lake City, EUA: Association for Computing Machinery, 2014. P. 127–144.

DEMICHIEL, L.G.; CHAMBERLIN, D.D.; LINDSAY, B.G. et al. Polyglot: extensions to relational databases for sharable types and functions in a multi-language environment. In: PROCEEDINGS of IEEE 9th International Conference on Data Engineering. Vienna, Áustria: IEEE Computer Society, 1993. P. 651–660.

DEVARAKONDA, M.V.; IYER, R.K. Predictability of process resource usage: a measurement-based study on UNIX. **IEEE Transactions on Software Engineering**, IEEE Computer Society, v. 15, n. 12, p. 1579–1586, 1989.

DIAS, Vinicius; MEIRA, Wagner; GUEDES, Dorgival. Janus: Diagnostics and reconfiguration of data parallel programs. **Journal of Parallel and Distributed Computing**, Elsevier Inc., v. 120, p. 196–210, 2018.

DUGGAN, Jennie; ELMORE, Aaron J.; STONEBRAKER, Michael et al. The BigDAWG Polystore System. **ACM SIGMOD Record**, Association for Computing Machinery, v. 44, n. 2, p. 11–16, ago. 2015.

DUGRÉ, Mathieu; HAYOT-SASSON, Valérie; GLATARD, Tristan. A Performance Comparison of Dask and Apache Spark for Data-Intensive Neuroimaging Pipelines. In: 2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS). Denver, EUA: IEEE Computer Society, 2019. P. 40–49.

EDEL, Marcus; SONI, Anand; CURTIN, Ryan R. An automatic benchmarking system. In: NIPS 2014 Workshop on Software Engineering for Machine Learning (SE4ML'2014). Montreal, Canada: [s.n.], 2014. v. 1.

GHOSHAL, Devarshi; WU, Kesheng; POUYOUL, Eric et al. Analysis and Prediction of Data Transfer Throughput for Data-Intensive Workloads. In: 2019 IEEE International

Conference on Big Data. Los Angeles, EUA: IEEE Computer Society, 2019. P. 3648–3657.

GOG, Ionel; SCHWARZKOPF, Malte; CROOKS, Natacha et al. Musketeer: all for one, one for all in data processing systems. In: EUROSYS '15: Proceedings of the Tenth European Conference on Computer Systems. Bordeaux, França: Association for Computing Machinery, 2015. P. 1–16.

GRAF, Sebastian; GLASS, Michael; TEICH, Jürgen et al. Multi-variant-based design space exploration for automotive embedded systems. In: 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE). Dresden, Alemanha: IEEE Computer Society, 2014. P. 1–6.

HALL, Mark; FRANK, Eibe; HOLMES, Geoffrey et al. The WEKA Data Mining Software: An Update. **ACM SIGKDD Explorations Newsletter**, Association for Computing Machinery, New York, EUA, v. 11, n. 1, p. 10–18, nov. 2009.

HERNANDO, Clara Aguirre. **Backstage to the Panama Papers: big data analytics and collaborative journalism**. The London School of Economics e Political Science. 2017. Disponível em: https://t.ly/bgfry. Acesso em: 9 abr. 2025.

HOLZER, M.; KNERR, B.; RUPP, M. Design Space Exploration for Real-Time Reconfigurable Computing. In: 2007 Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers. [S.l.]: IEEE Computer Society, 2007. P. 1981–1985.

HUANG, Shengsheng; HUANG, Jie; DAI, Jinquan et al. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In: 2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010). Long Beach, EUA: IEEE Computer Society, 2010. P. 41–51.

HUANG, Tsung- Wei; LIN, Dian-Lun; LIN, Chun-Xun et al. Taskflow: A Lightweight Parallel and Heterogeneous Task Graph Computing System. **IEEE Transactions on Parallel and Distributed Systems**, IEEE Computer Society, v. 33, n. 6, p. 1303–1320, 2022.

JUSTUS, Daniel; BRENNAN, John; BONNER, Stephen et al. Predicting the Computational Cost of Deep Learning Models. In: 2018 IEEE International Conference on Big Data. Seattle, EUA: IEEE Computer Society, 2018. P. 3873–3882.

KAOUDI, Zoi; QUIANÉ-RUIZ, Jorge-Arnulfo; CONTRERAS-ROJAS, Bertty et al. ML-based Cross-Platform Query Optimization. In: 2020 IEEE 36th International Conference on Data Engineering (ICDE). Dallas, EUA: IEEE Computer Society, 2020. P. 1489–1500.

KARAU, Holden et al. **Spark Advanced Topics: Force Computations**. 2024. Disponível em: https://holdenk.github.io/spark-flowchart/details/forced-computations/. Acesso em: 9 abr. 2025.

KARAU, Holden; KONWINSKI, Andy; WENDELL, Patrick et al. **Learning Spark: Lightning-Fast Big Data Analytics**. 2nd. Sebastopol, EUA: O'Reilly Media, Inc., 2020.

KASSELA, Evdokia; PROVATAS, Nikodimos; KONSTANTINOU, Ioannis et al. General-Purpose vs. Specialized Data Analytics Systems: A Game of ML amp; SQL Thrones. In: 2019 IEEE International Conference on Big Data. Los Angeles, EUA: IEEE Computer Society, 2019. P. 317–326.

KHINE, Pwint Phyu; WANG, Zhaoshun. A Review of Polyglot Persistence in the Big Data World. **Information**, MDPI, v. 10, n. 4, 2019.

KIEHN, Felix et al. Polyglot Data Management: State of the Art & Open Challenges. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 15, n. 12, p. 3750–3753, ago. 2022.

KIM, Bogyeong; KOO, Kyoseung; ENKHBAT, Undraa et al. M2Bench: A Database Benchmark for Multi-Model Analytic Workloads. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 16, n. 4, p. 747–759, dez. 2022.

KRANJC, Janez; PODPEČAN, Vid; LAVRAČ, Nada. ClowdFlows: A Cloud Based Scientific Workflow Platform. In: PROCEEDINGS of the 2012 European Conference on Machine Learning and Knowledge Discovery in Databases. Bristol, Reino Unido: Springer, Berlin, Heidelberg, 2012. P. 816–819.

KROSS, Johannes; KRCMAR, Helmut. Pertract: Model extraction and specification of big data systems for performance prediction by the example of Apache Spark and Hadoop. **Big Data and Cognitive Computing**, MDPI, v. 3, n. 3, p. 1–24, 2019.

KRUSE, Sebastian; KAOUDI, Zoi; QUIANE-RUIZ, Jorge-Arnulfo et al. Optimizing Cross-Platform Data Movement. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE). Macao, China: IEEE Computer Society, 2019. P. 1642–1645.

KWOK, Yu-Kwong; AHMAD, I. Benchmarking the task graph scheduling algorithms. In: PROCEEDINGS of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing. Orlando, EUUA: IEEE Computer Society, 1998. P. 531–537.

LATTUADA, Marco; GIANNITI, Eugenio; HOSSEINI, Marjan et al. Gray-Box Models for Performance Assessment of Spark Applications. In: PROCEEDINGS of the 9th

International Conference on Cloud Computing and Services Science. Heraklion, Grécia: Science e Technology Publications, Lda, 2019. (CLOSER 2019), p. 609–618.

LUCAS, Ji; IDRIS, Yasser; CONTRERAS-ROJAS, Bertty et al. RheemStudio: Cross-Platform Data Analytics Made Easy. In: 2018 IEEE 34th International Conference on Data Engineering. Paris, França: IEEE Computer Society, 2018. P. 1573–1576.

MAHAPATRA, Tanmaya et al. Graphical Spark Programming in IoT Mashup Tools. In: 2018 Fifth International Conference on Internet of Things: Systems, Management and Security. Valencia, Espanha: IEEE Computer Society, 2018. P. 163–170.

MARCUS, Ryan; PAPAEMMANOUIL, Olga. Plan-structured deep neural network models for query performance prediction. **Proceedings of the VLDB Endowment**, Association for Computing Machinery, v. 12, n. 11, p. 1733–1746, jul. 2019.

MAROS, Alexandre; MURAI, Fabricio; COUTO DA SILVA, Ana Paula et al. Machine Learning for Performance Prediction of Spark Cloud Applications. In: 2019 IEEE 12th International Conference on Cloud Computing. Milan, Itália: IEEE Computer Society, 2019. P. 99–106.

MATSUNAGA, Andréa; FORTES, José A.B. On the Use of Machine Learning to Predict the Time and Resources Consumed by Applications. In: 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing. Melbourne, Austrália: IEEE Computer Society, 2010. P. 495–504.

MCKINNEY, Wes. Data Structures for Statistical Computing in Python. In: PROCEEDINGS of the 9th Python in Science Conference. Austin, EUA: SciPy Conferences, 2010. P. 56–61.

MCKINNEY, Wes. **Apache Arrow and the "10 Things I Hate About pandas"**. 2017. Disponível em: https://t.ly/8NN08. Acesso em: 9 abr. 2025.

MCKINNEY, Wes. **Python for Data Analysis**. 3. ed. Sebastopol, EUA: O'Reilly Media, Inc., 2022.

MENG, Xiangrui; BRADLEY, Joseph; YAVUZ, Burak et al. MLlib: Machine Learning in Apache Spark. **Journal of Machine Learning Research**, JMLR.org, v. 17, n. 34, p. 1–7, 2016. Disponível em: http://jmlr.org/papers/v17/15-237.html. Acesso em: 9 abr. 2025.

MUND, Sumit. **Microsoft Azure Machine Learning**. 1. ed. Birmingham, Reino Unido: Packt Publishing, 2015. v. 1.

NAMBIAR, Raghunath Othayoth; POESS, Meikel. The Making of TPC-DS. In: PROCEEDINGS of the 32nd International Conference on Very Large Data Bases. Seul, Coreia do Sul: VLDB Endowment, 2006. (VLDB'06), p. 1049–1058.

NVIDIA RAPIDS. **cuDF pandas Accelerator Mode**. 2025. Disponível em: https://docs.rapids.ai/api/cudf/stable/cudf_pandas/how-it-works/. Acesso em: 9 abr. 2025.

NVIDIA RAPIDS. **cuDF Pandas Compatibility Notes**. 2025. Disponível em: https://docs.rapids.ai/api/cudf/stable/user_guide/pandascompat/. Acesso em: 9 abr. 2025.

NVIDIA RAPIDS. **cuML on GPU and CPU**. 2025. Disponível em: https://docs.rapids.ai/api/cuml/stable/execution_device_interoperability/. Acesso em: 9 abr. 2025.

NYSTROM, Nathaniel; CLARKSON, Michael R.; MYERS, Andrew C. "Polyglot: An Extensible Compiler Framework for Java". In: COMPILER Construction, 12th International Conference, CC 2003, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings. [S.l.]: Springer, 2003. (Lecture Notes in Computer Science), p. 138–152.

OPENMETADATA DEVELOPMENT TEAM. **OpenMetadata**. 2024. Disponível em: https://open-metadata.org. Acesso em: 9 abr. 2025.

PANDAS DEVELOPMENT TEAM. **Release notes**. 2024. Disponível em: https://t.ly/5-jMm. Acesso em: 9 abr. 2025.

PANDAS DEVELOPMENT TEAM. **What kind of data does pandas handle?** 2024. Disponível em: https://t.ly/Q6vmI. Acesso em: 9 abr. 2025.

PEDREGOSA, Fabian; VAROQUAUX, Gaël; GRAMFORT, Alexandre et al. Scikit-learn: Machine Learning in Python. **Journal of Machine Learning Research**, JMLR.org, v. 12, p. 2825–2830, 2011. Disponível em: http://jmlr.org/papers/v12/pedregosa11a.html.

PETERSOHN, Devin; MACKE, Stephen; XIN, Doris et al. Towards Scalable Dataframe Systems. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 13, n. 12, p. 2033–2046, jul. 2020.

PFEIFFER, Wayne; WRIGHT, Nicholas J. Modeling and predicting application performance on parallel computers using HPC challenge benchmarks. In: 2008 IEEE International Symposium on Parallel and Distributed Processing. Miami, EUA: IEEE Computer Society, 2008. P. 1–12.

PODPEČAN, Vid; ZEMENOVA, Monika; LAVRAČ, Nada. Orange4WS environment for service-oriented data mining. **The Computer Journal**, Oxford University Press, v. 55, n. 1, p. 82–98, 2011.

PONCE, Lucas. **Extensão de um ambiente de computação de alto desempenho para o processamento de dados massivos**. Jun. 2018. Dissertação de Mestrado – Universidade Federal de Minas Gerais. Disponível em: http://hdl.handle.net/1843/ESBF-B6CGGA.

PONCE, Lucas M.; LEZZI, Daniele; BADIA, Rosa M. et al. DDF Library: Enabling functional programming in a task-based model. **Journal of Parallel and Distributed Computing**, Elsevier Inc., v. 151, p. 112–124, 2021.

PREKOPCSAK, Zoltan; MAKRAI, Gabor; HENK, Tamas et al. Radoop: Analyzing big data with rapidminer and hadoop. In: PROCEEDINGS of the 2nd RapidMiner Community Meeting and Conference (RCOMM). Dublin, Irlanda: Shaker Verlag, 2011. P. 1–12.

ROBINSON, David. **Why is Python Growing So Quickly?** Stack Overflow. 2017. Disponível em: https://t.ly/yvxWH. Acesso em: 9 abr. 2025.

RONG, Gu; SHI, Jun; CHEN, Xiaofei et al. Octopus-DF: Unified DataFrame-based cross-platform data analytic system. **Parallel Computing**, Elsevier Inc., v. 110, 2022.

RULE, Adam; TABARD, Aurélien; HOLLAN, James D. Exploration and Explanation in Computational Notebooks. In: (CHI'18), p. 1–12.

SADALAGE, P.J.; FOWLER, M. **NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence**. EUA: Pearson Education, Inc, 2012.

SANTOS, Walter dos; CARVALHO, Luiz F. M.; AVELAR, Gustavo de P. et al. Lemonade: A Scalable and Efficient Spark-Based Platform for Data Analytics. In: 17TH IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. Madrid, Espanha: IEEE Computer Society, 2017. P. 745–748.

SARITHA, Bethineni; BONAGIRI, Rajitha; DEEPIKA, Rangaraju. Open source technologies in data science and big data analytics. **Materials Today: Proceedings**, Elsevier Inc., 2021. ISSN 2214–7853.

SCHRANK, Peter. **A brand new game: As people spend more time on social media, advertisers are following them**. The Economist. 2015. Disponível em: https://econ.st/3NFn4jK. Acesso em: 9 abr. 2025.

SCIKIT-LEARN DEVELOPERS. **GradientBoostingRegressor**. 2024. Disponível em: https://t.ly/waY7K. Acesso em: 9 abr. 2025.

SCIKIT-LEARN DEVELOPERS. **GridSearchCV**. 2024. Disponível em: https://t.ly/9uEP-. Acesso em: 9 abr. 2025.

SCIKIT-LEARN DEVELOPERS. **Linear Models**. 2024. Disponível em: https://t.ly/q3tps. Acesso em: 9 abr. 2025.

SELINGER, David. **Why big data means big business for online retailers**. The Guardian. 2012. Disponível em: https://t.ly/V4xAu. Acesso em: 9 abr. 2025.

SHAPI, Mel Keytingan M.; RAMLI, Nor Azuana; AWALIN, Lilik J. Energy consumption prediction by using machine learning for smart building: Case study in Malaysia. **Developments in the Built Environment**, Elsevier Inc., v. 5, 2021.

SHEN, Chao; TONG, Weiqin; CHOO, Kim-Kwang Raymond et al. Performance prediction of parallel computing models to analyze cloud-based big data applications. **Cluster Computing**, Springer US, v. 21, n. 2, p. 1439–1454, 2018.

STANČIN, I.; JOVIĆ, A. An overview and comparison of free Python libraries for data mining and big data analysis. In: 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). Opatija, Croácia: IEEE Computer Society, 2019. P. 977–982.

TEJEDOR, Enric; BECERRA, Yolanda; ALOMAR, Guillem et al. PyCOMPSs: Parallel computational workflows in Python. **The International Journal of High Performance Computing Applications**, SAGE Publications, v. 31, n. 1, p. 66–82, 2017.

THE FUGUE DEVELOPMENT TEAM. **Welcome to the Fugue Tutorials!** 2022. Disponível em: https://t.ly/_1E1t. Acesso em: 9 abr. 2025.

THE FUGUE DEVELOPMENT TEAM. **Fugue API in 10 minutes**. 2023. Disponível em: https://t.ly/QGPep. Acesso em: 9 abr. 2025.

THE POLARS DEVELOPMENT TEAM. **Polars - DataFrames for the new era**. 2023. Disponível em: https://www.pola.rs. Acesso em: 9 abr. 2025.

THE R FOUNDATION. **The R Project for Statistical Computing**. 2024. Disponível em: https://www.R-project.org/. Acesso em: 9 abr. 2025.

TOTONI, Ehsan et al. HiFrames: High Performance Data Frames in a Scripting Language. **CoRR**, abs/1704.02341, 2017. arXiv: 1704.02341.

TRANSACTION PROCESSING PERFORMANCE COUNCIL. **TPC-H**. 2024. Disponível em: https://www.tpc.org/tpch. Acesso em: 9 abr. 2025.

TRANSACTION PROCESSING PERFORMANCE COUNCIL. **TPCx-BB**. 2024. Disponível em: https://www.tpc.org/tpcx-bb/. Acesso em: 9 abr. 2025.

VENKATARAMAN, Shivaram; YANG, Zongheng; FRANKLIN, Michael et al. Ernest: Efficient performance prediction for large-scale advanced analytics. **Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation**, USENIX Association, Santa Clara, EUA, p. 363–378, 2016.

VIANNA, Emanuel; COMARELA, Giovanni; PONTES, Tatiana et al. Analytical Performance Models for MapReduce Workloads. **International Journal of Parallel Programming**, Springer, Berlin, Heidelberg, v. 41, n. 4, p. 495–525, 2013.

WATSON, Alex; BABU, Deepigha Shree Vittal; RAY, Suprio. Sanzu: A Data Science Benchmark. In: 2017 IEEE International Conference on Big Data (Big Data). Boston, EUA: IEEE Computer Society, 2017. P. 263–272.

WHITE, Tom. **Hadoop: The Definitive Guide**. 4ª. Sebastopol, EUA: O'Reilly Media, Inc., 2015.

XGBOOST DEVELOPERS. **XGBRegressor**. 2022. Disponível em: https://t.ly/u5Fc2. Acesso em: 9 abr. 2025.

YANG, Fangjin; TSCHETTER, Eric; LÉAUTÉ, Xavier et al. Druid: A Real-Time Analytical Data Store. In: PROCEEDINGS of the 2014 ACM SIGMOD International Conference on Management of Data. Snowbird, EUA: Association for Computing Machinery, 2014. (SIGMOD'14), p. 157–168.

YU, Xiang; CHAI, Chengliang; LI, Guoliang et al. Cost-based or learning-based? A hybrid query optimizer for query plan selection. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 15, n. 13, p. 3924–3936, 2022.

YU, Yuan; ISARD, Michael; FETTERLY, Dennis et al. DryadLINQ: a system for general-purpose distributed data-parallel computing using a high-level language. In: PROCEEDINGS of the 8th USENIX Conference on Operating Systems Design and Implementation. San Diego, EUA: USENIX Association, 2008. (OSDI'08), p. 1–14.

ZHAO, Bo; ZHOU, Hucheng; LI, Guoqiang et al. ZenLDA: An Efficient and Scalable Topic Model Training System on Distributed Data-Parallel Platform, 2015. arXiv: 1511.00440 [cs.DC].

ZHAO, Hang; RAO, Yu; LI, Donghua et al. A DAG Refactor Based Automatic Execution Optimization Mechanism for Spark. In: IFIP International Conference on Network and Parallel Computing. Hohhot, China: Springer, Cham, 2019. P. 338–344.

ZHENG, Xinnian; RAVIKUMAR, Pradeep; JOHN, Lizy K. et al. Learning-based analytical cross-platform performance prediction. In: 2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation. Samos, Grécia: IEEE Computer Society, 2015. P. 52–59.

ZHOU, Xuanhe; SUN, Ji; LI, Guoliang et al. Query Performance Prediction for Concurrent Queries Using Graph Embedding. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 13, n. 9, p. 1416–1428, mai. 2020.

Apêndice A

Fluxos de Dados

No capítulo 4, descrevemos o planejamento dos experimentos, apresentando os fluxos e os conjuntos de dados utilizados nos testes (seção 4.2), bem como os resultados experimentais obtidos (seção 4.4). O objetivo deste apêndice é complementar a apresentação dos fluxos, além de fornecer os resultados completos das métricas avaliadas em cada um dos cenários.

As seções deste apêndice são organizadas por fluxo de trabalho, contendo, além da descrição textual do fluxo, sua representação visual criada no Lemonade e uma tabela com os resultados das métricas coletadas. As tabelas A.1 a A.15 resumem esses resultados a partir da média de, no mínimo, três execuções para cada configuração. Nelas, são apresentados os tempos de execução considerando quatro cenários: apenas com a ferramenta Pandas (P), apenas com Spark (S), apenas com cuDF (C), ou utilizando nossa abordagem, que pode empregar múltiplas ferramentas (M).

Adicionalmente, são incluídas as métricas de speedup da configuração M em relação a cada uma das ferramentas individuais. Valores ausentes (representados por "–") indicam configurações nas quais a execução não foi concluída, seja por exceder o tempo limite de uma hora, seja por insuficiência de memória.

As figuras, além de ilustrar graficamente os fluxos, apresentam também um exemplo de configuração recomendada de ferramentas para o fluxo em questão, conforme a configuração de hardware utilizada em cada exemplo. Entre as figuras, a codificação de cores permanece constante: operações representadas em verde são execuções no Spark; em amarelo, no cuDF; e em azul, no Pandas.

A.1 Titanic - Fluxo de dados 1

No fluxo de dados 1 (figura A.1), é utilizada a base de dados do Titanic para criar um classificador capaz de prever a sobrevivência dos passageiros a partir de campos como idade, valor do ticket e número de parentes embarcados. Após aplicar a previsão do classificador, o resultado é salvo em um arquivo no HDFS ou sistema de arquivo convencional (a depender de ser a configuração em YARN ou não) contendo a coluna da previsão e o valor real. Na figura, operações em verde representam execuções em Spark. Os resultados completos das avaliações são apresentados na tabela A.1.

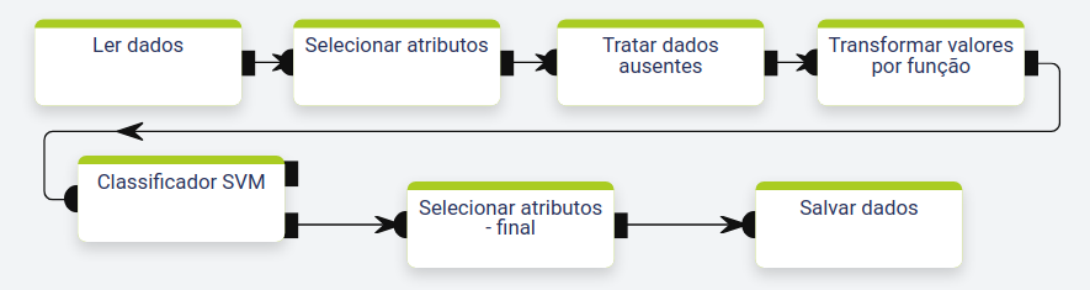


Figura A.1: Fluxo de dados 1. Exemplo de recomendação para o hardware Local[16] utilizando a carga de dados 5 no cenário PS.

Tabela A.1: Resultados dos experimentos do fluxo de dados 1.

Hardware	Métrica		Carga				
			1	2	3	4	5
Local[16]	Tempo (s)	Pandas (P)	78,0	166,7	361,0	1437,4	2834,4
		Spark (S)	46,4	74,7	130,0	468,0	930,7
		cuDF (C)	10,0	14,4	20,7	–	–
		M (Cen. PS)	46,4	74,7	130,0	468,0	930,7
		M (Cen. CPS)	10,0	14,4	20,7	150,7	297,4
	PS	Spd. S / M	1,0	1,0	1,0	1,0	1,0
		Spd. P / M	1,7	2,3	2,8	3,1	3,1
		Decisão / Acerto	S / ✓	S / ✓	S / ✓	S / ✓	S / ✓
	CPS	Spd. S / M	4,7	5,3	6,3	3,2	3,2
		Spd. P / M	7,8	11,7	17,5	9,6	9,6
		Spd. C / M	1,0	1,0	1,0	–	–
		Decisão / Acerto	C / ✓	C / ✓	C / ✓	M / ✓	M / ✓
Local[30]	Tempo (s)	Pandas (P)	80,9	170,5	365,2	1428,5	2843,9
		Spark (S)	47,0	75,4	132,4	434,4	929,4
		cuDF (C)	13,9	19,0	25,2	–	–
		M (Cen. PS)	47,0	75,4	132,0	465,4	938,0
		M (Cen. CPS)	13,9	19,0	25,2	130,7	281,7
	PS	Spd. S / M	1,0	1,0	1,0	1,0	1,0
		Spd. P / M	0,8	0,9	1,0	1,0	1,1
		Decisão / Acerto	S / ✓	S / ✓	S / ✓	S / ✓	S / ✓
	CPS	Spd. S / M	3,4	4,0	5,3	3,4	3,3
		Spd. P / M	5,9	9,0	14,6	11,0	10,1
		Spd. C / M	1,0	1,0	1,0	–	–
		Decisão / Acerto	C / ✓	C / ✓	C / ✓	M / ✓	M / ✓
YARN	Tempo (s)	Pandas (P)	83,7	174,4	369,4	1419,7	2853,4
		Spark (S)	60,0	87,7	134,4	454,0	882,4
		cuDF (C)	17,7	23,7	29,7	–	–
		M (Cen. PS)	60,0	87,7	134,4	454,0	882,4
		M (Cen. CPS)	17,7	23,7	29,7	155,4	332,0
	PS	Spd. S / M	1,0	1,0	1,0	1,0	1,0
		Spd. P / M	1,4	2,0	2,8	3,2	3,3
		Decisão / Acerto	S / ✓	S / ✓	S / ✓	S / ✓	S / ✓
	CPS	Spd. S / M	3,4	3,7	4,6	3,0	2,7
		Spd. P / M	4,8	7,4	12,5	9,2	8,6
		Spd. C / M	1,0	1,0	1,0	–	–
		Decisão / Acerto	C / ✓	C / ✓	C / ✓	M / ✓	M / ✓

A.2 Titanic – Fluxo de dados 2

O objetivo deste fluxo de dados (figura A.2) é comparar a quantidade de homens sobreviventes em relação ao grupo formado por mulheres e crianças na tragédia do Titanic. Para isso, são realizadas filtragens específicas a fim de separar cada grupo: crianças, mulheres adultas e homens adultos. Em seguida, é computado o número de registros de cada grupo, com base no estado de sobrevivência. Os valores correspondentes às mulheres e às crianças são posteriormente unificados por meio da mesclagem das respectivas tabelas, permitindo a obtenção de um resultado consolidado para esse grupo. Na figura, as operações representadas em azul representam execuções em Pandas. Os resultados

completos das avaliações estão apresentados na tabela A.2.

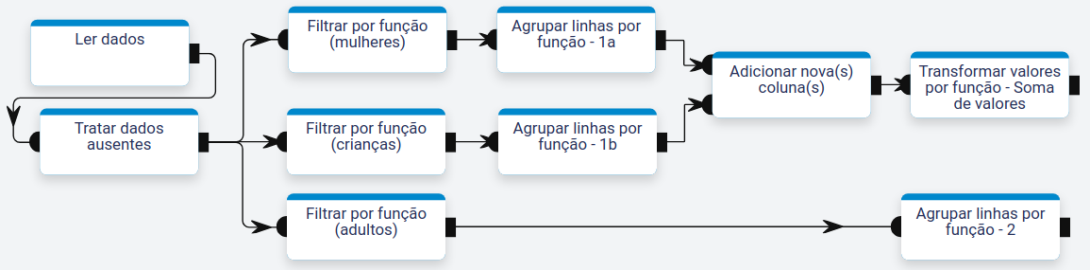


Figura A.2: Fluxo de dados 2. Exemplo de recomendação para o hardware Local[30] utilizando a carga de dados 4 no cenário PS.

Tabela A.2: Resultados dos experimentos do fluxo de dados 2.

Hardware	Métrica		Carga 1	2	3	4	5
Local[16]	Tempo (s)	Pandas (P)	14,0	21,0	35,7	116,0	257,0
		Spark (S)	18,0	25,4	39,4	127,0	255,7
		cuDF (C)	7,0	7,4	7,7	–	–
		M (Cen. PS)	18,0	25,4	39,4	127,0	255,7
		M (Cen. CPS)	7,0	7,4	7,7	127,0	255,7
	PS	Spd. S / M	1,0	1,0	1,0	1,0	1,0
		Spd. P / M	0,8	0,9	1,0	1,0	1,1
		Decisão / Acerto	S / ✓	S / ✓	S / ✓	S / ✗	S / ✓
	CPS	Spd. S / M	2,6	3,5	5,2	1,0	1,0
		Spd. P / M	2,0	2,9	4,7	1,0	1,1
		Spd. C / M	1,0	1,0	1,0	–	–
		Decisão / Acerto	C / ✓	C / ✓	C / ✓	S / ✗	S / ✓
Local[30]	Tempo (s)	Pandas (P)	13,4	20,0	33,4	116,7	271,4
		Spark (S)	17,7	25,0	39,7	125,0	257,0
		cuDF (C)	7,0	7,0	7,4	–	–
		M (Cen. PS)	13,4	20,0	33,4	116,7	271,4
		M (Cen. CPS)	7,0	7,0	7,4	116,7	271,4
	PS	Spd. S / M	1,4	1,3	1,2	1,1	1,0
		Spd. P / M	1,0	1,0	1,0	1,0	1,0
		Decisão / Acerto	P / ✓	P / ✓	P / ✓	P / ✓	P / ✗
	CPS	Spd. S / M	2,6	3,6	5,5	1,1	1,0
		Spd. P / M	1,9	2,9	4,6	1,0	1,0
		Spd. C / M	1,0	1,0	1,0	–	–
		Decisão / Acerto	C / ✓	C / ✓	C / ✓	P / ✓	P / ✗
YARN	Tempo (s)	Pandas (P)	21,0	28,4	44,4	136,4	288,7
		Spark (S)	26,0	34,0	48,7	135,7	249,7
		cuDF (C)	14,7	16,4	18,0	–	–
		M (Cen. PS)	21,0	28,4	48,7	135,7	249,7
		M (Cen. CPS)	14,7	16,4	18,0	135,7	249,7
	PS	Spd. S / M	1,3	1,2	1,0	1,0	1,0
		Spd. P / M	1,0	1,0	1,0	1,0	1,2
		Decisão / Acerto	P / ✓	P / ✓	S / ✓	S / ✓	S / ✓
	CPS	Spd. S / M	1,8	2,1	2,7	1,0	1,0
		Spd. P / M	1,5	1,8	2,5	1,0	1,2
		Spd. C / M	1,0	1,0	1,0	–	–
		Decisão / Acerto	C / ✓	C / ✓	C / ✓	S / ✓	S / ✓

A.3 Titanic – Fluxo de dados 3

A ideia desse fluxo de dados (figura A.3) é gerar uma lista ordenada da média de idades dos passageiros por classe social. Para isso, é feita uma seleção das colunas de interesse e uma remoção de registros ausentes. Posteriormente, a média das idades por classe social do passageiro é calculada a partir de um agrupamento por função, onde depois os dados são ordenados de forma decrescente. Na figura, operações em verde representam execuções em Spark. Os resultados das avaliações são apresentados na tabela A.3.

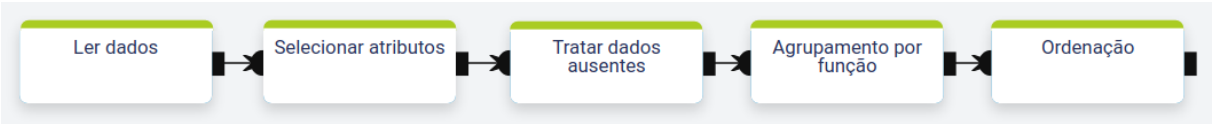


Figura A.3: Fluxo de dados 3. Exemplo de recomendação para o hardware Local[16] utilizando a carga de dados 4 no cenário PS.

Tabela A.3: Resultados dos experimentos do fluxo de dados 3.

Hardware	Métrica		Carga				
			1	2	3	4	5
Local[16]	Tempo (s)	Pandas (P)	12,7	19,7	32,0	111,0	230,4
		Spark (S)	14,0	19,0	28,0	83,0	157,0
		cuDF (C)	6,4	7,0	7,0	–	–
		M (Cen. PS)	14,0	19,0	28,0	83,0	157,0
		M (Cen. CPS)	6,4	7,0	7,0	83,0	157,0
	PS	Spd. S / M	1,0	1,0	1,0	1,0	1,0
		Spd. P / M	0,9	1,1	1,2	1,4	1,5
		Decisão / Acerto	S / ✓	S / ✓	S / ✓	S / ✓	S / ✓
	CPS	Spd. S / M	2,3	2,8	4,0	1,0	1,0
		Spd. P / M	2,0	2,9	4,6	1,4	1,5
		Spd. C / M	1,0	1,0	1,0	–	–
		Decisão / Acerto	C / ✓	C / ✓	C / ✓	S / ✓	S / ✓
Local[30]	Tempo (s)	Pandas (P)	12,4	18,7	31,4	104,4	201,0
		Spark (S)	15,0	21,0	32,7	103,0	195,0
		cuDF (C)	6,0	6,4	7,0	–	–
		M (Cen. PS)	15,0	21,0	35,7	–	195,0
		M (Cen. CPS)	6,0	6,4	7,0	–	195,0
	PS	Spd. S / M	1,0	1,0	1,0	–	1,0
		Spd. P / M	0,9	0,9	0,9	–	1,1
		Decisão / Acerto	S / ✓	S / ✓	M / ✓	M / ✓	S / ✓
	CPS	Spd. S / M	2,5	3,4	4,7	–	1,0
		Spd. P / M	2,1	3,0	4,5	–	1,1
		Spd. C / M	1,0	1,0	1,0	–	–
		Decisão / Acerto	C / ✓	C / ✓	C / ✓	M / ✓	S / ✓
YARN	Tempo (s)	Pandas (P)	20,0	27,0	42,0	125,0	241,4
		Spark (S)	23,7	28,7	39,7	109,4	192,0
		cuDF (C)	14,0	15,7	18,0	–	–
		M (Cen. PS)	23,7	28,7	39,7	109,4	192,0
		M (Cen. CPS)	14,0	15,7	18,0	109,4	192,0
	PS	Spd. S / M	1,0	1,0	1,0	1,0	1,0
		Spd. P / M	0,9	1,0	1,1	1,2	1,3
		Decisão / Acerto	S / ✓	S / ✓	S / ✓	S / ✓	S / ✓
	CPS	Spd. S / M	1,7	1,9	2,2	1,0	1,0
		Spd. P / M	1,5	1,8	2,4	1,2	1,3
		Spd. C / M	1,0	1,0	1,0	–	–
		Decisão / Acerto	C / ✓	C / ✓	C / ✓	S / ✓	S / ✓

A.4 TCP-H – Fluxo de dados 4

O fluxo de dados 4 (figura A.4) corresponde à adaptação da primeira consulta do *benchmark* TCP-H. A consulta gera um resumo de relatório de preços para todos os itens de linha faturados e enviados até uma determinada data. A data, definida pelo *benchmark*, corresponde ao dia de envio contido no banco de dados. Para essa consulta, é necessário utilizar apenas o arquivo *lineitem*. Após a filtragem de registros pela data limite, é realizada a criação de colunas para a computação do preço final, a partir de colunas como valor, desconto e os impostos. Após isso, os resultados são agrupados por status e categorias de vendas. O resultado final é ordenado e, em seguida, salvo em disco. Na figura, operações

em verde representam execuções em Spark. Os resultados completos das avaliações são apresentados na tabela A.4.

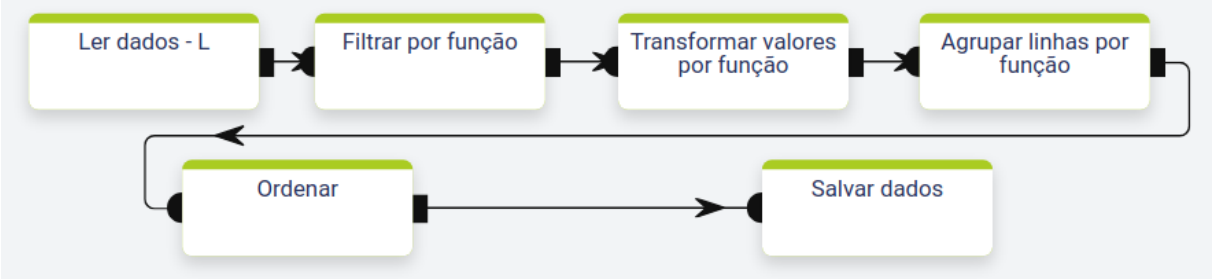


Figura A.4: Fluxo de dados 4. Exemplo de recomendação para o hardware Local[16] utilizando a carga de dados 3 no cenário PS.

Tabela A.4: Resultados dos experimentos do fluxo de dados 4.

Hardware	Métrica		Carga		
			1	2	3
Local[16]	Tempo (s)	Pandas (P)	75,7	358,7	766,0
		Spark (S)	27,0	90,7	190,7
		cuDF (C)	7,7	9,7	–
		M (Cen. PS)	27,0	90,7	190,7
		M (Cen. CPS)	7,7	9,7	190,7
	PS	Spd. S / M	1,0	1,0	1,0
		Spd. P / M	2,8	4,0	4,1
		Decisão / Acerto	S / ✓	S / ✓	S / ✓
	CPS	Spd. S / M	3,6	9,4	1,0
		Spd. P / M	9,9	37,1	4,1
		Spd. C / M	1,0	1,0	–
		Decisão / Acerto	C / ✓	C / ✓	S / ✓
Local[30]	Tempo (s)	Pandas (P)	77,7	364,0	770,4
		Spark (S)	27,0	90,4	189,4
		cuDF (C)	10,0	10,0	–
		M (Cen. PS)	27,0	90,4	189,4
		M (Cen. CPS)	10,0	10,0	189,4
	PS	Spd. S / M	1,0	1,0	1,0
		Spd. P / M	2,9	4,1	4,1
		Decisão / Acerto	S / ✓	S / ✓	S / ✓
	CPS	Spd. S / M	2,7	9,1	1,0
		Spd. P / M	7,8	36,4	4,1
		Spd. C / M	1,0	1,0	–
		Decisão / Acerto	C / ✓	C / ✓	S / ✓
YARN	Tempo (s)	Pandas (P)	121,7	545,0	1720,7
		Spark (S)	39,4	119,4	220,4
		cuDF (C)	18,7	44,0	–
		M (Cen. PS)	39,4	119,4	220,4
		M (Cen. CPS)	18,7	44,0	220,4
	PS	Spd. S / M	1,0	1,0	1,0
		Spd. P / M	3,1	4,6	7,9
		Decisão / Acerto	S / ✓	S / ✓	S / ✓
	CPS	Spd. S / M	2,2	2,8	1,0
		Spd. P / M	6,6	12,4	7,9
		Spd. C / M	1,0	1,0	–
		Decisão / Acerto	C / ✓	C / ✓	S / ✓

A.5 TCP-H – Fluxo de dados 5

O fluxo de dados 5 (figura A.5) corresponde à terceira consulta do *benchmark* TCP-H. A consulta recupera a prioridade de envio e a receita potencial, definida por uma expressão baseada no preço e no desconto, dos pedidos com maior receita entre aqueles que não foram enviados até uma determinada data. Os pedidos são listados em ordem decrescente de receita. Se existirem mais de 10 pedidos não enviados, apenas os 10 pedidos com maior receita serão listados. Para isso, são necessários três tipos de arquivos: *customer* (C), *orders* (O) e o *lineitem* (L). Na figura, as operações representadas em azul indicam execução com a ferramenta Pandas, enquanto aquelas em verde foram recomendadas

para execução com o Spark. Os resultados completos das avaliações são apresentados na tabela A.5.

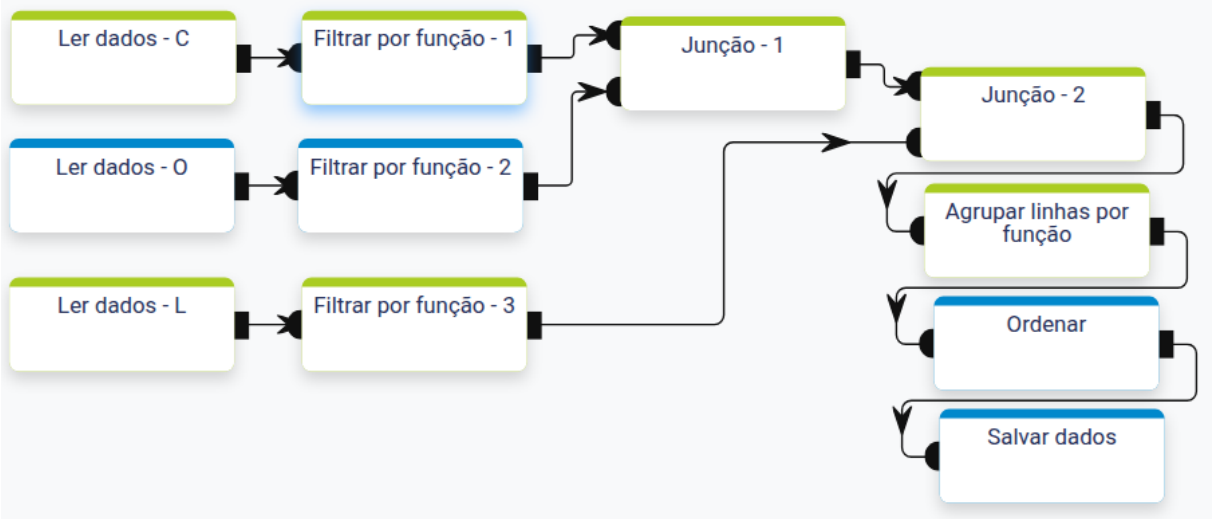


Figura A.5: Fluxo de dados 5. Exemplo de recomendação para o Hardware Local[16] utilizando a carga de dados 1 no cenário PS.

Tabela A.5: Resultados dos experimentos do fluxo de dados 5.

Hardware	Métrica		Carga		
			1	2	3
Local[16]	Tempo (s)	Pandas (P)	28,0	88,7	163,0
		Spark (S)	22,4	63,4	114,0
		cuDF (C)	8,7	9,0	–
		M (Cen. PS)	23,7	63,4	114,0
		M (Cen. CPS)	8,7	9,0	114,0
	PS	Spd. S / M	1,0	1,0	1,0
		Spd. P / M	1,2	1,4	1,5
		Decisão / Acerto	M / ✓	S / ✓	S / ✓
	CPS	Spd. S / M	2,6	7,1	1,0
		Spd. P / M	3,3	9,9	1,5
		Spd. C / M	1,0	1,0	–
		Decisão / Acerto	C / ✓	C / ✓	S / ✓
Local[30]	Tempo (s)	Pandas (P)	26,4	85,7	161,4
		Spark (S)	23,0	64,0	114,0
		cuDF (C)	7,0	9,0	–
		M (Cen. PS)	23,0	64,0	114,0
		M (Cen. CPS)	7,0	9,0	114,0
	PS	Spd. S / M	1,0	1,0	1,0
		Spd. P / M	1,2	1,4	1,5
		Decisão / Acerto	S / ✓	S / ✓	S / ✓
	CPS	Spd. S / M	3,3	7,2	1,0
		Spd. P / M	3,8	9,6	1,5
		Spd. C / M	1,0	1,0	–
		Decisão / Acerto	C / ✓	C / ✓	S / ✓
YARN	Tempo (s)	Pandas (P)	66,0	155,7	313,7
		Spark (S)	30,7	73,0	131,4
		cuDF (C)	19,0	29,4	–
		M (Cen. PS)	31,0	73,0	131,4
		M (Cen. CPS)	19,0	29,4	131,4
	PS	Spd. S / M	1,0	1,0	1,0
		Spd. P / M	2,2	2,2	2,4
		Decisão / Acerto	M / ✓	S / ✓	S / ✓
	CPS	Spd. S / M	1,7	2,5	1,0
		Spd. P / M	3,5	5,4	2,4
		Spd. C / M	1,0	1,0	–
		Decisão / Acerto	C / ✓	C / ✓	S / ✓

A.6 TCP-H – Fluxo de dados 6

O fluxo 6 (figura A.6) é uma nova consulta utilizando o arquivo *lineitem*. A ideia da consulta é gerar uma tabela com o preço médio das vendas de um dia sobre produtos até uma faixa de preço. Para isso, são inicialmente filtradas as vendas que satisfazem a condição de preço e, posteriormente, seu valor médio é contabilizado a partir do dia. Por fim, o resultado é ordenado por dia e, posteriormente, salvo. Na figura, operações em verde representam execuções em Spark. Os resultados completos das avaliações são apresentados na tabela A.6.

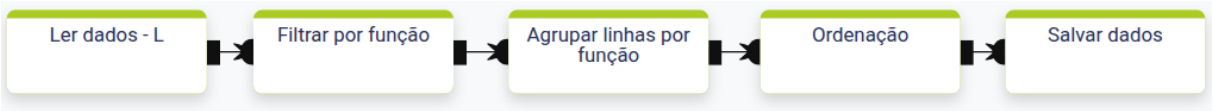


Figura A.6: Fluxo de dados 6. Exemplo de recomendação para o hardware Local[16] utilizando a carga de dados 2 no cenário PS.

Tabela A.6: Resultados dos experimentos do fluxo de dados 6.

Hardware	Métrica		Carga	
			1	2
Local[16]	Tempo (s)	Pandas (P)	20,7	80,0
		Spark (S)	18,7	53,0
		cuDF (C)	7,0	8,7
		M (Cen. PS)	18,7	53,0
		M (Cen. CPS)	7,0	8,7
	PS	Spd. S / M	1,0	1,0
		Spd. P / M	1,2	1,6
		Decisão / Acerto	S / ✓	S / ✓
	CPS	Spd. S / M	2,7	6,2
		Spd. P / M	3,0	9,3
		Spd. C / M	1,0	1,0
		Decisão / Acerto	C / ✓	C / ✓
Local[30]	Tempo (s)	Pandas (P)	21,0	79,4
		Spark (S)	18,7	52,7
		cuDF (C)	7,0	8,7
		M (Cen. PS)	18,7	52,7
		M (Cen. CPS)	7,0	8,7
	PS	Spd. S / M	1,0	1,0
		Spd. P / M	1,2	1,6
		Decisão / Acerto	S / ✓	S / ✓
	CPS	Spd. S / M	2,7	6,1
		Spd. P / M	3,0	9,2
		Spd. C / M	1,0	1,0
		Decisão / Acerto	C / ✓	C / ✓
YARN	Tempo (s)	Pandas (P)	57,4	144,0
		Spark (S)	42,4	64,7
		cuDF (C)	18,0	29,4
		M (Cen. PS)	42,4	64,7
		M (Cen. CPS)	18,0	29,4
	PS	Spd. S / M	1,0	1,0
		Spd. P / M	1,4	2,3
		Decisão / Acerto	S / ✓	S / ✓
	CPS	Spd. S / M	2,4	2,2
		Spd. P / M	3,2	5,0
		Spd. C / M	1,0	1,0
		Decisão / Acerto	C / ✓	C / ✓

A.7 TCP-H – Fluxo de dados 7

O fluxo 7 (figura A.7) tem como objetivo avaliar as similaridades entre vendas realizadas por meio de envio via caminhão. Na consulta, é aplicado o algoritmo K-Means para agrupar transações semelhantes com base em características específicas de cada venda,

permitindo a identificação de itens similares. Em seguida, é realizada a contagem do número de transações semelhantes inferidas em cada grupo. Esse resultado, ainda que fictício, poderia ser útil para auxiliar a empresa na otimização da logística de entregas. Na figura, as operações representadas em azul indicam execução com a ferramenta Pandas, enquanto aquelas em verde são executadas com o Spark. Os resultados completos das avaliações estão apresentados na tabela A.7.

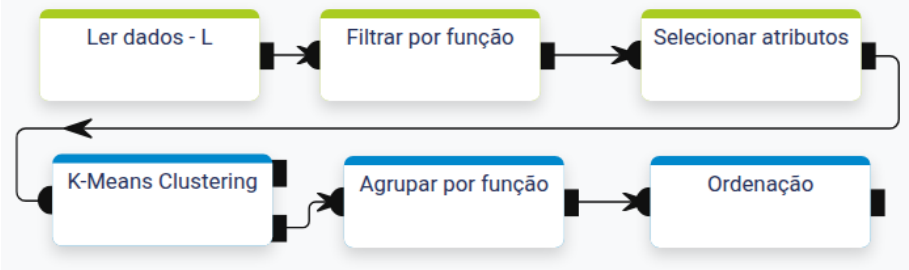


Figura A.7: Fluxo de dados 7. Exemplo de recomendação para o hardware Local[30] utilizando a carga de dados 3 no cenário PS.

Tabela A.7: Resultados dos experimentos do fluxo de dados 7.

Hardware	Métrica		Carga 1	Carga 2	Carga 3
Local[16]	Tempo (s)	Pandas (P)	23,0	87,4	170,4
		Spark (S)	64,7	275,7	559,4
		cuDF (C)	14,0	25,7	–
		M (Cen. PS)	64,7	107,4	204,0
		M (Cen. CPS)	14,0	25,7	198,7
	PS	Spd. S / M	1,0	2,6	2,8
		Spd. P / M	0,4	0,9	0,9
		Decisão / Acerto	S / ✗	M / ✗	M / ✗
	CPS	Spd. S / M	4,7	10,8	2,9
		Spd. P / M	1,7	3,4	0,9
		Spd. C / M	1,0	1,0	–
		Decisão / Acerto	C / ✓	C / ✓	M / ✗
Local[30]	Tempo (s)	Pandas (P)	22,7	86,4	168,4
		Spark (S)	65,0	272,4	548,4
		cuDF (C)	11,0	25,7	–
		M (Cen. PS)	29,7	106,4	203,0
		M (Cen. CPS)	11,0	25,7	204,0
	PS	Spd. S / M	2,2	2,6	2,7
		Spd. P / M	0,8	0,9	0,9
		Decisão / Acerto	M / ✓	M / ✓	M / ✓
	CPS	Spd. S / M	6,0	10,7	2,7
		Spd. P / M	2,1	3,4	0,9
		Spd. C / M	1,0	1,0	–
		Decisão / Acerto	C / ✓	C / ✓	M / ✓
YARN	Tempo (s)	Pandas (P)	42,4	152,4	303,4
		Spark (S)	76,0	274,7	578,4
		cuDF (C)	24,4	45,4	–
		M (Cen. PS)	38,4	110,7	205,0
		M (Cen. CPS)	24,4	45,4	196,4
	PS	Spd. S / M	2,0	2,5	2,9
		Spd. P / M	1,1	1,4	1,5
		Decisão / Acerto	M / ✓	M / ✓	M / ✓
	CPS	Spd. S / M	3,2	6,1	3,0
		Spd. P / M	1,8	3,4	1,6
		Spd. C / M	1,0	1,0	–
		Decisão / Acerto	C / ✓	C / ✓	M / ✓

A.8 Spotify – Fluxo de dados 8

O fluxo de dados 8 (figura A.8) busca entender o comportamento médio das *playlists* na base de dados do Spotify. Para isso, a partir do arquivo das músicas (*tracks*), são computadas informações individuais de cada *playlist*, como o número de músicas, artistas e álbuns. Em um segundo momento, esses valores são, então, agrupados novamente para um cálculo geral, buscando os valores médios de cada estatística. Na figura, operações em verde representam execuções em Spark. Os resultados completos das avaliações são apresentados na tabela A.8.

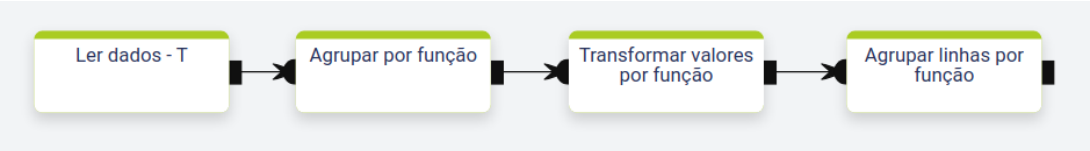


Figura A.8: Fluxo de dados 8. Exemplo de recomendação para o hardware Local[16] utilizando a carga de dados 2 no cenário PS.

Tabela A.8: Resultados dos experimentos do fluxo de dados 8.

Hardware	Métrica		Carga 1	Carga 2
Local[16]	Tempo (s)	Pandas (P)	31,7	122,4
		Spark (S)	24,0	71,7
		cuDF (C)	8,0	–
		M (Cen. PS)	24,0	71,7
		M (Cen. CPS)	8,0	71,7
	PS	Spd. S / M	1,0	1,0
		Spd. P / M	1,4	1,8
		Decisão / Acerto	S / ✓	S / ✓
	CPS	Spd. S / M	3,0	1,0
		Spd. P / M	4,0	1,8
		Spd. C / M	1,0	–
		Decisão / Acerto	C / ✓	S / ✓
Local[30]	Tempo (s)	Pandas (P)	46,4	121,7
		Spark (S)	24,0	71,7
		cuDF (C)	9,4	–
		M (Cen. PS)	24,0	71,7
		M (Cen. CPS)	9,4	71,7
	PS	Spd. S / M	1,0	1,0
		Spd. P / M	2,0	1,7
		Decisão / Acerto	S / ✓	S / ✓
	CPS	Spd. S / M	2,6	1,0
		Spd. P / M	5,0	1,7
		Spd. C / M	1,0	–
		Decisão / Acerto	C / ✓	S / ✓
YARN	Tempo (s)	Pandas (P)	56,4	144,4
		Spark (S)	56,4	116,7
		cuDF (C)	25,4	–
		M (Cen. PS)	56,4	116,7
		M (Cen. CPS)	25,4	116,7
	PS	Spd. S / M	1,0	1,0
		Spd. P / M	1,0	1,3
		Decisão / Acerto	S / ✓	S / ✓
	CPS	Spd. S / M	2,3	1,0
		Spd. P / M	2,3	1,3
		Spd. C / M	1,0	–
		Decisão / Acerto	C / ✓	S / ✓

A.9 Spotify – Fluxo de dados 9

O fluxo de dados 9, representado na figura A.9, cria um classificador baseado no SVM que prevê se uma faixa de música terá ao menos uma música do artista Drake (artista

mais frequente). Essa decisão considera metadados das *playlists* como número de seguidores, número de músicas e artistas. Para a criação do conjunto de treinamento, o dado de entrada é dividido em dois: os de *playlists* que possuem o Drake e os que não possuem. Após o preprocessamento das informações, os dados são unidos em um único conjunto de dados para a execução do SVM. Na figura, operações em azul representam execuções em Pandas, verde são em Spark, e amarelas em cuDF. Os resultados completos dos experimentos sobre esse fluxo são apresentados na tabela A.9.

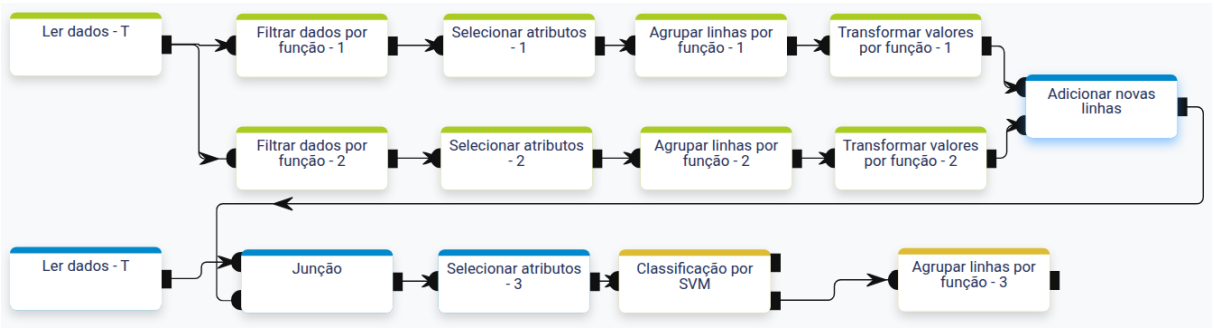


Figura A.9: Fluxo de dados 9. Exemplo de recomendação para o hardware Local[30] utilizando a carga de dados 2 no cenário CPS.

Tabela A.9: Resultados dos experimentos do fluxo de dados 9.

Hardware	Métrica		Carga 1	Carga 2
Local[16]	Tempo (s)	Pandas (P)	32,5	118,0
		Spark (S)	82,5	315,7
		cuDF (C)	10,4	–
		M (Cen. PS)	82,5	315,7
		M (Cen. CPS)	10,4	125,4
	PS	Spd. S / M	1,0	1,0
		Spd. P / M	0,4	0,4
		Decisão / Acerto	S / ✗	S / ✗
	CPS	Spd. S / M	8,0	2,6
		Spd. P / M	3,2	1,0
		Spd. C / M	1,0	–
		Decisão / Acerto	C / ✓	M / ✗
Local[30]	Tempo (s)	Pandas (P)	33,4	119,0
		Spark (S)	84,0	318,4
		cuDF (C)	14,4	–
		M (Cen. PS)	69,4	119,0
		M (Cen. CPS)	14,4	120,0
	PS	Spd. S / M	1,3	2,7
		Spd. P / M	0,5	1,0
		Decisão / Acerto	M / ✗	P / ✓
	CPS	Spd. S / M	5,9	2,7
		Spd. P / M	2,4	1,0
		Spd. C / M	1,0	–
		Decisão / Acerto	C / ✓	M / ✓
YARN	Tempo (s)	Pandas (P)	42,7	139,7
		Spark (S)	90,7	305,7
		cuDF (C)	30,4	–
		M (Cen. PS)	90,7	124,7
		M (Cen. CPS)	30,4	131,4
	PS	Spd. S / M	1,0	2,5
		Spd. P / M	0,5	1,2
		Decisão / Acerto	S / ✗	M / ✓
	CPS	Spd. S / M	3,0	2,4
		Spd. P / M	1,5	1,1
		Spd. C / M	1,0	–
		Decisão / Acerto	C / ✓	M / ✓

A.10 Spotify – Fluxo de dados 10

O fluxo de dados 10, representado na figura A.10, é responsável por gerar uma tabela que relaciona a popularidade de artistas em *playlists* e a sua média do número de músicas para todos os artistas presentes na amostra de 200 faixas de músicas com maiores seguidores e com no mínimo 50 músicas e 20 artistas. Na figura, operações em verde representam execuções em Spark e azul para Pandas. Os resultados completos dos experimentos sobre esse fluxo são apresentados na tabela A.10.

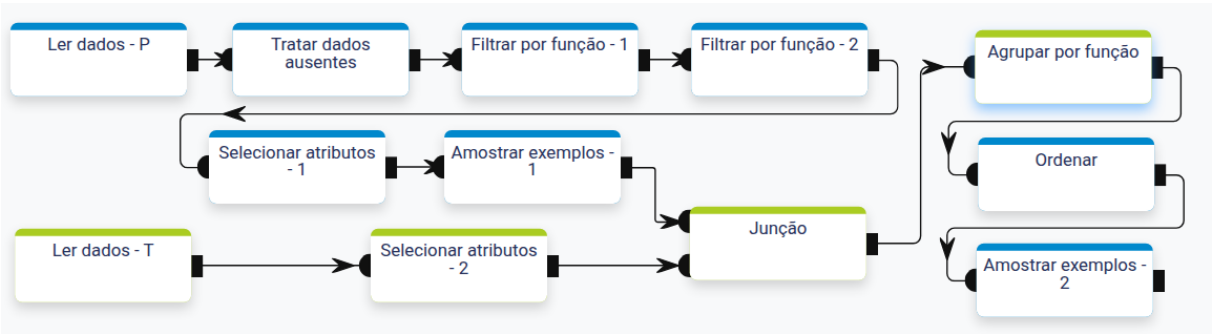


Figura A.10: Fluxo de dados 10. Exemplo de recomendação para o hardware Local[16] utilizando a carga de dados 1 no cenário PS.

Tabela A.10: Resultados dos experimentos do fluxo de dados 10.

Hardware	Métrica		Carga 1	Carga 2
Local[16]	Tempo (s)	Pandas (P)	29,4	107,7
		Spark (S)	27,0	86,4
		cuDF (C)	7,4	–
		M (Cen. PS)	27,7	84,7
		M (Cen. CPS)	7,4	89,4
	PS	Spd. S / M	1,0	1,1
		Spd. P / M	1,1	1,3
		Decisão / Acerto	M / ✓	M / ✓
	CPS	Spd. S / M	3,7	1,0
		Spd. P / M	4,0	1,3
		Spd. C / M	1,0	–
		Decisão / Acerto	C / ✓	M / ✓
Local[30]	Tempo (s)	Pandas (P)	29,7	106,4
		Spark (S)	27,0	85,0
		cuDF (C)	7,4	–
		M (Cen. PS)	28,0	86,4
		M (Cen. CPS)	7,4	86,4
	PS	Spd. S / M	1,0	1,0
		Spd. P / M	1,1	1,3
		Decisão / Acerto	M / ✓	M / ✓
	CPS	Spd. S / M	3,7	1,0
		Spd. P / M	4,1	1,3
		Spd. C / M	1,0	–
		Decisão / Acerto	C / ✓	M / ✓
YARN	Tempo (s)	Pandas (P)	39,0	126,0
		Spark (S)	33,7	83,0
		cuDF (C)	24,0	–
		M (Cen. PS)	–	95,7
		M (Cen. CPS)	24,0	93,0
	PS	Spd. S / M	–	0,9
		Spd. P / M	–	1,4
		Decisão / Acerto	M / ✓	M / ✓
	CPS	Spd. S / M	1,4	0,9
		Spd. P / M	1,7	1,4
		Spd. C / M	1,0	–
		Decisão / Acerto	C / ✓	M / ✓

A.11 Last.fm – Fluxo de dados 11

O fluxo de dados 11, representado na figura A.11, computa quatro informações: (*i*) os top 10 artistas mais escutados pelos usuários; (*ii*) as 10 faixas de idades que mais escutam músicas; (*iii*) os top 10 países com o maior número de usuários, e, por fim, (*iv*) a representatividade de cada sexo na plataforma. Na figura, operações em verde representam execuções em Spark e amarelas para cuDF. Os resultados completos dos experimentos sobre esse fluxo são apresentados na tabela A.11.

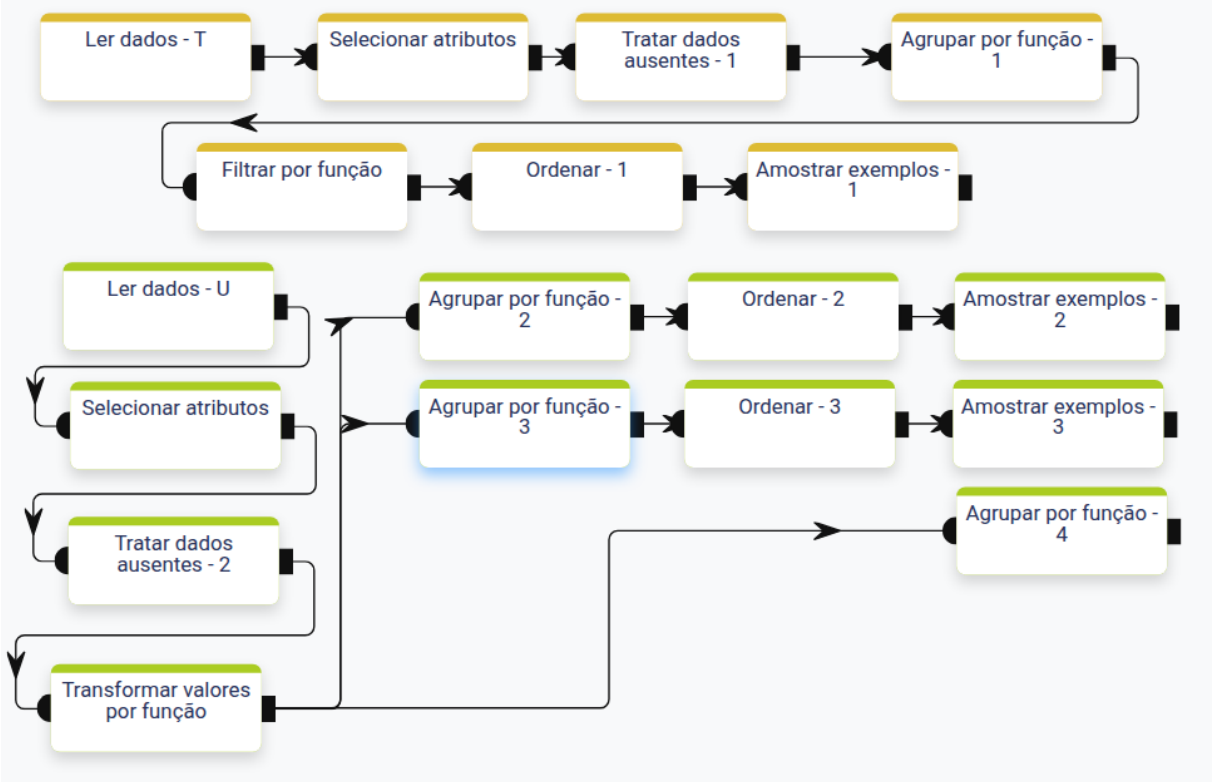


Figura A.11: Fluxo de dados 11. Exemplo de recomendação para o hardware Local[30] utilizando a carga de dados 1 no cenário CPS.

Tabela A.11: Resultados dos experimentos do fluxo de dados 11.

Hardware	Métrica		Carga 1
Local[16]	Tempo (s)	Pandas (P)	30,7
		Spark (S)	35,7
		cuDF (C)	8,0
		M (Cen. PS)	38,0
		M (Cen. CPS)	13,7
	PS	Spd. S / M	1,0
		Spd. P / M	0,9
		Decisão / Acerto	M / ✓
	CPS	Spd. S / M	2,7
		Spd. P / M	2,3
		Spd. C / M	0,6
		Decisão / Acerto	M / ✓
Local[30]	Tempo (s)	Pandas (P)	27,7
		Spark (S)	35,7
		cuDF (C)	8,4
		M (Cen. PS)	35,7
		M (Cen. CPS)	9,4
	PS	Spd. S / M	1,0
		Spd. P / M	0,8
		Decisão / Acerto	S / ✗
	CPS	Spd. S / M	3,9
		Spd. P / M	3,0
		Spd. C / M	0,9
		Decisão / Acerto	M / ✓
YARN	Tempo (s)	Pandas (P)	77,4
		Spark (S)	50,4
		cuDF (C)	27,7
		M (Cen. PS)	50,4
		M (Cen. CPS)	36,7
	PS	Spd. S / M	1,0
		Spd. P / M	1,6
		Decisão / Acerto	S / ✓
	CPS	Spd. S / M	1,4
		Spd. P / M	2,2
		Spd. C / M	0,8
		Decisão / Acerto	M / ✓

A.12 Last.fm – Fluxo de dados 12

O fluxo de dados 12, representado na figura A.12, relaciona a idade do usuário com a diversidade de artistas, computando estatísticas como número mínimo, médio e máximo de artistas diversos por idade para tentar identificar padrões de comportamentos. Utilizando os dois arquivos da base de dados do Last.fm, são primeiramente selecionados os usuários que possuem os campos de idade e sexo. Em seguida, computamos as estatísticas da quantidade de artistas distintos por usuário. Por fim, esses dois resultados intermediários são mesclados para a computação das estatísticas finais. Na figura, operações em azul representam execuções em Pandas e amarelas para cuDF. Os resultados completos dos

experimentos sobre esse fluxo são apresentados na tabela A.12.

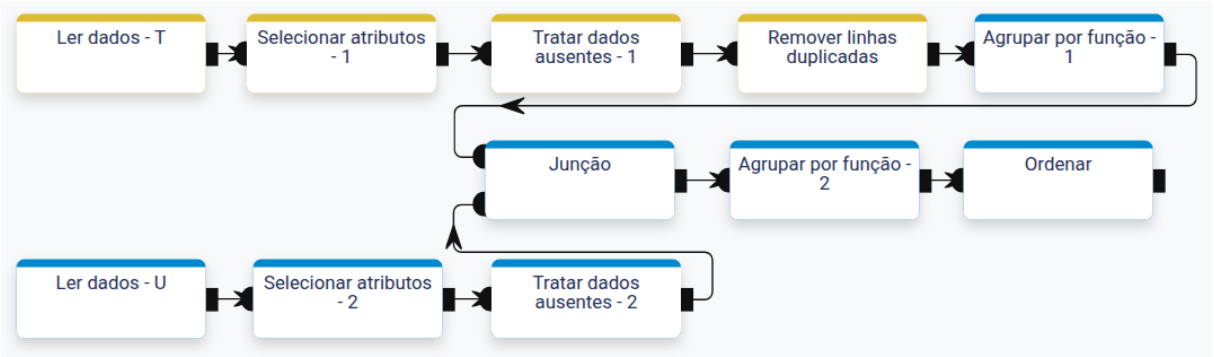


Figura A.12: Fluxo de dados 12. Exemplo de recomendação para o hardware Local[30] utilizando a carga de dados 1 no cenário CPS.

Tabela A.12: Resultados dos experimentos do fluxo de dados 12.

Hardware	Métrica		Carga
			1
Local[16]	Tempo (s)	Pandas (P)	27,7
		Spark (S)	36,7
		cuDF (C)	8,0
		M (Cen. PS)	37,0
		M (Cen. CPS)	8,4
	PS	Spd. S / M	1,0
		Spd. P / M	0,8
		Decisão / Acerto	M / ✗
	CPS	Spd. S / M	4,4
		Spd. P / M	3,4
		Spd. C / M	1,0
		Decisão / Acerto	M / ✓
Local[30]	Tempo (s)	Pandas (P)	27,0
		Spark (S)	38,0
		cuDF (C)	8,0
		M (Cen. PS)	36,7
		M (Cen. CPS)	8,4
	PS	Spd. S / M	1,1
		Spd. P / M	0,8
		Decisão / Acerto	M / ✓
	CPS	Spd. S / M	4,6
		Spd. P / M	3,3
		Spd. C / M	1,0
		Decisão / Acerto	M / ✓
YARN	Tempo (s)	Pandas (P)	72,7
		Spark (S)	45,4
		cuDF (C)	27,4
		M (Cen. PS)	72,7
		M (Cen. CPS)	27,4
	PS	Spd. S / M	0,7
		Spd. P / M	1,0
		Decisão / Acerto	P / ✗
	CPS	Spd. S / M	1,7
		Spd. P / M	2,7
		Spd. C / M	1,0
		Decisão / Acerto	C / ✓

A.13 Taxi-Rome – Fluxo de dados 13

O fluxo de dados 13, representado na figura A.13, tem como objetivo identificar as zonas das cidades com maiores concentrações de movimentações de táxis em Roma, utilizando o conjuntos de dados "Taxi-Rome". Nessa abordagem, após a seleção de alguns atributos, como o id do táxi e suas coordenadas (latitude e longitude), é utilizado o K-Means para dividir o fluxo de táxis de Roma em cinco regiões. Posteriormente, é realizado um agrupamento a partir do id do táxi para a geração de uma lista ordenada das regiões (centroides) das maiores concentrações de táxis. Na figura, operações em azul representam execuções em Pandas. Os resultados completos dos experimentos sobre esse fluxo são apresentados

na tabela A.13.

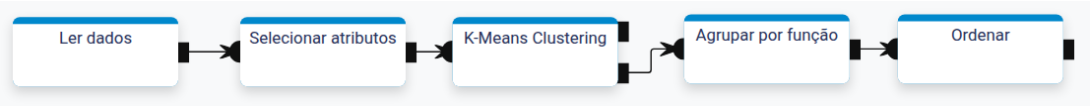


Figura A.13: Fluxo de dados 13. Exemplo de recomendação para o hardware Local[16] utilizando a carga de dados 1 no cenário PS.

Tabela A.13: Resultados dos experimentos do fluxo de dados 13.

Hardware	Métrica		Carga 1
Local[16]	Tempo (s)	Pandas (P)	57,7
		Spark (S)	272,4
		cuDF (C)	43,7
		M (Cen. PS)	57,7
		M (Cen. CPS)	43,7
	PS	Spd. S / M	4,8
		Spd. P / M	1,0
		Decisão / Acerto	P / ✓
	CPS	Spd. S / M	6,3
		Spd. P / M	1,4
		Spd. C / M	1,0
		Decisão / Acerto	C / ✓
Local[30]	Tempo (s)	Pandas (P)	59,4
		Spark (S)	267,7
		cuDF (C)	42,4
		M (Cen. PS)	59,4
		M (Cen. CPS)	42,4
	PS	Spd. S / M	4,6
		Spd. P / M	1,0
		Decisão / Acerto	P / ✓
	CPS	Spd. S / M	6,4
		Spd. P / M	1,4
		Spd. C / M	1,0
		Decisão / Acerto	C / ✓
YARN	Tempo (s)	Pandas (P)	68,4
		Spark (S)	243,0
		cuDF (C)	56,4
		M (Cen. PS)	68,4
		M (Cen. CPS)	56,4
	PS	Spd. S / M	3,6
		Spd. P / M	1,0
		Decisão / Acerto	P / ✓
	CPS	Spd. S / M	4,4
		Spd. P / M	1,3
		Spd. C / M	1,0
		Decisão / Acerto	C / ✓

A.14 US-Flights – Fluxo de dados 14

O fluxo de dados 14, ilustrado na figura A.14, utiliza o conjunto de dados US-Flights para prever o atraso na partida de voos. Inicialmente, são realizadas etapas de limpeza e preparação dos dados em colunas selecionadas, incluindo projeção, tratamento de valores ausentes, conversão de dados categóricos em valores numéricos e normalização via Min-Max scaler. Em seguida, é aplicada uma regressão linear com o objetivo de estimar o tempo de atraso. Por fim, é calculado o erro entre o valor previsto e o tempo real, sendo computado o erro médio das previsões obtidas. Na figura, operações em verde representam execuções no Spark, enquanto as azuis indicam execuções com Pandas. Os resultados completos dos experimentos para este fluxo são apresentados na tabela A.14.

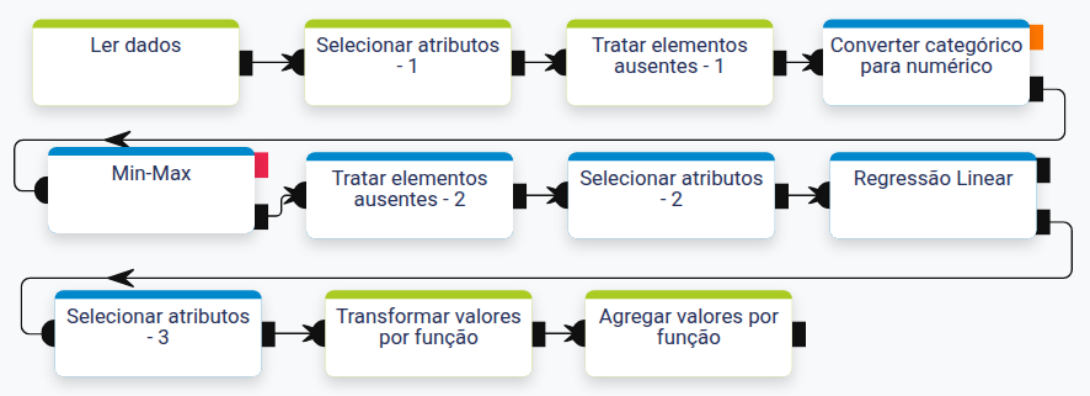


Figura A.14: Fluxo de dados 14. Exemplo de recomendação para o hardware Local[16] utilizando a carga de dados 2 no cenário PS.

Tabela A.14: Resultados dos experimentos do fluxo de dados 14.

Hardware	Métrica		Carga	
			2	3
Local[16]	Tempo (s)	Pandas (P)	221,7	–
		Spark (S)	627,4	1606,4
		cuDF (C)	–	–
		M (Cen. PS)	170,4	1606,4
		M (Cen. CPS)	96,4	1606,4
	PS	Spd. S / M	3,7	1,0
		Spd. P / M	1,3	–
		Decisão / Acerto	M / ✓	S / ✓
	CPS	Spd. S / M	6,6	1,0
		Spd. P / M	2,3	–
		Spd. C / M	–	–
		Decisão / Acerto	M / ✓	S / ✓
Local[30]	Tempo (s)	Pandas (P)	202,7	–
		Spark (S)	640,4	1598,4
		cuDF (C)	–	–
		M (Cen. PS)	145,0	1598,4
		M (Cen. CPS)	97,0	1598,4
	PS	Spd. S / M	4,5	1,0
		Spd. P / M	1,4	–
		Decisão / Acerto	M / ✓	S / ✓
	CPS	Spd. S / M	6,6	1,0
		Spd. P / M	2,1	–
		Spd. C / M	–	–
		Decisão / Acerto	M / ✓	S / ✓
YARN	Tempo (s)	Pandas (P)	217,6	–
		Spark (S)	624,0	1637,0
		cuDF (C)	–	–
		M (Cen. PS)	490,7	1637,0
		M (Cen. CPS)	141,0	1637,0
	PS	Spd. S / M	1,3	1,0
		Spd. P / M	0,5	–
		Decisão / Acerto	M / ✗	S / ✓
	CPS	Spd. S / M	4,5	1,0
		Spd. P / M	1,6	–
		Spd. C / M	–	–
		Decisão / Acerto	M / ✓	S / ✓

A.15 US-Flights – Fluxo de dados 15

O fluxo de dados 15 (figura A.15) possui o mesmo objetivo do fluxo anterior, porém adota uma abordagem mais simplificada. Neste fluxo, é aplicada uma regressão isotônica para prever o tempo de atraso de um voo com base no atraso na partida. De forma similar ao fluxo 14, o erro entre o valor previsto e o real é calculado, permitindo a estimativa do erro médio das previsões. Na figura, operações em verde representam execuções em Spark, enquanto as azuis indicam execuções com Pandas. Os resultados completos dos experimentos referentes a este fluxo estão apresentados na tabela A.15.

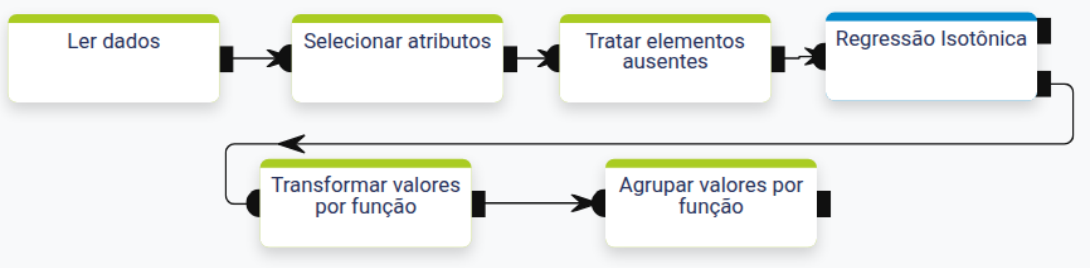


Figura A.15: Fluxo de dados 15. Exemplo de recomendação para o hardware Local[16] utilizando a carga de dados 1 no cenário PS.

Tabela A.15: Resultados dos experimentos do fluxo de dados 15.

Hardware	Métrica		Carga 1	Carga 2	Carga 3
Local[16]	Tempo (s)	Pandas (P)	71,7	179,4	–
		Spark (S)	66,7	156,7	429,0
		cuDF (C)	–	–	–
		M (Cen. PS)	49,4	103,0	249,0
		M (Cen. CPS)	48,7	101,4	246,7
	PS	Spd. S / M	1,4	1,6	1,8
		Spd. P / M	1,5	1,6	–
		Decisão / Acerto	M / ✓	M / ✓	M / ✓
	CPS	Spd. S / M	1,4	1,6	1,8
		Spd. P / M	1,5	1,8	–
		Spd. C / M	–	–	–
		Decisão / Acerto	M / ✓	M / ✓	M / ✓
Local[30]	Tempo (s)	Pandas (P)	69,4	193,7	–
		Spark (S)	67,4	160,4	407,0
		cuDF (C)	–	–	–
		M (Cen. PS)	65,0	155,0	418,0
		M (Cen. CPS)	67,4	160,4	407,0
	PS	Spd. S / M	1,0	1,0	1,0
		Spd. P / M	1,1	1,2	–
		Decisão / Acerto	S / ✓	S / ✓	S / ✓
	CPS	Spd. S / M	1,0	1,0	1,0
		Spd. P / M	1,1	1,3	–
		Spd. C / M	–	–	–
		Decisão / Acerto	S / ✓	S / ✓	S / ✓
YARN	Tempo (s)	Pandas (P)	88,0	177,0	–
		Spark (S)	75,4	163,4	435,0
		cuDF (C)	–	–	–
		M (Cen. PS)	67,4	118,4	273,7
		M (Cen. CPS)	59,7	113,0	268,4
	PS	Spd. S / M	1,2	1,4	1,5
		Spd. P / M	1,2	1,7	–
		Decisão / Acerto	M / ✓	M / ✓	M / ✓
	CPS	Spd. S / M	1,3	1,5	1,7
		Spd. P / M	1,5	1,6	–
		Spd. C / M	–	–	–
		Decisão / Acerto	M / ✓	M / ✓	M / ✓

Apêndice B

Prompts para a recomendação via LLM

No capítulo 4, mais especificamente na seção 4.4.3, realizamos uma comparação da capacidade da nossa solução em recomendar o melhor plano de execução, em relação aos planos de execução sugeridos por motores baseados em LLMs, como o ChatGPT e o Perplexity. Para essa comparação, elaboramos dois conjuntos de *prompts*, utilizados como entrada nas consultas realizadas a cada um desses motores. Neste apêndice, descrevemos os *prompts* utilizados em cada tipo de consulta.

B.1 Versão 1: abordagem simples

A primeira versão, exemplificada no Prompt B.1, adota uma abordagem mais simplificada. Nessa versão, são fornecidas informações em alto nível sobre o conjunto de dados, as operações que compõem o fluxo de execução, bem como as opções de ferramentas e configurações de hardwares disponíveis. Nesse exemplo, os termos destacados em negrito correspondem a informações variáveis da consulta. Ou seja, tratam-se de parâmetros que devem ser adaptados de acordo com o cenário avaliado, incluindo detalhes sobre os dados de entrada, as operações do fluxo, as ferramentas disponíveis e os hardwares possíveis.

Nos experimentos descritos na seção 4.4.3, foram consideradas apenas duas ferramentas de processamento de dados, Pandas e Spark, para uma única configuração de hardware: o Local[16], composto por 64 GB de memória RAM e 16 núcleos de processamento. Esse cenário está exemplificado no Prompt B.1 para o fluxo de dados 3 utilizado a carga de id 4. Caso o cenário avaliado disponibilizasse mais de uma opção de hardware, o trecho comentado presente no *prompt* deveria ser incluído na consulta realizada ao motor, de modo a refletir corretamente as opções disponíveis no momento da recomendação. Para o exemplo do Prompt B.1, a resposta B.2 contêm a solução recomendada do ChatGPT.

Prompt B.1: Exemplo de consulta em motores de LLM a partir da primeira abordagem.

```
Eu tenho um arquivo de 62.669.377 linhas, totalizando 4.2 GB,
 com as colunas: passengerId, survived, pclass, name, sex, age,
 sibsp, parch, ticket, fare, cabin, embarked.

Sobre esse arquivo, eu preciso executar as seguintes operações:
  - Operação 1: Ler o arquivo de entrada;
  - Operação 2: Selecionar as colunas age e survived;
  - Operação 3: Remover linhas em que ao menos um elemento das
                colunas age e survived, sejam nulos;
  - Operação 4: Fazer um agrupamento a partir da coluna survived,
                contabilizando o valor médio da coluna age;
  - Operação 5: Ordenar o resultado pela coluna survived.

Eu possuo um hardware onde posso executar esse experimento.
O hardware possui 64 GB de memória RAM e 16 VCPU.

Eu posso executar cada operação usando duas opções de ferramentas:
 Pandas/Scikit-Learn ou Spark. A cada operação executada eu posso
 decidir continuar na mesma ferramenta ou trocar para a outra.
 Mudanças entre ferramentas gastam tempo para a migração dos dados
 que variam de acordo com o tamanho do dado.

# Embora eu possa trocar de ferramenta a cada operação, o hardware
# deverá ser constante para todo o experimento. Qual a
# melhor opção de hardware para o meu fluxo de dados?

Crie um plano de execução que descreva qual ferramenta eu deveria
 usar em cada etapa para minimizar o tempo de execução.
```

B.2 Versão 2: abordagem com estatísticas das operações

A segunda versão de consulta, ilustrada no Prompt B.3, para o fluxo de dados 13 (carga 1), incorpora estatísticas de tempo de execução das diferentes operações realizadas por cada ferramenta nos hardwares avaliados em nossos experimentos. O objetivo dessa abordagem é fornecer aos motores de LLM informações adicionais, obtidas a partir de execuções históricas, que possam auxiliar o processo de tomada de decisão, complementando os dados de alto nível fornecidos na primeira versão do prompt. Essas informações são as mesmas utilizadas na etapa de calibração dos modelos desenvolvidos neste trabalho.

Os dados históricos são organizados em um arquivo no formato `CSV`, contendo as

Resposta B.2: Exemplo de resposta ao *Prompt* B.1.

```
Essa solução recomenda manter todo o processamento em uma única
 ferramenta para evitar custos de migração, neste caso, o uso
 contínuo do Pandas é a melhor escolha. Como o arquivo de 4 GB
 cabe confortavelmente na memória de um hardware com 64 GB de RAM
 e 16 vCPUs, as operações podem ser executadas de forma rápida e
 sem o overhead típico de iniciar um cluster distribuído, como o
 do Spark.

1. Leitura do arquivo: Utilize o Pandas para ler o arquivo CSV.
 Dado que 4 GB é um tamanho moderado para um hardware com 64 GB
 de RAM, o Pandas consegue carregar os dados em memória de
 forma eficiente.
2. Seleção das colunas: Após carregar os dados, selecione apenas
 as colunas age e survived usando o fatiamento do DataFrame.
3. Remoção de linhas com valores nulos: Utilize a função dropna
 para eliminar as linhas em que ao menos um dos valores nas
 colunas age e survived é nulo.
4. Agrupamento e cálculo da média: Realize um agrupamento pela
 coluna survived e calcule a média dos valores da coluna age
 com o método groupby.
5. Ordenação dos resultados: Ordene o DataFrame resultante com
 base na média de idade calculada.

Manter todas as operações no Pandas elimina o custo e a complexi-
 dade das migrações entre ferramentas. Esse custo não se compensa
 para o volume de dados (4 GB) e as operações simples envolvidas.

Quanto ao recurso computacional, o hardware com 64 GB de RAM e
 16 vCPUs é ideal para essa carga, habilitando o processamento em
 memória de forma rápida sem necessidade de escalabilidade
 adicional.
```

seguintes colunas: nome da operação; nome da ferramenta; número de linhas e de colunas dos dados de entrada; tamanho em bytes dos dados de entrada; e tempo médio de execução da operação (em segundos). Caso mais de uma opção de hardware estivesse disponível no cenário avaliado, seria necessária a inclusão de uma sexta coluna, indicando o hardware correspondente a cada registro da planilha.

A resposta B.4 exemplifica a solução do Perplexity a partir do *Prompt* B.3.

Prompt B.3: Exemplo de consulta em motores de LLM a partir da segunda abordagem.

Eu tenho um arquivo de **21.817.850** linhas, totalizando **1.6 GB**, com as colunas: **id, taxi, latitude, longitude, datetime**.

Sobre esse arquivo, eu preciso executar as seguintes operações:

- **Operação 1: Ler o arquivo CSV (data-reader);**
- **Operação 2: Selecionar as colunas latitude, longitude e taxi (projection);**
- **Operação 3: Rodar um K-Means utilizando as colunas latitude e longitude, com um número de clusters igual a 5. O algoritmo terá no máximo 100 iterações e o resultado do centroid será armazenado na coluna prediction (k-means-clustering-model);**
- **Operação 4: Realizar um groupby pela coluna taxi, contando a quantidade de centroids distintos para cada id. O resultado dessa contagem será a coluna n_zones (aggregation);**
- **Operação 5: Ordenar o resultado pela coluna taxi (sort).**

Eu possuo **um** hardware onde posso executar esse experimento. **O hardware possui 64 GB de memória RAM e 16 VCPU**.

Eu posso executar cada operação usando **duas** opções de ferramentas: **Pandas/Scikit-Learn ou Spark**. A cada operação executada eu posso decidir continuar na mesma ferramenta ou trocar para a outra. Mudanças entre ferramentas gastam tempo para a migração dos dados que variam de acordo com o tamanho do dado.

Crie um plano de execução onde descreva qual ferramenta eu devo usar em cada etapa para minimizar o tempo de execução. Para isso, considere o tempo de execução de cada operação disponível nas ferramentas descritos no arquivo **training_dataset.csv**.

Resposta B.4: Exemplo de resposta ao *Prompt* B.3.

```
# Plano de Execução para Minimizar o Tempo
Com base no tamanho do arquivo (21.817.850 linhas, 1.6 GB) e as
operações necessárias, o plano considera a eficiência de cada
ferramenta (Pandas/Scikit-Learn e Spark) para cada etapa,
conforme os tempos fornecidos no dataset training_dataset.csv.

# Plano de Execução
1. Operação 1: Ler o arquivo CSV (data-reader)
   Ferramenta: Spark
   Justificativa: Spark é mais eficiente para grandes volumes
                  de dados. Pandas seria mais lento (25.5s
                  contra 10s para Spark).
2. Operação 2: Selecionar as colunas latitude, longitude e
               taxi (projection)
   Ferramenta: Spark
   Justificativa: Spark mantém os dados distribuídos e
                  realiza projeções rapidamente em grandes
                  datasets (0.01s contra até 0.34s no Pandas).
3. Operação 3: Rodar K-Means com latitude e longitude
              (k-means-clustering-model)
   Ferramenta: Scikit-Learn
   Justificativa: Para 21M linhas, Scikit-Learn é mais rápido
                  (16.66s contra 94.31s no Spark). A migração
                  dos dados de Spark para Pandas será necessária.
4. Operação 4: Agrupar por taxi e contar os centroids distintos
              (aggregation)
   Ferramenta: Pandas
   Justificativa: Após a execução do K-Means no Scikit-Learn,
                  os dados já estarão em Pandas. A agregação é
                  mais eficiente em Pandas (0.67s contra até
                  6.38s no Spark).
5. Operação 5: Ordenar pela coluna taxi (sort)
   Ferramenta: Pandas
   Justificativa: Como os dados já estão em Pandas após a
                  agregação, a ordenação é mais eficiente
                  localmente (3.11s contra até 22.29s no Spark).
```