Otimização de *Dataflows* em *Frameworks* de *Big Data* por meio do Reúso de Dados

Gustavo Decarlo[1], **Daniel de Oliveira**[2], **Fabio Porto**[1]

[1] Laboratório Nacional de Computação Científica (LNCC) – DEXL Lab – Petrópolis, RJ – Brasil

[2]Universidade Federal Fluminense - Instituto de Computação (IC/UFF) – Niterói, RJ – Brasil

gdecarlo@posgrad.lncc.br, danielcmo@ic.uff.br, fporto@lncc.br

Abstract. *The adoption of Big Data frameworks has increased in recent years. They are considered the state of the art in supporting the parallel and distributed execution of data transformation applications. The latter are commonly composed of several activities that process a large volume of data. In order to optimize the execution of dataflows, Big Data framework schedules tasks exploring data locality, avoiding unnecessary transfers of data. However, such optimizations consider isolated-independent executions, i.e. do not consider sharing data among different dataflow executions. As it turns out, this kind of data reuse can speed up dataflows. This paper presents an approach for sharing temporary data among dataflows. We present an architecture that allows multiple executions of dataflows to share intermediate results, reducing execution time. We evaluated the proposed approach with a real COVID-19 data processing dataflows.*

Resumo. *O uso de arcabouços de Big Data tem aumentado nos últimos anos. Esses arcabouços representam um avanço no que tange o apoio à execução paralela e distribuída de aplicações. Essas aplicações são frequentemente compostas de diversas atividades, gerando assim um dataflow, que em geral processa um grande volume de dados. Por mais que os arcabouços sejam otimizados para explorar localidade dos dados evitar transferências desnecessárias no ambiente distribuído, tais otimizações são focadas em execuções isoladas, i.e. não consideram aproveitar dados de execuções anteriores. Esse tipo de reúso de dados pode acelerar dataflows, uma vez que o dado não precisa ser processado novamente caso já tenha sido produzido por uma execução anterior do mesmo dataflow. Este artigo apresenta uma abordagem para o compartilhamento de dados gerados nos dataflows. Discutimos e implementamos uma arquitetura que permite que múltiplas execuções de dataflows possam compartilhar resultados intermediários, reduzindo tempo de execução. Avaliamos a abordagem com dataflows reais de processamento de dados da COVID-19.*

1. Introdução

Nos últimos anos podemos observar um aumento do número de aplicações que processam um grande volume de dados (*i.e.*, *Big Data*), sejam elas da área científica ou da indústria [de Oliveira et al. 2019]. Muitas dessas aplicações são compostas por múltiplas tarefas encadeadas em um fluxo coerente, *i.e.*, um *dataflow*. A adoção de *dataflows* como modelos para implementação de aplicações permite a utilização de linguagens para

representação do fluxo, o que comumente simplifica o desenvolvimento da mesma. Existem diversos arcabouços que permitem a implementação de *dataflows* por meio de linguagens declarativas. Esses arcabouços permitem além da especificação do *dataflow*, o escalonamento das tarefas em ambientes distribuídos e frequentemente de alto desempenho. Alguns exemplos desses arcabouços são o Apache Spark [Zaharia et al. 2016], o Apache Storm [Jain 2017], o Apache Airflow [Airflow 2022], o Azkaban [Azkaban 2022] e o Prefect [Prefect 2022]. Apesar desses arcabouços oferecerem mecanismos eficientes para execuções distribuídas de *dataflows*, o tempo de execução de um *dataflow* ainda pode ser demasiadamente elevado, dependendo do volume de dados a ser processado. Esse tempo de execução poderia ser reduzido caso dados intermediários previamente produzidos pudessem ser reaproveitados, evitando assim a execução de certas tarefas computacionalmente intensivas do *dataflow* [Heidsieck et al. 2020]. A grande maioria dos arcabouços tratam execuções de múltiplos *dataflows* de forma independente, *i.e.*, o que foi produzido em uma execução não pode ser reutilizado em outra, mesmo em casos em que os parâmetros de entrada das duas execuções são idênticos, o que fará com que os resultados finais e intermediários sejam exatamente iguais. Até mesmo no caso do Spark e do Storm, que oferecem um certo nível de compartilhamento de dados, esse compartilhamento é restrito a uma mesma sessão em um mesmo *cluster* computacional [Gottin et al. 2018]. Entretanto, permitir tal compartilhamento de dados somente na mesma sessão e no mesmo *cluster* pode ser um limitante para otimizar a execução de diversos *dataflows*.

Considerando o cenário apresentado anteriormente, é possível observar que há uma oportunidade de otimização dos *dataflows* por meio do compartilhamento de dados intermediários, mesmo que sejam produzidos em diferentes sessões e diferentes *clusters*. Neste artigo, apresentamos uma abordagem chamada `FORESEE` (*dataFlOw REuSE layEr*), que é uma camada de *software* acessível por múltiplos *dataflows*, e que oferece uma visão global dos dados intermediários gerados de forma que possam ser reutilizados. A abordagem foi avaliada com um *dataflow* de processamento de dados da COVID-19 e os resultados apresentaram ganhos de até 50% no tempo de execução do *dataflow*. Esse artigo se encontra organizado em 4 seções além da Introdução. Na Seção 2 apresentamos a definição do problema. Em seguida, na Seção 3 apresentamos a arquitetura proposta do `FORESEE`. Já na Seção 4 são descritos os resultados experimentais. Por fim, na Seção 5 apresentamos a conclusão e os trabalhos futuros.

2. Definição do Problema

Neste artigo, consideramos como padrão os *dataflows* implementados no arcabouço Apache Spark. Nesse arcabouço são estabelecidas relações produtor-consumidor entre as operações de processamento de dados disponíveis. Em especial, no Spark existem dois tipos de operações: (i) Transformações e (ii) Ações. As transformações são responsáveis por mapear o conjunto de dados de entrada em um conjunto de dados de saída, enquanto ações retornam valores resultantes da execução da operação em uma estrutura de dados. O encadeamento de um par de operações, como uma relação produtor-consumidor, define o que chamamos neste trabalho de fragmento de *dataflow*. Além disso, ao conjunto de dados produzido após a execução das operações em um fragmento de *dataflow* chamamos de fragmento de dados.

Assim, podemos definir que uma tarefa $t(I),\ t : I \rightarrow O$ pode ser representada como uma função que mapeia um conjunto de dados de entrada I para um conjunto de

dados de saída O. Uma relação produtor-consumidor $\phi = \langle s, t, t' \rangle$ pode ser representada como uma tripla que é composta de um conjunto de dados s e duas tarefas t e t'. Dessa forma, um *dataflow* $D = \langle T, S, \Phi \rangle$ pode ser representado como uma tripla com as tarefas T um conjunto de dados de entrada S, e um conjunto de relações produtor-consumidor Φ.

Dessa forma, de forma a fomentar o reuso de dados entre fragmentos de múltiplos *dataflows*, precisamos responder algumas questões: (i) Uma vez que um dado intermediário se encontra disponível, como identificar que podemos reutilizar um dado armazenado ao invés de executar uma tarefa do *dataflow*? e (ii) Como avaliar o benefício de se acessar um fragmento de dados? Tomemos como exemplo os dois *dataflows* apresentados na Figura 1. Os dois *dataflows* consomem os mesmos dois conjuntos de dados como entrada (*i.e.*, i_1 e i_2). Consideremos que o *dataflow* A armazenou o resultado intermediário dc após a execução da tarefa t_3 e que o *dataflow* B pode reutilizar tais dados. Assim, temos que $dc \leftarrow t_3(t_2(t_1(i_1, i_2)))$. Para responder a questão (i) devemos verificar se $t_3(t_2(t_1(i_1, i_2))) \equiv t_7(t_6(t_5(i_1, i_2)))$. Caso os dois fragmentos produzam a mesma saída, para responder a questão (ii) devemos verificar se o tempo de transferência de dc para o *cluster* que o *dataflow* B está executando é menor que o tempo de execução do fragmento $t_7(t_6(t_5(i_1, i_2)))$, *i.e.*, $exectime(t_7(t_6(t_5(i_1, i_2)))) \geq tranf(dc)$. Na próxima seção apresentamos a arquitetura proposta da abordagem FORESEE para reuso de dados entre *dataflows*.

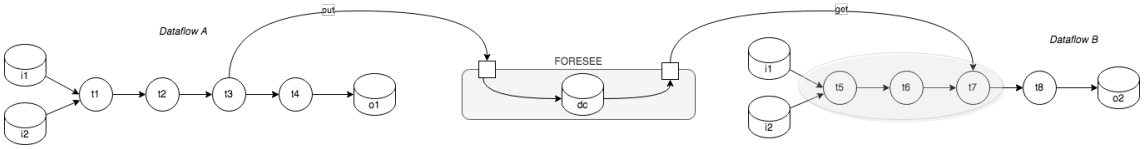

Figura 1. Exemplo de reuso de dados entre duas execuções de *dataflows*

3. Abordagem Proposta: FORESEE

Conforme discutido anteriormente, a abordagem FORESEE tem como objetivo prover dados intermediários produzidos por execuções de *dataflows* de forma que possam ser reutilizados por novas execuções de *dataflows*. A arquitetura da FORESEE é apresentada na Figura 2. A arquitetura é composta de seis componentes: (i) API, (ii) *Cache Broker*, (iii) Gerenciador de Fragmentos de Dados, (iv) Gerenciador de Metadados, (v) Sistema de Arquivos e (vi) Banco de Metadados.

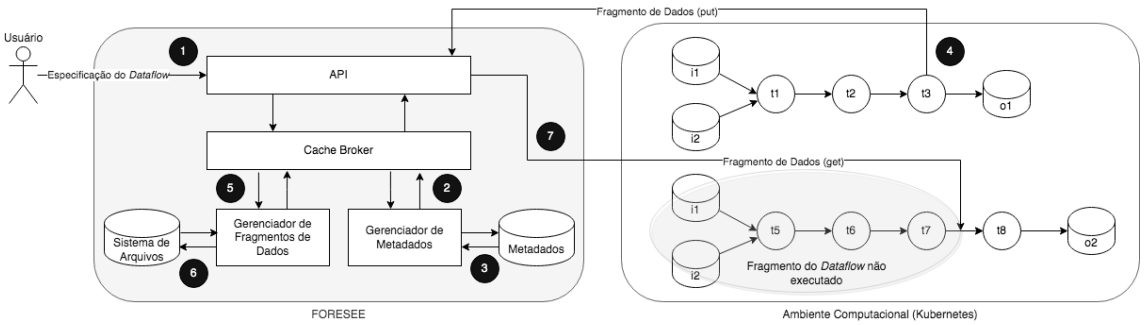

Figura 2. Arquitetura da Abordagem FORESEE

O processo de uso da FORESEE se inicia com o usuário submetendo a especificação do *dataflow* (Passo ❶) para a API do FORESEE. A API em sua versão atual foi desenvolvida para o *pyspark* e disponibiliza as funções de *salvar* (*i.e.*, put_datafrag, apresentada na Listing 1), quanto a função de *resgatar*. A especificação enviada para a API deve conter todas as tarefas e dependências de dados entre as tarefas. De posse da especificação do *dataflow*, a API envia essa informação para o *Cache Broker*, que identifica qual tipo de mensagem está recebendo e a encaminha para o componente gerenciador específico. Nesse caso, a especificação do *dataflow* é enviada ao Gerenciador de Metadados (Passo ❷), que armazena em um banco de metadados (Passo ❸) as informações. Em sua versão atual, o banco de metadados se encontra implementado no SGBD Cassandra, uma vez que ele apresenta baixa latência principalmente na ingestão de dados [Lakshman and Malik 2009].

Uma vez que os metadados se encontram carregados no banco de metadados, a execução dos *dataflows* pode ser iniciada. Uma vez que um *dataflow* se encontra em execução, chamadas são feitas a API (Passo ❹) para gravar dados intermediários (*i.e.*, *put*). Esses dados são repassados pela API ao *Cache Broker* que envia ao Gerenciador de Fragmentos de Dados (Passo ❺), que os grava no sistema de arquivos (Passo ❻), que pode ser local ou distribuído. Atualmente, o FORESEE utiliza o formato *Delta lake* para armazenar os fragmentos de forma que estejam otimizados para *storages* da nuvem como o *Google Cloud Storage* [Armbrust 2020]. Também adotamos o *cluster kubernetes* para facilitar a utilização de todos esses serviços com suas configurações próprias de recursos computacionais. No momento em que um segundo *dataflow* inicia sua execução, ele envia uma solicitação a API para verificar se há fragmentos de dados que podem ser reutilizados (Passo ❼). Caso existam fragmentos no sistema de arquivos, a API verifica se o mesmo apresenta menor custo de transferência em detrimento da reexecução das tarefas do *dataflow*.

Listing 1. Função em python da API que salva o fragmento de dados no sistema de arquivos

```
def put_datafrag(self, datafrag_ref: str, datafrag: DataFrame) -> None:
    metadata = self._extract_metadata(datafrag_ref, datafrag)
    schema = self._schema_operation()
    df_operation = self.spark_session.createDataFrame(
        data=[metadata], schema=schema
    )
    (df_operation.write.format('org.apache.spark.sql.cassandra')
        .mode('append')
        .options(table=self.table, keyspace=self.keyspace)
        .save()
        )
        (datafrag.write
        .format('delta')
        .mode('overwrite')
        .save(f'{self.datafrag_wareghouse}/{datafrag_ref}')
    )
```

4. Avaliação Experimental

De forma a avaliar a FORESEE, utilizamos um *dataflow* que processa o conjunto de dados abertos da COVID-19, oferecidos pela *Center for Systems Science and Engineering*

(CSSE) da Johns Hopkins University [Hopkins 2022]. O conjunto de dados descreve as ocorrências, mortes e recuperações de pacientes com COVID-19 por data, país e região do país. O *dataflow* para processamento desses dados possui duas tarefas. A primeira realiza uma agregação dos dados por país e no tempo. A saída dessa tarefa é um arquivo contendo os dados agregados. A segunda tarefa realiza o filtro de acordo com o país e a data desejada.

Para os experimentos, consideramos três execuções do *dataflow* previamente explicado: (i) Execução *Baseline* - o *dataflow* agrega os dados por país e depois filtra as ocorrências do Brasil sem gravar dados intermediários nem reutilizar dados, (ii) Somente Carga - o *dataflow* agrega os dados por país e depois filtra as ocorrências do Brasil, mas grava os dados intermediários, e (iii) Com Reuso - o *dataflow* reutiliza os dados agregados no *dataflow* do item (ii) e depois filtra as ocorrências da Alemanha. As execuções foram realizadas em uma máquina com processador Intel i7 com 16 GB RAM. A Figura 3 apresenta o tempo de execução em segundos para 10 execuções de cada um dos *dataflows*.

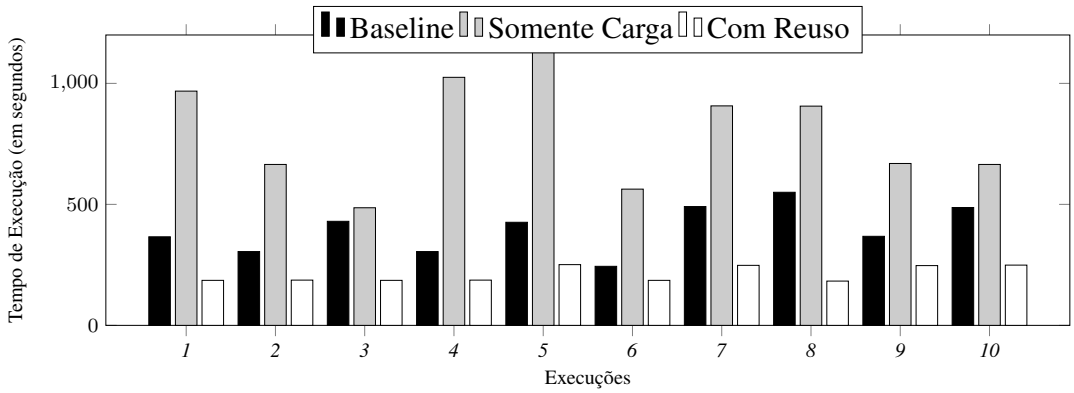

Figura 3. Tempo de Execução dos *dataflows* - 10 execuções

Ao analisarmos os resultados apresentados na Figura 3, podemos perceber que para execuções do *dataflow* que não reutilizam dados mas armazenam os fragmentos de dados, o *overhead* imposto no tempo de execução pode chegar a 270% no pior caso. Entretanto, é importante ressaltar que essa diferença em termos absolutos (no pior caso) é de 721 segundos (aproximadamente 12 minutos). Esse *overhead* é compensado pela aceleração obtida no caso em que o *dataflow* reutiliza os dados armazenados. No caso do *dataflow* com reúso, podemos perceber que o reúso dos fragmentos armazenados acelerou a execução do *dataflow* 50% em média, se comparado ao *baseline*. Cabe aqui também ressaltar, que o tamanho do fragmento de dados a ser salvo impacta diretamente no tempo de execução do *dataflow*, mas esse é um campo ainda aberto para otimizações. Além disso, é importante ressaltar que o benefício do reúso de um fragmento é diretamente dependente do número de *dataflows* que possam se beneficiar do processamento dos dados intermediários previamente armazenados. Essa relação custo x benefício é semelhante a criação de índices em bancos de dados, uma vez que o espaço ocupado em disco só é compensado caso existam consultas que utilizam tais índices.

5. Conclusão e Trabalhos Futuros

O uso de *dataflows* para processamento de grandes volumes de dados tem aumentado nos últimos anos, tanto na área científica quanto na indústria. Ainda que existam arcabouços

que processam esses *dataflows* em ambientes distribuídos, a sua execução ainda pode ser bastante demorada. Uma forma de reduzir esse tempo de execução é tentando reutilizar dados intermediários produzidos por execuções passadas de *dataflows*. Esse artigo apresenta a abordagem `FORESEE` que permite o registro de dados intermediários de *dataflows* para futuro reúso. Experimentos executados com a `FORESEE` e *dataflows* de processamento de dados da COVID-19 mostraram que podemos obter até 50% de aceleração na execução de um *dataflow* caso dados sejam reutilizados. Apesar de representar um avanço, experimentos em maior escala ainda se fazem necessários. Podemos também vislumbrar alguns pontos para trabalhos futuros. Um desses pontos é como comparar metadados de diferentes *dataflows* de uma forma simplificada, uma vez que a semântica da representação pode variar. Além disso, a identificação em tempo real de fragmentos semelhantes também se mostra como um desafio (nesse artigo assumimos que o usuário informou fragmentos semelhantes).

Referências

[Airflow 2022] Airflow, A. (2022). Apache airflow. https://airflow.apache.org/.

[Armbrust 2020] Armbrust, M. (2020). Delta lake: High-performance acid table storage over cloud object stores. In *Databricks*. Databricks.

[Azkaban 2022] Azkaban (2022). Azkaban. https://azkaban.github.io/.

[de Oliveira et al. 2019] de Oliveira, D. C. M., Liu, J., and Pacitti, E. (2019). *Data-Intensive Workflow Management: For Clouds and Data-Intensive and Scalable Computing Environments*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

[Gottin et al. 2018] Gottin, V., Pacheco, E., Dias, J., Ciarlini, A., Costa, B., Vieira, W., Souto, Y., Pires, P., Porto, F., and Rittmeyer, J. (2018). Automatic caching decision for scientific dataflow execution in apache spark. In *Proc. of the BeyondMR*, pages 1–10.

[Heidsieck et al. 2020] Heidsieck, G., de Oliveira, D., Pacitti, E., Pradal, C., Tardieu, F., and Valduriez, P. (2020). Distributed caching of scientific workflows in multisite cloud. In *Database and Expert Systems Applications*, volume 12392, pages 51–65. Springer.

[Hopkins 2022] Hopkins, U. J. (2022). Covid-19 data repository by the center for systems science and engineering at johns hopkins university. https://github.com/cssegisanddata/covid-19.

[Jain 2017] Jain, A. (2017). *Mastering Apache Storm: Real-Time Big Data Streaming Using Kafka, Hbase and Redis*. Packt Publishing.

[Lakshman and Malik 2009] Lakshman, A. and Malik, P. (2009). Cassandra - a decentralized structured storage system. In *Cs Cornell*. Cs Cornell.

[Prefect 2022] Prefect (2022). Prefect - the new standard in dataflow automation - prefect. https://www.prefect.io/.

[Zaharia et al. 2016] Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., Ghodsi, A., Gonzalez, J., Shenker, S., and Stoica, I. (2016). Apache spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65.