Reconciliação de dados usando MapReduce

Adriano L. Leão da Silva[1], **Dayse S. de Almeida**[1]

[1]Universidade Federal de Catalão (UFCAT) – Catalão, GO – Brasil

adrianolls.dev@gmail.com, daysesa@ufcat.edu.br

Abstract. *Currently, one of the main challenges in the field of data integration is the large data generated by several applications. In this paper, we propose Map and Reduce functions based on policies found in the literature for data integration using provenance data. These functions were applied in the distributed context. As a result, we obtained better computational performance with the Hadoop MapReduce compared to centralized computing of the functions, even considering a small data in experiments. Furthermore, the functions proved to be effective according to the policy considered, in both environments.*

Resumo. *Um dos principais desafios na área de integração de dados é o grande volume de dados gerado por aplicações atuais. Neste artigo são propostas funções de Map e Reduce para a integração de dados utilizando de dados de procedência. Essas funções são baseadas em políticas encontradas na literatura e, aplicadas no contexto distribuído. Como resultado, obteve-se um melhor tempo de execução com o Hadoop MapReduce, em comparação com a execução centralizada das funções, mesmo considerando um volume pequeno de dados nos experimentos. Além disso, as funções se mostraram eficazes de acordo com a política considerada, em ambos os ambientes.*

1. Introdução

Com o crescimento do poder computacional, a evolução dos sistemas de informação e o aumento da quantidade de serviços disponibilizados ao usuário, surgiram diferentes características nos dados gerados, como a falta de uma estrutura fixa e um grande volume, havendo necessidade de novas maneiras para manipulá-los. Para soluções de armazenamento permanente e gerenciamento de conjuntos de dados heterogêneos, sistemas de arquivos distribuídos e bancos de dados *Not Only* SQL (NoSQL) [Khan et al. 2023] têm sido usados. O modelo de processamento *MapReduce* [Dean 2008] se alinha com essas tecnologias devido à característica de atuar sobre dados não estruturados e ser capaz de operar grandes volumes de dados de maneira distribuída. Nesse modelo, o processamento dos dados é realizado em um *cluster* computacional pela aplicação de duas funções advindas do paradigma da linguagem funcional, as funções *Map* e *Reduce*.

Um dos passos primordiais para armazenamento, tratamento e análise de dados, é a integração dos dados, pois esses podem ter origem em fontes heterogêneas e apresentar diferentes formatos e domínios [Shrestha and Bhalla 2020, Yousif et al. 2021, Stojanović et al. 2022]. Um dos desafios da integração de dados, a resolução de conflitos [Kruse et al. 2020], refere-se ao problema de se decidir sobre inconsistências nos valores de atributos de uma mesma entidade, quando esses valores vêm de fontes diferentes. O foco neste trabalho está no problema de resolução de conflitos.

A resolução de conflitos envolve processos demorados, e normalmente realizados por um único usuário por meio de intervenção manual [Azuan 2021]. Assim, esses processos ficam inviabilizados diante de um grande volume de dados, abrindo uma possibilidade de experimentar-se o modelo *MapReduce*. Sendo assim, a proposta neste trabalho é utilizar o modelo de programação *MapReduce* para desenvolver funções de integração de dados. Especificamente, propõe-se funções *Map* e funções *Reduce* para a resolução de conflitos entre dados vindos de diferentes fontes. Essas funções são embasadas nas políticas de integração e reconciliação encontradas na literatura [Almeida et al. 2018] e executadas de forma distribuída utilizando o *framework* da Apache, Hadoop Mapreduce [Bhandarkar 2010, Apache Software Foundation 2025]. Os dados utilizados para a validação das funções desenvolvidas foram dados curriculares provenientes da plataforma Lattes.

O termo **reconciliação de dados** será usado no restante deste texto para se referir aos processos de integração realizados por múltiplos usuários, e o termo **integração de dados** será utilizado quando houver referência a processos realizados por único usuário diretamente sobre as fontes de dados ou cópias destas.

2. Fundamentação Teórica

O enfoque deste trabalho está na reconciliação e no compartilhamento de dados de maneira distribuída utilizando o modelo de programação *MapReduce*. Para isso, são propostas funções *MapReduce* para a reconciliação de dados, utilizando-se como base, políticas de reconciliação propostas no modelo AcCORD [Almeida et al. 2018].

O AcCORD é um modelo de reconciliação de dados colaborativo e assíncrono, que permite que diversos usuários trabalhem de forma cooperativa em um conjunto de dados de procedência tomando decisões sobre possíveis inconsistências. Para resolver inconsistências nos dados advindos de diferentes fontes, cada usuário pode utilizar uma ferramenta de integração de dados que o auxilie no processo e, armazena em um repositório, as ações realizadas. Normalmente esses processos, mesmo com uso de uma ferramenta de auxílio, requerem intervenção manual. A ferramenta utilizada para a integração de dados precisa fornecer as atualizações feitas pelo usuário no formato de dados de procedência [Cheney et al. 2009, Mahmood et al. 2013, Zheng et al. 2022]. Dados de procedência consistem no conjunto de metadados que possibilita identificar as fontes de dados e as transformações aplicadas sobre os dados, desde a criação até o estado atual desses.

O repositório utilizado no modelo AcCORD é formado por um conjunto de operações que armazenam os dados de procedência das atualizações feitas pelos usuários. Cada operação no repositório é composta pela sequência de atributos mostrados a seguir:

- **usuario:** identificador único do usuário que tomou a decisão e criou a operação;
- **op:** operação que reflete a decisão do usuário no processo de integração, sendo que as possíveis operações são edição, cópia, remoção ou inserção;
- **usuarioConf:** usuários que confiaram na operação em processos de reconciliação anteriores. Este campo não é utilizado neste trabalho;
- **origem:** fonte que provê o valor correto do atributo do objeto;
- **destino:** fonte na qual o valor do atributo foi corrigido pela *op*;
- **objeto:** valor da chave que identifica o objeto;
- **atributo:** nome do atributo no qual a *op* é realizada;

- **valorOrigem:** valor do atributo de *Origem*;
- **valorDestino:** valor do atributo de destino antes de ser sobrescrito pelo *valorOrigem*;
- **timestamp:** momento em que a *op* foi realizada.

Compartilhar decisões e trabalhar colaborativamente pode ser uma estratégia na economia de tempo e na confiança no resultado final. Mas, como cada usuário pode tomar decisões particularmente diferentes sobre as inconsistências nas fontes de dados, os repositórios individuais podem estar inconsistentes o final do processo de integração. Para solucionar o problema, foram propostas políticas de reconciliação no modelo AcCORD, que podem gerar uma visão única e consistente dos dados para todos os usuários, ou várias visões distintas para cada usuário, dependendo da necessidade particular de cada um deles. O modelo, no entanto, não se mostra adequado, em termos de eficiência, para ambientes com grande volume de dados. Isso se deve ao fato do processamento feito pelo módulo de reconciliação ser realizado localmente na máquina do usuário que solicita a reconciliação.

3. Funções de *MapReduce*

São propostas quatro funções de *MapReduce* para a reconciliação de dados, descritas a seguir.

1. A primeira função mantém as decisões do usuário que solicitou o processo de reconciliação. Sendo assim, se quaisquer operações p e o conflitam e p no valor do atributo é uma operação feita pelo usuário local, a política mantém p e remove o e as operações geradas a partir dessa subsequentemente. Essa política gera várias visões distintas para cada usuário.
2. A segunda remove todos os conflitos entre operações de atualização feitas por diferentes usuários. Se quaisquer operações p e o conflitam, a política remove p e o e todas as operações geradas subsequentemente a partir delas. Essa política gera uma única visão consistente dos dados para todos os usuários.
3. A terceira prioriza a ordem temporal das operações. Se quaisquer operações p e o conflitam e p é a operação mais recentemente realizada, a política mantém p e as operações subsequentes a essa. Atualizações necessárias para manter a consistência no repositório de dados são realizadas na operação o e nas operações subsequentes geradas a partir de o . Essa política gera uma única visão consistente dos dados.
4. A quarta função prioriza as operações que refletem a maioria das decisões sobre um valor, ou seja, aquelas operações cujo determinado valor atualizado aparece na maioria das operações. Se quaisquer operações p e o conflitam e p possui o valor sobre um atributo que representa a maioria das decisões dos usuários, a política mantém a operação p e suas operações subsequentes e remove a operação o. Atualizações necessárias para manter a consistência entre as operações, são realizadas nas operações susequentes à operação o . Caso haja empate na votação, a política remove as operações p e o e todas as operações subsequentes à elas. Essa política gera uma única visão consistente dos dados.

4. Materiais e Métodos

4.1. Base de Dados

Foram utilizados, neste trabalho, dados curriculares obtidos da plataforma Lattes, uma base pública de currículos de pesquisadores. Os repositórios de dados foram criados a partir da integração de dados de 4 currículos de pesquisadores do Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC-USP). Esses currículos foram selecionados entre 120 currículos de outros pesquisadores do instituto, devido à grande quantidade de publicações em comum, o que gerou um conjunto de dados com potencial para conflitos. Foram gerados 16 repositórios de dados com operações de *cópia*, *edição*, *remoção* e *inserção*, que representam as decisões de integração dos usuários. Os tamanhos dos repositórios variam de 390 KB com 749 operações a 40 KB com 79 operações.

Esses repositórios foram construídos e fornecidos por Almeida et al. [Almeida et al. 2018] e são os mesmos utilizados em seu trabalho para a avaliação de suas políticas de reconciliação. Eles foram empregados neste trabalho para a avaliação das funções de *MapReduce* desenvolvidas, permitindo assim, a análise da sua eficácia quando comparadas as políticas originais.

4.2. Ambiente Experimental

Para fins de comparação, as funções *MapReduce* foram executadas tanto utilizando o *framework* Hadoop *MapReduce* quanto em uma máquina local centralizada. O desempenho de cada abordagem foi avaliado com base no tempo de processamento gasto pelas políticas para tratar as operações conflitantes e na quantidade de operações consideradas corretas que permaneceram no repositório após a execução.

No primeiro experimento, as funções foram executadas em uma única máquina virtual, usando a aplicação VMWare para a virtualização. A máquina possui as seguintes configurações: 2 cores do processador Intel Core i5-9400F CPU de 2.90GHz, 4GB de memória RAM, 500GB de SSD e utiliza o sistema operacional Ubuntu 20.04.2 LTS 64-bit.

Na realização do segundo experimento, usou-se adicionalmente o *framework* Hadoop (versão 3.2.1) para a simulação do processamento distribuído. Essa ferramenta depende do Java OpenJDK e Javac (versão 1.8.0), que foram devidamente instalados. A configuração seguiu os padrões do modo *pseudo-distributed*, permitindo que cada conjunto de processos executados no Hadoop seja executado em um único processo Java, utilizando o mecanismo de *threads* disponível na arquitetura da linguagem de programação. Para a execução no Hadoop, foi utilizado o sistema HDFS do próprio *framework* para armazenar tanto o código das funções, quanto os repositórios de operações que constituem os dados de entrada da aplicação.

As funções de *Map* e *Reduce* foram desenvolvidas na linguagem Python (versão 3.8.5) e estão disponíveis no Github, no endereço: https://github.com/adriano-leao/MapreduceAcCORD.

5. Resultados e Discussões

Seguindo a metodologia descrita, o objetivo com os experimentos realizados foi avaliar a eficácia e a eficiência das funções *MapReduce* desenvolvidas, bem como o compor-

tamento das mesmas. Essas funções objetivam a reconciliação de dados em ambientes nos quais os dados são compartilhados por meio de repositórios. Para isso, é necessário que a ferramenta de integração utilizada previamente, gere um repositório de dados de procedência que representem as operações de atualização realizadas nos dados.

Cada uma das quatro funções implementadas, foi submetida a duas execuções diferentes. No primeiro experimento, as funções foram executadas em uma única máquina. No segundo, foi utilizado o *framework* Hadoop *MapReduce*, simulando o processamento em um ambiente distribuído. Nos dois experimentos considera-se que o mesmo usuário solicita o processo de reconciliação todas as vezes nas quais ele é aplicado. Todos os resultados foram obtidos por meio da média de cinco execuções em cada cenário, com um intervalo de confiança de 95%, assegurando assim, a sua confiabilidade.

Na Figura 1, são mostrados os tempos de execução gastos por cada política para manipular as operações de um número crescente de repositórios. Observa-se tempos melhores para as execuções de todas as funções no *framework* Hadoop *MapReduce*, em comparação com os tempos obtidos na execução centralizada das mesmas funções *MapReduce*, mesmo utilizando repositórios pequenos. Ambientes distribuídos tendem a prejudicar execuções realizadas sobre pequenos volumes de dados pois, muitas vezes, os tempos para distribuir os dados entre os nós e coletar e combinar os resultados, superam o tempo de execução em um único nó. Porém, não é isso que ocorre aqui, assegurando o melhor desempenho de funções *MapReduce* utilizando o Haddop.

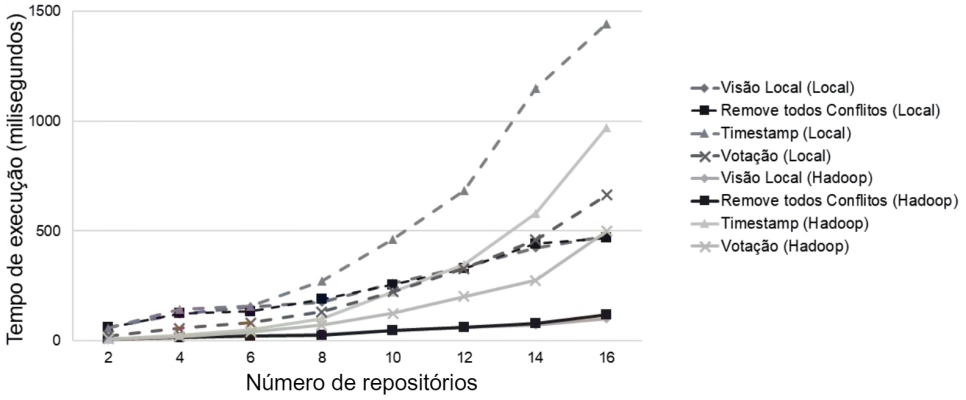

Figure 1. Comparação entre tempos gastos pelas funções *MapReduce* na execução centralizada e no Hadoop *MapReduce*.

Adicionalmente, destaca-se que na execução no Hadoop, as funções baseadas na visão local e na remoção de todos os conflitos exibiram o melhor desempenho. Isso se deve ao fato de que uma vez que as operações conflitantes são agrupadas, os conflitos não são verificados e essas operações não são atualizadas. Com o uso da primeira política, as operações realizadas pelo usuário que solicitou o processo de reconciliação são mantidas, e as operações conflitantes com essas, dos demais usuários, são descartadas. Com o uso da segunda política, todas as operações conflitantes são descartadas. Os tempos de execução confirmam a eficácia das funções, pois os seus comportamentos estão refletidos nos tempos gastos.

Também foi realizada uma análise do número de operações mantido no repositório após a conclusão do processo de reconciliação, considerando cada política e um número

crescente de repositórios envolvidos no processo, mostrados na Figura 2. Além disso, é mostrado o número inicial de operações existentes no repositório utilizado como referência. Ou seja, o número de operações no repositório do usuário que solicita o processo de reconciliação. Ele importa as operações dos demais repositórios dos demais antes de se realizar o processo de reconciliação.

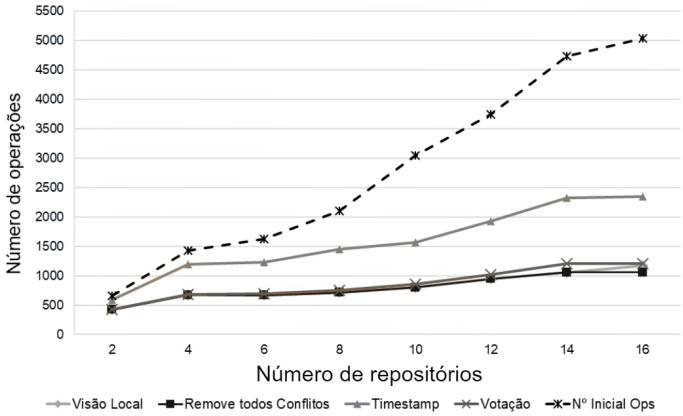

Figure 2. Número de operações mantidas no repositório após a aplicação de cada função de reconciliação.

Os resultados obtidos foram similares àqueles apresentados por Almeida et al. [Almeida et al. 2018] em relação ao comportamento das funções. A política que remove todas as operações conflitantes foi responsável por remover maior quantidade de operações do repositório, conforme o esperado. Esse resultado foi seguido de perto pelas funções baseadas na visão local e na votação. A primeira remove qualquer operação conflitante com as operações do usuário que solicita o processo de reconciliação. Já a segunda, apesar de se basear numa votação para decidir quais as operações conflitantes devem ser mantidas, em caso de empate na votação, ela remove todas as operações conflitantes assim como a política de remoção de todos os conflitos.

No cenário apresentado, a política baseada no *timestamp* mostrou-se mais eficaz em atualizar as operações que aquela baseada na votação, preservando-se assim um maior número no repositório resultante. Isso é intuitivamente, bem visto pelos usuários, pois eles não terão que tomar novas decisões para inconsistências que já foram tratadas por eles anteriormente.

6. Conclusão

Neste trabalho, políticas de reconciliação de dados em nível de instância foram exploradas no contexto do modelo de processamento *MapReduce* em um ambiente centralizado e em um ambiente distribuído. Os resultados obtidos revelaram que as estratégias distribuídas têm potencial para aprimorar significativamente a eficiência da reconciliação de dados, tendo em vista que em todos os casos testados, obteve-se um desempenho melhor ao utilizar-se a abordagem distribuída em comparação com a forma centralizada.

Tendo em vista que as principais limitações deste trabalho são, o pequeno volume de dados utilizado nos experimentos e a sua execução realizada em um único servidor Hadoop, o que resultou em uma simulação do processamento distribuído, recomenda-se que em pesquisas subsequentes, essas abordagens sejam exploradas em ambiente de

cluster distribuído, com foco na execução de dados em larga escala. O uso de conjuntos de dados substanciais é essencial para obter avaliações mais precisas do desempenho das funções de reconciliação em ambientes distribuídos. Essas investigações têm o potencial de preencher as lacunas deixadas por este trabalho e, ampliar a compreensão da reconciliação de dados em contextos mais desafiadores.

References

Almeida, D. S.; Hara, C. S., Ciferri, R. R., and Ciferri, C. D. A. (2018). An asynchronous collaborative reconciliation model based on data provenance. *Software: Pratice and-Experience*, 48(1):197–232.

Apache Software Foundation (2025). Apache hadoop. https://hadoop.apache.org. Acessado em: 13 jun. 2025.

Azuan, N. A. A. (2021). *Exploring Manual Correction as a Source of User Feedback in Pay-As-You-Go Integration*. PhD thesis, The University of Manchester.

Bhandarkar, M. (2010). Mapreduce programming with apache hadoop. In *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, pages 1–1.

Cheney, J., Chiticariu, L., Tan, W.-C., et al. (2009). Provenance in databases: Why, how, and where. *Foundations and Trends® in Databases*, 1(4):379–474.

Dean, J.; Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM - 50th anniversary issue: 1958 - 2008*, 51(1):107–113.

Khan, W., Kumar, T., Zhang, C., Raj, K., Roy, A. M., and Luo, B. (2023). Sql and nosql database software architecture performance analysis and assessments—a systematic literature review. *Big Data and Cognitive Computing*, 7(2).

Kruse, F., Hassan, A. P., Awick, J.-P., Gómez, J. M., and Bui, T. (2020). A qualitative literature review on linkage techniques for data integration. In *HICSS*, pages 1–11.

Mahmood, T., Jami, S. I., Shaikh, Z. A., and Mughal, M. H. (2013). Toward the modeling of data provenance in scientific publications. *Computer Standards & Interfaces*, 35(1):6–29.

Shrestha, S. and Bhalla, S. (2020). Survey on the evolution of models of data integration. *Int. J. Knowl. Based Comput. Syst*, 8:11–16.

Stojanović, A., Horvat, M., and Kovačević, Z. (2022). An overview of data integration principles for heterogeneous databases. In *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)*, pages 1111–1116.

Yousif, O., Zakaria, R., Aminudin, E., Yahya, K., Sam, A., Singaram, L., Munikanan, V., Yahya, M., Wahi, N., and Shamsuddin, S. (2021). Review of big data integration in construction industry digitalization. *Frontiers in Built Environment*, 7.

Zheng, L., Pan, J., and Zhang, K. (2022). Power data integration method based on database-table metadata semantic. *Journal of Physics: Conference Series*, 2179(1):012028.