DedupeGov: Um Ambiente para Deduplicação de Grandes Volumes de Dados de Pessoas Físicas e Jurídicas em Âmbito Governamental

Vitor Mangaravite[1], Marcos Carvalho[1], Luis Cantelli[1], Lucas M. Ponce[1], Bruno Campoi[1], Gabriel Nunes[1], Alberto H. F. Laender[1], Marcos André Gonçalves[1]

[1] Departamento de Ciência da Computação
Universidade Federal de Minas Gerais Gerais
31270-901 – Belo Horizonte – MG

{vitormangaravite,marcoscarvalho,luiscantelli,lucasmsp}@dcc.ufmg.br

{bruno.campoi,gabrielmendes,laender,mgoncalv}@dcc.ufmg.br

Abstract. *Record Deduplication (RD) aims to identify instances that represent the same real-world entity in data repositories. In the government environment, the RD process facilitates the identification of irregularities and reduces the consumption of computing resources in data integration tasks. In this context, we propose a scalable, effective and efficient environment for integrating large data repositories (i.e., with large volumes of data, in the order of millions of records) to unify duplicate entities from multiple and different sources. Our experimental results demonstrate a 21.8% reduction from the original repository with 99% of accuracy and 95% of recall when identifying duplicate records. In addition, the proposed architecture proved to be extremely efficient and scalable for large volumes of data, deduplicating a repository of more than 392 million records in about one hour, in addition to being easy to generalize to different types of entity.*

Resumo. *Deduplicação de registros (DR) tem como objetivo identificar instâncias que representam a mesma entidade do mundo real em repositórios de dados. No ambiente governamental, o processo de DR facilita a identificação de irregularidades e reduz o consumo de recursos computacionais em tarefas de integração de dados. Nesse contexto, propomos neste artigo um ambiente escalável, efetivo e eficiente para integração de grandes repositórios de dados (i.e., com grandes volumes de dados, da ordem de milhões de registros) para unificar entidades duplicadas oriundas de múltiplas e diferentes fontes. Nossos resultados experimentais demonstram uma redução de 21,8% do repositório original com 99% de precisão e 95% de revocação na identificação de registros duplicados. Além disso, a arquitetura proposta mostrou-se extremamente eficiente e escalável para grandes volumes de dados, deduplicando um repositório de mais de 392 milhões de registros em cerca de uma hora, além de ser de fácil generalização para diferentes tipos de entidade.*

1. Introdução

Nos dias atuais, o aumento da digitalização das empresas e instituições governamentais por meio do uso de tecnologias da informação tem contribuído para o aumento do volume

e da escala dos dados em todas as instâncias de controle e fiscalização governamental. Do ponto de vista dessas instâncias, os dados são obtidos a partir de diferentes fontes (inúmeros repositórios de pessoas físicas e pessoas jurídicas, pregões de licitações públicas, dados societários e de posse, dentre outros) e é neste contexto de grande volume de dados que os sistemas de integração de dados são utilizados [Ziegler & Dittrich 2007]. Esses sistemas visam unificar os dados para produzir informação concisa, bem como garantir uma visão mais realista dos mesmos, auxiliando tanto na eficiência, quanto na eficácia das análises.

Dentre as etapas envolvidas na integração de dados, a deduplicação de registros é essencial para garantir uma redução no custo do espaço de armazenamento e no tempo de processamento das instâncias, bem como para garantir um repositório de dados íntegro [Kaur et al. 2018]. Além disso, no ambiente governamental, registros duplicados podem dificultar a identificação de irregularidades. Especificamente, a deduplicação de registros do tipo PF/PJ pode contribuir para aspectos investigativos e de análise de contravenções dentro de orgãos governamentais, uma vez que essa etapa consiste em identificar, combinar e mesclar registros que referem-se à mesma entidade do mundo real [Christen 2009].

O processo de deduplicação de registros é intrinsecamente quadrático [Caldeira & Ferreira 2018], o que aumenta o uso de recursos computacionais e o tempo de processamento conforme a quantidade de dados cresce. Isso gera diversos desafios para o desenvolvimento e implementação de soluções em um ambiente de *Big Data*, caracterizado pelo grande volume de dados e pela velocidade de coleta, ingestão e análise desses dados, assim como pela sua grande diversidade. Dessa forma, existem vários desafios que precisam ser tratados de forma adequada para que essas soluções sejam efetivamente empregadas em um ambiente de produção.

Nossa argumentação é que uma solução de deduplicação de dados adequada a um ambiente de *Big Data* deve atender a quatro requisitos: (i) *eficácia*, (ii) *eficiência*, (iii) *escalabilidade* e (iv) *generalização.* *Eficácia* refere-se à capacidade da solução adotada de identificar os grupos de registros que representam a mesma entidade do mundo real. Existem diversos trabalhos que propõem diferentes formas para realizar a identificação de registros duplicados [Christen 2011, Papadakis et al. 2020]. As estratégias usuais utilizam duas funções específicas: (i) a *função de blocagem*, responsável por identificar grupos de instâncias de registros minimamente similares para serem consideradas duplicadas e (ii) a *função de similaridade*, que quantifica a similaridade entre duas instâncias como sendo a probabilidade delas serem duplicatas. Como forma de avaliação da eficácia, geralmente são utilizadas as métricas de *revocação* e *precisão*. Nesse contexto, vários modelos de aprendizado de máquina vêm ganhando notoriedade [Stonebraker et al. 2018]. Contudo, muitas dessas abordagens necessitam de uma grande quantidade de exemplos rotulados para fins de treinamento, tornando o tempo de rotulação de tais instâncias de treino, se não proibitivo, no mínimo altamente custoso.

Complementarmente, *eficiência* e *escalabilidade* referem-se ao tempo de execução e à capacidade de executar o processo de deduplicação em larga escala. Em outras palavras, referem-se à capacidade de identificar grupos de registros duplicados em grandes repositórios de dados em tempo de execução viável (geralmente definido no contexto da aplicação). Em relação a esses requisitos, espera-se que o tempo de execução

e o uso de recursos computacionais não aumentem na mesma proporção (por exemplo, quadraticamente) que a quantidade de dados.

Por fim, a maioria das soluções propostas para deduplicação de dados é voltada para domínios específicos, ou seja, tais soluções resolvem o problema em determinados contextos específicos [Ceccarelli et al. 2013] como, por exemplo, nos casos de desambiguação de nomes de autores de publicações científicas [Azeroual et al. 2022, Ferreira et al. 2020] e de sequenciamento genômico [Bartus & Arzuaga 2018]. Nesse sentido, avaliamos que a *generalização*, ou seja, a capacidade de identificar registros duplicados em diferentes contextos, é um requisito fundamental, principalmente no ambiente de dados governamentais onde o número de aplicações e tipos de registro a serem potencialmente deduplicados é bastante grande (e.g., registros de pessoas, empresas, licitações, leilões, etc.).

Em resumo, o nosso objetivo neste artigo é propor uma arquitetura geral para deduplicação de registros em um ambiente de *Big Data* que atenda aos quatro requisitos citados anteriormente (eficácia, eficiência, escalabilidade e generalização). Nossa proposta envolve dois processos: (i) Deduplicação Geral, compreendendo a deduplicação de todos os registros previamente armazenados, gerando ao final uma tabela denominada *Master Data Management Records (MDM Records)*, e (ii) Deduplicação Incremental, no qual novas cargas de dados são deduplicadas de forma adicional ao processo de geração da tabela *MDM Records*.

Assim, neste contexto, procuramos responder às seguintes questões de pesquisa:

- QP1: É possível criar uma arquitetura de deduplicação que atenda todos os requisitos (conflitantes) de eficácia, eficiência, escalabilidade e generalidade?
- QP2: É possível manter uma tabela *MDM Records* conforme novas cargas de diferentes fontes são ingeridas sem a necessidade de deduplicá-la totalmente a cada nova carga?

Para responder a essas questões, avaliamos a nossa arquitetura em um âmbito governamental, no qual os repositórios de dados que representam as entidades dos tipos Pessoa Física (PF) e Pessoa Jurídica (PJ) são advindos de diferentes órgãos governamentais. Diferentemente de outros trabalhos [Alexiou et al. 2022, Singhal et al. 2019], utilizamos um repositório bastante expressivo, contendo mais de 390 milhões de registros. Para as novas cargas, utilizamos repositórios com aproximadamente dois milhões de registros.

Em nossa arquitetura, para ambos os processos (Geral e Incremental), a eficácia é atendida por meio de um modelo de aprendizado de máquina que apresenta alta precisão e revocação, e que foi treinado utilizando uma amostra das instâncias dos próprios repositórios. Para tanto, é usado o *framework* de deduplicação Dedupe [Bilenko 2002] para se criar um conjunto de registros que representam a mesma entidade do mundo real. Esse conjunto de treino foi construído com base na rotulação de 1000 pares de registros (0,0002% do repositório), implicando um esforço mínimo de rotulação, mesmo considerando o custo inerente da rotulação manual que demanda tempo e conhecimento sobre o domínio do problema. Apesar disso, nossos resultados mostraram uma precisão de até 99% e uma revocação de 95%, o que contribuiu para uma redução de 21,8% do repositório original.

Já em termos de eficiência e escalabilidade, a arquitetura foi implementada em

um ambiente *Spark* utilizando o paradigma *Resilient Distributed Dataset*[1] *(RDD)* em Python. Nossa avaliação experimental mostrou que a solução é eficiente e escalável em função da quantidade de registros do repositório original, levando um tempo máximo de aproximadamente uma hora e quarenta minutos para deduplicar um repositório com mais de 392 milhões de registros. Por fim, a nossa arquitetura foi concebida para permitir a generalização para outros domínios. Em nossa avaliação, consideramos dois domínios específicos (Pessoa Física e Pessoa Jurídica), com campos e valores distintos.

Assim, as principais contribuições deste artigo são:

- Proposta de uma arquitetura de deduplicação de registros, denominada DedupeGov, integrada a uma arquitetura de *Big Data* que procura satisfazer os quatro requisitos propostos;
- Uma avaliação experimental do processo de deduplicação de registros que representam entidades dos tipos Pessoa Física (PF) e Pessoa Jurídica (PJ), advindos de diversas fontes governamentais;
- Um modelo de aprendizado de máquina eficaz para identificar registros duplicados nos domínios de PF e PJ com baixo custo de rotulação.

O restante deste artigo está organizado como se segue. Na Seção 2 apresentamos uma breve descrição de alguns trabalhos relacionados ao nosso. A seguir, na Seção 3 detalhamos os principais módulos da arquitetura DedupeGov, enquanto na Seção 4 descrevemos a sua avaliação experimental e os resultados obtidos. Por fim, na Seção 5 concluímos o artigo apresentando as nossas conclusões e apontando alguns trabalhos futuros.

2. Trabalhos Relacionados

Nesta seção, descrevemos alguns trabalhos relacionados à nossa proposta encontrados na literatura. A maioria desses trabalhos está voltada para domínios específicos, bem como não mensura escalabilidade e eficiência com base em um conjunto de dados representativo e de larga escala. Os trabalhos abordados a seguir representam uma visão atual e recente do estado-da-arte relacionado ao problema de deduplicação de registros.

Singhal et al. [2021] propuseram um método de deduplicação, denominado *Electronic Patient Matching System - EPMS*, que ajuda a solucionar o problema de troca de informação de pacientes entre diversas organizações de saúde. Assim como em nosso trabalho, o método proposto aplica o processo de blocagem, visando, portanto, a redução na quantidade de comparações entre os registros. No entanto, diferente da nossa avaliação, os autores consideram uma pequena amostra sintética de pacientes com 44.555 registros.

Por outro lado, Alexiou et al. [2021] projetaram um *framework* integrado ao ambiente SQL que visa diminuir o tempo de processamento desse tipo de consulta a partir do processo de deduplicação. O processo de deduplicação é realizado apenas sobre os dados que se relacionam diretamente com a consulta a ser processada, mantendo o repositório correspondente inalterado. Ao contrário do nosso trabalho, a proposta dos autores não reside em uma melhoria de espaço de armazenamento, dado que a deduplicação ocorre apenas durante a busca e não altera o repositório original.

Ai et al. [2021] propuseram um método de deduplicação de registros baseado em um subgrafo conectado para melhorar a acurácia, bem como a eficiência na detecção de

[1] https://databricks.com/glossary/what-is-rdd

repetições. Ao detectar repetições, esse método visa formar grupos de registros duplicados, concatenando assim os campos que os formam e produzindo um único registro representativo do grupo. Os eventos de ocorrência de palavras, assuntos, objetos, tempos e localizações são utilizados para formar as arestas do grafo cujos vértices são os registros. Quanto mais arestas, mais semelhanças os registros do subgrafo possuem e maior a possibilidade de serem registros duplicados. Apesar de resultados positivos na melhora da acurácia e na eficiência da deduplicação inicial do repositório, o método não apresenta nenhuma solução otimizada para adição de novos registros, sendo necessário uma deduplicação completa do repositório a cada adição de um novo registro, tornando-o ineficaz quando aplicado a um repositório com volume elevado de dados em constante utilização.

Com relação ao treinamento de modelos de aprendizado de máquina para tarefas de deduplicação de registros, Espiridião et al. [2021] utilizaram a técnica *data augmentation* para melhorar a performance de modelos usados na identificação de autores homônimos de artigos científicos. Conforme demonstrado por esses autores, a técnica utilizada melhora a acurácia e a revocação dos modelos quando aplicados em um conjunto de dados pequeno e pouco representativo. Ao contrário dessa abordagem, optamos por utilizar uma técnica de aprendizado ativo através da qual conseguimos treinar um modelo eficaz sem ampliar o conjunto de dados de treino (como ocorre no caso de *data augmentation*).

Finalmente, Ngueilbaye et al. [2021] propuseram um novo modelo, denominado *Stacked Dedupe Learning* (SDL), que elimina a necessidade de rotulação manual durante o treinamento do modelo de deduplicação. A eliminação dessa rotulação, se bem sucedida, torna o treinamento independente da disponibilidade de rotuladores humanos, possibilitando um treinamento mais eficiente. SDL utiliza duas técnicas para alcançar seu objetivo. A primeira considera a média simples da representação da distribuição dos *tokens* que representam as palavras, enquanto a segunda utiliza redes neurais bidirecionais (*Long Short-Term Memory)*. Esse trabalho de certa forma propõe uma estratégia ortogonal aos nossos objetivos, de modo que, eventualmente, poderá ser utilizado no futuro em nosso processo de aprendizado ativo como forma de gerar uma semente inicial.

3. Arquitetura Proposta

Nesta seção, apresentamos a arquitetura do ambiente DedupeGov proposto e implementado no trabalho reportado neste artigo. A Figura 1 ilustra todos os módulos que compõem essa arquitetura, os quais são brevemente descritos a seguir.

O Módulo de Treino é responsável pelas tarefas pertinentes à criação do modelo de aprendizado de máquina, o qual é brevemente descrito na Subseção 4.1 e pode ser facilmente substituído sem a necessidade de alteração da arquitetura. Como parte desse módulo temos os processos de amostragem do repositório, de rotulação manual das instâncias de treino e de treino de uma função de inferência cujo objetivo é identificar se um par de registros é formado por duplicatas.

O Módulo de Integração abrange três processos específicos: *Pré-processamento*, *Processo Ponta-a-Ponta* e *Processo Incremental*. O *Pré-processamento* padroniza os diversos campos de acordo com um mesmo formato (por exemplo, padronização do formato de datas, nomes de Pessoas Físicas e CPF's). Além disso, definimos um ar-

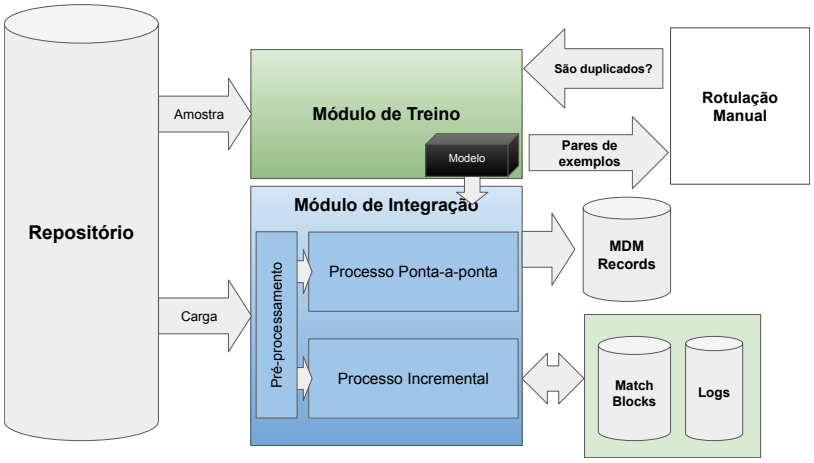

Figura 1. Arquitetura Proposta - DedupeGov.

quivo de configuração (no formato JSON) para o mapeamento dos campos do repositório de origem de acordo com o esquema de identificação *Entity Identity Structure* (EIS) [Zhou & Talburt 2011] utilizado no processo de deduplicação[2]. Isso permite corrigir a nomenclatura e a dimensionalidade esperada para criar e manter a tabela *MDM Records*. Além disso, o mapeamento contribui para a generalização da solução, uma vez que no arquivo de configuração é possível realizar os ajustes necessários e específicos de cada domínio.

Já o Processo Ponta-a-Ponta é responsável por identificar todos os registros duplicados no repositório de origem, para então criar a tabela *MDM Records*. Como ilustrado na Figura 2, esse processo adota uma abordagem *top-down* e é dividido em duas etapas, Computação por *Batch* e Computação por Bloco, cada qual com suas respectivas subetapas. Na etapa de Computação por *Batch* é realizado o processo de blocagem dos registros, onde cada registro tem suas respectivas chaves de bloco calculadas, o que nos permite processar cada *batch* separadamente. Para gerar cada chave de bloco, utilizamos o método *fingerprint* da ferramenta Dedupe, considerando para isso o campo CPF para Pessoa Física e CNPJ para Pessoa Jurídica. Na Figura 2, cada retângulo representa um registro e cada cor representa a sua respectiva chave de bloco. Registros da mesma cor são registros duplicados.

Após a criação dos blocos, é dado início à etapa de Computação por Bloco. Nessa etapa é realizada a deduplicação dos registros propriamente dita, o que ocorre através de três subetapas: Cálculo do Escore, Cálculo dos Componentes Conectados e Combinação. Na primeira subetapa, cada bloco é processado paralelamente e, para cada par de registro no bloco, é calculado um escore de similaridade com base em uma regressão logística treinada para identificar pares de registros duplicados. O resultado é então repassado para a próxima subetapa que consiste em identificar os componentes conectados, ou seja, todos os grupos de registros dentro de um bloco que são considerados registros duplicados de uma mesma entidade. É importante ressaltar que, para identificar os pares de registros em cada bloco, calcular a similaridade entre eles e identificar os componentes

[2]O EIS de um registro define os campos e seus respectivos valores necessários para a identificação dos registros duplicados.

conectados, utilizamos, respectivamente, as funções *pairs*, *score* e *cluster* da ferramente Dedupe. Por fim, para cada grupo de registros considerados duplicados, a última subetapa (Combinação) constrói um registro mais completo e representativo, denominado registro *master*.

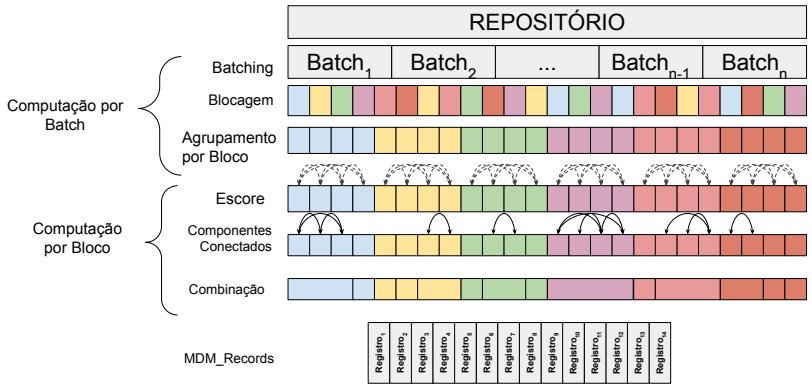

Figura 2. Módulo de Integração - Processo Ponta-a-Ponta.

O Processo Ponta-a-Ponta não cria apenas a tabela *MDM Records*, mas também duas tabelas intermediárias denominadas *Match Blocks* e *Logs* (Figura 1). De modo geral, enquanto a tabela *MDM Records* armazena um identificador único (UUID_Master), a tabela *Match Blocks* armazena o mapeamento da chave de bloco com a respectiva lista de identificadores UUID_Master que pertencem ao bloco. Adicionalmente, a tabela *Logs* registra o resultado do Processo Incremental, principalmente o par UUID da carga inserida e os respectivos identificadores UUID_Master alterados na tabela *MDM Records*. Com base nesses valores, é possível realizar uma auditoria e identificar erros no processo de deduplicação, bem como, se necessário, reconstruir a tabela *MDM Records*.

Por outro lado, enquanto o Processo Ponta-a-Ponta tem como objetivo deduplicar um repositório completo, o Processo Incremental (PI) visa inserir um novo conjunto de registros à tabela *MDM Records* já criada, garantindo que não serão inseridos duplicatas. Assim como no Processo Ponta-a-Ponta, a primeira etapa consiste na execução da blocagem (em *batch*) sobre o novo conjunto de registros, seguido pelo agrupamento dos registros por bloco. Uma vez que a tabela *Match Blocks* também está agrupada pelas chaves de bloco dos registros da tabela *MDM Records*, executamos um *Right Join* entre a tabela *Match Blocks* com a blocagem da carga de entrada, extraindo da tabela *MDM Records* somente os blocos de registros que ocorreram pelo menos uma vez na carga. Este tipo de computação evita a re-deduplicação de registros que já foram deduplicados na tabela *MDM Records*, bem como auxilia na identificação dos novos registros (i.e., registros não encontrados nessa tabela, uma vez que blocos com apenas um registro podem ser considerados não duplicados e serem diretamente armazenados na tabela *MDM Records*).

Todos os demais blocos com mais de um registro passarão pela etapa de Computação por Bloco, ou seja, cálculo da similaridade de todos os pares (Escore), agrupamento dos escores (Componentes Conectados) e Combinação. No entanto, diferente do Processo Ponta-a-Ponta, após a combinação dos registros, verificamos se o registro resultante é baseado na combinação com um registro já conhecido, ou seja, que já estava nas tabelas *MDM Records* e *Match Blocks*, ou se é apenas uma combinação de registros da nova carga. Quando trata-se de um registro advindo da nova carga, podemos realizar

o armazenamento direto nas tabelas *MDM Records* e *Match Blocks*. Porém, quando a combinação ocorre com um registro já conhecido, faz-se necessário a atualização desses registros tanto na tabela *MDM Records*, quanto na *Match Blocks*. Essa atualização é, então, salva na tabela *Log*, na qual armazenamos o novo UUID do registro e o seu respectivo UUID que estava na tabela *MDM Records* (UUID_Master).

Finalmente, é importante ressaltar que a modularização prevista na arquitetura proposta garante a independência de seus módulos e de suas respectivas etapas e subetapas, permitindo a alteração individual de cada um deles sem impacto no funcionamento dos demais.

4. Avaliação Experimental

Nesta seção, apresentamos uma visão geral da arquitetura do ambiente DedupeGov proposta neste artigo. A arquitetura DedupeGov foi implementada em Apache Spark[3], um arcabouço popular de código-aberto para processamento de grandes volumes de dados. Esse arcabouço permite a execução de um conjunto de operações nos dados armazenados no HDFS[4], possibilitando distribuir o processamento dessas operações em um *cluster* de modo a obter um melhor aproveitamento dos recursos computacionais.

Na nossa experimentação, buscamos cumprir os requisitos apresentados, bem como responder às perguntas de pesquisa propostas neste artigo. Para isso, utilizamos um repositório de dados reais contendo 392.602.483 de registros vindos de diferentes fontes. Desse total, aproximadamente 343 milhões são registros que representam entidades do tipo Pessoa Física e 48 milhões do tipo Pessoas Jurídica. As entidades do tipo Pessoa Física (PF) possuem nove atributos (CPF, nome da pessoa, titulo de eleitor, número NIS/NIT, identificador da CEMIG, Nome do pai/mãe e data de nascimento), enquanto as entidades do tipo Pessoa Jurídica (PJ) possuem apenas dois (CNPJ e nome da empresa). Desse repositório foram geradas quatro amostras, nas quais variamos a quantidade de registros conforme sumarizado na Tabela 2 (Cenário/Entrada). Todas as amostras foram mantidas em formato Parquet no HDFS (v3.1), onde eram lidas e processadas pela nossa solução em um *cluster* Spark (v2.3.2) utilizando dois nós executores, cada um com oito GB de memória RAM e oito núcleos de processamento. Além disso, a nossa solução foi implementada em Python sob o paradigma RDD.

A seguir, na Subseção 4.1, discutimos a avaliação do modelo utilizado para deduplicação de registros dos tipos PF e PJ, enquanto que na Subseção 4.2 apresentamos os resultados do Processo Ponta-a-Ponta (deduplicação) e do Processo Incremental. Por fim, na Subseção 4.3, detalhamos os resultados de desempenho da solução de deduplicação.

4.1. Avaliação do Modelo de Aprendizado de Máquina

Para este trabalho, criamos um modelo de aprendizado de máquina para detectar pares de registros duplicatas dos tipos PF e PJ. Para isso, utilizamos a técnica de pareamento probabilístico que, por sua vez, usa como atributos a diferença de alinhamento entre os campos que compõem os registros. Especificamente, utilizamos a ferramenta Dedupe[5]

[3]Apache Spark. `https://spark.apache.org`

[4]Sistema de armazenamento distribuído do ecossistema Hadoop. `https://hadoop.apache.org`

[5]`https://docs.dedupe.io/en/latest/how-it-works/Matching-records.html`

que implementa o método *Affine Gap Distance* para identificar a distância entre os campos dos registros. Com base nessas distâncias, treinamos uma regressão logística que retorna a probabilidade dos registros serem iguais. Para o processo de treinamento do modelo geramos uma amostragem por diversidade, cuja regra de seleção foi baseada no agrupamento de pares de registros utilizando o algoritmo *K-means* com 1000 *clusters*. No final, o par de registros mais próximo à média do *cluster* foi selecionado para rotulação manual, totalizando assim 1000 pares de instâncias de treino rotuladas. Em relação à rotulação, cada par de registro da amostra selecionada foi rotulado por três pessoas, sendo selecionado aquele que atendia ao consenso da maioria (duas concordâncias).

Realizamos a avaliação da eficácia do modelo criado considerando as métricas Precisão ($P(\cdot)$), Revocação ($R(\cdot)$) e F1, definidas pelas Equações 1 e 2, onde $\mathcal{E}$ é o conjunto resultante dos pares de registros deduplicados pelo DedupeGov e $\hat{\mathcal{E}}$ é o conjunto de entidades reais (duplicadas).

$$P(\mathcal{E}, \hat{\mathcal{E}}) = \frac{|\mathcal{E} \cap \hat{\mathcal{E}}|}{|\mathcal{E}|}; R(\mathcal{E}, \hat{\mathcal{E}}) = \frac{|\mathcal{E} \cap \hat{\mathcal{E}}|}{|\hat{\mathcal{E}}|} \quad (1)$$

$$F1_{Micro}(\mathcal{E}, \hat{\mathcal{E}}) = \frac{2 * P(\mathcal{E}, \hat{\mathcal{E}}) * R(\mathcal{E}, \hat{\mathcal{E}})}{P(\mathcal{E}, \hat{\mathcal{E}}) + R(\mathcal{E}, \hat{\mathcal{E}})}; F1_{Macro}(\mathcal{E}, \hat{\mathcal{E}}) = \frac{F1_{PAR} + F1_{NPAR}}{2} \quad (2)$$

Especificamente no processo de deduplicação, a precisão da classe PAR ($r_i \equiv r_j$) considera a proporção do número de pares de registros duplicados que realmente correspondem à mesma entidade no mundo real. Já a revocação corresponde à proporção do número de pares de registros reais que foram identificados corretamente pelo algoritmo. A métrica $F1_{Micro}$ pode ser definida como a média harmônica da precisão e da revocação considerando que todos os pares são igualmente importantes, enquanto a $F1_{Macro}$ considera que todas as classes têm importância equivalente.

Assim, avaliamos o modelo considerando duas amostras distintas. A primeira foi criada a partir do conjunto de treino e a segunda a partir da propagação dos rótulos dos 1000 pares (centroides) para todos os respectivos registros de cada *cluster*. A Tabela 1 sintetiza os resultados do nosso modelo, em termos da revocação e precisão, para ambas as amostras.

Tabela 1. Resultados do modelo proposto.

	MICROF1	MACROF1	REVOCAÇÃO(PAR)	PRECISÃO(PAR)
AMOSTRA POR DIVERSIDADE	0,880	0,880	0,797	0,943
REPOSITÓRIO COMPLETO	0,987	0,984	0,959	0,997

Os resultados mostram a eficácia do modelo ao atingir 94,3% de precisão na identificação de pares de registros duplicados. Além disso, há uma melhora em todas as métricas quando aplicado ao repositório completo, o que pode ser explicado pela característica do pareamento probabilístico. Na prática, os algoritmos de pareamento aprendem regras de deduplicação distintas. Assim, enquanto o pareamento exato realiza a blocagem dos registros considerando a exatidão dos atributos dos registros, o pareamento probabilístico aprende a executar essa etapa considerando a similaridade. Isso implica que, mesmo que os valores de alguns atributos estejam mascarados, o pareamento

probabilístico é capaz de "desempatar" considerando o restante dos atributos. Além disso, o pareamento probabilístico permite detectar erros de digitação entre dois registros duplicados.

4.2. Qualidade da Solução

Um dos aspectos mais importantes derivados do nosso processo de deduplicação é a redução da quantidade de registros gerados ao seu final, especificamente quando comparamos a quantidade de registros do repositório original (Entrada) com o total da tabela *MDM Records*. A Tabela 2 apresenta os resultados do Processo Ponta-a-Ponta sobre cada amostra.

Tabela 2. Sumário das amostras utilizadas na experimentação.

	Entrada		MDM Records	
Cenário	# de registros (milhões)	Utilização em Disco (GB)	# de registros (milhões)	Utilização em Disco (GB)
Amostra 1	24	1,8	18,9	1,7
Amostra 2	130,8	9,9	102,6	9,2
Amostra 3	261,7	19,8	204,7	18,3
Amostra 4	392,6	28,6	306,9	27,5

Em todos os cenários houve uma redução de aproximadamente 21% na quantidade de registros. Além disso, como mencionado, a deduplicação contribui para a diminuição da utilização de espaço em disco, o que foi perceptível nos resultados dos nossos experimentos. É importante notar que a redução na quantidade de registros duplicados está diretamente interligada com a eficácia do modelo em identificar grupos de tais registros. Portanto, a alta eficácia do nosso modelo pode explicar a qualidade da solução de deduplicação de registros.

Avaliamos também a qualidade do Processo Incremental (PI) sob três novas cargas contendo, em média, dois milhões de registros. Na experimentação, consideramos a tabela *MDM Records* criada a partir da Amostra 4, conforme indicado na Tabela 2. A Tabela 3 sumariza os principais resultados do PI. Do ponto de vista da quantidade de registros novos identificados, houve um aumento de mais de 900 mil registros na tabela *MDM Records*, o que significa uma relação de 14,5% em função da quantidade total de registros que passaram pelo PI. Além disso, do total de registros das cargas, 85,5% foram considerados duplicados, ou seja, aqueles registros das cargas novas que casaram com os registros da tabela *MDM Records* (quarta coluna da Tabela 3). Essa grande quantidade de registros duplicados já era esperada, tendo em vista que a tabela *MDM Records* já continha uma quantidade significativa de registros, aumentando as chances de haver casamento com os registros advindos das cargas novas.

Tabela 3. Resultados do Processo Incremental.

Carga	# de registros	# novos registros	# registros Carga Nova × MDM Records
Carga A	2.011.418	383.797	1.627.621
Carga B	2.345.165	94.343	2.250.822
Carga C	2.124.386	465.378	1.659.008

4.3. Desempenho

A deduplicação de registros é em geral um processo oneroso em função da quantidade de registros envolvidos. Assim sendo, avaliamos o desempenho do Processo Ponta-a-ponta

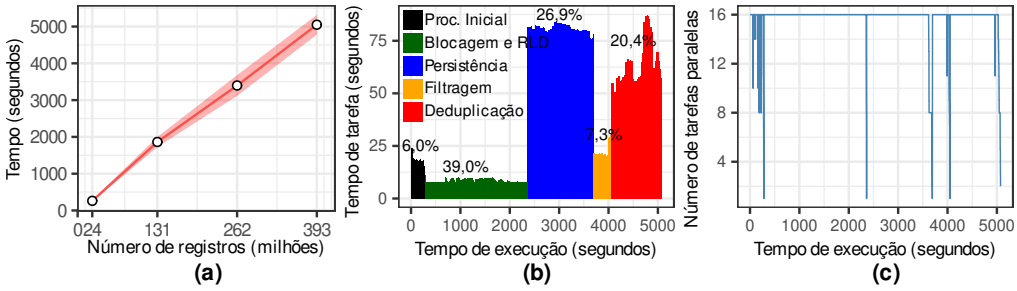

Figura 3. Testes de desempenho. (a) Tempo de execução em função do número de registros. (b) Etapas de execução. (c) Número de tarefas executadas concorrentemente em um dado instante de tempo.

sobre diversos cenários, a fim de verificar a sua eficiência e escalabilidade. O gráfico da Figura 3(a) apresenta o tempo de execução da solução em função do aumento de carga (quantidade de registros). Os valores apresentados no gráfico são valores médios obtidos a partir de 10 execuções. Para todas as execuções, o coeficiente de variação ficou abaixo de 0,10, indicando uma homogeneidade no tempo de execução em cada configuração.

Pelo gráfico podemos perceber um comportamento de execução escalável. A expressão da regressão do tempo de execução (em segundos) em função do número de registros pode ser dada por $t(x) = 1,3 \times 10^{-5}x + 47,2$. Tal função possui um coeficiente de determinação (R^2) de 0,998. A taxa de crescimento (1,3) da reta sugere que a solução, com a paralelização no Spark, possui um comportamento linear onde o tempo de execução cresce próximo da mesma taxa de aumento de dados.

Já a Figura 3(b) representa um *trace* que mapeia as tarefas e seus respectivos custos no decorrer de uma execução para uma amostra de tamanho 4. Embora o *trace* contemple apenas uma execução, esse comportamento pode ser extrapolado para outras execuções devido à sua variância baixa. Os valores medidos nessa figura foram obtidos a partir da instrumentação do Spark e podem ser categorizados nas seguintes cinco etapas: (i) processamento inicial, na qual é executado o preprocessamento e a blocagem; (ii) identificação de Registros Largamente Duplicados (RLD) – embora não mencionada na descrição da arquitetura, consiste em uma operação para eliminar registros com todos os campos iguais; (iii) persistência intermediária do estado do processamento; (iv) filtragem de blocos com registros únicos; e por fim (v) deduplicação dos blocos com mais de um registro. A representatividade de cada etapa, apresentada no gráfico a partir das porcentagens, foi calculada considerando o tempo médio de 10 execuções.

Esse segundo gráfico nos ajuda a compreender o comportamento da aplicação. O processamento inicial constitui uma etapa rápida da aplicação (6,0% do tempo de execução total). Já a segunda parte, é uma das etapas mais custosas para a aplicação, correspondendo a 39,0% do tempo total. A etapa de blocagem em si é rápida e corresponde a cerca de 17,7% do tempo dessa segunda parte. No entanto, o restante desse custo é devido à etapa RLD que exige um reagrupamento dos dados e, por consequência, transferências de dados pela rede via Spark. Por exemplo, podemos observar pelo gráfico que embora o custo de execução de uma tarefa dessa etapa (em verde) seja de pouco menos de 13 segundos, muitas tarefas são necessárias devido à redistribuição dos dados.

A terceira etapa, que envolve a persistência intermediária dos resultados, repre-

senta cerca de 26,9% do tempo de execução. Essa é uma etapa voltada para os aspectos de otimização do *framework* Spark. Assim sendo, uma vez que o resultado obtido na etapa anterior precisará ser utilizado em momentos futuros, os ganhos para persistir o resultado intermediário em disco sobressaem aos custos envolvidos. Por causa dessa persistência, a etapa de filtragem e escrita em disco tornam-se rápida, impactando apenas 7,3% do tempo de execução. Por fim, a etapa de deduplicação e escrita em disco (em vermelho) possui um custo de 20,4% no tempo de execução. Nessa etapa, a parte específica da deduplicação corresponde a uma fração de 87,6% do seu custo. Em nossa instrumentação, apenas 0,4% do tempo da aplicação não pôde ser explicado.

O custo medido da deduplicação nos reitera a importância da paralelização dessa atividade via Spark. Sem essa abordagem, o tempo de execução e o custo de manter os dados em sua totalidade em memória poderia ser proibitivo ao lidar com cargas muito volumosas, como as avaliadas. Por exemplo, o gráfico da Figura 3(c) mapeia o número de tarefas no Spark sendo executadas concorrentemente. Esse gráfico nos mostra que a aplicação utiliza quase a totalidade dos núcleos disponíveis (em nosso experimento, 16 *cores*). Os pontos onde isso não ocorre é no início da aplicação, em que acontece um reparticionamento inicial dos dados, e entre cada estágio de execução, como mostrado no gráfico da Figura 3(b).

5. Conclusões e Trabalhos Futuros

Neste artigo, propusemos uma arquitetura para deduplicação de registros, denominada DedupeGov, que se mostrou adequada ao ambiente de *Big Data*, atendendo diferentes requisitos conflitantes como eficácia, eficiência, escalabilidade e generalização. Além disso, demonstramos tratar-se de uma solução incremental capaz de deduplicar novas cargas sem a necessidade de reprocessar a deduplicação sobre uma tabela *MDM Records* já criada.

O processo de deduplicação foi avaliado considerando diversos cenários com variações na quantidade de registros, permitindo verificar a eficiência e a escalabilidade da solução. Além disso, nosso modelo mostrou-se eficaz, atingindo uma alta precisão e revocação (acima 95%), o que contribuiu para a redução de registros duplicados em até 21.8%. Além disso, consideramos as entidades do tipo Pessoa Física e Pessoa Jurídica no processo de deduplicação, demonstrando que a arquitetura proposta é capaz de generalizar para diferentes domínios. Para isso, a nossa arquitetura permite adaptações para diferentes domínios por meio de pequenos ajustes nas configurações e no desenvolvimento de modelos de aprendizado de máquina adequados aos respectivos domínios.

Como trabalhos futuros, pretendemos desenvolver modelos de aprendizado de máquina capazes de identificar registros duplicados para entidades de outros tipos, por exemplo, Endereço. Além disso, consideramos modelar a solução proposta em uma arquitetura *lambda*, permitindo que a execução da deduplicação de novas cargas seja feita em tempo real.

Agradecimentos

Gostaríamos de agradecer o apoio do Ministério Público de Minas Gerais, por meio do projeto Capacidades Analíticas, bem como à CAPES, CNPq e FAPEMIG pelo apoio individual aos autores.

Referências

Ai, W., Xu, J., Shao, H., Wang, Z., & Meng, T. (2021). An Entity Event Deduplication Method Based on Connected Subgraph. In *Proceedingsns of the 7th International Conference on Systems and Informatics (ICSAI)*, pages 1–6. IEEE.

Alexiou, G., Papastefanatos, G., Stamatopoulos, V., Koutrika, G., & Koziris, N. (2022). QueryER: A Framework for Fast Analysis-Aware Deduplication over Dirty Data. *arXiv preprint arXiv:2202.01546*.

Azeroual, O., Jha, M., Nikiforova, A., Sha, K., Alsmirat, M., & Jha, S. (2022). A Record Linkage-Based Data Deduplication Framework with DataCleaner Extension. *Multimodal Technologies and Interaction*, 6(4):27.

Bartus, P. & Arzuaga, E. (2018). Gdedup: Distributed File System Level Deduplication for Genomic Big Data. In *2018 IEEE International Congress on Big Data (BigData Congress)*, pages 120–127. IEEE.

Bilenko, M. Y. (2002). *Learnable Similarity Functions and Their Application to Record Linkage and Clustering*. PhD thesis, The University of Texas, Austin.

Caldeira, L. S. & Ferreira, A. A. (2018). Melhorias no Processo de Blocagem para Resolução de Entidades Baseadas na Relevância dos Termos. In *Anais do XXXIII Simpósio Brasileiro de Bancos de Dados*, pages 61–72. SBC.

Ceccarelli, D., Lucchese, C., Orlando, S., Perego, R., & Trani, S. (2013). Dexter: An Open Source Framework for Entity Linking. In *Proceedings of the Sixth International Workshop on Exploiting Semantic Annotations in Information Retrieval*, pages 17–20.

Christen, P. (2009). Development and user experiences of an open source data cleaning, deduplication and record linkage system. *ACM SIGKDD Explorations Newsletter*, 11(1):39–48.

Christen, P. (2011). A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 24(9):1537–1555.

Espiridião, L. V., Dias, L. L., & Ferreira, A. A. (2021). Applying Data Augmentation for Disambiguating Author Names. In *Anais do XXXVI Simpósio Brasileiro de Bancos de Dados*, pages 109–120. SBC.

Ferreira, A. A., Gonçalves, M. A., & Laender, A. H. F. (2020). *Automatic Disambiguation of Author Names in Bibliographic Repositories*. Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan & Claypool Publishers.

Kaur, R., Chana, I., & Bhattacharya, J. (2018). Data deduplication techniques for efficient cloud storage management: a systematic review. *The Journal of Supercomputing*, 74(5):2035–2085.

Ngueilbaye, A., Wang, H., Mahamat, D. A., & Elgendy, I. A. (2021). SDLER: stacked dedupe learning for entity resolution in big data era. *The Journal of Supercomputing*, 77(10):10959–10983.

Papadakis, G., Skoutas, D., Thanos, E., & Palpanas, T. (2020). Blocking and filtering techniques for entity resolution: A survey. *ACM Computing Surveys)*, 53(2):1–42.

Singhal, H., Ravi, H., Chakravarthy, S. N., Balasundaram, P., & Babu, C. (2019). EPMS: A Framework for Large-scale Patient Matching. In *31st IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1096–1101. IEEE.

Stonebraker, M., Ilyas, I. F., et al. (2018). Data Integration: The Current Status and the Way Forward. *IEEE Data Eng. Bull.*, 41(2):3–9.

Zhou, Y. & Talburt, J. R. (2011). Entity Identity Information Management (EIIM). In *Proceedings of the International Conference on Information Quality*, pages 327–241.

Ziegler, P. & Dittrich, K. R. (2007). Data Integration - Problems, Approaches, and Perspectives. In *Conceptual Modelling in Information Systems Engineering*, pages 39–58. Springer.