ANÁLISE DE DADOS CIENTÍFICOS SOBRE MÚLTIPLAS FONTES DE DADOS AO LONGO DA EXECUÇÃO DE SIMULAÇÕES COMPUTACIONAIS

Vítor Silva Sousa

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientadores: Marta Lima de Queirós Mattoso
Daniel Cardoso Moraes de Oliveira
Patrick Valduriez

Rio de Janeiro
Junho de 2018

ANÁLISE DE DADOS CIENTÍFICOS SOBRE MÚLTIPLAS FONTES DE DADOS AO LONGO DA EXECUÇÃO DE SIMULAÇÕES COMPUTACIONAIS

Vítor Silva Sousa

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Profª. Marta Lima de Queirós Mattoso, D.Sc.

Prof. Daniel Cardoso Moraes de Oliveira, D.Sc.

Prof. Alexandre de Assis Bento Lima, D.Sc.

Profª. Maria Cristina Silva Boeres, Ph.D.

Prof. Leonardo Guerreiro Azevedo, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
JUNHO DE 2018

Sousa, Vítor Silva

Análise de Dados Científicos sobre Múltiplas Fontes de Dados ao longo da Execução de Simulações Computacionais / Vítor Silva Sousa. – Rio de Janeiro: UFRJ/COPPE, 2018.

XV, 198 p.: il.; 29,7 cm.

Orientadores: Marta Lima de Queirós Mattoso

Daniel Cardoso Moraes de Oliveira

Patrick Valduriez

Tese (doutorado) – UFRJ / COPPE / Programa de Engenharia de Sistemas e Computação, 2018.

Referências Bibliográficas: p. 192-198.

1. Simulações computacionais. 2. Fluxo de dados. 3. Dados científicos. I. Mattoso, Marta Lima de Queirós *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III Título.

À minha família.

AGRADECIMENTOS

À Profª. Marta Mattoso, Prof. Patrick Valduriez e Prof. Daniel de Oliveira, meus orientadores, por terem acreditado no meu potencial, pelos comentários e críticas valiosos no amadurecimento desta tese, pelas sugestões quanto ao meu encaminhamento acadêmico e profissional, assim como pelas excelentes oportunidades oferecidas;

Ao Prof. Alvaro Coutinho por ter me proporcionado um ambiente incomparável para aquisição de conhecimento, aplicação dos conceitos aprendidos em aulas, experiências de vida e mesmo a difícil habilidade de separar o ambiente do trabalho com o da amizade;

À minha família por todo o apoio emocional para o desenvolvimento desta tese e a realização das atividades em diversos projetos de pesquisa;

Aos membros da banca por aceitarem em participar da defesa da minha tese de doutorado;

Ao Thaylon Guedes, Luciano Leite, José Vitor Delgado, Thiago Perrotta, Débora Pina, Vinícius Campos e Renan Souza pela parceria no desenvolvimento dos diversos projetos de pesquisa, assim como por terem me auxiliado na obtenção dos resultados desta tese;

À Mara Prata, Gutierrez da Costa, Claudia Prata e Ricardo Vieira por sempre me ajudarem com as questões administrativas;

E a todos os amigos, que me apoiaram ao longo do curso e acreditaram no meu potencial para o desenvolvimento desta tese. Amigos que manifestaram em pequenos atos esse sentimento tão difícil de ser mensurado;

Agradeço.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

ANÁLISE DE DADOS CIENTÍFICOS SOBRE MÚLTIPLAS FONTES DE DADOS AO LONGO DA EXECUÇÃO DE SIMULAÇÕES COMPUTACIONAIS

Vítor Silva Sousa

Junho/2018

Orientadores: Marta Lima de Queirós Mattoso
Daniel Cardoso Moraes de Oliveira
Patrick Valduriez
Programa: Engenharia de Sistemas e Computação

Simulações computacionais em larga escala são caracterizadas pelo encadeamento de programas que executam modelos computacionais cada vez mais complexos. Muitos dos dados produzidos por esses programas precisam ser analisados pelos usuários do domínio científico a fim de validar as suas hipóteses científicas. Entretanto, esta não é uma tarefa trivial, pois outros programas precisam ser desenvolvidos para acessar e capturar esses dados científicos. Em muitos casos, os usuários também precisam relacionar dados produzidos por diferentes programas de simulação. Esta tese propõe uma abordagem capaz de monitorar, depurar e analisar o fluxo de elementos de dados produzido pelos diferentes programas de simulação. Propomos também uma arquitetura baseada em componentes, nomeada como ARMFUL, que permite extrair e relacionar dados científicos produzidos nessas diversas etapas por meio da abstração de fluxo de dados e de técnicas de captura de dados científicos. Os seus componentes podem ser instanciados em um sistema de *workflows* científicos (A-Chiron) ou uma biblioteca de componentes (DfAnalyzer). Avaliamos essas instâncias utilizando simulações em ambientes de processamento de alto desempenho. Os resultados experimentais mostram que a nossa abordagem introduz uma sobrecarga negligenciável em relação ao tempo de execução da simulação, além de permitir o processamento de consultas aos dados científicos.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

ANALYSIS OF RAW DATA FROM MULTIPLE DATA SOURCES DURING THE EXECUTION OF COMPUTATIONAL SIMULATIONS

Vítor Silva Sousa

June/2018

Advisors: Marta Lima de Queirós Mattoso

Daniel Cardoso Moraes de Oliveira

Patrick Valduriez

Department: Systems and Computer Engineering

Large-scale computational simulations are characterized by the chaining of programs that execute increasingly complex computational models. Much of the data produced by these programs need to be analyzed by scientific domain users to validate their scientific hypotheses. However, it is not trivial since other programs must be developed to access and to capture these scientific data. In many cases, users also need to relate data produced by different simulation programs. This thesis proposes an approach that monitors, debugs, and analyzes the data element flow produced by different simulation programs. We also propose a component-based architecture, named as ARMFUL, to extract and relate scientific data generated in these several simulation steps considering a dataflow abstraction and techniques for scientific data capture. ARMFUL's components can be instantiated on a scientific workflow system (*e.g.*, A-Chiron) or a library of components (*e.g.*, DfAnalyzer). We evaluate these instances using simulations in high performance computing environments. In our experimental results, our approach introduced a negligible overhead of the simulation execution time, and we perform complex queries to the scientific data.

ÍNDICE

LISTAGEM DE FIGURAS

LISTAGEM DE TABELAS

Capítulo 1 - Introdução

As aplicações de Ciência Computacional e Engenharia (CSE) são baseadas em modelos computacionais que resolvem problemas que normalmente requerem Processamento de Alto Desempenho (PAD) (RÜDE *et al.*, 2016). As aplicações de CSE não estão vinculadas a um domínio específico. Elas podem ser encontradas em biologia, astronomia, geologia, várias áreas de engenharia, etc. Elas têm a natureza exploratória de aplicações científicas, ao mesmo tempo em que têm que lidar com execuções em grande escala, que duram por um longo tempo, mesmo quando usam PAD. O ecossistema de software para desenvolver essas aplicações envolve muito mais que escrever *scripts* ou invocar uma cadeia de códigos científicos legados (RÜDE *et al.*, 2016). Cientistas computacionais desenvolvem seus códigos de aplicações de CSE baseados em uma modelagem matemática complexa que resulta na chamada de componentes de ambientes e bibliotecas de CSE. As aplicações de CSE (ou simulações computacionais) são caracterizadas pelo encadeamento de chamadas desses componentes que muitas vezes já estão preparados para a execução paralela otimizada.

No entanto, vários parâmetros precisam ser definidos para chamar esses componentes. Essa escolha de valores é muito difícil de ser predefinida e precisa de monitoramento para ajustes em tempo de execução. Para realizar o monitoramento de uma simulação computacional é necessário acompanhar a evolução da geração de dados científicos ao longo da execução. Uma aplicação de CSE gera um grande volume de dados científicos (PENNISI, 2011). Esses dados se caracterizam por estarem fragmentados em um grande número de partições devido ao PAD. Além das partições de dados de arquivos, associadas a uma etapa da simulação, outros arquivos são gerados a cada etapa do encadeamento de componentes da aplicação de CSE. Acompanhar a evolução desses dados requer acessar os dados científicos diretamente no sistema de arquivos, o que é uma atividade custosa devido à quantidade, tamanho, estrutura de dados e formato padrão desses arquivos. Esse acesso necessita da escrita de programas específicos para identificar e percorrer os arquivos envolvidos na análise, relacionando os dados de seus conteúdos, o que se caracteriza por um acesso *ad-hoc* aos dados. Outra dificuldade é associar os dados dos arquivos aos parâmetros e etapas da simulação em andamento, além de arquivos de visualizações que são gerados para o monitoramento (SILVA *et al.*, 2016c).

O monitoramento da execução de uma aplicação de CSE necessita de um apoio centrado em dados para lidar com vários desafios de prover uma análise de dados capaz de oferecer, durante a execução, um panorama da evolução com o comportamento dos dados, imagens e parâmetros definidos na composição da simulação. As soluções existentes se encontram em dois extremos. De um lado está o acesso *ad-hoc* aos dados em arquivos, sem recursos de análise. Do outro lado estão os Sistemas de Gerência de Bancos de Dados (SGBD) com muitos recursos de análise para dados em grandes volumes, mas sem recursos de acesso às estruturas e partições de arquivos de dados científicos.

Existem algumas soluções de SGBDs para a análise de dados científicos, como o SciDB (BROWN, 2010), PostgresRaw (ALAGIANNIS *et al.*, 2015) e SDS/Q (DONG *et al.*, 2013), que caracterizam-se por soluções *post-mortem*, ou seja, só são viáveis de serem adotadas após a execução da simulação. Além disso, essas soluções são voltadas para a análise de dados do arquivo final resultante da aplicação. Elas não são preparadas para realizar consultas envolvendo dados de múltiplos arquivos, principalmente quando os relacionamentos entre os arquivos de dados são implícitos, ou seja, decorrem do encadeamento das etapas da simulação computacional. Aplicações de CSE necessitam da análise de dados durante a execução, pois mesmo em ambientes de PAD, elas podem durar horas, dias, ou, às vezes, meses em execução. Essa análise de dados necessita ter acesso aos relacionamentos implícitos entre os arquivos de dados.

Surge então o problema geral que norteia esta tese, que consiste em como prover uma solução para a análise de dados científicos sobre múltiplos arquivos ao longo da execução de simulações computacionais. Nesse contexto, tal solução possui os seguintes desafios: (i) a dificuldade de transformar os dados de forma que eles sejam passíveis de serem consultados em tempo de execução, (ii) como é possível relacionar dados gerados em diferentes etapas da simulação, e (iii) como prover uma solução que não interfira no desempenho computacional das simulações computacionais.

Diante disso, esta tese defende a hipótese de que, ao prover uma visão global dos dados com seus relacionamentos gerados na simulação computacional, a análise de dados científicos sobre múltiplos arquivos pode ser realizada durante a execução das simulações. Assim, tal solução passa por uma alternativa centrada em dados, que disponibilize os dados

científicos relacionados entre si para análise, durante a execução em ambiente PAD, sem as limitações do acesso *ad-hoc* sobre os arquivos e do acesso ao SGBD *post-mortem*.

Uma primeira abordagem para o problema de relacionar dados gerados em diferentes etapas de uma simulação computacional seria o uso de modelos de dados de proveniência (FREIRE *et al.*, 2008). Modelos de dados de proveniência permitem a representação de fluxos de dados registrando o histórico dos dados consumidos, propagados e produzidos pelos componentes da simulação. A grande vantagem no uso desse modelo advém do esforço de padronização de sua representação, realizado pelo consórcio W3C com o PROV (MOREAU and GROTH, 2013). Ao adotar um modelo genérico de representação, a identificação dos atores das consultas fica facilitada, independente do domínio de dados da aplicação, tornando-se uma abordagem genérica. Outra vantagem dessa representação é que os relacionamentos entre os dados são registrados a partir de identificadores ou valores escalares dos dados científicos que fluem ao longo da execução, o que facilita o seu registro em um SGBD qualquer. Ou seja, ao invés de se inserir todos os dados na base de dados de proveniência, apenas alguns valores são inseridos. Esses dados e seus relacionamentos oferecem um amplo panorama da evolução dos dados e seus relacionamentos decorrentes das etapas de execução. Finalmente, dados de proveniência também são importantes para melhorar as soluções de análise de dados *post-mortem* e principalmente para a reprodução dos resultados da simulação.

O Projeto *Interoperable Design of Extreme-scale Application Software* ("IDEAS productivity") é uma iniciativa de 2018 que contempla uma família de projetos, envolvendo várias instituições e laboratórios nos EUA, preocupados com a complexidade do desenvolvimento de software para aplicações de CSE. O IDEAS visa a "permitir uma atitude fundamentalmente diferente de criar e apoiar aplicações de CSE" com características desejáveis, como proveniência e reprodutibilidade (BERNHOLDT *et al.*, 2017). Na verdade, os dados de proveniência podem ajudar no registro de opções de parâmetros da simulação. Associá-los aos resultados pode melhorar tanto o ajuste fino quanto as análises de dados em tempo de execução.

São muitas as soluções de análise de dados baseadas no registro dos dados de proveniência (OLIVEIRA *et al.*, 2018). Para o uso de dados de proveniência na análise de dados, é necessário: um mecanismo de captura de dados; a carga desses dados em uma

ferramenta de análise de dados, tipicamente um SGBD; e um mecanismo de consulta. No entanto, tais soluções possuem diversas limitações para o uso em análises de dados ao longo da execução de aplicações de CSE. As duas principais são: (i) a dificuldade em capturar e registrar os dados junto a um ambiente de PAD, ao mesmo tempo em que a simulação é executada, sem prejudicar o desempenho da simulação; e (ii) a limitação aos tipos de análise sobre os dados.

A primeira limitação é que as abordagens de proveniência são em sua grande maioria do tipo *post-mortem*, com as exceções do Chiron e SciCumulus. Dados são capturados e registrados em arquivos de *log* para, somente após a execução, estruturá-los e disponibilizá-los para consulta. A segunda limitação é que, por ser um modelo genérico, o poder de análise é limitado a consultas do tipo quais sãos os dados de entrada da segunda etapa da simulação ou quais são os parâmetros ou dados de saída da terceira etapa. Na realidade, uma análise de dados científicos requer mais acesso aos "conteúdos" desses dados, como qual é a média dos valores de A, sendo A um resultado da segunda etapa. Ou ainda quais os arquivos de visualização de dados para a décima iteração da simulação, ou quais os valores de X (valor dentro de um arquivo de saída da terceira etapa) que estão relacionados a valores de A > y. Ao analisar a literatura, não foram encontradas soluções genéricas que permitam a captura e o registro de dados de domínio junto aos dados de proveniência. Realizamos, inclusive um levantamento sobre essa limitação em (DE OLIVEIRA *et al.*, 2015). Existe ainda uma outra categoria de consultas durante a execução que requer uma análise ligada ao comportamento do desempenho, como qual é a média do tempo de execução da terceira etapa, considerando que essa etapa é executada em milhares de iterações. Esse tipo de análise permite ao cientista computacional optar por mudar valores de parâmetros que tornem a execução mais curta.

Considerando as características das aplicações de CSE, para que os dados de proveniência sejam efetivamente usados em análises de dados, é necessário que sejam capturados em ambientes de PAD, disponibilizados para consulta durante a execução e que sejam acrescidos de dados de arquivos ou variáveis para favorecer análises mais específicas ao domínio da aplicação. Ainda, para que sejam realizadas consultas sobre o desempenho, os dados sobre a execução também precisam ser capturados, registrados e todos relacionados entre si.

Dessa forma, a análise exploratória de dados científicos envolve comumente três tipos de consultas, que consideram o acesso (i) ao conteúdo dos arquivos de dados científicos, (ii) aos múltiplos arquivos relacionados pelos programas de simulação, e (iii) aos elementos de dados relacionados pelos arquivos. Essas consultas são referenciadas ao longo desta tese pelos seguintes termos, respectivamente, (i) conteúdo de arquivos, (ii) múltiplos arquivos e (iii) elementos de dados relacionados.

A Figura 1 mostra um exemplo que contempla os três tipos de acesso aos dados científicos produzidos por uma simulação no domínio da astronomia, baseando-se no conjunto de ferramentas do Montage (JACOB *et al.*, 2009). O *workflow* Montage visa construir mosaicos personalizados no formato de arquivo FITS. Na Figura 1, o arquivo de dados de entrada *images.tbl* contém uma lista de imagens no formato FITS para uma região específica do espaço. A primeira atividade do *workflow* Montage consome esse arquivo de entrada e gera um conjunto de arquivos do tipo FITS, para cada arquivo que está relacionado ao valor CRVAL1 que é um elemento de dados de *images.tbl*. A segunda atividade do *workflow* converte arquivos do tipo FITS em uma extensão específica, chamada HDU, que exige a representação dos dados da imagem em valores inteiros positivos. Em seguida, a terceira atividade realiza transformações lineares gerando projeções de imagens no arquivo *projected_images.tbl*. Por último, a quarta atividade gera um mosaico da região do espaço no formato JPG. Cada um dos tipos de acesso a dados é descrito a seguir para consultas sobre dados gerados ao longo das atividades do *workflow* Montage.

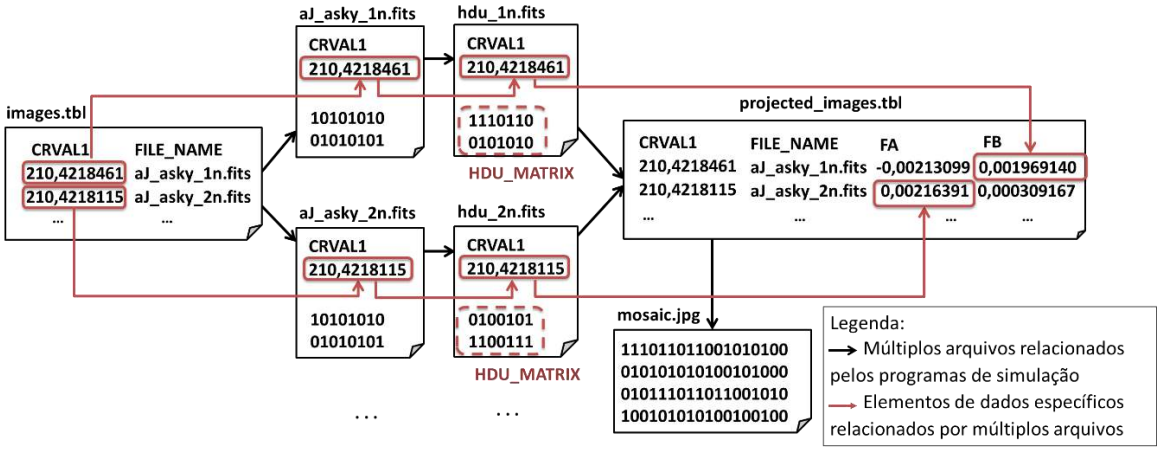


Figura 1. Exemplo de simulação de astronomia para a análise de dados científicos.

Exemplo de consulta ao conteúdo de arquivos: O usuário deseja realizar uma consulta que obtém as transformações lineares realizadas pelo programa de projeção,

atributos *FA* e *FB* (marcados pelos retângulos no arquivo *projected_images.tbl*), quando o valor do comprimento de onda em um pixel de referência é maior que 210,4218200 (*i.e.*, CRVAL1 > 210,4218200). Para isso, é necessário acessar o arquivo *projected_images.tbl*, percorrê-lo para encontrar os valores de *CRVAL1*, *FA* e *FB* e verificar a condição CRVAL1 > 210,4218200. Caso o conteúdo do arquivo *projected_images.tbl* estivesse estruturado dentro de uma base de dados utilizando um Sistema de Gerência de Banco de Dados (SGBD), seria trivial realizar essa consulta.

Exemplo de consulta envolvendo múltiplos arquivos: As setas com a cor preta representam a sequência dos arquivos nos formatos TBL e FITS, que são projetados em outros arquivos TBL e transformados para criar um mosaico no formato de arquivo JPG. Quando esses relacionamentos são registrados, eles proveem o acesso necessário para a realização das consultas que consideram o rastro dos arquivos intermediários que lidam com a geração do arquivo *mosaic.jpg*, ou seja, o resultado final dessa simulação computacional.

Exemplo de consulta aos elementos de dados relacionados: A Figura 1 mostra o fluxo de elementos de dados científicos em caminhos a serem percorridos por esse tipo de acesso – por meio das setas na cor vermelha entre arquivos. Nesse exemplo, o atributo *CRVAL1* é usado como chave para relacionar os dados científicos em diferentes arquivos. Portanto, o arquivo *hdu_1n.fits* pode ser relacionado ao elemento de dados *FB*: 0,001969140. Consequentemente, é possível analisar o conteúdo do arquivo *Header Data Unit* (HDU) (atributo *HDU_MATRIX*) em relação às transformações lineares (*e.g.*, atributos *FA* e *FB*). Por outro lado, sem o apoio ao fluxo de elementos de dados, os usuários teriam que escrever programas para buscar cada valor dessas chaves ao longo do fluxo de arquivos, assim como para relacionar e analisar esses elementos de dados.

Caso o conteúdo de *projected_images.tbl* estivesse estruturado dentro de um SGBD, seria trivial fazer a consulta que envolve o acesso ao conteúdo do arquivo. No entranto, na maioria dos arquivos de dados científicos, não é viável inserir todos os dados brutos no SGBD. Além disso, muitos desses dados podem não ser nunca consultados. Soluções do tipo NoDB (ALAGIANNIS *et al.*, 2015) e RAW (KARPATHIOTAKIS *et al.*, 2014) seriam indicadas nesse caso. No entanto, essas soluções não conseguiriam identificar os relacionamentos entre os arquivos, como as relações de dependência ou de associação de

elementos de dados. Além disso, não foram encontradas soluções para consultas que envolvem o acesso aos elementos de dados relacionados, muito menos que apoiassem as consultas que envolvem o acesso envolvendo múltiplos arquivos e elementos de dados relacionados ao mesmo tempo.

O estado da arte em soluções para a análise sobre dados científicos se concentra em apoiar as consultas que acessam o conteúdo de arquivos. Não foram encontradas soluções para consultas que acessam elementos de dados relacionados, muito menos que apoiassem as consultas que envolvem múltiplos arquivos e acessam elementos de dados relacionados ao mesmo tempo. As diversas análises da literatura realizadas ao longo desta tese indicaram que as soluções existentes não possibilitam as consultas necessárias para aplicações de CSE (CAMATA *et al.*, 2018, DE OLIVEIRA *et al.*, 2015, OCAÑA *et al.*, 2015, OLIVEIRA *et al.*, 2014, SILVA *et al.*, 2016a, 2016c, 2017b). Por exemplo, soluções focadas na captura de dados de proveniência em simulações computacionais têm se destacado no apoio à análise *post-mortem* dos dados envolvidos em diferentes etapas de um mesmo programa ou *script*, porém sem apoio a ambientes PAD, como a ferramenta noWorkflow (MURTA *et al.*, 2014) e a biblioteca Tigres (HENDRIX *et al.*, 2016). Como complicador nesse cenário temos a falta de interação com ferramentas de análise por meio de visualização de dados científicos (*e.g.*, ParaView), complementando as análises via consultas. Somando-se a isso, o grande volume de dados, requer muitas vezes que o dado seja capturado *in situ*, ou seja, compartilhando o endereço de memória da simulação computacional, não permitindo que o mesmo dado seja movimentado, e muito menos replicado e manipulado em diferentes formatos e meios de armazenamento.

Ademais, os desafios de soluções para a análise de dados científicos ao longo da execução de aplicações de CSE podem ser classificados em: (i) levantamento do tipo de análise de dados desejada; (ii) escolha dos dados a serem monitorados para análise; (iii) modelagem dos relacionamentos desses dados junto aos dados de proveniência; (iv) extração de dados durante a execução para a análise; (v) armazenamento dos dados de forma a serem consultados durante a execução, à medida que são gerados; (vi) mecanismos de acesso eficiente aos dados científicos; e (vii) recursos de consulta a esses dados durante a execução. Aliado a esses desafios, existe o requisito de que a solução de análise de dados

não prejudique o desempenho de PAD da aplicação de CSE. A solução proposta nesta tese aborda os sete desafios conforme a seguir.

Como solução proposta nesta tese, foram propostas inicialmente uma abstração teórica para registrar o fluxo de dados da simulação computacional (Seção 3.1) e uma modelagem do fluxo com dados de proveniência (Seção 3.2), permitindo representar os dados científicos gerados pelas simulações computacionais como conjuntos de dados e o processamento computacional de cada etapa da simulação como transformações de dados. Em seguida, analisou-se o apoio aos dados de proveniência dos SGWfC e foi proposta uma metodologia para viabilizar a realização dos passos (i) a (iii). Além disso, analisou-se os mecanismos de extração de dados científicos (Capítulo 2) e finalmente foi proposta uma arquitetura (Capítulo 4) contemplando (iv) a (vii) para a captura e a análise de dados científicos produzidos por simulações computacionais.

Além desta introdução, esta tese está organizada em outros seis capítulos. O Capítulo 2 apresenta os trabalhos relacionados a esta tese quanto a análise exploratória de dados científicos, considerando os três tipos de acesso aos dados frequentemente realizados nesse contexto. O Capítulo 3 apresenta a fundamentação teórica necessária para o melhor entendimento do restante da tese, abordando os conceitos de fluxo de dados, proveniência de dados e *workflow* científico. O Capítulo 4 descreve a abordagem adotada nesta tese para a análise exploratória de dados científicos por meio da abstração de fluxo de dados e a sua primeira instância em um SGWfC. O Capítulo 5 apresenta a implementação da abordagem independente de um SGWfC, considerando as restrições comumentes encontradas em aplicações de CSE. O Capítulo 6 discute os resultados experimentais obtidos a partir de quatro simulações computacionais. O Capítulo 7 conclui esta tese e apresenta as oportunidades de trabalhos futuros nesse tópico de pesquisa.

Capítulo 2 - Análise exploratória de dados científicos

A partir da motivação e do problema de interesse desta tese, este capítulo discute os principais trabalhos relacionados. Mais especificamente, as seções deste capítulo discutem os trabalhos existentes quanto ao apoio aos três tipos de consultas necessárias para a análise exploratória de dados científicos, enfatizando as suas diferenças e as suas limitações. Apesar do foco ser na análise de dados científicos, este capítulo também discute o processo de captura dos dados científicos ou mesmo da especificação do fluxo de dados, enumerando as vantagens e as desvantagens dos trabalhos relacionados. Vale ressaltar também que os passos adotados nessa análise da literatura foi baseada em um artigo do tipo *survey* que já havia sido publicado pelo nosso grupo de pesquisa (MATTOSO *et al.*, 2015).

2.1. Análise de dados científicos em apenas um arquivo

Diversas propostas lidam com a análise de dados científicos em arquivos de forma isolada, mantendo o seu conteúdo no formato original (ALAGIANNIS *et al.*, 2015, BLANAS *et al.*, 2014, KIM *et al.*, 2011, MA *et al.*, 2012, WU *et al.*, 2009). Em geral, essas propostas acessam elementos de dados específicos em arquivos (também conhecidos como região de interesse), depois elas indexam esses dados usando linguagens específicas de consulta, máquinas ou APIs (do termo em inglês *Application Programming Interface*), aliviando assim o esforço de desenvolvimento de códigos para cada tipo de análise. Como exemplos dessa abordagem temos o FastBit (WU *et al.*, 2009), o *Attribute-based Unified Data Access Service* (AUDAS) (MA *et al.*, 2012), o FastQuery (CHOU *et al.*, 2011), o SDS/Q (DONG *et al.*, 2013) e o AQUAdex (HONG *et al.*, 2015). Mais especificamente, o FastQuery e o SDS/Q lidam com a indexação e a consulta de dados de forma paralela, reduzindo o tempo de resposta ao processar consultas, quando comparado com abordagens sequenciais.

Em relação às técnicas de indexação, existem duas propostas de índices largamente utilizadas nessas soluções: *bitmap* e posicional. A indexação baseada em um mapa de bits, conhecida como indexação *bitmap*, consiste na análise do domínio de determinados atributos presentes em arquivos de dados científicos para gerar índices booleanos por meio da verificação de equações algébricas. Por exemplo, assumindo-se que os valores do atributo *X* para todos os arquivos de dados científicos apresentam um domínio com apenas

quatro valores possíveis, assumindo-se uma estrutura de dados de números inteiros, então o mapa de *bits* necessita de quatro colunas para avaliar a presença ou não de um determinado valor, assumindo-se uma equação de equidade, conforme apresentado na Tabela 1. Existem casos também em que o uso de inequações para a indexação *bitmap* é mais vantajoso. Por exemplo, no caso de números em ponto flutuante com um domínio muito extenso, o uso de inequações pode favorecer tanto a geração de índices (menor número de colunas de *bits* no mapa), como consultas que buscam um determinado intervalo de valores para o atributo indexado (restrição do espaço de busca pelos índices gerados). Como tecnologias que empregam a indexação *bitmap*, pode-se citar o FastBit (WU *et al.*, 2009), o FastQuery (CHOU *et al.*, 2011) e o SDS/Q (DONG *et al.*, 2013).

ID	X	Mapa de bits			
		Bit 0 (X=0)	Bit 1 (X=1)	Bit 2 (X=2)	Bit 3 (X=3)
1	2	0	0	1	0
2	3	0	0	0	1
3	1	0	1	0	0
4	2	0	0	1	0
5	0	1	0	0	0
6	0	1	0	0	0
7	1	0	1	0	0

Tabela 1. Mapa de *bits* para indexação *bitmap* do atributo X em um exemplo hipotético.

O SDS/Q executa consultas diretamente sobre arquivos, eliminando a sobrecarga relacionada à conversão da estrutura de dados do conteúdo bruto e a sua carga em uma estrutura de dados diferente. No que diz respeito à execução paralela, o SDS/Q explora os recursos providos pelo uso da memória volátil, uma vez que ele emprega a indexação *bitmap* (ELMASRI e NAVATHE, 2010) e o processamento de consultas em memória. Ele também apresenta melhorias de desempenho no processamento de consultas pelo uso de técnicas para paralelizar a execução intra- e inter-nós, além de implementar índices *bitmap* usando o FastBit. Por outro lado, a sua máquina é restrita ao formato de arquivo HDF5.

Outra proposta presente em trabalhos mais recentes sobre a indexação de dados científicos considera a geração de índices posicionais. De uma forma simplificada, esses índices utilizam informações que facilitam a localização (*i.e.*, posição) dos dados científicos nos arquivos. Por exemplo, um índice posicional pode ser o *bit*, em que o valor de um determinado atributo precisa começar a ser lido do arquivo. Comparando-se esse tipo de índice com o mapa de *bits* gerado pela indexação *bitmap*, os índices posicionais tendem a

fornecer uma sobrecarga de dados menor para a gerência dos dados científicos, pois, diferentemente do índice *bitmap*, este não cria *bits* redundantes (*i.e.*, sequência de zeros no mapa de bits) que crescem de acordo com o aumento do domínio dos atributos indexados (*i.e.*, aumento do número de colunas a serem representadas no mapa de *bits*). Como exemplos de sistemas que empregam índices posicionais, pode-se citar o NoDB (ALAGIANNIS *et al.*, 2015) e o RAW (KARPATHIOTAKIS *et al.*, 2014).

O NoDB (ALAGIANNIS *et al.*, 2015) propõe a extração de conteúdo específico presente em arquivos de dados científicos e a sua carga em uma versão modificada do SGBD relacional PostgreSQL, conhecido como PostgresRaw. Apesar de evitar mudanças nas estruturas originais dos dados científicos, o NoDB baseia-se no processamento de consultas adaptativas, que utiliza estatísticas e estratégias de *caching* para ter acesso aos dados científicos e realizar as transformações de dados estritamente necessárias. Alagiannis *et al.* (2015) apresentam melhorias de desempenho significativas ao utilizar o NoDB, uma vez que apenas os dados científicos necessários pelas consultas são efetivamente indexados. Entretanto, como limitação, o NoDB realiza a carga dos dados científicos acessados no PostgresRaw, o que caracteriza uma sobrecarga de armazenamento dos dados em um repositório usando outras estruturas de dados (*i.e.*, estruturas de dados que podem ser diferentes das adotadas no arquivo com dados científicos).

Para lidar com essas limitações, o mesmo grupo de pesquisa propôs o RAW (KARPATHIOTAKIS *et al.*, 2014), uma máquina de processamento de consulta que adapta o plano de execução das consultas para os formatos dos dados científicos, sem apresentar a sobrecarga de armazenar os dados científicos em outro repositório externo ou base de dados. Essa abordagem é similar ao SDS/Q, ao mesmo tempo em que não é restrita a apenas um formato de arquivo. Em relação ao seu desenvolvimento, o RAW implementa caminhos para acesso aos dados durante a execução da consulta (conhecidos pelo termo em inglês, *just-in-time access paths*), assim como apresenta heurísticas para selecionar as colunas relevantes para a consulta desejada. A proposta dos caminhos de acesso considera a geração deles em tempo de execução para direcionar o mapeamento dos operadores de varredura aos dados científicos no plano de execução da consulta. Além disso, o RAW busca adiantar o máximo possível as operações de seleção de elementos de dados no plano de execução da consulta.

Com outra técnica de indexação, o AQUAdex (*Accelerating Query Using Area pixelization indexing*) (HONG *et al.*, 2015) propõe a indexação e a recuperação de séries temporais de imagens a partir de conjuntos de dados astronômicos, tendo como motivação a descoberta de imagens presentes em uma região específica do espaço e capturadas em um determinado intervalo de tempo. Para isso, o AQUAdex realiza o acesso, a extração e a indexação espacial de arquivos no formato FITS que foi construída com base no HEALPix[1] (*Hierarchical Equal Area isoLatitude Pixelization of a sphere*), para realizar análises em séries temporais.

Recentemente, o AQUAdex propôs uma extensão da sua proposta original a fim de reduzir a sobrecarga em termos de tempo de processamento por carregar os índices e outros metadados em uma base de dados e eventuais operações que envolviam o uso de disco. Essa extensão, conhecida como AQUAdexIM (AQUAdex In-Memory) (HONG *et al.*, 2016), tira proveito de dados já alocados em memória pela simulação computacional para realizar a indexação dos dados e o processamento de consultas por meio do uso da memória principal. Além disso, algumas otimizações foram realizadas em suas estruturas de dados para a redução do custo no processamento de consultas, como o uso de mapas baseados em uma função *hash* por meio da estrutura de dados *HashMap*. Por outro lado, ambas as propostas, AQUAdex e AQUAdexIM, apresentam limitações quanto ao apoio à indexação de apenas um formato de arquivo e o processador de consultas não considera análises que envolvem elementos de dados de diferentes transformações de dados da simulação computacional.

A partir de outra abordagem, o SimDB (LUSTOSA, 2015) consiste em um sistema para otimizar o armazenamento de malhas a partir de uma estrutura de dados baseada em *octrees* utilizando o SciDB (BROWN, 2010), considerando dados de simulação em larga escala representados por matrizes e malhas. O SimDB apresenta como principais contribuições a redução de células sem conteúdo no armazenamento de dados pelo SciDB, em função da representação dos dados em *octrees*; um melhor desempenho no processamento de consultas que não cabem em memória, quando comparado com outros SGBD; e a integração com o ParaView (AYACHIT, 2015), permitindo análises visuais de resultados de consultas por parte do usuário. Nesse mesmo grupo de pesquisa, o DEXL, do

[1] http://healpix.sourceforge.net

Laboratório Nacional de Computação Científica[2] (LNCC), pode-se observar outro trabalho que argumenta a importância do uso de bases de dados probabilísticas para a gerência de dados científicos de natureza incerta (GONÇALVES e PORTO, 2014).

Entretanto, pelo fato desses trabalhos serem focados em arquivos "isolados", eles não permitem análises sobre o fluxo de dados. Para que o fluxo de dados fosse representado nos SGBD adaptados aos dados científicos, os programas de simulação, que geram os dados científicos, teriam que ser reescritos, mudando todo o seu acesso às estruturas de dados de acordo com a implementação do SGBD, algo que não seria viável no panorama atual de programas de simulação computacional, em função do esforço de desenvolvimento e desses programas serem vistos, em várias circunstâncias, como caixas-pretas (ou seja, código privado e inacessível). A outra alternativa seria manter os dados em seus formatos originais e replicá-los nos sistemas de gerência e análise de dados científicos, algo que não se mostrou efetivo na experiência do NoDB (ALAGIANNIS *et al.*, 2015), nem parece compatível com a larga escala de dados no contexto de processamento de dados *in situ*, em sintonia com os sistemas de simulação computacional (KIM *et al.*, 2011).

Dessa forma, os trabalhos existentes não consideram a análise do fluxo dos elementos de dados a partir da execução dos programas de simulação que produzem arquivos, cujo conteúdo deve ser acessado, extraído, indexado e consultado ao final da simulação computacional. Para apoiar a análise do fluxo de dados, essas propostas precisariam permitir a gerência dos diversos arquivos envolvidos e dos elementos de dados relacionados pelos programas de simulação. No que diz respeito às simulações computacionais, esta tese apoia aplicações que acompanham a execução escalável de problemas de CFD difíceis, como em (KIM *et al.*, 2011), em que aplicam-se otimizações em estruturas *octrees* adaptativas para modelos computacionais otimizados que executam com milhares de núcleos computacionais (do termo em inglês *cores*) em cálculos baseados em Equações Diferenciais Parciais (EDP) não lineares, heterogêneas e mal condicionadas.

2.2. Análise de múltiplos arquivos

As abordagens para a gerência do fluxo de arquivos têm como meta o apoio às transformações de dados pelo sistema de arquivos, ignorando os conteúdos específicos do

[2] www.lncc.br

domínio presentes nos arquivos. Por consequência, essa abordagem trata os arquivos como caixas-pretas, uma vez que não apresentam índices para os dados científicos ou apoio às consultas para os conteúdos dos arquivos de dados científicos. Considerando esse assunto, Vahi *et al.* (2013) propõem duas abordagens baseadas no armazenamento de dados, em que os usuários do domínio científico não precisam modificar seus programas para armazenar os dados produzidos em dispositivos de armazenamento.

A primeira abordagem considera o armazenamento de objetos, que se caracteriza pela captura de todos os arquivos consumidos e produzidos na simulação computacional. Desse modo, essa solução permite gerenciar os arquivos manipulados em tempo de execução de forma distribuída, em que os arquivos necessários são requisitados ao mecanismo de armazenamento de objetos. Já a segunda abordagem é baseada em um sistema de arquivos compartilhados para armazenar os arquivos manipulados em tempo de execução, também utilizando dispositivos de armazenamento de objetos.

Nesse trabalho de Vahi *et al.* (2013), as simulações computacionais foram executadas em um ambiente de nuvem computacional (Amazon EC2), considerando uma simulação no domínio da astronomia (*i.e.*, Montage) caracterizada por muitas operações de entrada e saída (*i.e.*, centrada em dados) e outra centrada em CPU (*i.e.*, Rosetta). Os resultados apresentaram um desempenho melhor para o sistema de arquivos compartilhados em comparação com o sistema de arquivos distribuídos devido ao custo de comunicação, ou seja, o custo de transferência de arquivos entre recursos computacionais.

Ademais, o arcabouço AWARD (ASSUNCAO *et al.*, 2012) provê uma solução baseada no fluxo de dados por meio de uma representação orientada a tuplas para gerenciar a execução de simulações computacionais. Esse arcabouço apoia a distribuição dos dados em diferentes centros de dados (*i.e.*, diferentes regiões espaciais). Assim, para controlar a execução da simulação, o AWARD captura as tuplas (conjunto de valores de parâmetros) em tempo de execução e gerencia a responsabilidade de propagação dos dados entre os programas de simulação de acordo com as dependências de dados existentes (*i.e.*, transferência de dados entre centros de dados). Entretanto, a integração do AWARD com uma base de dados de proveniência não é trivial, pois esse arcabouço apresenta um esquema fixo e qualquer mudança para inserir novos metadados requer modificações substanciais na sua máquina. Logo, na perspectiva da gerência do fluxo de arquivos, o AWARD permite

que as referências para os arquivos consumidos e produzidos em cada transformação de dados sejam representadas como valores de parâmetros nas tuplas.

Dessa forma, nenhuma dessas soluções considera a gerência do fluxo dos elementos de dados no contexto da análise exploratória de dados científicos. Apesar de representar os valores dos parâmetros, o AWARD não lida com valores de parâmetros como elementos de dados na sua representação relacional, o que inviabiliza a reconstrução do fluxo de dados nesse nível de abstração. Além disso, nenhuma dessas soluções discute a captura de dados científicos presentes em arquivos produzidos pelos programas de simulação.

2.3. Análise de elementos de dados relacionados pelos arquivos

Os Sistemas de Gerência de *Workflows* Científicos (SGWfC), apresentados formalmente na Seção 3.3.2, oferecem a abstração de *workflows* científicos para apoiar a composição, execução, gerência e análise de simulações computacionais. Atualmente, alguns SGWfC têm apresentado funcionalidades para a gerência do fluxo dos elementos de dados, como o Kepler (BOWERS *et al.*, 2008), o Panda (IKEDA e WIDOM, 2010) e o SCC, sendo que o último consiste na integração dos SGWfC Chiron (OGASAWARA *et al.*, 2013) e SciCumulus (DE OLIVEIRA *et al.*, 2010). Por outro lado, essas propostas não apoiam a gerência do fluxo de elementos de dados presentes em arquivos de dados científicos, o que limita o seu potencial analítico.

O Kepler define formalmente o encadeamento de atividades como um *workflow* científico, que corresponderia à simulação computacional com os seus programas encadeados, consumindo e produzindo arquivos com dados científicos. Além de apoiar a gerência de arquivos, esse SGWfC também gerencia o fluxo dos elementos de dados. Entretanto, o Kepler não permite o acesso e a captura de dados científicos presentes em arquivos de dados científicos, característico da análise exploratória de dados científicos. Consequentemente, esse sistema não permite consultas ao conteúdo de arquivos de dados científicos e apresenta limitações para as consultas aos elementos de dados relacionados, conforme apresentadas no Capítulo 1.

Com o intuito de gerenciar o rastro do fluxo de arquivos e de elementos de dados produzidos por uma simulação computacional, o Panda propõe um elegante e poderoso formalismo para dados de proveniência (IKEDA e WIDOM, 2010). Todavia, esse sistema

também possui a mesma desvantagem do Kepler: o apoio ao acesso aos dados científicos presentes em arquivos ou mesmo alocados em memória. Ao considerar outras soluções existentes que apoiam a gerência da proveniência de dados, como o Pegasus Lite (VAHI *et al.*, 2013) e o Wings (GIL *et al.*, 2011), percebe-se que elas não permitem o processamento de consultas direcionadas ao fluxo de elementos de dados. Como exceção, o LabelFlow (ALPER *et al.*, 2015) permite, por um sistema de anotações, o armazenamento de metadados quanto à estrutura do fluxo de dados, assim como os dados de domínio manipulados pela simulação computacional. Em contrapartida, esse sistema não permite a captura de dados científicos armazenados em arquivos, embora os elementos de dados acessíveis nos programas de simulação (como caixas brancas) possam ser rastreados.

Já o SCC (SciCumulus + Chiron) contempla o uso de uma álgebra de *workflows* centrada em dados para compor e executar simulações computacionais em ambientes de PAD. Nessa máquina paralela, os *workflows* são representados e executados como expressões algébricas compostas de operações, que encapsulam programas de simulação e operandos para representar o fluxo de dados. Somando-se a isso, o SCC permite a captura de dados de proveniência e de determinados dados de domínio em tempo de execução. Portanto, essa máquina paralela de *workflows* científicos apoia as consultas ao conteúdo de arquivos e a múltiplos arquivos, enquanto que o apoio às consultas aos elementos de dados relacionados (ou fluxo dos elementos de dados) apresentam limitações nessa máquina de *workflows*.

Portanto, de um modo geral, o uso de SGWfC requer um esforço considerável de aprendizagem por parte dos usuários para modelar as suas simulações computacionais para permitir as análises focadas em dados de proveniência e de domínio. Por esse motivo, outras soluções têm sido propostas com a estratégia de prover recursos para que os usuários não precisem se adaptar a outros ambientes de trabalho (*i.e.*, aprendizado de outra linguagem de programação ou de como modelar um *workflow* científico) para realizarem as suas análises. Como exemplos dessas soluções temos o YesWorkflow (MCPHILLIPS *et al.*, 2015), o noWorkflow (MURTA *et al.*, 2014) e o RDataTracker (LERNER e BOOSE, 2015).

Primeiramente, o YesWorkflow (MCPHILLIPS *et al.*, 2015) consiste em uma ferramenta capaz de capturar e analisar dados de proveniência prospectiva de *scripts*

desenvolvidos independentemente da linguagem de programação. Para isso, essa ferramenta exige que o usuário adicione anotações em forma de comentários em seus *scripts* para representar o *workflow* científico da simulação computacional. Dessa forma, o usuário deve definir os blocos de programa (ou transformações de dados) envolvidos no *workflow*, assim como as portas de entrada e de saída de cada programa (ou atributos consumidos e produzidos), sendo que as dependências de dados (conhecidas como canais pelo YesWorkflow) são inferidas de acordo com a propagação de portas entre os programas.

Ademais, essa ferramenta também conta com três visualizações possíveis de acordo com o *workflow* modelado, sendo baseadas (i) nos blocos de programa, (ii) nos atributos consumidos e produzidos por cada programa, e (iii) em ambos, ou seja, apresenta os blocos e os atributos em uma mesma visualização. Mesmo permitindo a investigação da estrutura da simulação computacional executada, o YesWorkflow não permite análises que envolvem uma granularidade mais fina (*i.e.*, proveniência retrospectiva), relacionando o consumo e a produção dos elementos de dados pelos programas. Ademais, o processo de captura dos dados requer uma etapa de ajuste manual dos programas de simulação por parte do usuário.

De forma complementar ao YesWorkflow, o noWorkflow (MURTA *et al.*, 2014) consiste em uma ferramenta que permite a análise dos dados de proveniência prospectiva e retrospectiva em *scripts*, sem que o usuário precise modificar os seus *scripts* na linguagem de programação Python. Para isso, o interpretador original do Python foi modificado para permitir a gerência das principais operações dos *scripts* em tempo de execução, como em métodos de leitura e escrita de dados em arquivos. Outros recursos também são centrados na análise da proveniência retrospectiva por meio da visualização do fluxo dos elementos dados no controle de versões (ou execuções) dos *scripts*. Entretanto, pelo fato do usuário não especificar as transformações de dados (ou funções em Python) de interesse, o noWorkflow captura os dados de proveniência em uma granularidade muito fina, monitorando informações sobre invocações de funções e atribuições de variáveis que podem não ser necessárias do ponto de vista analítico. Além de ser específica para *scripts* em Python, essa ferramenta também não considera a captura de dados científicos de arquivos, a execução de *scripts* de natureza paralela em ambientes de PAD, ou mesmo apresenta uma modelagem da sua base de dados de proveniência que favoreça a análise do fluxo de elementos de dados.

De forma análoga, o RDataTracker (LERNER e BOOSE, 2015) consiste em uma biblioteca para a captura do fluxo de elementos de dados oriundos da execução de *scripts* em R. Para isso, o RDataTracker baseia-se em um processo de instrumentação dos *scripts*, que pode ter diferentes níveis de intrusão. Em uma abordagem menos intrusiva, o usuário precisa especificar no *script* em que linhas de código se deve começar e finalizar a captura de dados por essa biblioteca. Nesse caso, o grafo de proveniência obtido pode ser muito detalhado, dificultando o desenvolvimento das análises requeridas pelo usuário. Assim, uma abordagem mais intrusiva pode ser realizada, em que apenas os processos (ou transformação de dados em nossa abstração) e os atributos de interesse devem ser especificados. Vale ressaltar que as mesmas limitações observadas para o noWorkflow são observadas no RDataTracker.

Diferentemente das propostas anteriores, o Tigres (HENDRIX *et al.*, 2016) é uma biblioteca na linguagem de programação Python que exige a modelagem da simulação computacional como um *workflow* científico. Para isso, o usuário precisa especificar as tarefas a serem executadas em cada programa ou transformação de dados do *workflow*. Em relação à essa especificação, o Tigres propõe um conjunto de padrões para as funções ou transformações de dados a serem executadas (*sequence*, *parallel*, *merge* e *split*), além de conter uma camada em sua arquitetura focada na adaptação do *script* em Python para ser executado em diferentes ambientes computacionais, desde em máquinas locais até em ambientes de PAD. Logo, essa biblioteca assume que o usuário comumente executa programas caixas-pretas e precisa especificar, por meio de um *script*, a estrutura do *workflow* científico, assim como as tarefas e os dados de entrada para cada tarefa. Por esse motivo, o Tigres requer que o usuário tenha conhecimento da linguagem de programação Python para especificar a execução do *workflow* em um *script*; e saiba, a priori, todas as tarefas que serão executadas (e os seus dados de entradas), o que não é trivial em simulações computacionais que envolvam, especialmente, intervenções dos usuários em tempo de execução.

Diante desses trabalhos relacionados à análise do fluxo de elementos de dados, observa-se que eles estão inseridos em dois cenários distintos. Primeiramente, temos as soluções baseadas em SGWfC, que assumem que os programas de simulação são caixas-pretas e precisam ser invocados de forma paralela em ambientes de PAD, variando-se os

valores dos atributos de entradas. Nesse caso, os dados de proveniência são capturados respeitando-se a estrutura do *workflow* modelado e não exigem um esforço adicional do ponto de vista de instrumentação do código fonte dos programas de simulação. Por outro lado, essas soluções exigem um conhecimento dos usuários para modelar os *workflows* científicos e os SGWfC apresentam limitações para apoiar os três tipos de consultas na análise exploratória de dados científicos.

Enquanto isso, outras ferramentas e bibliotecas têm sido propostas para permitir a captura de dados de proveniência por meio da instrumentação ou a análise dos dados manipulados pelos *scripts* em tempo de execução. Nesse cenário, os usuários têm acesso ao conteúdo dos *scripts* para poder ajustá-los e extrair os dados científicos de interesse. Contudo, essas ferramentas comumente apresentam limitações relacionadas ao fato de serem exclusivas para uma linguagem de programação, de apresentarem um grafo de proveniência muito detalhado quando são pouco intrusivas no código fonte, de não terem apoio à execução paralela em ambientes de PAD ou mesmo à captura de dados científicos em arquivos. Por outro lado, elas apresentam contribuições quanto à análise do fluxo dos elementos de dados, considerando a gerência da proveniência dos dados.

2.4. Análise global da literatura

Somando-se às análises pontuais de cada categoria de soluções existentes, realizadas nas seções anteriores, esta seção apresenta uma análise mais global da literatura. Nesta tese, seguimos uma categorização dos diferentes sistemas estudados, sendo estes divididos em basicamente três categorias: sistemas tradicionais de análise de dados científicos em arquivos, SGWfC e sistemas de captura de dados de proveniência. Em relação aos sistemas tradicionais de análise de dados científicos, como o FastBit, o FastQuery, o SDS/Q, o PostgresRaw e o AQUAdex, observou-se que essas soluções consideram apenas o acesso e a análise de dados científicos presentes em arquivos de forma isolada. Além disso, esses sistemas apresentam uma sobrecarga relacionada à leitura dos dados dos arquivos, à transformação desses dados em estruturas de dados mais adequadas para o repositório externo ou a base de dados, e à carga dos dados nesse repositório externo. Dessa forma, esses sistemas não estão no mesmo contexto desta tese, que considera a análise de dados científicos durante a execução de simulações computacionais, pois eles consistem em abordagens *post-mortem*.

Enquanto isso, os SGWfC com apoio à captura de dados de proveniência, como o Kepler, o Chiron e o Pegasus Lite, permitem registrar o fluxo de arquivos e o fluxo de elementos de dados produzidos pela execução de simulações computacionais. Contudo, eles apresentam limitações associadas à captura de dados científicos em arquivos. Somando-se a isso, esses SGWfC exigem ajustes nos programas de simulação para que a execução paralela em ambientes de PAD seja gerenciada pelos seus próprios controles de execução.

Em função dessa limitação dos SGWfC existentes, observa-se mais recentemente o surgimento de sistemas focados na captura de dados de proveniência em tempo de execução, como o noWorkflow e o RDataTracker. Esses sistemas contribuem tanto em manter o ambiente de desenvolvimentos dos usuários do domínio científico ou do próprio desenvolvedor da aplicação, como em prover recursos que não exijam a especificação de quais dados precisam ser capturados durante a execução das aplicações de CSE, por meio de recursos de instrumentação automática de código fonte. Apesar de todo o seu apoio à captura e à análise dos dados, tais sistemas normalmente não são desenvolvidos para apoiar a execução paralela em ambientes de PAD e são dependentes de uma linguagem de programação para o acoplamento dos programas de simulação com esses sistemas.

Diante dessa realidade, esta tese visa propor uma solução focada na captura e na análise de dados científicos ao longo da execução de simulações. Mais especificamente, considera-se uma abordagem que permita apoiar os três tipos de consultas discutidos para a análise exploratória de dados científicos, sendo que nenhum dos sistemas estudados permitem análises baseadas em todos os tipos de consultas. Tais consultas almejam acessar (i) o conteúdo de arquivos de dados científicos, (ii) múltiplos arquivos relacionados pelas etapas da simulação e (iii) elementos de dados relacionados pelos múltiplos arquivos.

Capítulo 3 - Referencial teórico

Esse capítulo tem o propósito de apresentar a fundamentação teórica para melhor compreensão desta tese. Por isso, os conceitos de fluxo de dados, proveniência e *workflows* científicos são definidos formalmente. Ao mesmo tempo, o SGWfC SCC (SciCumulus + Chiron) é apresentado em detalhes na Seção 3.3.4 pelo fato dele ser usado no desenvolvimento dos primeiros experimentos desta tese. Ademais, as abordagens existentes para a captura e a análise de dados científicos são descritas nesse capítulo para facilitar a compreensão do cenário alvo desta tese.

3.1. Fluxo de dados

Esta tese assume a abstração de fluxo de dados para permitir a gerência dos dados científicos e dos seus relacionamentos ao longo da execução de simulações computacionais. Essa abstração adotada foi adaptada de um formalismo já proposto por Ikeda *et al.* (2013) para o cenário de *workflows* científicos, a fim de atender à nossa nomenclatura e ser mais direcionada ao fluxo de elementos de dados. Nesta seção, apresentamos uma definição formal de fluxo de dados (Subseção 3.1.1) e os dois níveis existentes para a gerência de fluxos de dados (Subseção 3.1.2).

3.1.1. Definição

Na abstração do fluxo de dados, a menor unidade de interesse consiste em um *elemento de dados* (Definição 3.1.1) que apresenta uma sequência de valores para os atributos predefinidos no conjunto de dados ao qual esse elemento de dados pertence. Assim, um conjunto de dados apresenta uma sequência de atributos, de modo que o i-ésimo valor de um elemento de dados corresponde ao i-ésimo atributo desse conjunto. Um conjunto de elementos de dados consiste em uma *coleção de dados* (Definição 3.1.2), sendo que um *conjunto de dados* (Definição 3.1.3) é composto por coleções de dados e por uma sequência de atributos. Além disso, uma *transformação de dados* (ou, de forma abreviada, *transformação*) (Definição 3.1.4) realiza o processamento de dados baseado em procedimentos de um algoritmo ou modelo computacional, consumindo dados de um ou mais conjuntos de dados como entrada e produzindo um ou mais conjuntos de dados como saída. Ao mesmo tempo, as transformações podem apresentar *dependências de dados* (Definição 3.1.5) entre si em função de um conjunto de dados, que é produzido por uma

transformação de dados e consumido por outra transformação de dados. A partir desses conceitos, um *fluxo de dados* (Definição 3.1.6) é definido pela composição de transformações de dados, manipulando conjuntos de dados e respeitando as dependências de dados dessa composição.

De forma semelhante, essa especificação de fluxo de dados pode ser representada por um grafo direcionado $G = (V, E)$, em que os vértices V correspondem às transformações de dados, e as arestas E correspondem aos conjuntos de dados. Uma transformação de dados pode ser instanciada de acordo com o consumo de um subconjunto de suas coleções de dados de entrada e a produção de um subconjunto de suas coleções de dados de saída, sendo este conceito definido como *instância de uma transformação de dados* (Definição 3.1.7). Em outras abstrações, o conceito de instância de uma transformação de dados é conhecido como *tarefa*. Vale ressaltar que as definições adotam as letras maiúsculas para representar um conjunto ou uma sequência de entidades (*e.g.*, T representa um conjunto de transformações de dados), enquanto que uma instância de uma entidade é representada em letra minúscula (*e.g.*, t representa uma transformação de dados). Como exceção temos a representação de fluxo de dados, que é descrito por uma letra maiúscula, isto é, D_F.

Definição 3.1.1. (Elemento de Dados) Um elemento de dados $e = (v \mid v \in V)$ é composto de uma sequência de valores V.

Definição 3.1.2. (Coleção de Dados) Uma coleção de dados $c = \{e \mid e \in E\}$ é composta de um conjunto de elementos de dados E.

Definição 3.1.3. (Conjunto de Dados) Um conjunto de dados $s = (A, C)$ é um par composto de uma sequência de atributos $A \mid \forall a \in A: a(name, type)$, e de um conjunto de coleções de dados $C \mid \forall e \in C: card(e.V) = card(A)$.

Definição 3.1.4. (Transformação de Dados) Uma transformação de dados $t = (I, O)$ é um par composto de conjuntos de dados de entrada I e de conjuntos de dados de saída O.

Definição 3.1.5. (Dependência de Dados) Uma dependência de dados $\varphi = (s, t_{previous}, t_{next})$ é uma tripla composta de um conjunto de dados s, de uma transformação de dados $t_{previous}$ e de uma transformação de dados t_{next}, em que $s \subseteq t_{previous}.O \wedge s \subseteq t_{next}.I$.

Definição 3.1.6. (Fluxo de Dados) Um fluxo de dados $D_F = (T, S, \Phi)$ é uma tripla composta de transformações de dados T, de conjuntos de dados S, e de dependências de dados Φ.

Definição 3.1.7. (Instância da Transformação de Dados) Uma instância de transformação de dados $t^* = (I', O')$, or $O' = t^*(I')$, consiste em um par com um subconjunto das coleções de dados dos conjuntos de dados de entrada $I' = \bigcup_{k=1}^{card(t.I)}(i_k.C' \mid i_k.C' \subseteq i_k.C)$ e um subconjunto das coleções de dados dos conjuntos de dados de saída $O' = \bigcup_{k=1}^{card(t.O)}(o_k.C' \mid o_k.C' \subseteq o_k.C)$ manipulados pela transformação de dados t.

Exemplo 3.1.1. Uma aplicação científica é desenvolvida para contar o número de vezes que determinadas palavras aparecem em cada arquivo de texto de entrada. Nesse caso, a aplicação pode ser baseada em duas transformações de dados t_1 e t_2, também conhecidas como *FindWords* e *Count*, que realizam, respectivamente, a identificação de cada ocorrência de uma palavra no arquivo de texto de entrada e a contagem do número de ocorrências para cada palavra. Logo, a transformação de dados *FindWords* consome um conjunto de dados s_1 e produz um conjunto de dados s_2, enquanto que a transformação de dados *Count* consome os elementos de dados de s_2 e produz um conjunto de dados s_3. O conjunto de dados s_1 apresenta o atributo *FILE* que aponta para o arquivo de texto a ter o conteúdo analisado, enquanto o esquema de s_2 possui a sequência de atributos *WORD* e *OCCURRENCE* que representam a ocorrência de uma palavra no arquivo investigado, e s_3 possui a sequência de atributos *WORD* e *COUNT* que representam o número de vezes que cada palavra esteve presente em todos os arquivos de entrada. A Figura 2 apresenta esse exemplo de fluxo de dados, assim como a representação de cada conceito (transformação de dados, conjunto de dados, dependência, atributos, entre outros) de acordo com as definições apresentadas.

3.1.2. Gerência do fluxo de dados

Somando-se às questões de especificação formal do fluxo de dados, as soluções precisam ser capazes de gerenciar o fluxo adotado pelos dados consumidos e produzidos por cada transformação de dados durante a sua execução. Semelhante aos tópicos discutidos no Capítulo 2, discutimos o potencial analítico associado à gerência do fluxo de dados em SILVA *et al.* (2016). Ademais, definimos nesse artigo dois níveis de abstração para apoiar a gerência do fluxo de dados – *i.e.*, físico e lógico –, sendo que eles fazem parte das principais contribuições desta tese.

A *gerência do fluxo de dados no nível físico* consiste no apoio às transformações de dados no nível do sistema de arquivos. Logo, as transformações de dados são analisadas apenas em relação ao consumo e à produção de arquivos por cada programa de simulação,

sem considerar o conteúdo de arquivos específicos do domínio da simulação. Nessa mesma perspectiva, esse nível de gerência do fluxo de dados trata os arquivos como caixas-pretas, uma vez que não existem índices ou apoio a consulta com acesso aos dados científicos presentes nesses arquivos. Consequentemente, as análises oferecidas aos usuários do domínio científico ficam restritas a ponteiros para os arquivos envolvidos no fluxo de dados, sendo necessário que eles analisem cada arquivo individualmente ou desenvolvam programas específicos para ter acesso aos dados científicos presentes nos arquivos. Em ambas as alternativas para a análise específica de domínio, os usuários apresentam tarefas que são propensas a erros e serão capazes de realizar apenas um tipo de análise.

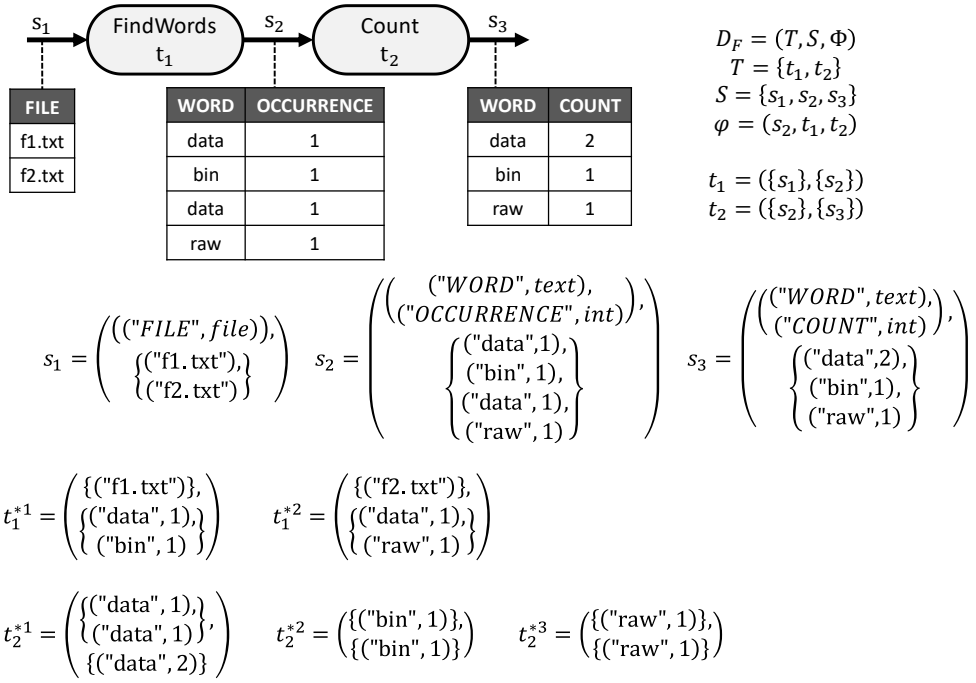


Figura 2. Exemplo de fluxo de dados.

Já a *gerência do fluxo de dados no nível lógico* lida com o monitoramento de como os elementos de dados são consumidos, produzidos e propagados pelos programas de simulação. Esses elementos de dados contemplam tanto os conteúdos de arquivos de dados científicos, como a propagação de dados pelos programas de simulação. A partir do monitoramento dos elementos de dados, os seus relacionamentos podem ser utilizados para reconstruir o fluxo de dados, agregando valor ao potencial analítico desse nível de gerência. Do ponto de vista dos usuários do domínio científico, eles podem realizar consultas direcionadas aos aspectos do domínio da simulação computacional, sem precisar do desenvolvimento de programas específicos para os arquivos de dados científicos, como

constatado na gerência do fluxo de dados no nível físico. Por outro lado, o custo computacional é maior para a gerência no nível lógico devido a sua granularidade mais fina, em que os elementos de dados precisam ser capturados e monitorados em tempo de execução.

Ainda em relação ao nível lógico, ele pode ser dividido em *fluxo de elemento de dados* e *fluxo de coleção de dados*. O fluxo de elemento de dados representa as transformações em um elemento de dados ao longo da execução do fluxo de dados por meio de diferentes programas em uma simulação computacional. Nesta tese, define-se que um *elemento de dados* é um item de dado que apresenta uma sequência de valores, sendo que o valor na i-ésima posição corresponde ao atributo na i-ésima posição da sequência de atributos do conjunto de dados ao qual o elemento de dados pertence. Já o fluxo de coleção de dados rastreia o conteúdo específico do domínio que pode ser organizado como uma coleção de dados, ou seja, representado por dois ou mais elementos de dados.

3.2. Proveniência

Apesar das definições relacionadas à abstração do fluxo de dados, a proveniência de dados é que efetivamente permite o registro do histórico das transformações de dados, especificando quando uma determinada transformação foi instanciada, quais coleções de dados foram consumidas e produzidas, entre outras informações relacionadas à execução. Nesse sentido, apoiar a gerência da proveniência de uma forma correta, precisa e mínima atrelada à abstração de fluxo de dados é essencial para garantir o potencial analítico no contexto desta tese. Sendo assim, esta seção apresenta as noções de proveniência de dados.

3.2.1. Definição

Como apresentado por Buneman *et al.* (2001), a proveniência de dados pode ser definida pela "descrição das origens de uma porção de dados e do processo pelo qual ela alcança uma base de dados" (tradução da seguinte passagem do texto em inglês "*the description of the origins of a piece of data and the process by which it arrived in a database*"). Nesse contexto, a gerência da proveniência dos dados pode ser baseada em duas granularidades, sendo conhecidas como *proveniência em granularidade grossa* e *fina* (FREIRE *et al.*, 2008). A proveniência em granularidade grossa é definida pela descrição das origens de uma porção de dados de acordo com as operações do conjunto de dados. Ou

seja, a proveniência em granularidade grossa compreende a gerência das transformações de dados a partir de uma visão geral de consumo e produção de conjuntos de dados, sem considerar a gerência do fluxo de dados no nível lógico. Como proveniência em granularidade fina, pode-se definir que os dados de proveniência são capturados em relação às derivações dos elementos de dados em cada transformação de dados. Logo, essa granularidade considera as transformações de dados do ponto de vista do fluxo de elementos ou de coleções de dados.

Para permitir a padronização, o compartilhamento e a interoperabilidade de dados de proveniência, diferentes grupos, como o *World Wide Web Consortium* (W3C), têm proposto modelos de dados que consistem em uma representação de como os dados de proveniência devem ser capturados e armazenados em uma base de dados. A partir da aplicação de domínio, um modelo de dados de proveniência pode ser integrado a artefatos que estão associados ao domínio da simulação computacional. Como um exemplo, o PROV-DM (MOREAU e GROTH, 2013) é uma iniciativa do W3C PROV para representar diferentes tipos de dados de proveniência de forma a não ser específico para apenas um domínio. No cenário de simulações computacionais, o PROV-Df (SILVA *et al.*, 2016c) é nesta tese (Seção 4.4) como um modelo de dados baseado nas recomendações do PROV-DM para capturar e consultar dados de proveniência e dados específicos do domínio em um mesmo repositório durante a execução de simulações computacionais, permitindo que dados de domínio possam ser relacionados aos dados de proveniência e, consequentemente, a análise do fluxo de dados no nível lógico.

3.2.2. Propriedades da proveniência

Uma especificação de proveniência de dados precisa apoiar três propriedades básicas conhecidas como corretude, precisão e minimalidade, para que os usuários sejam capazes de analisar o histórico das transformações de dados em simulações computacionais de uma forma correta e mínima. Nesse sentido, uma proveniência de dados é dita correta, quando ela apresenta todos os elementos de dados de entradas envolvidos na produção de um elemento de dados de saída específico em uma determinada transformação de dados (Definição 3.2.1). Uma proveniência de dados é mais precisa que outra, quando os seus elementos de dados estão contidos no conjunto de elementos de dados da outra proveniência (Definição 3.2.2). Consequentemente, uma proveniência é mínima, quando não existe outra

proveniência correta tão precisa quanto ela (Definição 3.2.3). Esses conceitos foram propostos originalmente no trabalho de Ikeda *et al.* (2013) e, nesta tese, adaptamos o seu formalismo para a abstração de fluxo de dados, conforme apresentado a seguir.

Definição 3.2.1. (Corretude) Assume-se que uma instância da transformação de dados t é representada por $c_o = t^*(c_i)$. Considerando um elemento de dados de saída $e \in c_o$, a proveniência $c_i' \subseteq c_i$ é correta para e com relação à t^*, se $\forall c_i' \subseteq c_i : e \in t^*(c_i) \Leftrightarrow e \in t^*(c_i \cap c_i')$.

Definição 3.2.2. (Precisão) Assume-se que uma instância da transformação de dados t é representada por $c_o = t^*(c_i)$ e que $e \in c_o$. Supondo que os subconjuntos $c_i' \subseteq c_i$ e $c_i'' \subseteq c_i$ sejam ambas proveniências corretas para e com relação à t^*, c_i' é, pelo menos, tão preciso quanto c_i'', se $c_i' \subseteq c_i''$. c_i' é mais preciso que c_i'', se $c_i' \subset c_i''$.

Definição 3.2.3. (Minimalidade) Assume-se que uma instância da transformação de dados t é representada por $c_o = t^*(c_i)$ e que $e \in c_o$. Supondo que o subconjunto $c_i' \subseteq c_i$ seja uma proveniência correta para e com relação à t^*, c_i' é mínimo para e, se não existir uma proveniência correta c_i'' para e com relação à t^*que seja mais precisa que c_i'.

3.2.3. Proveniência dos elementos de dados

Como proposto por Ikeda *et al.* (2013), um *workflow orientado a dados* é composto de transformações de dados, considerando alguns conjuntos de dados de entrada e outros conjuntos de dados de saída. Do mesmo modo, um fluxo de dados, conforme definido na Seção 3.1, pode ser representado por um *workflow* orientado a dados. Ao assumir que cada transformação de dados t_i (ou simplesmente t) contém uma especificação de proveniência $prov_t$, também é possível descrever a proveniência de um elemento de dados de saída $e \epsilon o.C$ a partir de instâncias da transformação de dados t por $prov_t(e) \subseteq \bigcup_{k=1}^{card(t.I)} i_k.C$, onde i_k representa um conjunto de dados de entrada da transformação t. Adotando-se esse conceito, pode-se definir a *proveniência do elemento de dados* de uma forma recursiva.

Definição 3.2.4. (Proveniência do Elemento de Dados) Assumindo um fluxo de dados D_F, a proveniência do elemento de dados e em D_F, denotada por $prov_{D_F}(e)$, é um subconjunto de elementos de dados de entrada a partir de instâncias das transformações de dados anteriores. Considerando que t seja a transformação de dados que produz o elemento de dados de saída e, então $prov_{D_F}(e) = \bigcup_{e' \epsilon\, prov_t(e)} prov_{D_F}(e')$, onde $prov_t(e)$ é a proveniência em apenas um nível de e com relação às instâncias de t. Se e é um elemento de dados de entrada, então $prov_{D_F}(e) = \{e\}$.

3.2.4. Categorias de dados de proveniência

Considerando os diversos aspectos do processamento de dados, existem duas categorias de proveniência: prospectiva e retrospectiva. A proveniência prospectiva tem o objetivo de apresentar uma visão geral sobre a composição do fluxo de dados, uma vez que ela descreve os componentes estruturais do fluxo de dados (*e.g.*, transformações de dados, os seus conjuntos de dados e as suas dependências). Desse modo, o uso dos dados dessa categoria permite apenas verificar a representação ou a especificação do fluxo de dados. Ou seja, as análises são restritas a quais são as transformações de dados definidas na simulação computacional, assim como quais conjuntos são consumidos e produzidos por cada transformação em um nível mais abstrato (*e.g.*, nome do conjunto de dados, sem considerar os elementos de dados). Consequentemente, os usuários do domínio científico não são capazes de analisar informações associadas à execução das transformações (*e.g.*, o tempo de execução de um programa e a linha de comando para a invocação desse programa).

A proveniência retrospectiva, por outro lado, captura informações sobre os dados processados durante a execução de cada transformação de dados, como o tempo decorrido para uma transformação de dados específica, os arquivos consumidos por cada transformação de dados, ou mesmo o diretório em que acontece a invocação de uma transformação de dados. Outra característica consiste em permitir que os usuários do domínio científico monitorem o desempenho, assim como os dados de domínio manipulados por uma instância de uma transformação de dados em tempo de execução. Além disso, os usuários podem utilizar esses dados de proveniência retrospectiva para realizar análises mais aprofundadas quanto à simulação computacional executada, a fim de validar suas hipóteses científicas.

Em alguns casos, os resultados obtidos pelos programas de simulação, sejam eles dados específicos do domínio ou de desempenho, podem não apresentar o comportamento esperado do ponto de vista científico. Assim, os usuários do domínio precisam interagir com os especialistas em ciência da computação para realizar ajustes na própria configuração do fluxo de dados ou abortar a execução da simulação, caso ela ainda não tenha sido finalizada (MATTOSO *et al.*, 2013). Após tais ajustes e novas submissões de execução da simulação, eles conseguiriam analisar e validar os novos resultados obtidos. Nesse sentido, diversos estudos têm sido desenvolvidos para destacar a importância da intervenção humana

no processo exploratório de simulações computacionais a partir dos dados de proveniência. Esta proposta de trabalho é conhecida pelo termo *putting the Human-In-the-Loop* (HIL) (W.F.J., 2007), que faz uma alusão ao processo de inserir o ser humano nesse desenvolvimento iterativo. Para que isso seja possível, os dados de proveniência são imprescindíveis para garantir a rastreabilidade, a confiabilidade e a reprodutibilidade das simulações computacionais, em especial os associados à proveniência retrospectiva.

3.2.5. Rastro de proveniência

No cenário de fluxo de dados, os dados de proveniência podem ser capturados durante a execução das transformações de dados, de forma que *rastro de proveniência* seja registrado. Desse modo, eles permitem que os usuários do domínio científico realizem análises baseadas no fluxo dos elementos de dados (ou das coleções de dados) em diferentes transformações de dados. Além disso, o rastro de proveniência pode ser classificado em duas granularidades: grossa e fina. O *rastro de proveniência em granularidade grossa* considera a análise dos conjuntos de dados consumidos e produzidos pelas transformações ao longo do fluxo de dados. Enquanto isso, o *rastro de proveniência em granularidade fina* define um nível de abstração mais baixo, uma vez que esse rastro é baseado nas transformações dos elementos de dados, ou seja, em fragmentos dos conjuntos de dados consumidos e produzidos por instâncias das transformações de dados.

Para apoiar a gerência do rastro de proveniência em granularidade fina, os elementos de dados dos conjuntos de dados manipulados pelas transformações de dados precisam ser relacionados. Para isso, os esquemas dos conjuntos de dados (Definição 3.2.5) precisam ser considerados e as três propriedades seguintes são fundamentais nesse contexto: mapeamento de atributo, mapeamento de múltiplos atributos e transitividade nos mapeamentos de múltiplos atributos. Esses conceitos foram propostos por Ikeda *et al.* (2013) e foram adaptados nesta tese para a abstração de fluxo de dados, conforme apresentado nas Definições 3.2.6 e 3.2.7, assim como no Teorema 3.2.1. As propriedades de mapeamento de atributos e de mapeamento de múltiplos atributos definem como os atributos de um conjunto de dados de entrada são transmitidos para o conjunto de dados de saída. A transitividade em mapeamentos de atributos assume a existência de dependências funcionais entre atributos dos elementos de dados de saída referentes à uma transformação de dados anterior e atributos dos elementos de dados de entrada de uma próxima

transformação de dados. Essa condição também implica que a próxima transformação de dados precisa que os elementos de dados sejam gerados pela transformação de dados anterior para começar a sua execução.

Definição 3.2.5. (Esquema do Conjunto de Dados) Um conjunto de dados s possui um esquema de dados $\mathcal{R} = s.A \cup A_{T'}$, que consiste de uma sequência de atributos de domínio a partir de s e de atributos identificadores $A_{T'}$ para as instâncias de cada transformação de dados de T'.

Definição 3.2.6. (Mapeamento de Atributo) Uma transformação de dados t tem um mapeamento de atributo $i.a' \leftrightarrow o.a''$, se $\forall x \in domain(a'') \wedge (a'.name = a''.name) \wedge (a'.type = a''.type) \wedge \left(\forall i' \in i : \left(\sigma_{a''=x} t(i') = \sigma_{a''=x} t\left(\sigma_{a'=x}(i')\right)\right)\right)$.

Definição 3.2.7. (Mapeamento de Múltiplos Atributos) Uma transformação de dados t tem um mapeamento múltiplo de atributos $i.A' \leftrightarrow o.A''$, se $card(i.A') = card(o.A'') \wedge$
$(\forall a' \in i.A', \forall a'' \in i.A'' : i.a' \leftrightarrow o.a'')$.

Teorema 3.2.1. (Transitividade em Mapeamentos de Múltiplos Atributos) Se a transformação de dados t' tem um mapeamento de múltiplos atributos $s'.A' \leftrightarrow s''.A''$, a transformação de dados t'' tem um mapeamento de múltiplos atributos $s''.A'' \leftrightarrow s'''.A'''$, e o fluxo de dados D_F tem uma dependência de dados $\varphi = (s'', t', t'')$, então existe também, por transitividade, o mapeamento de múltiplos atributos $s'.A' \leftrightarrow s'''.A'''$ em D_F.

Diferentemente de Ikeda *et al.* (2013), que argumentam que a adição do apoio à proveniência física (ou seja, uso de identificadores para as instâncias da transformação de dados) adiciona uma sobrecarga de tempo considerável à carga de dados, esta tese apresenta uma abordagem que considera o armazenamento de dados, tanto da proveniência lógica (*i.e.*, atributos do domínio científico) como da proveniência física (Definição 3.2.5). Ao mesmo tempo, Ikeda *et al.* (2013) também enumeram as limitações relacionadas à preservação da precisão e, consequentemente, da minimalidade ao utilizarem uma abordagem exclusivamente baseada na proveniência lógica. Por esse motivo, a nossa abordagem segue uma proveniência híbrida (ou seja, física e lógica).

A Figura 3 mostra a diferença entre as abordagens de proveniência física e lógica a partir do exemplo de fluxo de dados para a contagem de palavras em arquivos de texto (Exemplo 3.1.1). Mais especificamente, a Figura 3(a) mostra um exemplo de mapeamentos físicos em que os atributos das instâncias das transformações de dados t_1 e t_2 (*i.e.*,

T1_TASKID e *T2_TASKID*) são relacionados entre os conjuntos de dados s_1, s_2 e s_3, respectivamente. Já a Figura 3(b) mostra um exemplo de mapeamentos lógicos em que os atributos específicos do domínio (*i.e.*, *FILE* e *WORD*) são relacionados pelos conjuntos de dados s_1, s_2 e s_3.

Diante dessas definições, a função $prov_t(e)$, que analisa a proveniência de um elemento de dados, investiga o mapeamento de múltiplos atributos na transformação de dados t, define quais atributos apresentam relação entre o consumo e a produção de dados, e identifica, por meio desse mapeamento, quais elementos de dados de entrada foram responsáveis pela produção de e. Esse comportamento seria semelhante ao uso dos atributos do mapeamento como atributos de equidade ao realizar a junção (operação da álgebra relacional) entre um conjunto de dados de entrada e um conjunto de dados de saída para uma determinada transformação, realizando a projeção apenas dos elementos de dados de entrada. De forma semelhante, os resultados dessas junções poderiam ser utilizados em outras operações de junções com conjuntos de dados manipulados por outras transformações de dados. Assim, essa proposta seria capaz de analisar a proveniência do fluxo de dados e, consequentemente, de apoiar a análise de elementos de dados relacionados por múltiplas transformações de dados.

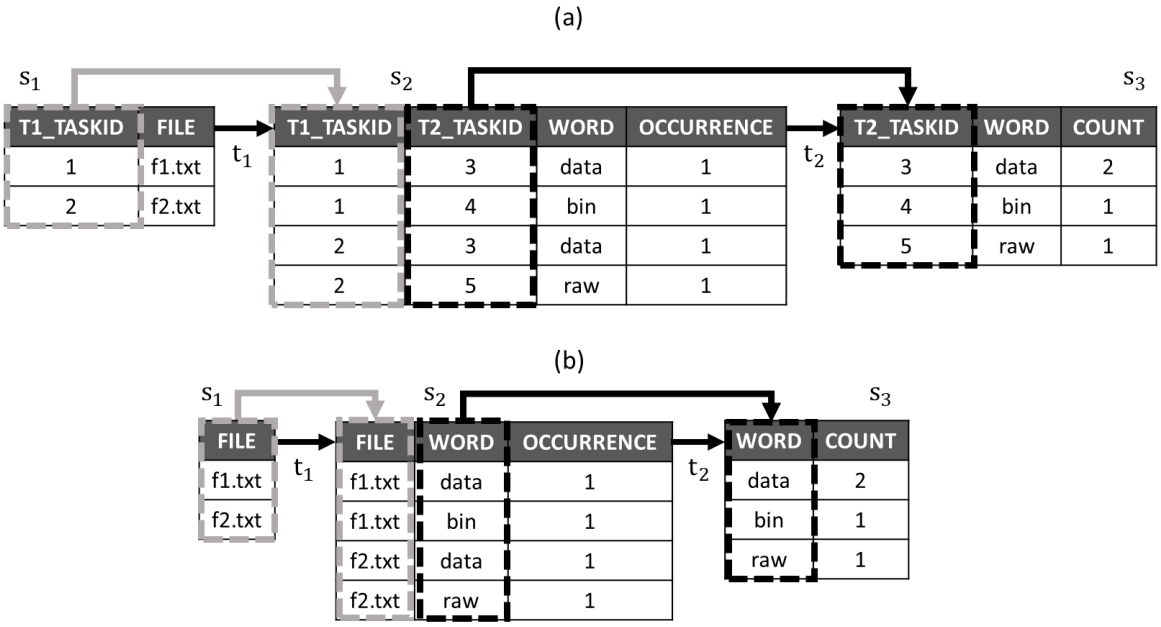


Figura 3. Proveniência de dados no nível (a) físico e (b) lógico.

3.2.6. Análise de dados de proveniência

A partir dos conceitos apresentados nas subseções anteriores, essa subseção discute o potencial analítico ao capturar dados de proveniência, assumindo-se que os dados científicos sejam acessíveis (*e.g.*, uso de ferramentas alternativas para capturar tais dados armazenados frequentemente em arquivos – FastBit). Tais potenciais e benefícios de utilizar dados de proveniência são discutidos no estudo proposto por Mattoso *et al.* (2015). Nessa linha, quatro tipos de análises de dados de proveniência são discutidos nessa seção, sendo os seguintes: monitoramento, *awareness*, depuração e investigação de dados de domínio.

O monitoramento da execução do fluxo de dados baseia-se no processo de verificar o estado da execução em um determinado momento da simulação computacional para identificar se ocorreu (ou está ocorrendo) algum comportamento específico (MATTOSO *et al.*, 2015). Para que seja possível apoiar esse tipo de análise, os componentes responsáveis pelo monitoramento dependem do sistema de fluxo de dados. Em função dessa análise ser realizada em tempo de execução, a ferramenta de monitoramento precisa coletar os dados durante a condução da simulação computacional. Se considerarmos um caso real, espera-se que, com essas informações, os usuários do domínio científico sejam capazes de analisar comportamentos inesperados em tempo de execução.

Diferentemente do monitoramento, que é caracterizado por atuar de uma forma mais passiva, as análises de *awareness* visam informar o estado atual da execução da simulação computacional (e do fluxo de dados gerado) aos usuários do domínio científico, a fim de que eles possam planejar ações por meio do rastro dos dados de proveniência. Essas ações são normalmente baseadas em consultas, a fim de permitir que os usuários do domínio científico com o auxílio de especialistas em ciência da computação evitem falhas, antecipem problemas ou melhorem os resultados. No caso das melhorias dos resultados específicos do domínio, eles podem realizar refinamentos na composição do fluxo de dados (*i.e.*, na sua estrutura) ou ajustes finos nos elementos de dados (*i.e.*, valores assumidos para atributos específicos do domínio da simulação computacional).

As análises de depuração são caracterizadas pela avaliação de resultados que, por meio de métricas, descrevem se o sistema está apresentando o comportamento esperado durante a execução do fluxo de dados. As métricas para depuração precisam ser definidas, coletadas e salvas durante a execução do fluxo de dados, a fim de que os cientistas possam

melhorar o desempenho do sistema para a execução paralela e resolver eventuais falhas relacionadas ao próprio fluxo de dados.

Por último, as análises com dados de domínio investigam o conteúdo dos elementos de dados de saída, relacionando-os com outros tipos de dados de proveniência. Nesse caso, os usuários estão normalmente dedicados a verificar um fragmento de conjunto de dados referentes ao domínio da simulação (*i.e.*, seleção de elementos de dados). Em cenários mais complexos, eles desejam investigar esse fragmento rastreando as suas origens e a sua evolução ao longo da simulação computacional (*i.e.*, junção de conjuntos de dados). A partir de análises minuciosas, os usuários são capazes de compreender o comportamento de um fragmento específico em um determinado domínio da ciência, restringindo a sua análise apenas aos atributos de interesse (*i.e.*, projeção de atributos em um conjunto de dados). De acordo com a conclusão obtida, eles ainda conseguem sugerir ajustes nos elementos de dados de domínio, para que os resultados irrelevantes sejam filtrados, reduzindo a quantidade de dados intermediários analisados ao longo do fluxo de dados.

A investigação de dados de domínio também pode utilizar dados de proveniência associados às informações da execução da simulação computacional, o que potencializa a capacidade analítica. Nessa circunstância, vale destacar a importância de um modelo de dados de proveniência capaz de integrar esses diferentes tipos de dados (Seção 3.2.1) (DE OLIVEIRA *et al.*, 2015, OCAÑA *et al.*, 2015, SILVA *et al.*, 2016b, SOUZA *et al.*, 2015).

3.3. *Workflow* Científico

Outra forma de lidar com a complexidade das simulações computacionais em larga escala consiste na sua modelagem como *workflows* científicos (DEELMAN *et al.*, 2009). *Workflows* científicos estendem o conceito original de *workflows*. *Workflows* são tradicionalmente vistos como a definição do processo para a criação, o armazenamento, o compartilhamento e a revisão da informação. Analogamente, *workflow* científico é um método de abstração e sistematização do processo experimental ou de parte dele (DIAS *et al.*, 2015). Esta seção tem o propósito de apresentar uma definição de *workflow* científico no cenário de simulação computacional, assim como detalhes sobre o funcionamento de Sistemas de Gerência de *Workflows* Científicos (SGWfC) e os seus recursos para o monitoramento da execução de simulações computacionais.

3.3.1. Definição

Um *workflow* científico é definido pelo encadeamento de atividades responsáveis pela invocação de programas de simulação, onde o encadeamento de duas atividades descreve uma dependência de dados entre elas, ou seja, os dados produzidos por um programa são consumidos por outro. De forma análoga, um *workflow* científico pode ser representado por um fluxo de dados, em que as atividades correspondem às transformações de dados e as dependências de dados correspondem ao fluxo de dados entre duas transformações de dados.

Um *workflow* científico com essas características é geralmente centrado em dados e conduzido por dados ou, como definido por Ogasawara *et al.* (2011), ele seria um *workflow* científico centrado em dados. Ogasawara *et al.* (2011) afirmam também que tal *workflow* centrado em dados pode ser modelado como um grafo direcionado acíclico (conhecido pela sigla DAG, do termo em inglês *Directed Acyclic Graph*), em que os vértices representam as atividades e as arestas representam o fluxo de dados entre as atividades.

3.3.2. Sistema de Gerência de *Workflows* Científicos (SGWfC)

Devido à complexidade do volume de dados processados pelos *workflows* científicos em larga escala, observa-se a necessidade de um sistema para gerenciar a sua composição (*i.e.*, definição da estrutura do *workflow* – atividades e dependências de dados), execução e análise (*e.g.*, mecanismo para o processamento de consultas ou ferramentas de visualização), conforme argumenta Mattoso *et al.* (2010). Esses sistemas também precisam implementar diretivas específicas para lidar com cada ambiente de PAD. Os sistemas com essas características são conhecidos como Sistemas de Gerência de *Workflows* Científicos (SGWfC) (DEELMAN *et al.*, 2009). Como exemplos de SGWfC, podemos citar o Pegasus (DEELMAN *et al.*, 2015), o Swift/T (ZHAO *et al.*, 2008) e o SCC (SILVA *et al.*, 2014).

Assumindo-se a abstração de fluxo de dados para *workflows* científicos, os usuários do domínio científico com o auxílio de especialistas em ciência da computação, que conhecem um determinado SGWfC, realizam a especificação das transformações de dados e as suas dependências de dados usando esse SGWfC. De acordo com a especificação do *workflow* científico, o SGWfC gera um plano de execução respeitando um determinado modelo de execução, que também é conhecido como modelo de computação (NAJJAR *et*

al., 1999). Cada SGWfC pode apresentar um ou mais modelos de computação e a execução de um modelo determina como os dados são processados, considerando as restrições de especificação do *workflow* científico.

Ao considerar a definição de fluxo de dados, os elementos de dados de entrada podem ser subdivididos em tarefas, que consomem um subconjunto de elementos de dados e processam esses elementos de dados de acordo com a especificação da transformação de dados. Assumindo-se que as tarefas tenham sido geradas, elas podem ser distribuídas para os recursos computacionais a fim de realizar o processamento dos dados. Dessa forma, esta tese está direcionada aos estudos de SGWfC que realizam a distribuição de tarefas para recursos computacionais remotos, ou seja, os que apresentam recursos para a execução paralela e distribuída tipicamente requisitada pelos ambientes de PAD.

3.3.3. Monitoramento de *workflows* científicos

Para monitorar a execução de *workflows* científicos, os SGWfC têm apresentado basicamente três propostas para representar informações quanto à estrutura do *workflow*, à sua execução e ao domínio da simulação computacional, sendo as seguintes: arquivos de *log*, apoio à proveniência de dados e uso de ferramentas alternativas. Dessas propostas, a maioria dos SGWfC geram arquivos de *log* durante a execução de *workflows* científicos. Esses arquivos apresentam, em geral, informações sobre o desempenho ou domínio da simulação computacional. Ao mesmo tempo, esses sistemas frequentemente lidam apenas com operações de escrita nesses arquivos (operações de uma única via), uma vez que não é trivial desenvolver aplicações para a análise desses arquivos de *log*, assim como o custo adicional de acessos concorrentes, caso haja o apoio às operações de leitura.

Por outro lado, no cenário científico, os usuários do domínio científico geralmente necessitam avaliar os seus programas de simulação de acordo com questões de desempenho ou mesmo com os resultados obtidos. Logo, esses usuários precisam analisar o conteúdo dos arquivos de *log* durante ou após a execução do *workflow*. Em ambos os casos, os SGWfC não são capazes de capturar esses tipos de dados, pois os programas de simulação comumente escrevem tais dados em arquivos de *log* da sua maneira, a menos que um padrão de escrita seja imposto pelo SGWfC. VisTrails (CALLAHAN *et al.*, 2006), Kepler e Swift/T são exemplos de SGWfC baseados em arquivos de *log*. De forma similar, o Spark

(ZAHARIA *et al.*, 2010) é um arcabouço de fluxo de dados que também baseia-se no uso de arquivos de *log* para monitorar a execução de aplicações em ambientes de PAD.

Outra abordagem com apoio ao monitoramento de *workflows* científicos considera a captura de dados de proveniência, que podem apresentar dois níveis de granularidade, conforme apresentado na Seção 3.2.1. De acordo com o tipo de granularidade adotado, os SGWfC adotam um padrão para a representação dos dados de proveniência. Por exemplo, a *World Wide Web Consortium* (W3C) apresentou um conjunto de recomendações conhecido como PROV (MOREAU e GROTH, 2013) para permitir a interoperabilidade de dados de proveniência em ambientes heterogêneos, como a própria *Web*. No contexto de simulações computacionais, a heterogeneidade estaria vinculada à possibilidade de desenvolver *workflows* científicos em diferentes domínios da ciência, como a bioinformática (OCAÑA *et al.*, 2011) e a astronomia (JACOB *et al.*, 2009), enquanto apenas um modelo seria capaz de descrever os dados de proveniência em todos os domínios.

Outro aspecto importante consiste no momento em que ocorre a captura de dados de proveniência. Nesse sentido, a proveniência pode ser capturada de uma forma *offline* ou *online*. A captura *offline* de dados de proveniência consiste em armazenar os dados em um determinado repositório após a execução do *workflow* científico, como observado nos SGWfC VisTrails e Kepler. Enquanto isso, a captura *online* de dados de proveniência considera a obtenção e o armazenamento dos dados em tempo de execução, conforme implementado no Swift/T, Pegasus e SCC (SciCumulus + Chiron).

Em função da possível sobrecarga de armazenar os dados de proveniência concorrentemente com o processamento de programas de simulação, a captura de dados de proveniência *online* normalmente apresenta um desempenho pior que a proposta *offline*. Por fato, essa diferença de desempenho é mais aparente em SGWfC que gerenciam a proveniência em uma granularidade mais fina. Ao mesmo tempo, vale enfatizar o potencial analítico de se capturar a proveniência de forma *online*, pois os usuários são capazes de realizar consultas de monitoramento ou mesmo do próprio domínio em tempo de execução. Destaca-se também que existem SGWfC com apoio à captura de dados de proveniência em tempo de execução, mas que não são capazes de obter dados do domínio ou de proveniência em uma granularidade fina, como o Swift/T.

Somando-se a essas abordagens, existem SGWfC que integram o seu mecanismo de monitoramento com ferramentas alternativas, a fim de atender a eventuais limitações. Por exemplo, o Pegasus é um SGWfC paralelo que não é capaz de capturar o conteúdo de arquivos de dados científicos, que podem ser muito relevantes para análises específicas do domínio. Portanto, o uso de soluções para o processamento de consultas em dados científicos poderia permitir a captura do conteúdo presente em arquivos e o seu armazenamento em uma base de dados. Assim, um processador de consultas poderia ser acoplado ao Pegasus para apoiar as análises de dados científicos, enquanto considera também os dados de proveniência e de desempenho já armazenados na sua base de dados. Nesse caso, os próprios especialistas em ciência da computação requerem um conhecimento técnico para realizar a integração dessas tecnologias com sucesso, dado que esse trabalho de desenvolvimento não é trivial. Em relação às análises de dados científicos, existem algumas tecnologias emergentes que poderiam ser empregadas, como o DiNoDB (TIAN *et al.*, 2014), o RAW, o FastQuery e o SDS/Q. Além disso, os SGWfC também podem ser integrados a ferramentas alternativas de outros tipos de dados. Por exemplo, o componente *PerfMetricEval* (SILVA *et al.*, 2016b, SOUZA *et al.*, 2015) foi desenvolvido e integrado ao SCC com o objetivo de integrar dados de proveniência em granularidade fina com informações de desempenho e de consumo de recursos computacionais (*e.g.*, consumo de CPU, quantidade de dados alocados em memória e número de operações de entrada e saída).

3.3.4. SGWfC SCC

Desenvolvido pelos grupos de pesquisa de Computação de Alto Desempenho e Banco de Dados[3] da COPPE / Universidade Federal do Rio de Janeiro e da Universidade Federal Fluminense (SILVA *et al.*, 2014), o SGWfC SCC consiste na integração dos SGWfC Chiron (OGASAWARA *et al.*, 2013) e SciCumulus (DE OLIVEIRA *et al.*, 2010). Como sistema pioneiro do grupo, o Chiron foi desenvolvido para apoiar a execução de *workflows* científicos usando uma máquina paralela dedicada a ambientes de *clusters* e grades computacionais. Já o SciCumulus apresenta as principais características e componentes do Chiron, além de contribuir com algoritmos que tiram proveito de ambientes

[3] hpcdb.wordpress.com

de nuvens computacionais, considerando, *e.g.*, a elasticidade dos recursos computacionais e o custo financeiro (OLIVEIRA *et al.*, 2012).

Com outra meta, o SCC visa uma maior flexibilidade para o usuário ao modelar, submeter, monitorar e analisar *workflows* científicos em ambientes de PAD (SILVA *et al.*, 2014). Além disso, o SCC defende a integração de dados da composição do *workflow*, da sua execução e do próprio domínio em apenas uma base de proveniência, de forma semelhante às publicações do Chiron e do SciCumulus (OCAÑA *et al.*, 2015, OGASAWARA *et al.*, 2013). Para o desenvolvimento da solução proposta nesta tese, utilizou-se a versão mais recente do SCC, que também assume dados de desempenho e de consumo dos recursos computacionais armazenados na base de dados de proveniência.

Em relação ao seu funcionamento, o SCC é responsável pela gerência de um *workflow* científico e dos dados que são consumidos, transformados e produzidos ao longo de cada atividade constituinte desse *workflow*. Além disso, as oportunidades de otimização devem ser consideradas tanto na especificação, como na execução paralela de *workflows* científicos. Para lidar com essa questão, Ogasawara *et al.* (2011) propôs uma abordagem algébrica com o intuito de facilitar a definição de *workflows* científicos, a gerência de dados em paralelo presentes em um fluxo de dados e a otimização de *workflows* em tempo de execução. Essa abordagem algébrica caracterizou-se pela extensão da álgebra relacional tradicional, sendo conhecida como *Scientific Workflow Algebra* (SciWfA), para permitir a otimização e o processamento de consultas em SGBD (ÖZSU e VALDURIEZ, 2011).

Uma vez que um *workflow* científico possa ser representado por um fluxo de dados, então a SciWfA também permite expressar as transformações de dados e como os elementos de dados fluem entre elas. Como principal vantagem, pode-se destacar a adição de semântica às transformações de dados. De acordo com a álgebra proposta por Ogasawara *et al.* (2011), cada transformação de dados $t_i \in T$ (*i.e.*, atividade de um *workflow*) é regida por uma operador algébrico ϕ. Portanto, T apresenta o conjunto de todas as transformações de dados e $\phi \in \{Map, SplitMap, Reduce, Filter, Query\}$. Além disso, t_i consome um conjunto de dados (ou relação, como é tradicionalmente chamado na álgebra relacional) I_i de elementos de dados de entrada e produz um conjunto de dados I_{i+1} de elementos de dados de saída. De forma similar, um conjunto de dados de saída pode ser o conjunto de dados de entrada do operador associado a uma próxima transformação de dados. Dependendo do

operador algébrico, um operando adicional γ também precisa ser definido. Logo, uma transformação de dados algébrica pode ser representada pela seguinte expressão:

$$I_{i+1} \leftarrow \phi(t_i, \gamma, I_i) \quad (1)$$

Um operador algébrico ϕ é definido de acordo com a relação entre o número de elementos de dados consumidos e produzidos para uma determinada invocação da transformação de dados, conforme observado na Tabela 2. Diferentemente do Chiron e do SciCumulus, o SCC apresenta os operadores que usam expressões algébricas relacionais, antes nomeados de SRQuery e MRQuery, implementados como apenas um operador algébrico, conhecido como Query. Tal mudança foi motivada pela redundância semântica desses operadores, pois ambos realizavam o processamento de consultas na base de proveniência. A sua única diferença estava relacionada ao número de conjuntos de dados de entrada. Nessa circunstância, o operador SRQuery considerava apenas um conjunto de dados de entrada, enquanto o operador MRQuery apresentava múltiplos conjuntos de dados.

Operador	**Tipos de atividades**	**Operandos adicionais**	**Relação entre os elementos de dados consumidos e produzidos**
Map	Aplicação	Nenhum	1:1
SplitMap	Aplicação	Atributo que referencia arquivo	1:m
Reduce	Aplicação	Conjunto de atributos de agregação	n:1
Filter	Aplicação	Nenhum	1:(0-1)
Query	Expressão da álgebra relacional	Nenhum	n:m

Tabela 2. Operações da SciWfA (adaptado de Ogasawara *et al.*, 2011)

A relação entre os elementos de dados consumidos e produzidos é calculada a partir da cardinalidade (*i.e.*, número de elementos de dados) do conjunto de dados de entrada e do conjunto de dados de saída. Na execução de um *workflow* científico, um programa de simulação é invocado múltiplas vezes, sendo que cada invocação realiza o consumo de um ou mais elementos de dados de entrada e a geração de um ou mais elementos de dados de saída. Entretanto, os programas de simulação invocados pelo SCC são normalmente feitos por empresas ou grupos de pesquisas que não estão cientes da especificação da SciWfA. Consequentemente, cada invocação pode não produzir os elementos de dados de saída necessários para o operador algébrico definido, que são importantes para o mecanismo de

captura dos dados de proveniência e para servir como entrada para uma próxima transformação de dados. Essa instanciação da execução de uma atividade regida por um operador algébrico da SciWfA é chamada de *ativação*.

De acordo com Özsu e Valduriez (2011), uma ativação é a menor unidade de processamento de dados que não pode ser mais fragmentada, *i.e.*, não pode ser mais paralelizada. No Chiron, o núcleo do SCC, uma ativação pode ser simplificada como a invocação de uma transformação de dados (*i.e.*, programa de simulação). Ademais, Ogasawara *et al.* (2011) afirmam que ela é um objeto autocontido que apresenta informações fundamentais para permitir a execução de uma transformação de dados em qualquer recurso computacional. Assim, uma ativação contém qual aplicação é invocada, quais dados serão consumidos, quais dados serão produzidos e qual é o esquema dos dados produzidos.

A partir da especificação da SciWfA, pode-se observar que uma ativação no SCC é mais que simplesmente a invocação de uma transformação de dados. De fato, ela é caracterizada por três procedimentos: *instrumentação*, *invocação* e *extração* (OGASAWARA *et al.*, 2011). Antes da execução do *workflow* científico, os usuários definem o esquema dos conjuntos de dados de entrada e saída para todas as transformações de dados. Na instrumentação, os elementos de dados de entrada são obtidos e a invocação do programa de simulação é preparada de acordo o esquema dos dados de entrada. Após isso, a invocação de uma ativação consiste em chamar um programa de simulação em um determinado recurso computacional, consumindo os elementos de dados de entrada definidos na instrumentação e produzindo elementos de dados de saída. Como último procedimento, a extração realiza a captura de dados produzidos pelos programas de simulação para serem representados em relações. Mais detalhes sobre a definição formal de ativação podem ser encontrados em (OGASAWARA *et al.*, 2011).

Vale ressaltar que outros dados de proveniência também são capturados durante os procedimentos de uma ativação e podem ser utilizados para apoiar os cientistas em diversas análises. Como exemplos de análises, podemos citar as seguintes questões:

- Existe alguma ativação que ainda esteja executando?

- Algum erro foi constatado nas ativações? Se positivo, qual foi o código do erro observado e a sua mensagem?
- Quanto tempo uma ativação necessita, em média, para executar com sucesso um determinado programa de simulação?
- Existe alguma anomalia associada ao uso dos recursos computacionais?

Mais importante que isso, os dados de proveniência associados aos dados de domínio conseguem auxiliar os usuários na depuração de programas (*i.e.*, dados de desempenho ou erros em função dos valores de atributos do domínio). Isso viabiliza análise mais ricas em tempo de execução, além de permitir o monitoramento da execução (OLIVEIRA *et al.*, 2014, SILVA *et al.*, 2016b, SOUZA *et al.*, 2015).

Em relação à distribuição de ativações para favorecer o paralelismo (Seção 3.3.2), o SCC conta com duas estratégias para gerenciar a execução paralela de *workflows* científicos, sendo as seguintes: *First Tuple First* (FTF) e *First Activity First* (FAF) (Ogasawara *et al.*, 2011). Para explicar a diferença entre essas estratégias, utilizaremos um exemplo simples de *workflow* científico: duas transformações de dados t_1 e t_2, em que t_2 depende de t_1. Na estratégia FTF, uma ativação de t_2 apenas aguarda que os seus elementos de dados de entrada sejam produzidos pela ativação correspondente à t_1 (ou seja, da transformação de dados anterior), sendo esse o comportamento típico de *pipeline* em fluxo de dados. Por outro lado, na estratégia FAF, t_2 apenas começa a execução das suas ativações, quando todas as ativações de t_1 tiverem sido executadas por completo. Logo, t_2 permanece bloqueada até que t_1 termine.

3.4. Abordagens para a captura e a análise de dados científicos

Para apoiar a captura e a análise de dados científicos, existem diferentes abordagens para determinar o momento em que os dados científicos devem ser acessados, extraídos, carregados e analisados. Como premissa para essas etapas, os programas de simulação já devem ter produzido os dados científicos relevantes. A partir dessa condição, as abordagens existentes para a captura e a análise de dados científicos podem ser classificadas em três categorias: *post mortem*, *in situ* e *in transit*. Essas abordagens são apresentadas em mais detalhes nas demais subseções.

3.4.1. Post mortem

As soluções tradicionais, conhecidas como *post mortem*, caracterizam-se por capturar e analisar os dados científicos após a execução dos programas de simulação (ALAGIANNIS *et al.*, 2015, CHOU *et al.*, 2011, KARPATHIOTAKIS *et al.*, 2014, WU *et al.*, 2009). Nesse cenário, os programas de simulação produzem os dados científicos e os armazenam diretamente em arquivos, em formatos específicos de acordo com o domínio da simulação computacional. Em seguida, as soluções dessa categoria acessam o conteúdo desses arquivos, realizam as análises léxica e sintática, extraem conjuntos de dados científicos relevantes, indexam dados de regiões específicas e preparam esses dados para serem consultados posteriormente. Dessa forma, essa abordagem caracteriza-se por ter, pelo menos, duas vezes a interação com meios de armazenamento não-voláteis, que tendem a ser mais lentos, como apresentado na Figura 4(a).

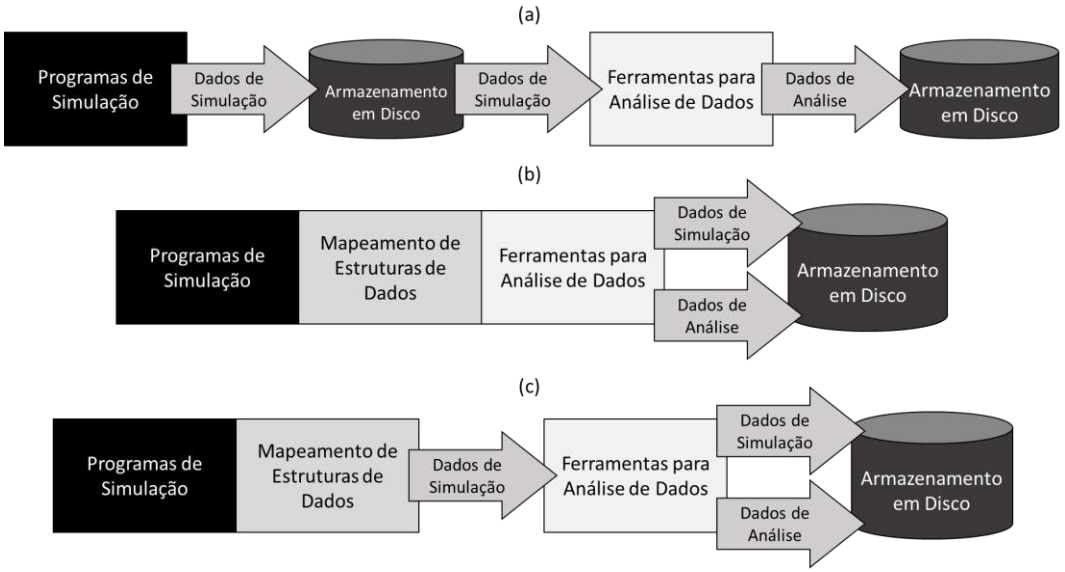


Figura 4. Captura e análise de dados científicos: (a) *post mortem*; (b) *in situ*; e (c) *in transit*. Adaptado de Oldfield *et al.* (2014). As setas representam a movimentação de dados entre recursos computacionais, enquanto os retângulos unidos correspondem ao uso de dados ainda alocados em memória.

Ao mesmo tempo, a principal vantagem dessa categoria consiste em não adicionar sobrecarga de tempo para executar os programas de simulação, pois a captura e a análise de dados científicos podem ser realizadas de forma concorrente, em outros recursos computacionais. Todavia, essa categoria requer mais espaço de armazenamento, pois os dados científicos produzidos pelos programas de simulação precisam ser armazenados em

arquivos (para os programas de extração) para estarem disponíveis em análises futuras. Além disso, essa categoria apresenta como outra desvantagem o fato de não apoiar análises durante a execução das simulações computacionais, sendo esta uma característica essencial na análise exploratória de dados científicos (MATTOSO *et al.*, 2015).

Considerando os trabalhos relacionados à captura e à análise de dados científicos apresentados na Seção 2.1, *e.g.*, o NoDB (ALAGIANNIS *et al.*, 2015) consiste em um processador de consultas a dados científicos que utiliza uma extensão do SGBD PostgreSQL[4], conhecida como PostgresRaw, e respeita as propriedades da categoria *post mortem*. Assim, por meio desse SGBD, o NoDB permite que, ao invés de realizar a carga completa dos dados científicos presentes em arquivos, apenas os atributos presentes em consultas submetidas pelo usuário tenham seus valores extraídos dos arquivos. Consequentemente, o volume de dados carregados no SGBD pode ser menor que as soluções baseadas na extração total dos dados científicos presentes em arquivos. Semelhante ao NoDB, outras soluções têm empregado técnicas de indexação para reduzir o volume de dados carregados e para favorecer o acesso eficiente (*i.e.*, mais rápido) aos dados científicos, como o FastBit (WU *et al.*, 2009) e o AQUAdex (HONG *et al.*, 2015).

3.4.2. In situ

Diante dos avanços nas simulações, os modelos computacionais têm se tornado cada vez mais complexos, produzindo, consequentemente, um volume de dados maior. Mesmo assim, as melhorias de desempenho associadas ao armazenamento de dados (*i.e.*, operações de leitura e escrita em disco) não têm acompanhado o progresso em termos de processamento computacional (*i.e.*, uso de CPU). Assim, simulações em larga escala têm apresentado uma dificuldade maior em lidar com latências entre a geração dos dados científicos e o seu efetivo armazenamento, como discutido em (AYACHIT *et al.*, 2016, BENNETT *et al.*, 2012). Com o objetivo de tirar proveito de dados já alocados na memória pelos programas de simulação, ou seja, antes do efetivo armazenamento em arquivos, soluções baseadas na categoria *in situ* buscam se acoplar aos programas de simulação para serem capazes de capturar e analisar os dados científicos relevantes durante a execução das simulações computacionais (AYACHIT *et al.*, 2016, BAUER *et al.*, 2016, LASLUISA *et*

[4] https://www.postgresql.org/

al., 2015, RUBEL *et al.*, 2016). Desse modo, as soluções *in situ* utilizam os mesmos recursos computacionais dedicados aos programas de simulação para analisar dados científicos e evitam que todos os dados acessados sejam armazenados em disco rígido (*i.e.*, armazenamento não-volátil), como apresentado na Figura 4(b).

Ademais, para que seja possível relacionar os dados produzidos pelos programas de simulação com as estruturas de dados utilizadas pelas ferramentas de análise e visualização de dados científicos, as soluções dessa categoria necessitam de um mapeamento das estruturas de dados entre os programas de simulação e as ferramentas de análise. Uma vez definido este mapeamento, os dados podem ser compartilhados entre eles por meio do uso da memória volátil, enquanto que as ferramentas de análise de dados podem executar suas rotinas para validarem hipóteses científicas, extrair determinados elementos de dados de interesse, ou produzir visualizações. Como exemplo dessa categoria, o SENSEI (AYACHIT *et al.*, 2016) consiste em uma interface genérica entre os programas de simulação e as soluções de análise de dados científicos, permitindo o compartilhamento de dados em memória volátil. Para realizar o mapeamento dos dados, o SENSEI requer que o usuário desenvolva um *adaptador de dados*, responsável por definir quais dados do programa de simulação são relevantes e devem ser mantidos em memória volátil de acordo com a estrutura do modelo de dados do VTK (VTK, 2009). Somando-se a isso, essa solução também requer a especificação de um *adaptador de análise* que utiliza os dados descritos em objetos do VTK em códigos de programas para analisar dados científicos. O ParaView Catalyst (AYACHIT *et al.*, 2015), o VisIt[5] e o ADIOS (LIU *et al.*, 2014) são exemplos de ferramentas para a análise e a visualização de dados que o SENSEI oferece suporte.

3.4.3. In transit

Em outros cenários, as análises de dados científicos podem exigir uma capacidade de processamento computacional, ou mesmo de gerência dos dados, com potencial de interferir no desempenho dos modelos computacionais executados pelos programas de simulação. Ou seja, elas podem aumentar o tempo de execução da simulação computacional significativamente. Nesses casos, as soluções conhecidas como *in transit* permitem que os dados científicos acessados sejam mapeados e utilizados pelas ferramentas de análise de

[5] https://visit.llnl.gov

dados em outros recursos computacionais, como apresentado na Figura 4(c). Com isso, a principal diferença para a abordagem *in situ* consiste em não executar os programas de simulação nos mesmos recursos computacionais das ferramentas de análise de dados. Em relação à abordagem *post mortem*, essa abordagem também apresenta potencial de redução do volume de dados armazenados em disco rígido, pois os usuários do domínio científico podem analisar (ou investigar) os dados ainda alocados em memória volátil para armazenar apenas os dados relevantes, como também observado na abordagem *in situ*.

Em contrapartida, essa abordagem introduz um custo de comunicação entre os recursos computacionais, já que os dados gerados pelos programas de simulação precisam ser transferidos para os recursos computacionais em que serão utilizadas as ferramentas de análise de dados. Ao mesmo tempo, esse custo é proporcional ao volume de dados envolvido nas análises desejadas e depende da configuração da rede. Do ponto de vista do usuário, essa abordagem apresenta uma latência para prover os resultados das análises aos usuários do domínio científico em função do custo de comunicação. Diante dessas abordagens, destaca-se que a escolha da solução a ser aplicada depende do momento em que se deseja realizar as análises, do tipo de análises a serem realizadas (assim como o volume de dados envolvido) e dos impactos em termos de desempenho ao introduzir esse apoio analítico.

Capítulo 4 - ARMFUL: uma arquitetura para a captura e a análise de dados científicos

A partir da motivação, da caracterização do problema e do referencial teórico, este capítulo apresenta a abordagem utilizada para proporcionar a análise exploratória de dados científicos relacionados ao longo do fluxo de dados. Mais especificamente, este capítulo apresenta uma visão geral da arquitetura baseada em componentes, conhecida como ARMFUL (do termo em inglês, *Analysis of Raw data from Multiple Files* ou, em português, Análise de Dados Científicos a partir de Múltiplos Arquivos), que lidam com a captura e a análise de dados de proveniência e dados científicos (Seção 4.1). Após essa visão geral da nossa arquitetura, esse capítulo detalha as características de cada componente, considerando o acesso aos dados científicos, a geração de índices, o armazenamento dos dados capturados e o processamento de consultas. Além disso, esse capítulo apresenta o diagrama PROV-Df, que foi proposto para representar os fluxos de dados gerados pelas simulações computacionais, assumindo os dados de proveniência e os dados científicos capturados de arquivos. A arquitetura ARMFUL pode ser instanciada em diferentes ambientes computacionais. Resumidamente, duas instâncias da ARMFUL foram desenvolvidas, sendo elas conhecidas como SCC-A e DfAnalyzer. A primeira instância (Seção 4.7), SCC-A, consistiu em uma extensão do Sistema de Gerência de *Workflows* Científicos (SGWfC) SCC para apoiar a captura e análise de dados científicos. A segunda instância (Capítulo 5) corresponde à biblioteca de componentes DfAnalyzer para analisar o fluxo de dados gerado por simulações computacionais, que se caracteriza por ser uma implementação da ARMFUL independente de um SGWfC.

4.1. Visão Geral

A partir da abstração de fluxo de dados e da importância da gerência dos dados de proveniência em uma granularidade fina, conforme apresentado respectivamente nas Seções 3.1 e 3.2, esta tese apresenta a arquitetura baseada em componentes ARMFUL[6] (do termo em inglês, *Analysis of Raw data from Multiple Files* ou, em português, Análise de Dados Científicos a partir de Múltiplos Arquivos) (SILVA *et al.*, 2017b). A ARMFUL

[6] https://hpcdb.github.io/armful

apoia a gerência dos dados de proveniência e dos dados científicos envolvidos no encadeamento dos programas de simulação a fim de que mecanismos subsequentes também sejam capazes de analisar esses dados durante ou após as simulações computacionais.

Mais especificamente, essa arquitetura apoia o processamento de consultas aos dados científicos presentes em arquivos ou alocados em memória, assim como analisa o fluxo de elementos de dados científicos oriundos da execução da simulação computacional (pertencentes a diferentes transformações de dados). Para isso, adotamos a gerência da proveniência dos elementos de dados no nível físico e lógico, monitorando, respectivamente, os identificadores das instâncias das transformações de dados (também conhecidas como tarefas) e os valores dos atributos específicos do domínio (*i.e.*, dados científicos). Essa abordagem de gerência da proveniência dos elementos de dados é descrita na Seção 3.2.

Ademais, esta tese defende que a ARMFUL seja baseada em duas etapas para apoiar a análise exploratória de dados científicos. Primeiramente, considera-se uma etapa dedicada à captura de dados científicos de forma integrada aos dados de proveniência. Portanto, o fluxo de arquivos e os relacionamentos necessários para permitir a gerência do fluxo de dados no nível lógico (*i.e.*, fluxo dos elementos de dados) são requisitos dessa etapa, conforme discutido no Capítulo 3. Na segunda etapa, propõe-se um mecanismo capaz de processar consultas que assumem o acesso aos dados de proveniência e aos dados científicos, ainda considerando o fluxo de dados nos níveis físico e lógico.

A Figura 5 mostra os comportamentos esperados na etapa de captura de dados de proveniência de uma forma integrada com os dados científicos por meio das seguintes camadas: *Captura de Dados Científicos*, *Captura de Dados de Proveniência* e *Armazenamento de Dados*. Semelhante aos SGWfC existentes que apoiam a gerência de dados de proveniência, os componentes da camada *Captura de Dados de Proveniência* são responsáveis pela captura de dados de proveniência e pelo armazenamento dos mesmos em um determinado repositório (componente da camada *Armazenamento de Dados*), que pode ser, *e.g.*, uma base de dados. Nesse caso, considera-se apenas dados associados à composição e à execução do fluxo de dados, como a especificação dos programas de simulação utilizados em cada transformação de dados e o tempo de execução de uma determinada transformação de dados (*i.e.*, dados de proveniência prospectiva e

retrospectiva). Ademais, essa camada requer um mecanismo capaz de validar os dados de proveniência que estão sendo capturados, a fim de que não existam inconsistências na base de dados (componente *Validador de Dados de Proveniência*).

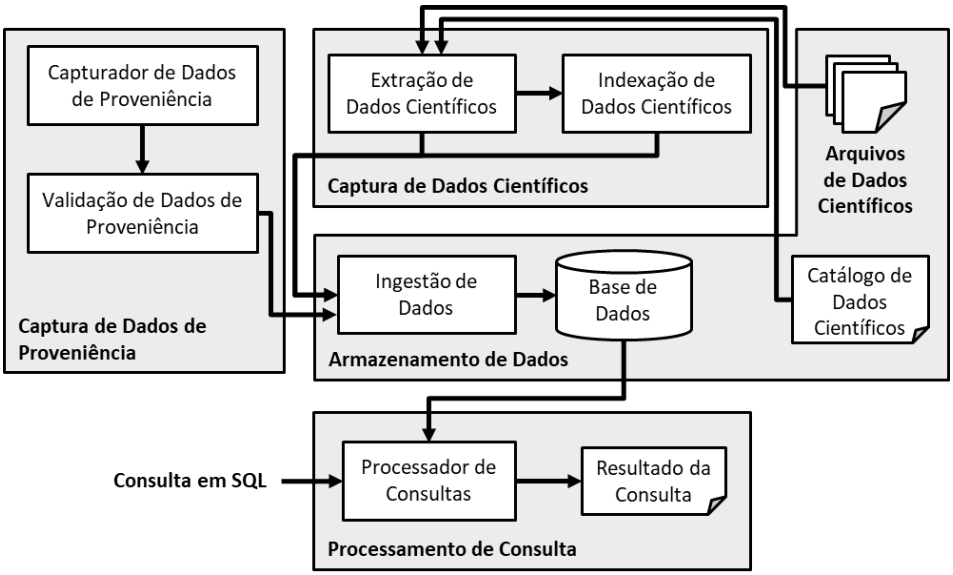


Figura 5. Camadas da arquitetura ARMFUL.

Já os componentes da camada *Captura de Dados Científicos* são responsáveis por acessar, extrair e, eventualmente, indexar os dados científicos manipulados pelos programas de simulação, que podem estar armazenados em arquivos ou mesmo alocados em memória (processamento *in situ*). Uma vez que os dados científicos tenham sido capturados, eles podem ser carregados em uma base de dados de proveniência integrada. Vale enfatizar que essa etapa depende bastante da camada *Armazenamento de Dados*, pois os dados científicos são comumente armazenados em arquivos e o catálogo para esses dados são armazenados em disco (componente *Catálogo de Dados Científicos*).

Somando-se a isso, o volume de dados científicos gerado durante a execução das simulações computacionais é um fator essencial para definir a melhor estratégia de representação desses dados. Assumindo-se, por exemplo, uma simulação que gera um pequeno volume de dados em cada transformação de dados, ao mesmo tempo em que apresentam poucos atributos a serem capturados em cada arquivo de dados científicos e uma estrutura de dados "padrão" (*e.g.*, ponto flutuante, inteiro ou cadeia de caracteres), a melhor abordagem nesse caso seria carregar os dados científicos com o próprio valor de cada atributo dos seus arquivos na base de dados. Ressalta-se que, nesta tese, o termo ingestão de dados compreende a carga dos dados em uma base de dados de proveniência.

Entretanto, no caso em que um grande volume de dados precisa ser capturado e as estruturas de dados são mais complexas (*e.g.*, *octrees* e malhas), pode-se empregar técnicas de indexação de dados científicos (CHOU *et al.*, 2011, KIM *et al.*, 2011). Essas técnicas baseiam-se na geração de índices para referenciar os dados científicos nos arquivos. Assim, os valores armazenados na base de dados de proveniência integrada seriam ponteiros (*i.e.*, índices) que serviriam como referências para consultas nos arquivos de dados científicos.

Nesse cenário, o acesso e a captura dos elementos de dados de interesse (*i.e.*, valores dos atributos) podem ser realizados apenas pelo componente *Extração de Dados Científicos* da Figura 5, enquanto que o uso de técnicas de indexação de dados exige que esse componente seja integrado ao componente de *Indexação de Dados Científicos* para gerar índices apenas para os elementos de dados de interesse. Ao mesmo tempo, o componente *Ingestão de Dados* também precisa estar ciente da abordagem de captura de dados científicos, ou seja, extração ou indexação, para que os elementos de dados ou os índices gerados sejam carregados de forma apropriada na base de dados.

Uma vez que os índices gerados para os dados científicos ou mesmo os seus valores tenham sido armazenados em uma base de dados integrada, os usuários podem desenvolver e submeter consultas em SQL para o componente *Processador de Consulta*, que retornará os resultados obtidos da base de dados. De forma mais específica, o processador de consulta precisa ter ciência de qual abordagem de captura de dados científicos foi adotada para definir se ele usará apenas os recursos de processamento de consultas do Sistema de Gerência de Banco de Dados (SGBD) empregado para a base de dados, ou se ele precisará dos recursos de ferramentas alternativas utilizadas para indexar os dados científicos (só que agora para acessar os dados por meio dos índices), como o FastBit.

4.2. Camada de captura de dados científicos

Conforme discutido no Capítulo 2, a análise exploratória de dados científicos caracteriza-se pela investigação por parte dos usuários do domínio científico de resultados parciais e finais obtidos a partir da execução de simulações computacionais. Nesse cenário, os dados científicos produzidos pelos programas de simulação são comumente armazenados em arquivos ou alocados em memória. Dessa forma, para apoiar a análise desses dados, as soluções existentes nessa linha de pesquisa precisam se concentrar no acesso e na captura dos dados científicos relevantes que estão presentes nessas fontes de dados, a fim de que os

dados capturados sejam carregados em repositórios externos ou bases de dados para estarem disponíveis no processamento de consultas. Nessa seção apresentamos as abordagens existentes para a captura de dados científicos que se dedicam à extração dos dados (Subseção 4.2.1) e ao uso de técnicas de indexação (Subseção 4.2.2) para reduzir o volume de dados carregados em repositórios externos e favorecer o acesso eficiente aos dados científicos por meio dos índices gerados.

4.2.1. Extração de dados científicos

Tendo como objetivo a captura dos dados científicos, presentes em arquivos, manipulados por transformações de dados, o componente *Extração de Dados Científicos* realiza o acesso a esses dados por meio de duas análises, conhecidas como léxica e sintática. Na ciência da computação, a análise léxica (conhecida pelo termo em inglês *tokenizing*) caracteriza-se pela leitura da entrada de linhas de caracteres e pela geração de uma sequência de símbolos léxicos, conhecidos pelo termo em inglês *lexical tokens* (BLANAS *et al.*, 2014). Logo, em outras palavras, a análise léxica permite a verificação de um determinado alfabeto. Na Figura 5, o alfabeto está presente no *Catálogo de Dados Científicos*, que consiste em um conjunto de metadados necessários para a compreensão do alfabeto adotado em cada formato de arquivo (*e.g.*, arquivos no formato HDF5 em simulações de dinâmica de fluidos computacionais). Assim, a análise léxica consiste na capacidade de acessar os dados científicos de acordo com o formato do arquivo e de verificar se os dados científicos obtidos estão de acordo com o domínio da simulação computacional.

A partir dos símbolos gerados na análise léxica, a análise sintática (conhecida pelo termo em inglês *parsing*) baseia-se na verificação da sequência de caracteres de entrada para determinar estruturas segundo uma determinada gramática formal (BLANAS *et al.*, 2014). Em termos práticos, um analisador sintático pode ser utilizado para decompor o texto em unidades estruturais a serem organizadas dentro de um bloco de disco. No cenário de acesso aos dados científicos, pode-se utilizar um analisador sintático para transformar a sequência de caracteres de entrada em uma estrutura de dados conhecida (*e.g.*, grafo), assim como para fornecer informações pertinentes para a indexação posterior dos dados científicos, se aplicável. Semelhante à análise léxica, a análise sintática precisa ter acesso ao catálogo de dados científicos, uma vez que cada formato de arquivo ou estrutura de dados

em memória apresenta um conjunto de atributos e suas estruturas de dados características. Logo, o uso de metadados nessas análises facilitam a conversão dos dados científicos em estruturas de dados específicas a serem armazenadas na base de dados.

Assumindo-se que os dados científicos estejam acessíveis após as análises léxica e sintática, essa etapa também permite que apenas os atributos de interesse sejam extraídos para apoiar análises futuras. Para isso, uma condição de seleção (ou de filtragem) dos dados precisa ser definida para indicar quais valores de atributos (ou regiões de interesse) são importantes para os usuários em suas análises exploratórias, assim como uma lista de atributos deve ser informada para obter apenas os valores de atributos relevantes. Esse processo de seleção de determinados atributos e dos seus respectivos valores a partir de arquivos, respeitando-se as condições de seleção e de projeção, é conhecido pelo termo *extração de dados científicos*. O processo de extração de dados científicos é vantajoso, principalmente, em cenários de simulações computacionais em larga escala, em que, apesar do grande volume de dados produzidos, os usuários estão mais interessados em regiões específicas de interesse dos dados gerados.

Considerando a instanciação de cartuchos para esse componente da arquitetura ARMFUL, ela exige o desenvolvimento de três métodos para apoiar a extração de dados científicos, sendo conhecidos como *extract*() – do termo em português *extrair* (dados científicos de um arquivo) –, *run*() – do termo em português rodar/executar (um programa de extração de dados) – e *access*() – do termo em português acessar (dados científicos presentes em arquivos). O método *extract*() é responsável por identificar e extrair elementos de dados a partir da sequência de atributos escolhidos pelos usuários e que estão presentes em um determinado arquivo, conforme apresentado no Algoritmo 1. Dessa forma, esse método assume que todo o conteúdo especificado está presente no arquivo e obtém os valores de atributos de interesse por meio da invocação de um programa externo ou *script* de extração de dados científicos (argumento *program*).

O método *run*() é usado com o método *extract*() para extrair dados científicos de diversos arquivos que seguem o mesmo formato, *e.g.*, arquivos no formato HDF5. Assim, o método *run*() se caracteriza por invocar o método *extract*() a cada arquivo que precisa ter determinado conteúdo extraído dele. Pela sua generalidade de varrer todos os arquivos de dados científicos relevantes, esse método é genérico e não há a necessidade de implementá-

lo novamente ao adicionar cartuchos nesse componente da ARMFUL. Por último, o método *access*() consulta ou acessa dados científicos extraídos pelo método *extract*().

Algoritmo 1. Extração de dados científicos de um arquivo.

Entrada:

program: programa para extração de dados científicos

source: arquivo de dados científicos

atts: lista de atributos, cujos dados serão extraídos

Saída:

dataset: conjunto de dados de saída

```
1.  function extract(program, source, atts):
2.      dataset ← Dataset(atts)
3.      for each element in getElements(program, source) do:
4.          values ← { }
5.          position ← 0
6.          for each att in atts do:
7.              values[position] ← getAttributeValue(att, element)
8.              position++
9.          end do
10.         dataset.addElement(values)
11.     end do
12.     return dataset
13. end function
```

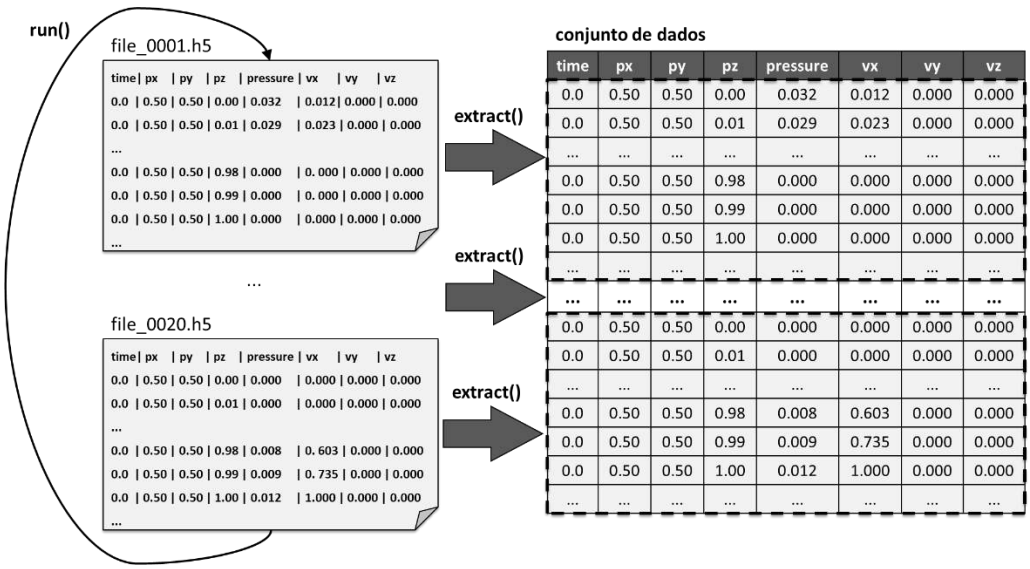


Figura 6. Exemplo de extração de dados científicos em arquivos no formato HDF5.

4.2.2. Indexação de dados científicos

Uma vez que os dados científicos tenham sido acessados, provendo informações sobre as estruturas de dados e os metadados para a geração de índices, o componente *Indexação de Dados Científicos* implementa técnicas de indexação dos dados científicos capturados com o propósito de reduzir o volume de dados carregados em um repositório externo ou uma base de dados, e de prover uma forma de acesso eficiente aos dados científicos. Mais especificamente, a geração de índices oferece vantagens relacionadas principalmente ao processamento de consultas, ao mesmo tempo em que introduz uma sobrecarga durante o acesso aos dados científicos em diferentes arquivos. Além disso, a indexação de dados científicos só deve ser aplicada em arquivos, pois os dados são armazenados de forma não-volátil, permitindo a geração de índices consistentes, ou seja, que podem ser utilizados posteriormente na consulta aos dados científicos.

Como o cenário em foco desta tese envolve arquivos com grande volume de dados, assumindo-se estruturas de dados que, em geral, não são vantajosas de serem armazenadas em um Sistema de Gerência de Banco de Dados – da sigla SGBD – (alto custo de ingestão dos dados), as técnicas existentes de indexação dos dados científicos (BLANAS *et al.*, 2014, KIM *et al.*, 2011, SILVA *et al.*, 2016c) evita exatamente essa sobrecarga na ingestão de dados. Nesse sentido, existem diferentes algoritmos para apoiar a indexação de dados científicos, como os índices *bitmap* e posicional apresentados na Seção 2.1.

Considerando os aspectos de desenvolvimento de cartuchos usando o componente Indexação de Dados Científicos da arquitetura ARMFUL, três métodos são definidos para permitir o acesso e a geração de índices para dados científicos presentes em arquivos, sendo conhecidos como *index*(), *run*() e *access*(). O método *index*() gera índices para todos os atributos escolhidos pelo usuário, sendo que esses atributos precisam estar presentes no arquivo de dados científicos. O Algoritmo 2 apresenta a implementação desse método, assumindo que o método *index*() invoca um algoritmo de indexação escolhido pelo usuário (argumento *algorithm*) e gera índices para os atributos de interesse (argumento *atts*).

Algoritmo 2. Geração de índices para acessar dados científicos em um arquivo.

Entrada:

algorithm: algoritmo de indexação

data$_{file}$: elementos de dados extraídos de um arquivo de dados científicos

atts: lista de atributos

Saída:

dataset: conjunto de dados de saída

```
1.  function index(algorithm, data_file, atts):
2.      dataset ← DataSet(atts)
3.      for each element in data_file do:
4.          values ← { }
5.          position ← 0
6.          for each attribute in atts do:
7.              values[position] ← generateIndex(algorithm, attribute, element)
8.              position++
9.          end do
10.         dataset.addElement(values)
11.     end do
12.     return dataset
13. end function
```

Ademais, semelhante ao comportamento dos cartuchos de Extração de Dados Científicos, o método *run*() invoca o método *index*() para gerar índices para todos os atributos em cada registro (ou elemento de dados) presente em diversos arquivos de dados científicos. Já o método *access*() obtém os dados científicos usando os índices gerados pelo método *index*(), conforme apresentado no Algoritmo 3. Portanto, diferentemente da extração de dados que tem acesso direto aos elementos de dados a partir do conjunto de dados (armazenado na base de dados), a técnica de indexação de dados exige que os índices gerados sejam obtidos, primeiramente, do conjunto de dados para serem utilizados no acesso aos elementos de dados presentes em arquivos de dados científicos.

Apesar das tecnologias permitirem a indexação de dados científicos e as suas máquinas de processamento de consultas usarem esses índices para melhorar o acesso aos dados, elas não são capazes de gerenciar dados científicos relacionados em múltiplos arquivos. Ou seja, as soluções existentes são focadas em analisar dados científicos em arquivos de uma forma isolada. Logo, aspectos associados ao fluxo de arquivos e ao fluxo de elementos de dados não são apoiados por essas tecnologias. Tendo em vista essa

limitação, a nossa abordagem representa esses dados científicos em uma base de dados integrada, ou seja, que também considera (i) dados de proveniência prospectiva (Seção 3.2.4) – tipo de dado que permite especificar os principais programas de simulação como transformações de dados e as dependências de dados entre as transformações –; e dados de proveniência retrospectiva (Seção 3.2.4) – tipo de dado obtido durante a execução de uma simulação computacional.

Algoritmo 3. Acesso aos dados científicos usando índices.

Entrada:

algorithm: algoritmo para acesso aos dados científicos usando índices
path: caminho para o arquivo de entrada
file$_{name}$: nome do arquivo de entrada
dataset: conjunto de dados, em que cada elemento contém os índices gerados
atts: attributos de interesse

Saída:

output: conjunto de dados de saída

```
1.  function access(algorithm, path, file_name, dataset, atts):
2.      output ← DataSet(atts)
3.      if exists file_name in path then:
4.          for each element in dataset do:
5.              values ← { }
6.              position ← 0
7.              for each attIndex in element.values do:
8.                  if dataset.attributes[position] in atts:
9.                      attValue ← getValueByIndex(algorithm, attIndex)
10.                     values ← values + {attValue}
11.                 end if
12.                 position++
13.             end do
14.             output.addElement(values)
15.         end do
16.     end if
17.     return output
18. end function
```

4.3. Camada de captura de dados de proveniência

Assim como os dados científicos precisam ser capturados de arquivos, os sistemas com apoio ao monitoramento de fluxos de dados, como os SGWfC pela abstração de

workflows científicos, também precisam gerenciar os dados de proveniência gerados, em tempo de execução, pelas simulações computacionais (camada *Captura de Dados de Proveniência*). Na nossa abordagem, temos como meta capturar os dados de proveniência em uma granularidade fina (componente *Captura de Dados de Proveniência*), a fim de que os três tipos de análises exploratórias em arquivos de dados científicos sejam apoiados, conforme apresentado no Capítulo 2. Ademais, após a captura dos dados de proveniência, essa camada requer que esses dados sejam validados por meio do componente *Validação de Dados de Proveniência* com o objetivo de garantir que o fluxo de dados modelado siga as definições da Subseção 3.1.1 e os dados de proveniência retrospectiva sejam consistentes com a especificação do fluxo de dados (*i.e.*, transformações de dados e suas dependências de dados) ou, em outras palavras, com os dados de proveniência prospectiva.

Outra preocupação da nossa proposta consiste na representação dos dados de proveniência e dos dados científicos capturados de arquivos. No que diz respeito a essa representação, nossa abordagem segue as recomendações de grupos e instituições para que o modelo de dados de proveniência seja padronizado, estruturado e interoperável. Nesta tese apresentamos uma extensão do diagrama de classes dos dados de proveniência para *workflows* científicos, conhecido como PROV-Wf, que foi desenvolvido em colaboração com o D.Sc. Flavio Costa (COSTA *et al.*, 2013). A adaptação desse diagrama teve o propósito de estender a representação de dados de proveniência em granularidade fina para descrever os dados científicos acessados, extraídos e indexados a partir de arquivos e de permitir análises direcionadas ao fluxo de dados nos níveis físico e lógico (Subseção 3.1.2). Portanto, para seguir a abstração de fluxo de dados apresentada na Subseção 3.1.1, a Seção 4.4 apresenta mais detalhes dessa adaptação do PROV-Wf, que foi nomeada como PROV-Df (SILVA *et al.*, 2016c), uma vez que o foco está nas noções de fluxo de dados e não de *workflows* científicos.

4.4. PROV-Df: diagrama de classes para modelar o fluxo de dados

Como apresentado anteriormente, a arquitetura ARMFUL permite o acesso, a indexação e o armazenamento de dados científicos por meio de cartuchos, que utilizam algoritmos desenvolvidos pelos usuários ou ferramentas alternativas. Entretanto, as análises exploratórias de dados científicos não se restringem ao acesso isolado dos dados científicos presentes em arquivos consumidos ou gerados por transformações de dados, conforme

discutido no Capítulo 2. Os usuários do domínio científico regularmente necessitam analisar o fluxo de arquivos ao longo da execução da simulação, assim como relacionar os elementos de dados presentes em diferentes arquivos.

Nessa perspectiva, o armazenamento dos dados é um aspecto importante na captura de dados de proveniência em granularidade fina. Ademais, iniciativas têm sido desenvolvidas para orientar a representação desses dados, como as recomendações do PROV (MOREAU e GROTH, 2013) pela *World Wide Web Consortium* (W3C). A proposta dessas iniciativas é permitir que os modelos de dados de proveniência sejam padronizados, estruturados e interoperáveis. Assim, uma vez que os estereótipos possam ser definidos de acordo com um metamodelo (*e.g.*, PROV-DM[7]), diferentes modelos de dados de proveniência podem ser relacionados de acordo com a nomenclatura do metamodelo.

Seguindo as recomendações do PROV W3C, um diagrama de classes, conhecido como PROV-Wf (COSTA *et al.*, 2013), já havia sido proposto e empregado pelo SGWfC SCC para representar os dados de proveniência gerados por *workflows* científicos, em particular, com execuções paralelas em larga escala. Entretanto, considerando-se as definições de fluxo de dados apresentadas na Seção 3.1, esta tese estendeu o PROV-Wf para apoiar tais conceitos (SILVA *et al.*, 2017b). Esse diagrama de classes para os dados de proveniência estendido foi nomeado como PROV-Df, conforme mostrado na Figura 7. Comparando-se o PROV-Wf com o PROV-Df, as suas contribuições estão exatamente no mapeamento dos conceitos de *workflow científico* para a abstração de fluxo de dados, além das extensões para permitir a representação de conjuntos de dados, com coleções e elementos de dados, assim como o processo de extração de dados científicos em arquivos.

O PROV-Df é composto de três partes: a estrutura do fluxo de dados (classes em branco do diagrama de classe), os dados de execução referentes a um fluxo de dados com os dados de domínio (classes em cinza escuro) e as configurações do ambiente computacional (classes em cinza claro). O estereótipo de cada classe corresponde a um componente do PROV-DM. As classes em branco descrevem a especificação do fluxo de dados (classe *Dataflow*), que é composto de transformações de dados (classe *DataTransformation*), conjuntos de dados (classe *DataSetSchema*), atributos (classe

[7] https://www.w3.org/TR/prov-dm/

Attribute), tipos de arquivos manipulados (classe *FileType*) e das extrações / indexações de dados realizadas (classe *Extractor* e *ExtractorCombination*). Por uma questão de simplificação, não foi apresentada a classe *Indexer* para expressar a indexação de dados científicos, já que ela apresenta os mesmos tipos de relacionamento que a classe *Extractor*. As dependências de dados entre as transformações de dados são representadas nesse modelo de dados pelo relacionamento *WasAttributedTo* entre as classes *DataTransformation* e *DataSetSchema*.

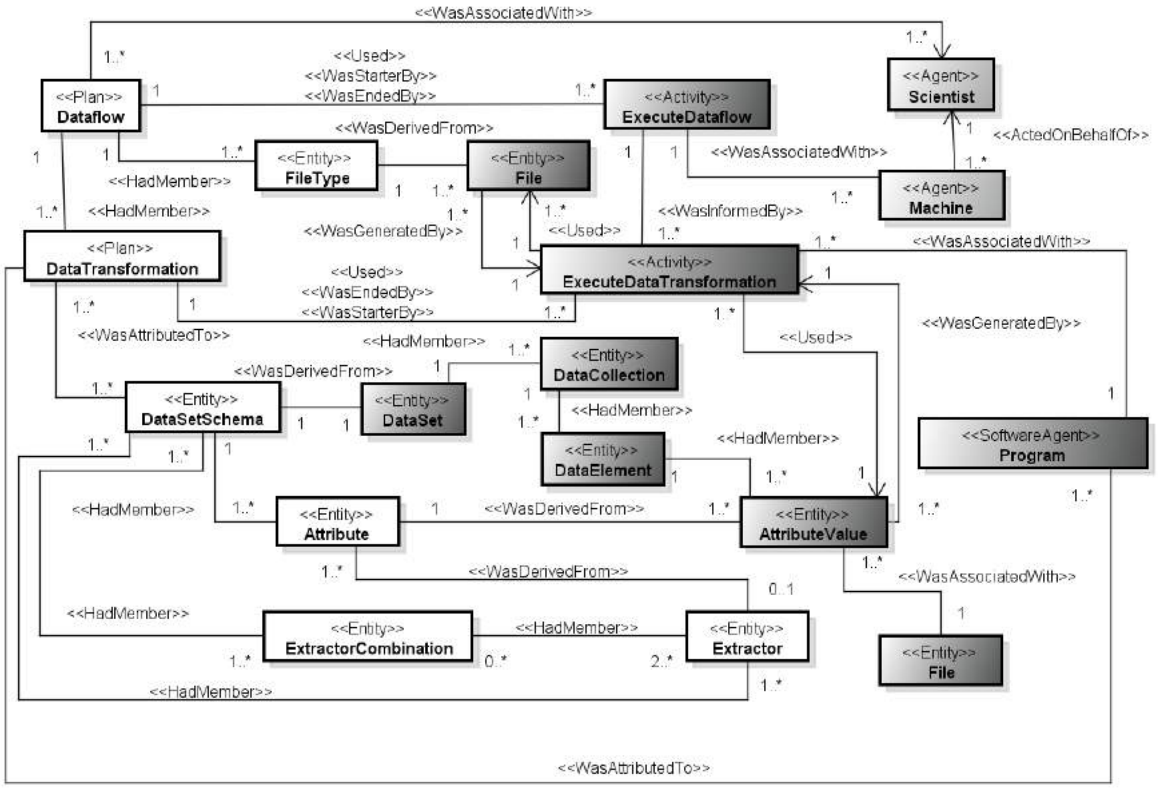


Figura 7. Diagrama de classe PROV-Df (SILVA *et al.*, 2016c).

As classes em cinza escuro descrevem os dados de execução referente a um fluxo de dados (atividade *ExecuteDataflow*), que considera as propriedades sobre a execução das transformações de dados (atividade *ExecuteDataTransformation*), os conjuntos de dados instanciados (classe *DataSet*), as coleções de dados (classe *DataCollection*) e os elementos de dados a partir de cada conjunto de dados (classe *DataElement*), os valores de atributos referentes a um determinado elemento de dados, ou seja, os dados científicos (classe *AttributeValue*), os programas invocados pelas transformações de dados (agente de software *Program*) e os arquivos consumidos e produzidos (classe *File*).

As classes em cinza claro descrevem informações sobre o ambiente computacional. Mais especificamente, o PROV-Df descreve os recursos computacionais utilizados para executar uma simulação computacional (agente *Machine*), assim como os usuários do domínio científico envolvidos na especificação e execução da simulação (agente *Scientist*). Em relação ao PROV-Wf, ajustou-se também alguns relacionamentos para apoiarem a gerência do fluxo de dados nos níveis físico e lógico. Como exemplo, temos os relacionamentos entre as classes *DataTransformation, DataSetSchema, Attribute, DataSet, DataElement* e *AttributeValue*.

Em termos práticos, o PROV-Df mapeia suas classes para tabelas na base de dados de proveniência. Para cada conjunto de dados consumido e produzido em um fluxo de dados, uma tabela física é gerada no SGBD escolhido com seus respectivos atributos e valores. Logo, as tabelas referentes aos conjuntos de dados permitem o armazenamento de dados específicos do domínio em tempo de execução. Além disso, a partir do diagrama PROV-Df, os usuários poderiam analisar o fluxo de dados por completo e informações sobre a sua execução (como o tempo de duração ou a existência de erros ao executar uma transformação de dados).

Ao mesmo tempo, uma base de proveniência que segue o diagrama PROV-Df permite o acesso aos arquivos de dados científicos, que podem coexistir com diversos outros arquivos gerados por transformações de dados em diferentes espaços de trabalho (*i.e.*, diretórios). As referências para os arquivos podem ser registradas em uma base por meio de ponteiros (*e.g.,* URI), a fim de utilizá-los na análise do fluxo de arquivos e dos elementos de dados presentes nesses arquivos. Dessa forma, os usuários são capazes de elaborar consultas analíticas com caráter investigativo (mais profundas) em sua base de proveniência, considerando tanto os dados específicos de domínio (*i.e.*, dados científicos presentes em arquivos), como informações do fluxo de dados, para observar os dados de diferentes arquivos manipulados ao longo da execução da simulação computacional.

4.5. Ingestão de dados científicos

Apesar das diferentes técnicas de indexação de dados científicos, existem circunstâncias em que a ingestão dos dados científicos (*i.e.*, carga dos valores assumidos pelos atributos) junto com os dados de proveniência em um determinado repositório (*e.g.*, base de dados) seja o mais recomendado por questões de desempenho. Como exemplo,

pode-se considerar a ingestão dos dados científicos como mais vantajosa, se a indexação de dados científicos com estruturas de dados padrões (*e.g.*, cadeia de caracteres e inteiros) ocupar mais espaço de armazenamento que a ingestão do próprio dado científico e o tempo de consulta usando índices for maior que o custo de consultar diretamente os dados científicos na base de dados de proveniência integrada. Em função desse cenário, a nossa abordagem permite que os usuários definam se pretendem indexar os dados científicos ou se preferem carregar os elementos de dados diretamente na base de dados. Além de permitir que o usuário defina se realizará a indexação ou a ingestão dos dados científicos, a nossa abordagem é capaz de escolher a técnica de indexação de dados científicos a ser utilizada e, consequentemente, modificar a proposta de armazenamento na base de dados.

No caso da ingestão de dados científicos, os índices são gerados diretamente pelo SGBD e, depois, eles são usados para acessar os dados científicos dos arquivos armazenados na base de dados. Por conseguinte, a nossa abordagem representa os valores dos atributos no esquema da base de dados de proveniência de acordo com a natureza dos dados científicos: estruturas de dados dos valores de atributos carregados de arquivos ou estruturas de dados necessárias para representar os índices gerados a partir de dados científicos presentes em arquivos. Vale ressaltar que o resultado desse processo com extração ou indexação dos dados científicos, portanto, é uma base de dados de proveniência com os registros de fluxo de dados de diferentes simulações computacionais, sendo que o esquema dessa base de dados segue a especificação do PROV-Df.

4.6. Processamento de consultas aos dados científicos

Uma vez que os dados científicos tenham sido capturados e armazenados em um SGBD, os usuários podem consultá-los para validar ou refutar as suas hipóteses científicas. Para que isso seja possível, alguns mecanismos precisam ser adicionados como uma etapa de pré-processamento das consultas para permitir as análises focadas nos dados científicos, quando técnicas de indexação com ferramentas alternativas são aplicadas. Semelhante à primeira etapa existente em processadores de consultas, o pré-processamento também deve contemplar as análises léxica e sintática. No caso de consultas para permitir o acesso aos dados científicos, esse componente precisa ser capaz de considerar os seguintes aspectos, assumindo-se que os dados científicos foram capturados por técnicas de indexação:

- As estruturas dos dados científicos não correspondem necessariamente às estruturas de dados adotadas pelos atributos armazenados na base de proveniência, pois os valores armazenados correspondem aos índices gerados ao acessar o conteúdo dos arquivos de dados científicos; e
- A referência para o arquivo utilizado na indexação de determinados atributos normalmente não está presente na especificação desses atributos na base de dados. Logo, a base de dados precisa ser capaz de representar essas referências aos arquivos e relacionar esses arquivos aos atributos capturados (*i.e.*, que apresentam os índices para os dados científicos).

A partir dessas análises, esse mecanismo de pré-processamento é capaz de identificar os índices gerados em cada conjunto de dados, assim como de subdividir a consulta de acordo com a natureza dos dados. Nesse caso, os dados de proveniência, os dados de execução e os dados científicos extraídos diretamente de arquivos de dados podem ser consultados pela subconsulta que é executada pelo SGBD, em que temos a base de dados da ARMFUL seguindo as especificações do PROV-Df. Já as subconsultas, que consideram o acesso aos dados científicos por meio dos índices gerados, serão executadas pelos mecanismos de processamento de consultas das ferramentas alternativas utilizadas, como a ferramenta FastBit. Ao final de cada um desses comportamentos, os resultados obtidos precisam ser unidos de acordo com a especificação da consulta submetida. Caso haja qualquer tipo de erro ao processar a consulta, uma mensagem de erro deve ser gerada por esse mecanismo em tempo de execução.

4.7. SCC-A: uma instância da ARMFUL usando o SGWfC SCC

Como primeira instância da ARMFUL, o SGWfC SCC foi estendido para permitir a captura e a análise de dados científicos, sendo essa implementação conhecida como SCC-A[8]. Como o SCC já apresentava um recurso de extração de dados, sem considerar o acesso ao conteúdo presente em arquivos de dados científicos, o seu modelo arquitetural (Subseção 4.7.1) foi preservado. Ao mesmo tempo, como a álgebra de *workflows* científicos empregada na especificação das atividades usando o SCC não continha operadores algébricos para representar a extração e a indexação de dados científicos, o SCC-A também

[8] https://hpcdb.github.io/armful/a-chiron.html

exigiu a extensão dessa álgebra. Mais especificamente, dois novos operadores algébricos foram propostos, conhecidos como Raw e RawI (Subseção 4.7.2), que permitem extrair e indexar dados científicos de arquivos, respectivamente. Vale ressaltar também que os resultados dessa seção são apresentados por Silva *et al.* (2017).

4.7.1. Modelo arquitetural do SGWfC SCC-A

O SGWfC paralelo SCC (apresentado em mais detalhes na Seção 3.3.4) é composto das seguintes camadas, conforme mostrado na Figura 8: Gerência de *Workflow*, Escalonamento de *Workflow*, Processamento de Ativação, Invocação de Programa, Invocação de Extrator e Processamento de Dados de Proveniência. Primeiramente, a camada Gerência de *Workflow* é responsável por controlar a execução paralela dos *workflows* científicos em ambientes de *cluster* e nuvem computacional, determinando o momento adequado de executar cada atividade. Esse momento é definido segundo as dependências de dados e a estratégia de execução paralela adotada pelo SCC, que pode variar em duas estratégias de fluxo de dados (FTF e FAF) e duas estratégias de despacho (estática e dinâmica). Tais estratégias são detalhadas em Ogasawara *et al.* (2011). Abaixo dessa camada, o SCC apresenta a camada Escalonamento de *Workflow*, que realiza a subdivisão das atividades em unidades de dados autocontidas e a distribuição dessas unidades para os recursos computacionais disponíveis, a fim de favorecer o paralelismo de dados. Cada uma dessas unidades de dados autocontidas é conhecida como ativação, semelhante ao conceito de tarefa (RAICU *et al.*, 2008) em outros SGWfC paralelos. Dessa forma, uma ativação contempla o menor conjunto de dados necessário para executar o programa de simulação de uma determinada atividade.

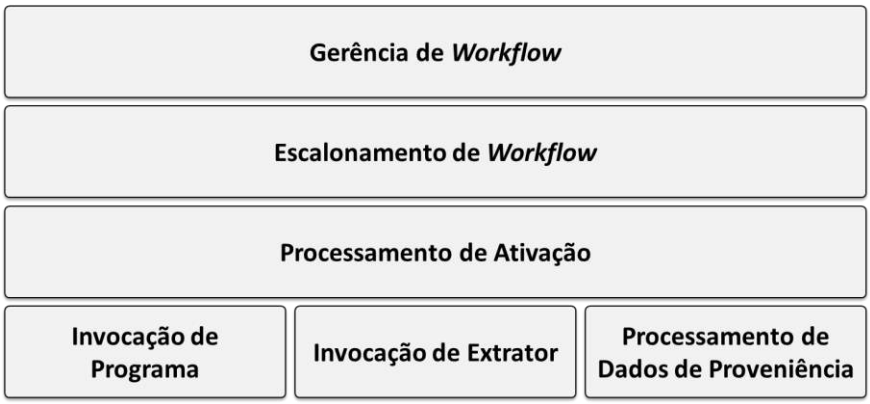


Figura 8. Representação em camadas do SCC.

Já a camada Processamento de Ativação lida com a identificação dos comportamentos necessários para permitir a execução de uma ativação: a invocação dos programas de simulação, a extração de dados científicos e a captura dos dados de proveniência. Nesse sentido, a camada Invocação de Programa é responsável pela instrumentação e execução propriamente dita dos programas de simulação envolvidos em uma ativação, segundo a especificação de uma atividade. Ademais, a camada Invocação de Extrator tem o objetivo de permitir a carga de dados gerados durante a execução de *workflows* científicos em uma base de dados de proveniência relacional. Por último, a camada Processamento de Dados de Proveniência apoia o mecanismo de processamento de consultas do SCC, garantindo a captura de dados de proveniência em tempo de execução. Nesse caso, o SCC implementa o processamento de consultas por meio de uma conexão com um Sistema de Gerência de Banco de Dados (SGBD) relacional, mais especificamente, o PostgreSQL[9].

Apesar dessas camadas, a álgebra relacional de *workflows* científicos, proposta inicialmente e implementada no Chiron (OGASAWARA *et al.*, 2011), não é suficiente para realizar a captura de dados científicos em arquivos por meio da camada Invocação de Extrator. A etapa original de extração de dados do SCC consiste apenas na captura de dados por meio da invocação de programas externos que acessam dados produzidos (normalmente em arquivos de *log*) e os representam como relações de saída das atividades do *workflow*. Após a execução desses programas, esse SGWfC paralelo exige a escrita dos dados capturados em um arquivo no formato tabular, conhecido como *ERelation.txt*.

Diferentemente dessa proposta original do SCC, defendemos que o processo de extração de dados científicos deve acessar e capturar os dados científicos em arquivos com dados em sua forma bruta, assim como representar esses dados extraídos também no arquivo *ERelation.txt*. Mais especificamente, esse processo é composto de três etapas: acesso aos dados científicos de arquivos com dados em sua forma bruta, indexação dos dados extraídos e a carga dos dados científicos em um repositório externo ou uma base de dados. No cenário de captura de dados científicos, o acesso e a indexação dos dados científicos requerem um conhecimento prévio das estruturas de dados presentes nos

[9] https://www.postgresql.org/

formatos de arquivo de interesse. Além disso, em casos de arquivos com grande volume de dados a serem extraídos, o SCC pode empregar técnicas de indexação para evitar a carga de muitos dados científicos ou de estruturas de dados complexas e densas em repositórios externos ou em bases de dados. Ao mesmo tempo, o SCC deve fornecer funcionalidades que permitam os usuários selecionarem os atributos de dados científicos relevantes para as suas análises futuras.

Diante dessas questões, esta tese propõe uma extensão do SGWfC SCC, conhecida como SCC-A, que contemple dois mecanismos possíveis na etapa de extração de dados: carga e indexação de dados científicos. A carga de dados consiste no mesmo comportamento existente nesse SGWfC paralelo, que é caracterizada pela invocação de um programa externo, mas com o acesso a dados presentes em arquivos de dados científicos. Dessa forma, os valores dos atributos de interesse (*i.e.*, dados científicos) seriam carregados diretamente no SGBD relacional utilizado pelo SCC-A. Enquanto isso, a indexação de dados científicos lida com o acesso aos dados científicos presentes em arquivos em formatos específicos do domínio, a geração de índices (*e.g.*, *bitmap* e posicional) para permitir o acesso eficiente a esses dados posteriormente e a carga dos índices na base de proveniência do SCC-A. Logo, ao invés de armazenar os valores dos atributos na base de proveniência, o SCC-A carrega os índices referentes aos dados científicos. Além disso, essa base de proveniência gerencia quais atributos tiveram seus valores indexados, a fim de apoiar o acesso aos dados científicos presentes nos arquivos pelo uso dos índices gerados.

Ao acoplar a arquitetura ARMFUL com o SCC (Seção 4.7), esta tese tirou proveito das relações utilizadas pelo SCC para representar os dados consumidos e produzidos pelas transformações de dados em uma base de proveniência, com o objetivo de monitorar, depurar e analisar o fluxo de dados em simulações computacionais. Essa abordagem relacional também favorece consultas aos dados científicos de uma forma estruturada (*i.e.*, uma relação contém todos os dados científicos referentes a um conjunto de dados para uma transformação de dados específica), o que facilita as análises desenvolvidas pelos usuários. Além disso, o SCC-A utiliza as expressões algébricas para direcionar a execução paralela do *workflow* científico, armazenando os dados na base de proveniência. Nesse aspecto, o SCC-A armazena dados de proveniência prospectiva e retrospectiva, que podem ser representados, respectivamente, pelas especificações das expressões algébricas realizadas

em cada transformação de dados (e as suas dependências de dados) e os metadados relacionados à execução do *workflow* científico (COSTA *et al.*, 2013).

Dado que esta tese considera a gerência do fluxo de arquivos e dos elementos de dados, estendeu-se a base de proveniência do SCC para armazenar metadados relacionados aos dados específicos do domínio; arquivos consumidos e produzidos; e elementos de dados extraídos a partir de arquivos de dados científicos (*i.e.*, formatos de arquivos específicos do domínio). Seguindo as recomendações do PROV W3C, um diagrama de classes, conhecido como PROV-Wf (COSTA *et al.*, 2013), já havia sido proposto e empregado no SCC para representar os dados de proveniência para *workflows* científicos, em particular, com execuções paralelas em larga escala. Assim, considerando-se as definições de fluxo de dados (Seção 3.1), esta tese estendeu o PROV-Wf para apoiar tais conceitos (SILVA *et al.*, 2017b). Esse modelo de dados de proveniência estendido é conhecido como PROV-Df, conforme discutido na Seção 4.4.

Já no que diz respeito à análise dos dados científicos nessa base integrada de dados de proveniência, o processador de consultas terá a responsabilidade de utilizar os índices gerados e as referências para os arquivos para obter os valores assumidos pelos atributos de dados científicos. Do mesmo modo, o modelo de dados de proveniência deve permitir que os metadados referentes ao tipo de índice empregado e ao formato do arquivo com dados científicos estejam acessíveis, para que o processador de consulta possa obtê-los ao gerar o plano de execução da consulta. Em circunstâncias em que as análises necessitam relacionar dados científicos de diferentes arquivos, o modelo de dados também precisa representar os fluxos de dados nos níveis físico e lógico gerados pela simulação computacional.

Considerando a representação em camadas do SCC, pode-se observar que a contribuição desta tese se concentra nas camadas Invocação de Extrator e Processamento de Dados de Proveniência. Mais especificamente, a abordagem proposta tem o objetivo de permitir o acesso e a indexação de dados científicos para apoiar análises específicas em dados científicos, enquanto que mantém a carga de dados científicos desenvolvida originalmente no SCC. Por isso, o componente de Captura de Dados Científicos da ARMFUL foi instanciado como uma arquitetura baseada em componentes, conhecida como *Raw Data Capture* (*RDC*) (em português, Captura de Dados Científicos). A arquitetura RDC é dividida em cartuchos de dois tipos, como mostrado na Figura 9, que realizam a

indexação de dados científicos (Indexador de Dados Científicos do termo, em inglês, *Raw Data Indexer* – RDI) e a extração de dados científicos (Invocador de Extrator do termo, em inglês, *Extractor Invoker* – EI). Tais componentes são extensíveis, de forma que diferentes cartuchos podem ser desenvolvidos para extrair ou indexar dados científicos no SCC-A (BIRSAN, 2005).

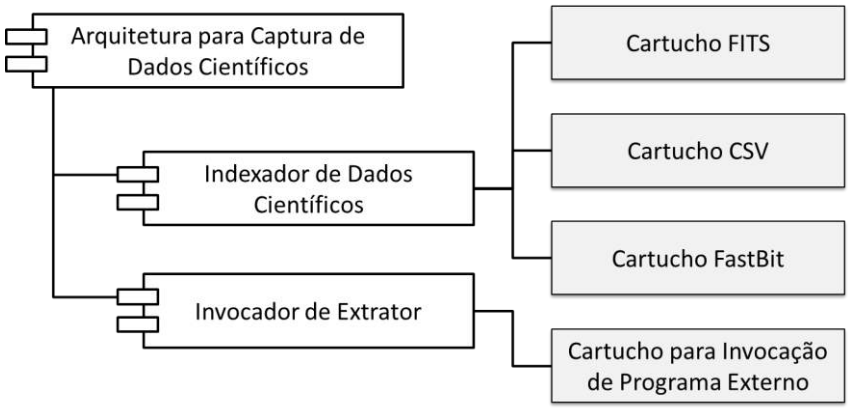


Figura 9. Cartuchos implementados na arquitetura RDC.

O componente RDI realiza a indexação dos dados científicos por meio do acesso aos conteúdos presentes em arquivos. Nesta implementação do SCC-A, utilizou-se duas técnicas de indexação, que foram baseadas no mapeamento posicional proposto por Karpathiotakis *et al.* (KARPATHIOTAKIS *et al.*, 2014), nos cartuchos para os formatos de arquivo FITS e CSV, e em mapas de *bits* usando a ferramenta FastBit. No caso do uso da ferramenta FastBit, esse cartucho foi resultado de um trabalho de pesquisa em colaboração com o M.Sc. José Vitor Leite (SILVA *et al.*, 2017b), em que se utilizou essa ferramenta alternativa para gerar índices do tipo mapa de *bits* para os dados científicos de interesse. De forma semelhante, outras ferramentas alternativas podem ser utilizadas pela arquitetura RDC para indexar os dados científicos, como o NoDB, o Slalom e o SDS.

Diferentemente do RDI, o componente EI permite a invocação de programas de extração de dados científicos, conhecidos como extratores, para a carga dos valores dos atributos de interesse diretamente na base de dados do SCC-A. Consequentemente, o Cartucho para Invocação de Programa Externo tem o objetivo de invocar programas externos para extrair dados científicos de arquivos produzidos pelos programas de simulação. Ao contrário, o componente RDI propõe o uso de técnicas de indexação para reduzir a carga de dados na base de proveniência e, consequentemente, o tempo de execução

da simulação computacional. Somando-se a isso, o RDI apresenta como desvantagem uma sobrecarga na etapa de consulta aos dados científicos, devido ao custo de recuperação dos índices gerados e armazenados na base de dados de proveniência para depois utilizá-los no acesso aos dados científicos armazenados nos arquivos.

4.7.2. Raw e RawI: operadores algébricos estendidos na SciWfA para extrair e indexar dados científicos

Apesar da abordagem algébrica do SCC apresentar três procedimentos para descrever a execução de uma ativação (*i.e.*, instrumentação, execução e extração), a álgebra relacional para *workflows* científicos, conhecida como SciWfA (OGASAWARA *et al.*, 2011), não permite descrever a extração e a indexação de dados científicos em uma transformação dos dados. Nesse sentido, essa álgebra foi estendida nesta tese, adicionando-se dois operadores algébricos para permitir a extração e a indexação de dados científicos, sendo esses novos operadores algébricos conhecidos como *Raw* e *RawI*, respectivamente. Semelhante aos operadores originais do SCC, os operadores Raw e RawI devem apresentar o tipo da atividade operada, os operandos adicionais e a relação de entrada. Como operandos adicionais, esses operadores necessitam de duas informações: o nome do atributo da relação de entrada que apresenta como valores as referências para os arquivos a terem seu conteúdo extraído e os nomes dos atributos que precisam ter seus valores capturados desses arquivos.

Ademais, o tempo de processamento computacional pode variar de acordo com o operador algébrico. Para o operador Raw, existe apenas um comportamento possível: a invocação de programas externos para extrair dados científicos de arquivos. Logo, a especificação da transformação de dados deve contemplar a linha de comando para invocar os programas externos. Já o operador RawI permite dois comportamentos possíveis: a invocação de ferramentas alternativas, como o FastBit; ou o uso de algoritmos desenvolvidos em determinados cartuchos da arquitetura RDC para indexar dados científicos acessados em arquivos. Consequentemente, a invocação de ferramentas alternativas exige a especificação da linha de comando para executar essas ferramentas, ao especificar a transformação de dados, enquanto que o uso de cartuchos com algoritmos próprios requer apenas a identificação do cartucho a ser utilizado. No contexto desta tese, desenvolveu-se dois cartuchos com algoritmos próprios na arquitetura RDC, mas inspirados

em soluções existentes, como o Cartucho FITS[10] e o Cartucho CSV baseado na geração de índices posicionais, conforme a proposta do NoDB (ALAGIANNIS *et al.*, 2015). Vale ressaltar que, caso seja de interesse do usuário, as invocações para as ferramentas alternativas podem ser acopladas como um cartucho com algoritmo próprio, sem que o usuário tenha que ter conhecimento de como deve invocar essas ferramentas para indexar os dados científicos.

Além disso, a relação do número de tuplas consumidas e produzidas varia consideravelmente nesses operadores algébricos, pois depende da quantidade de arquivos analisados e do próprio volume de dados presente nesses arquivos. Portanto, essa relação pode ser representada por *1:n*, a fim de que a operação desempenhada pelo operador Raw ou RawI apresente o comportamento mais genérico possível (*i.e.*, expressando que cada arquivo de entrada pode extrair um ou mais elementos de dados científicos). Nessa circunstância, assume-se que cada invocação desses operadores é responsável por acessar e, eventualmente, indexar dados científicos presentes em apenas um arquivo. Diante dessas informações, os operadores algébricos Raw e RawI podem ser especificados formalmente, conforme apresentado abaixo:

$$Relação_{saída} \leftarrow Raw\left(\begin{matrix} Programa_{externo}, \\ \{\{Atributo_{arquivos}\}, \{Atributos_{acessados}\}\}, \\ Relação_{entrada} \end{matrix}\right) \quad (2)$$

$$Relação_{saída} \leftarrow RawI\left(\begin{matrix} Ferramenta_{alternativa}\ ou\ Cartucho_{indexação}, \\ \{\{Atributo_{arquivos}\}, \{Atributos_{acessados}\}\}, \\ Relação_{entrada} \end{matrix}\right) \quad (3)$$

Diferentemente da álgebra original de *workflows* científicos, *i.e.*, SciWfA, essa extensão da álgebra também considerou que os operadores Raw e RawI só podem ser especificados em uma certa atividade do *workflow*, quando um dos outros operadores algébricos (Map, SplitMap, Reduce, Filter e Query) também for definido anteriormente nessa mesma atividade. Essa especificação caracteriza a extração e a indexação de dados como um pós-processamento de dados para uma determinada atividade, a fim de representar os dados produzidos em uma relação de saída.

[10] https://wiki.duraspace.org/display/ISLANDORA713/FITS+Extractor

A Figura 10(a) mostra um exemplo de atividade (denominada de *A*) sem extração de dados, que é composta apenas pelo operador algébrico Map. Enquanto isso, a Figura 10 (b) e a Figura 10(c) mostram atividades, denominadas de A_e e A_i, sendo regidas pelos operadores algébricos Raw e RawI para permitir a extração e a indexação de dados científicos, respectivamente. A extração de dados desse exemplo considera a invocação de um programa externo, conhecido como *extractData*, enquanto que a indexação de dados é baseada na invocação da ferramenta alternativa FastBit. Os arquivos de dados científicos seguem o formato CSV, sendo especificados como operandos adicionais, enquanto os valores dos atributos a_1, a_2 e a_3 são capturados dos arquivos CSV.

Além das contribuições associadas ao formalismo das transformações de dados ao longo da simulação computacional, a extensão da álgebra de *workflows* científicos apresenta vantagens relacionadas às análises exploratórias de dados científicos. Dado que os operadores algébricos Raw e RawI permitem a captura de dados científicos presentes em arquivos, análises, que levam em consideração o conteúdo específico do domínio, podem ser tratadas a partir do processamento de consultas à base de dados de proveniência. Ao mesmo tempo, estruturas de dados que antes apresentavam um alto custo para o seu armazenamento na base de dados de proveniência por meio da extração de dados, agora podem contar com a extensão da álgebra para aplicar técnicas de indexação. Dessa forma, o uso do operador RawI permite o armazenamento dos índices referentes ao conteúdo desses arquivos na base de proveniência, assumindo um espaço reduzido em comparação com a carga dos valores dos atributos.

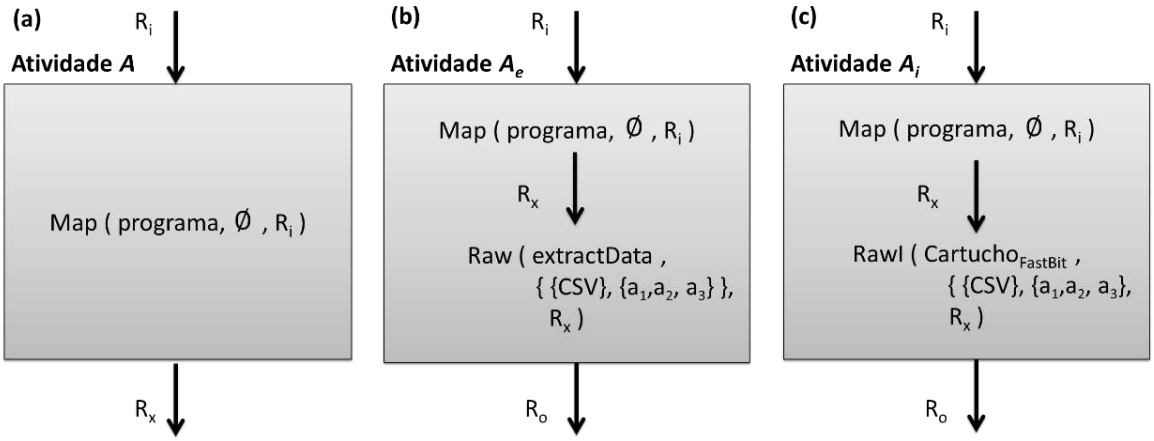


Figura 10. Atividades usando a SciWfA estendida: (a) sem extração/indexação, (b) com extração e (c) com indexação de dados científicos.

Por exemplo, ao invés de armazenar os valores dos elementos de uma matriz, o operador algébrico pode ser responsável por indexar a posição de início da matriz, o número de elementos e o tamanho de cada elemento de dados. Ao ser necessário utilizar o valor propriamente dito do atributo por uma consulta ou mesmo pela própria máquina de execução paralela, o acesso a esse valor de atributo pode ser obtido por meio dos índices gerados e armazenados na base de dados de proveniência. Como protótipo de índices posicionais desenvolvidos nesta tese, considerou-se as estruturas de índices para os seguintes tipos de dados (Tabela 3): numérico, textual, arquivo, lista, matriz e malha.

Estrutura de dados	**Formato do índice**
Numérico	[posição_início, tamanho]
Textual	[posição_início, tamanho]
Arquivo	[posição_início, tamanho]
Lista	[posição_início, tamanho, elementos, delimitador]
Matriz	[posição_início, tamanho, linhas, colunas, delimitador]
Malha	[posição_início, tamanho, xdepth, ydepth, zdepth, delimitador]

Tabela 3. Estruturas de dados apoiadas pelo operador RawI.

Destaca-se também que cada estrutura de dados apresenta sua própria construção de índices. No caso de valores numéricos, textuais e arquivos, os índices apresentam a posição de início e o número de posições ocupadas por ele, ou seja, o seu tamanho. Já as estruturas de dados para listas, matrizes e malhas, além de possuírem a especificação da posição inicial e do tamanho de cada elemento, elas apresentam um delimitador e o número de elementos em cada uma de suas dimensões de representação. Ou seja, para uma matriz, duas dimensões são definidas para expressar o número de elementos (*e.g.*, linhas e colunas). Vale ressaltar que assumimos tamanhos fixos para a representação de conteúdos que seguem as estruturas de dados de lista, matriz e malha. Portanto, o segundo elemento (tamanho) no formato do índice não deveria estar presente na representação dessas estruturas de dados.

Existem circunstâncias também em que após o processamento de um modelo computacional, os usuários necessitam extrair dados de diferentes formatos de arquivos e que, para isso, eles devem utilizar um algoritmo de extração ou de indexação de dados científicos para cada formato de arquivo. Assim, o pós-processamento pode ser

caracterizado pela execução de extratores e/ou indexadores de forma paralela, conforme mostrado na Figura 11. Entretanto, a definição de uma atividade pela álgebra de *workflows* científicos corresponde a uma transformação de dados, que consome uma relação de entrada e produz uma relação de saída (OGASAWARA *et al.*, 2011). Logo, a execução paralela dos extratores produz várias relações de saída, o que contradiz a definição de uma atividade. Ao mesmo tempo, do ponto de vista de uma atividade, os resultados de extração apresentam um significado semântico para o modelo computacional executado pelo operador algébrico antes dos operadores Raw e RawI. Logo, tais operadores não podem ser separados conceitualmente em uma outra atividade.

Somando-se a isso, os dados científicos obtidos nos operadores Raw e RawI normalmente estão relacionados. Logo, considerou-se a definição de outro operador algébrico na SciWfA, conhecido como *Join*, com o objetivo de associar os elementos de dados (*i.e.*, dados científicos) obtidos por mais de um operador dos tipos Raw ou RawI. Logo, esse operador permite que apenas uma relação de saída seja gerada ao término de uma atividade. A Figura 11 mostra um exemplo de atividade com dois operadores de extração/indexação de dados e o uso do operador Join. Para a especificação do operador Join, deve-se informar o algoritmo de junção a ser aplicado (*e.g.*, equi-junção), os atributos de junção (representados como operandos adicionais, *e.g.*, a_1) e as relações de entrada (*i.e.*, relações de saída após a execução dos operadores Raw e RawI). Formalmente, o operador Join pode ser definido da seguinte forma:

$$Relação_{saída} \leftarrow Join\left(Algoritmo_{junção}, \{\ Atributos\ \}, Relações_{entrada}\right) \qquad (4)$$

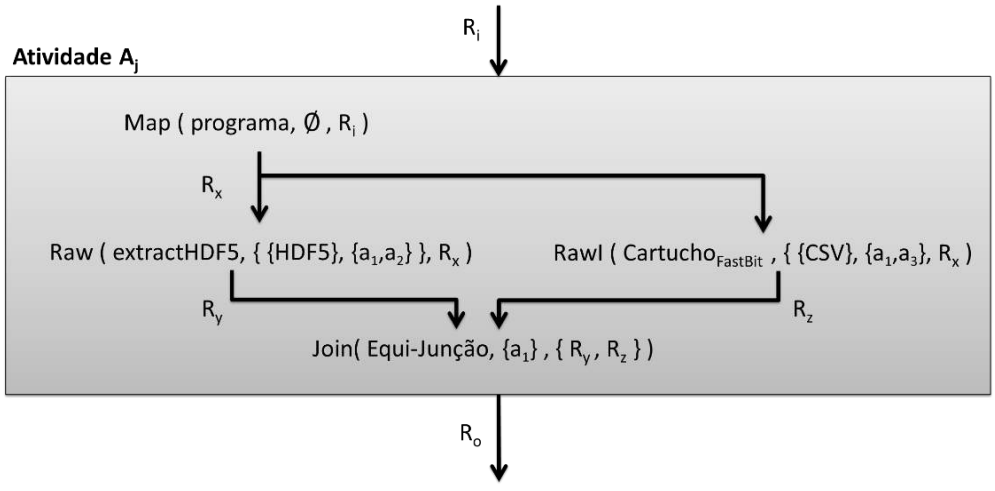


Figura 11. Exemplo de uso do operador algébrico Join.

Capítulo 5 – DfAnalyzer: componentes para a análise de dados em simulacões computacionais

Este capítulo apresenta a biblioteca de componentes DfAnalyzer (SILVA *et al.*, 2018), a segunda instância da ARMFUL desenvolvida nesta tese, que permite o monitoramento, a depuração e a análise do fluxo de dados durante a execução de simulações computacionais. Para isso, os usuários do domínio científico podem trabalhar em parceria com o especialista em ciência da computação para modelar as suas simulações como um fluxo de dados. Em muitos casos, inclusive, os usuários do domínio já utilizam ferramentas de análise e visualização de dados científicos com recursos de programação ou de integração com os seus programas de simulação parecidos com a abordagem adotada pela DfAnalyzer. Assim, eles frequentemente apresentam autonomia para utilizar diretamente a DfAnalyzer, sem o auxílio de um especialista em ciência da computação.

Diante dessa realidade, introduzimos uma metodologia orientada ao fluxo de dados que visa capturar e analisar os dados de proveniência e os dados científicos presentes em diferentes arquivos ou ainda alocados em memória, que seguem formatos ou estruturas de dados padronizadas de acordo com o domínio da simulação. Assumimos o cenário em que os usuários do domínio científico requerem auxílio dos especialistas em ciência da computação. Mais especificamente, a DfAnalyzer apresenta um conjunto de componentes para apoiar a captura de dados de proveniência, a captura de dados científicos, o processamento de consultas e a visualização das especificações de fluxo de dados. Este capítulo é subdividido em quatro seções, que apresentam uma visão geral dessa biblioteca de componentes (Seção 5.1); a nossa metodologia orientada ao fluxo de dados em simulações computacionais (Seção 5.2); as operações da DfAnalyzer para a captura de dados de proveniência e de dados científicos (Seção 5.3); e os componentes da sua arquitetura (Seção 5.4).

5.1. Visão Geral

Como mencionado anteriormente, a biblioteca de componentes DfAnalyzer[11] permite o monitoramento, a depuração e a análise dos fluxos de dados gerados pelas

[11] https://hpcdb.github.io/armful/dfanalyzer.html

simulações computacionais em tempo de execução. Para isso, a DfAnalyzer realiza a captura de dados de proveniência e de dados científicos presentes em arquivos, assim como o processamento de consultas de forma *online* – ou seja, durante a execução da simulação – para investigar o fluxo de dados nos níveis físico e lógico (*i.e.*, fluxo de arquivos e de elementos de dados, respectivamente), conforme discutido no Capítulo 2.

Diferentemente do SCC-A (Seção 4.7), a DfAnalyzer consiste em uma instância da ARMFUL independente de um SGWfC. Logo, ao invés de ter um SGWfC para gerenciar a execução paralela dos programas de simulação, esses programas já apresentam recursos para apoiar o paralelismo de dados ao executar seus modelos computacionais em diversos nós computacionais (*e.g.*, em um ambiente de *cluster* computacional). Como exemplos de tecnologias que são comumente integradas a esses programas de simulação, pode-se citar o Open MPI[12], o OpenMP[13] ou mesmo outra biblioteca capaz de executar os métodos computacionais de forma paralela, como a libMesh[14], sendo específica para métodos de elementos finitos.

Dessa forma, a DfAnalyzer assume que os programas de simulação são caixas cinzas (do termo em inglês, *gray boxes*), em que um programa caixa cinza apresenta parte do seu código fonte passível de ser instrumentado ou acessado, enquanto que a outra parte envolve um código fonte privado (caixa preta), que pode, por exemplo, realizar invocações a outros programas ou bibliotecas. Mais especificamente, esses programas são comumente baseados em *scripts*, pois eles apenas apresentam invocações para bibliotecas externas, como MPI, libMesh e ParaView Catalyst (para visualização de dados *in situ*). Assim, a DfAnalyzer está na categoria de sistemas baseados na invocação de serviços ou bibliotecas, em que as partes do código fonte acessíveis nesses programas de simulação podem ser ajustadas a fim de invocar os serviços disponibilizados pela DfAnalyzer.

Ademais, essa biblioteca de componentes defende uma solução agnóstica à linguagem de programação, ou seja, os usuários podem modificar os seus programas de simulação independentemente da linguagem de programação. Para isso, essa biblioteca de

[12] https://www.open-mpi.org

[13] http://www.openmp.org

[14] https://libmesh.github.io

componentes conta com um conjunto de serviços RESTful para a captura de dados de proveniência e científicos, e com programas binários ou ferramentas alternativas que os usuários podem invocar a partir de seus códigos fontes para realizar a extração e, eventualmente, a indexação de dados científicos presentes em arquivos. Essa proposta difere das soluções existentes para a análise de dados de proveniência em *scripts* e programas, que tendem a seguir uma linguagem de programação específica e serem sequenciais, como o noWorkflow (MURTA *et al.*, 2014), específico para *scripts* sequenciais na linguagem de programação Python.

Para apoiar a captura de dados científicos comumente presentes em arquivos, a DfAnalyzer apresenta uma operação, denominada *extract*, que oferece o acesso, a extração e, eventualmente, a indexação dos dados científicos. Como primeiro passo, portanto, essa operação acessa os conteúdos presentes em cada arquivo, sendo que apenas os valores dos atributos de interesse são selecionados e extraídos. De acordo com uma sequência de atributos predefinidos, a extração de dados científicos representa uma sequência de valores para esses atributos como um elemento de dados, segundo a abstração de fluxo de dados (Seção 3.1). Já na indexação de dados científicos, ao invés de extrair diretamente os valores dos atributos, técnicas de indexação podem ser aplicadas para criar índices, a fim de permitir um acesso eficiente aos dados científicos. Assim, a indexação de dados tende a ser mais vantajosa em estruturas de dados mais avançadas, como malhas, ou quando o volume de dados acessados é muito grande (*i.e.*, grande quantidade de elementos de dados que poderiam ser referenciados apenas por um arquivo de índices).

Além disso, a DfAnalyzer foi integrada ao ParaView Catalyst, uma ferramenta para a análise e a visualização de dados *in situ*. Dessa forma, além de permitir a análise de dados científicos em arquivos, a nossa ferramenta é capaz de tirar proveito de dados alocados em memória, reduzindo o custo de operações em disco e, consequentemente, o tempo de execução da simulação computacional, conforme apresentado nas avaliações experimentais desta tese. Outra contribuição é a geração de visualizações para estruturas de dados mais complexas, como malhas, facilitando a análise de dados científicos.

Motivados também pelas tecnologias relacionais padrão apresentarem um bom alicerce na gerência e na manipulação de uma grande variedade de conjuntos de dados, a DfAnalyzer representa os conjuntos de dados como relações. Ademais, essa biblioteca de

componentes tira proveito do poder analítico da linguagem SQL, permitindo análises focadas em múltiplos arquivos e diversos elementos de dados relacionados (armazenados em relações). Do ponto de vista de implementação, a DfAnalyzer adota um Sistema de Gerência de Banco de Dados (SGBD) relacional orientado a coluna e apresenta um conjunto de componentes para apoiar as funcionalidades discutidas anteriormente. Esses componentes são descritos em mais detalhes na Subseção 5.4.

5.2. Metodologia orientada ao fluxo de dados em simulações computacionais

Como discutido anteriormente, as simulações computacionais são caracterizadas pelo processamento de modelos computacionais encadeados cada vez mais complexos, consumindo e produzindo grande volume de dados. Muitos desses dados são armazenados em diferentes arquivos. Nesse contexto, o usuário do domínio científico comumente precisa analisar o conteúdo desses arquivos para validar ou refutar as suas hipóteses científicas. Com esse propósito, eles frequentemente utilizam programas *ad-hoc* e ferramentas alternativas para que os conteúdos desses arquivos sejam capturados. Ou seja, esses programas e ferramentas são empregados para acessar os dados científicos presentes em um arquivo, extrair apenas os dados científicos relevantes e indexar os dados científicos quando apenas a extração desses dados pode tornar custosa, ou mesmo impraticável, a carga dos dados em um repositório externo ou em uma base de dados.

Além da análise de dados científicos relevantes presentes em um arquivo, esses usuários necessitam investigar dados armazenados em diferentes etapas da simulação computacional e, consequentemente, de diferentes arquivos. Dessa forma, embora os dados científicos estejam acessíveis, eles também precisam estar disponíveis, ou seja, eles precisam seguir estruturas de dados e estar armazenados em um repositório externo de modo que o usuário do domínio científico possa relacionar dados de diferentes arquivos. Ao mesmo tempo, eles necessitam de recursos que favoreçam a modelagem e a execução das suas consultas, assim como a visualização dos dados.

Diante dessas questões, observou-se as seguintes limitações no cenário de simulações computacionais em larga escala: (i) a composição explícita dos programas de simulação não é normalmente representada; (ii) a captura de dados científicos é frequentemente realizada após a execução da simulação por meio de programas *ad-hoc* e ferramentas alternativas, como uma etapa de pós-processamento; e, consequentemente, (iii)

existem desafios no apoio à análise exploratória de dados científicos, uma vez que os dados de interesse não estão disponíveis em tempo de execução e, normalmente, não são relacionados pelos diferentes arquivos produzidos ao longo das etapas da simulação computacional. A partir dessas limitações, a DfAnalyzer segue uma abstração de fluxo de dados (Seção 3.1) – uma das contribuições desta tese – para tornar explícita a representação das diferentes etapas de processamento em uma simulação computacional (*i.e.*, transformações de dados), que envolvem o consumo e a produção de dados (*i.e.*, conjunto de dados) importantes na análise exploratória de dados científicos.

Mesmo com essa abstração de fluxo de dados, ainda não estão claras as diferentes etapas envolvidas na explicitação do fluxo de dados presente em uma simulação computacional, no apoio à extração de dados científicos e no processamento de consultas de acordo com as necessidades do usuário do domínio científico. Nesse sentido, esta tese propõe a extensão de uma metodologia para permitir a captura de dados de proveniência e científicos usando a DfAnalyzer a partir de simulações computacionais que não apresentam o seu fluxo de dados explicitados e não apoiam a proveniência dos dados. Essa metodologia estendida foi baseada na PrIMe (MUNROE *et al.*, 2006), uma técnica de engenharia de software amplamente conhecida na comunidade de proveniência para adaptar a modelagem de aplicações a fim de permitir a captura de dados de proveniência. As principais etapas da PrIMe consistem em identificar os tipos de dados necessários para responder às consultas baseadas na proveniência da simulação computacional, em analisar a modelagem da simulação para determinar em quais pontos os dados de proveniência devem ser registrados e em adaptar os programas de simulação para capturar os dados de proveniência de interesse. Motivados por essas etapas da PrIMe, a nossa metodologia é dividida em três etapas: modelagem do fluxo de dados, modelagem dos dados de domínio e modelagem das consultas.

Em um primeiro momento, também assumimos que o usuário do domínio científico não conhece o fluxo de dados da sua simulação computacional, pois esse fluxo de dados não está explicitamente definido. Desse modo, a primeira fase da nossa metodologia envolve um trabalho colaborativo entre o usuário do domínio científico e o especialista em ciência da computação para modelar a simulação computacional como um fluxo de dados, semelhante a uma etapa de pré-processamento para depois realizar a execução da simulação.

Nessa fase, esses usuários identificam as transformações de dados, os conjuntos de dados, os atributos presentes em cada conjunto de dados, que podem ser capturados de arquivos, e as dependências de dados entre as transformações. Para apoiar a captura de dados científicos por meio do uso de um programa *ad-hoc* ou uma ferramenta alternativa, a nossa abstração de fluxo de dados foi estendida para contemplar o conceito de *extrator de dados científicos*, ou, simplesmente, *extrator*. Assim, os extratores também fazem parte da especificação do fluxo de dados.

Como resultado, essa colaboração entre os usuários obtém a especificação do fluxo de dados para a simulação computacional de interesse. Essa especificação do fluxo de dados é normalmente representada por um modelo conceitual de dados, por meio de um diagrama Entidade-Relacionamento (ER) para, em seguida, ser mapeado em um modelo de dados lógico (HEUSER, 2009). Considerando a abstração de fluxo de dados, a definição de conjunto de dados segue uma representação próxima de uma relação e as tecnologias relacionais apresentam um bom alicerce na gerência de uma grande variedade de dados, favorecendo que a DfAnalyzer siga uma metodologia baseada no armazenamento dos metadados do fluxo de dados em uma base de dados relacional.

Uma vez que a especificação do fluxo de dados tenha sido obtida, o especialista em ciência da computação realiza ajustes nos programas de simulação com o propósito de capturar os dados de proveniência produzidos pelas diferentes instâncias das transformações de dados, também conhecidas como *tarefas*. Assim, enquanto a especificação do fluxo de dados contempla dados de proveniência prospectiva (Seção 3.2), os ajustes dessa fase são importantes para apoiar a captura de dados de proveniência retrospectiva (quando a simulação computacional for executada). Vale ressaltar que essa fase não depende do usuário do domínio científico, pois o especialista em ciência da computação já estaria ciente dos requisitos da simulação computacional.

Ademais, esse passo é importante na obtenção das dependências de dados em tempo de execução, ao especificar, no código fonte dos programas de simulação, quais tarefas são responsáveis pela produção e pelo consumo de elementos de dados em um conjunto. Nesse sentido, esses ajustes apresentam frequentemente um comportamento dinâmico, a fim de capturar as dependências de dados de diversas instâncias de uma transformação de dados (ou tarefas). Como exemplo, temos os cenários de varredura de parâmetros (RAICU *et al.*,

2008), em que diferentes elementos de dados de entrada (diversas combinações de valores dos atributos) são utilizados para avaliar um modelo computacional, realizando uma iteração na primeira transformação de dados para cada combinação de valores dos atributos de entrada.

Na fase seguinte, o usuário do domínio científico retoma a sua parceria com o especialista em ciência da computação para modelarem apenas os dados de domínio. Como primeiro passo, esses usuários definem quais valores de atributos podem ser obtidos diretamente dos programas de simulação, sem especificar um extrator de dados científicos. Em seguida, eles trabalham para definir os programas *ad-hoc* ou ferramentas alternativas a serem utilizadas pelos extratores de dados científicos e como associar os dados científicos (ou valores de atributos) extraídos aos valores de atributos obtidos sem extratores.

Como resultado, os usuários são capazes de compreender como dados científicos de diferentes origens, como um arquivo de dados científicos ou dados ainda alocados em memória pelo programa de simulação, são relacionados e, posteriormente, representados como elementos de dados em um conjunto de dados. Além disso, eles aproveitam o trabalho em colaboração para definir o trecho do código fonte em que as invocações dos extratores serão realizadas nos programas de simulação e como associá-las a um conjunto de dados, além de ajustes para carregar os elementos de dados dos conjuntos de dados na base de dados da DfAnalyzer. Dado que as fases anteriores tenham sido concluídas, os usuários podem executar a simulação computacional. Assim, a DfAnalyzer é capaz de capturar os dados de proveniência e os dados científicos produzidos pelos programas de simulação modificados e de armazenar esses dados em uma base de dados relacional, seguindo o nosso modelo de dados lógico. Somando-se a isso, a nossa ferramenta disponibiliza um conjunto de visões para apoiar a análise de dados científicos em tempo de execução. Nesse sentido, a última fase da nossa metodologia corresponde à modelagem de consultas por meio das visões oferecidas pela DfAnalyzer.

Apesar desta tese apresentar essa metodologia no contexto da DfAnalyer, em qualquer solução de análise de dados é necessária uma etapa de modelagem que represente as atividades e as dependências de um *workflow* na aplicacao científica. Isso é necessário quando se usa um SGWfC e, principalmente, quando se usa uma abordgem específica para a captura e a análise de dados (*e.g.*, DfAnalyzer). Portanto, um SGWfC também deveria ter

uma metodologia, tanto para identificar as atividades quanto os dados a serem capturados. Nesse sentido, a metodologia apresentada nas subseções a seguir também poderia ser aplicada ao utilizar-se um SGWfC.

5.2.1. Modelagem do fluxo de dados

Pelo trabalho em colaboração entre o usuário do domínio científico e o especialista em ciência da computação, a primeira fase da nossa metodologia consiste na modelagem da simulação computacional de interesse como um fluxo de dados para representar os seus dados de proveniência (FREIRE *et al.*, 2008). Para isso, esses usuários trabalham em uma especificação do fluxo de dados, podendo ser representada por um diagrama ER. Por conseguinte, esse diagrama ER pode ser mapeado em um modelo de dados lógico, representando os principais conceitos da abstração de fluxo de dados e a estrutura da simulação computacional. Por último, esses usuários precisam realizar os ajustes necessários nos programas de simulação para capturar os dados de proveniência. Nessa subseção, apresentamos de forma detalhada esses passos envolvidos na modelagem dos dados de proveniência, considerando um exemplo de *script* em Python usando Apache Spark (ZAHARIA *et al.*, 2010) para a contagem de palavras em arquivos informados como entrada, conforme a Figura 12.

```
# get input_files
input_files = sc.textFile("hdfs://../input_files.txt").collect()
occurrences = sc.parallelize([]) # empty RDD

# loop - for each input file
for file in input_files:
    # get lines
    lines = sc.textFile("hdfs://../" + file)

    # get word occurrences in each line
    words = lines.flatMap(lambda line: line.split(" "))
    tempOccurrences = words.map(lambda word: (word, 1))
    occurrences = occurrences.union(tempOccurrences)

# count occurrences of each
wordcounts = occurrences.reduceByKey(lambda a, b: a + b)

# save output
datasetcounts.saveAsRawDataFile("hdfs://../output.bin")
```

Figura 12. *Script* em Python usando Spark para a contagem de palavras.

Como mencionado anteriormente, a primeira fase da nossa metodologia consiste na modelagem da simulação computacional de interesse como um fluxo de dados a partir da gerência dos dados de proveniência. Entretanto, a especificação do fluxo de dados, até o momento, não está explícita para a simulação computacional de interesse. Por exemplo, a Figura 12 não mostra quais são as principais etapas de processamento computacional (*i.e.*, transformações de dados na abstração de fluxo de dados), os dados consumidos e produzidos em cada etapa (*i.e.*, conjuntos de dados e seus atributos) e as dependências de dados entre essas etapas. Tais conceitos são essenciais para expressar o fluxo dos elementos de dados e, eventualmente, de arquivos na simulação computacional de interesse.

Para atender a essa questão, o usuário do domínio científico trabalha com o especialista em ciência da computação em uma especificação do fluxo de dados. Frequentemente, eles apresentam um conjunto de passos bem definidos para obter essa especificação, sendo descrito nesta subseção. Como primeiro passo nesse processo, esses usuários identificam os trechos de código fonte em seus programas e *scripts* com um processamento computacional que exigem análises futuras, de acordo com dados que serão consumidos e produzidos. Nesse momento, esses usuários identificam, portanto, apenas as transformações de dados e o seu comportamento em termos de processamento computacional. Por exemplo, a Figura 13 mostra as três transformações de dados envolvidas no *script* em Python para a contagem de palavras, sendo essas transformações responsáveis por identificar as linhas presentes nos arquivos de texto de entrada (*Obter linhas*), obter as ocorrências de palavras em cada linha (*Identificar ocorrências de palavras*) e agrupar as ocorrências por palavra (*Contar ocorrências de palavras*).

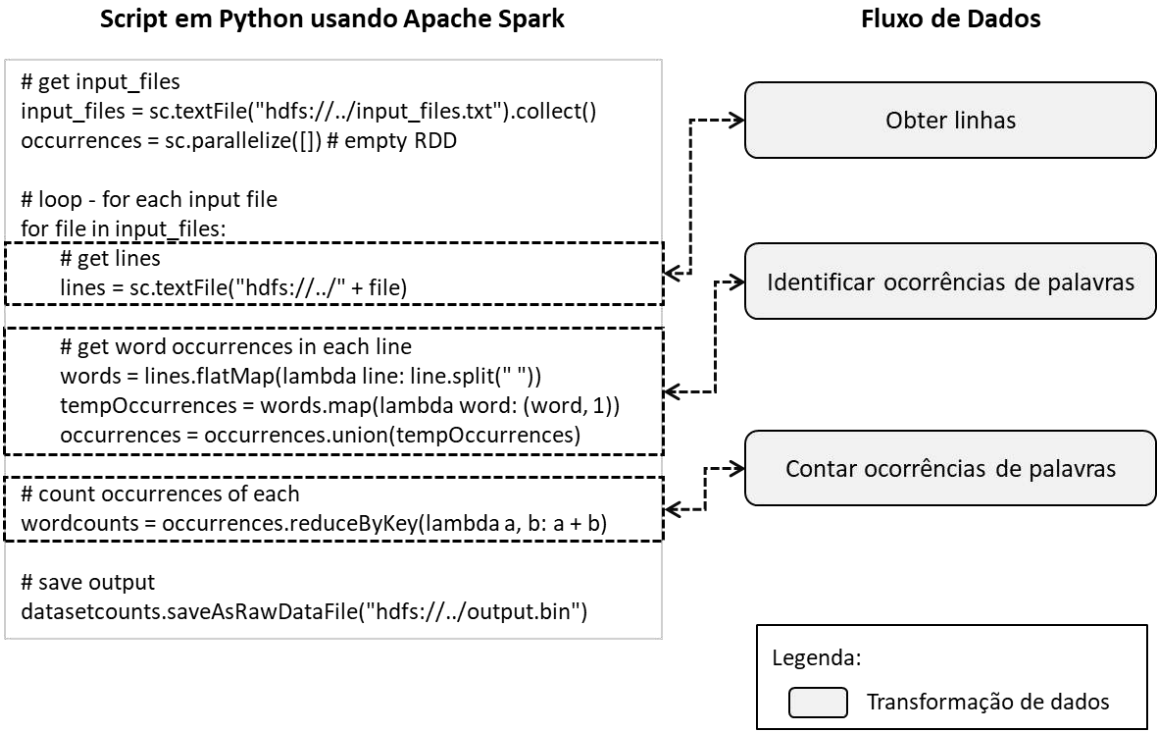


Figura 13. Especificação das transformações de dados em uma simulação computacional.

Após a identificação das transformações de dados, os usuários trabalham para identificar os conjuntos de dados (e os seus atributos) que serão consumidos e produzidos em cada transformação de dados. Além disso, quando conjuntos de dados são consumidos por uma transformação e produzidos por outra, eles aproveitam para definir as dependências de dados. Nesse contexto, uma dependência de dados consiste em dizer que uma transformação depende de dados produzidos por outra transformação em função de um conjunto de dados (Definição 3.1.5 da Seção 3.1). A Figura 14 mostra os conjuntos de dados manipulados pelas transformações de dados identificadas no passo anterior, assim como expressa as dependências de dados ao relacionar as transformações pelas setas. Além disso, destacamos em negrito o rótulo (ou nome) dos conjuntos de dados, pois, ao adotarmos uma abstração de fluxo de dados, o nosso foco está na análise do fluxo de elementos de dados manipulados pelos conjuntos de dados.

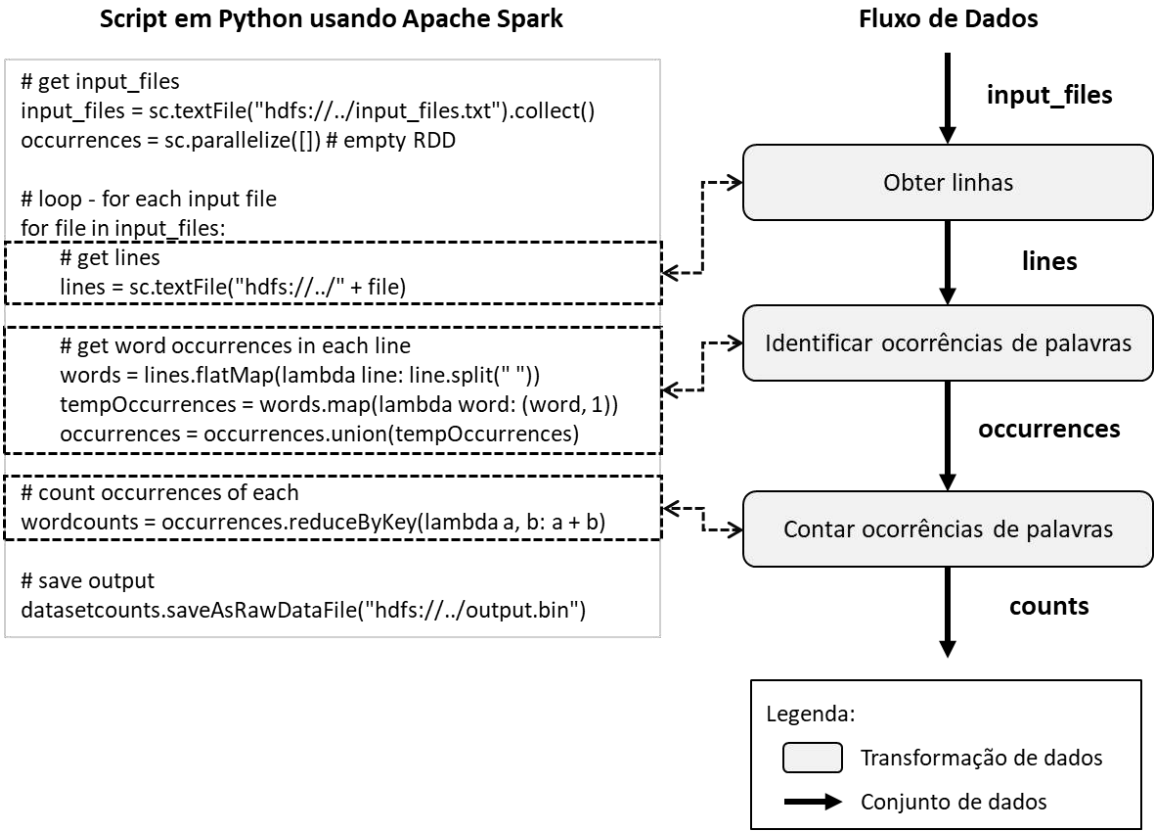


Figura 14. Especificação dos conjuntos de dados e das suas dependências em uma simulação computacional.

Mais especificamente, a primeira transformação de dados desse *script* (*Obter linhas*) consome os arquivos de texto presentes no arquivo *input_files.txt*, que são representados por um conjunto de dados do Spark, ou RDD (do termo, em inglês, *Resilient Distributed Dataset*). Como resultado, o conjunto de dados *lines* é produzido com as linhas de texto obtidas dos arquivos de entrada (atributo *line*, não apresentado na figura). Para cada linha do conjunto de dados *lines*, a segunda transformação identifica as palavras existentes e atribui o valor um para a sua ocorrência (ou seja, obtemos dois atributos, nomeados *word* e *occurrence*). Como saída dessa transformação, todas as palavras identificadas são armazenadas no conjunto de dados *occurrences*, mesmo que existam redundâncias de elementos de dados (tuplas com os mesmos valores de atributos). Por último, a terceira transformação de dados agrupa as ocorrências por palavra, gerando o conjunto de dados *counts*, que apresenta os atributos *word* (palavra encontrada) e *count* (número de ocorrências da palavra em todas as linhas analisadas, mesmo que a palavra de interesse esteja presente em arquivos diferentes).

Além dos conceitos da abstração de fluxo de dados, observamos que o processo de captura de dados científicos comumente realizado pelos usuários não é representado na especificação do fluxo de dados. Basicamente, em cenários reais, o usuário do domínio científico utiliza programas *ad-hoc* ou ferramentas alternativas para acessar, extrair e, eventualmente, indexar dados científicos de arquivos. Tais dados científicos correspondem, na abstração de fluxo de dados, aos valores de atributos presentes em elementos de um conjunto de dados. Sendo assim, definimos os programas de captura de dados científicos como *extratores de dados científicos* (ou, de uma forma abreviada, *extrator*), sendo cada extrator associado a determinados atributos de um conjunto de dados. Para formalizar o conceito de extrator de dados científicos, nós estendemos a abstração de fluxo de dados (Definições 5.2.1 e 5.2.2). Existem casos também em que os dados científicos obtidos de dois extratores precisam ser relacionados.

Definição 5.2.1. (Extrator de Dados) Um extrator de dados $\psi = (\rho, A)$ é composto de um programa de extração de dados científicos e da sequência de atributos $A \mid \forall a \in A: a = (name, type)$ extraídos por ρ.

Definição 5.2.2. (Conjunto de Dados) Um conjunto de dados $s = (A, C, \Psi)$ é uma tripla composta de atributos predefinidos $A \mid \forall a \in A: a = (name, type)$, de coleções de dados $C \mid \forall e \in C :$ $card(e.V) = card(A \cup \Psi.\mathrm{A})$, e de extratores de dados científicos Ψ, em que $\mathrm{s.A} \cap \Psi.\mathrm{A} = \emptyset$.

Como passo seguinte dessa fase, o especialista em ciência da computação utiliza a especificação do fluxo de dados para realizar a *modelagem conceitual* da sua simulação computacional. A modelagem conceitual é frequentemente realizada usando um diagrama ER, que consiste em uma descrição em alto nível dos dados a serem armazenados na base de dados, considerando, inclusive, as restrições associadas a esses dados. Considerando novamente a simulação computacional da Figura 14, o especialista em ciência da computação representa os principais conceitos do fluxo de dados desse *script* em Python em um diagrama ER. Esse diagrama é mostrado na Figura 15 e contempla a especificação do fluxo de dados, envolvendo entidades associadas aos conceitos da abstração de fluxo de dados (Seção 3.1). Logo, esse diagrama não sofre alterações nas suas entidades e relacionamentos ao modificarmos a simulação computacional, embora as instâncias das suas entidades e relacionamentos sejam diferentes de acordo com a estrutura do fluxo de dados modelado. Por exemplo, a Figura 16 apresenta as instâncias da entidade *Dataset* do

diagrama ER para a especificação do fluxo de dados a partir da simulação computacional de contagem de palavras.

Enquanto isso, os relacionamentos desse diagrama representam como os diferentes conceitos (*e.g.*, transformação de dados e conjunto de dados) estão associados ou mesmo apresentam uma ideia de composição. Por exemplo, o relacionamento ternário entre as entidades *Transformation* e *Dataset* representam como duas transformações de dados são dependentes entre si em função de elementos de dados de um determinado conjunto de dados. Ou seja, esse relacionamento representa o conceito de dependência de dados. Já no sentido de composição, o relacionamento entre as entidades *Dataset* e *Attribute* evidencia que um conjunto de dados é composto de uma sequência de atributos.

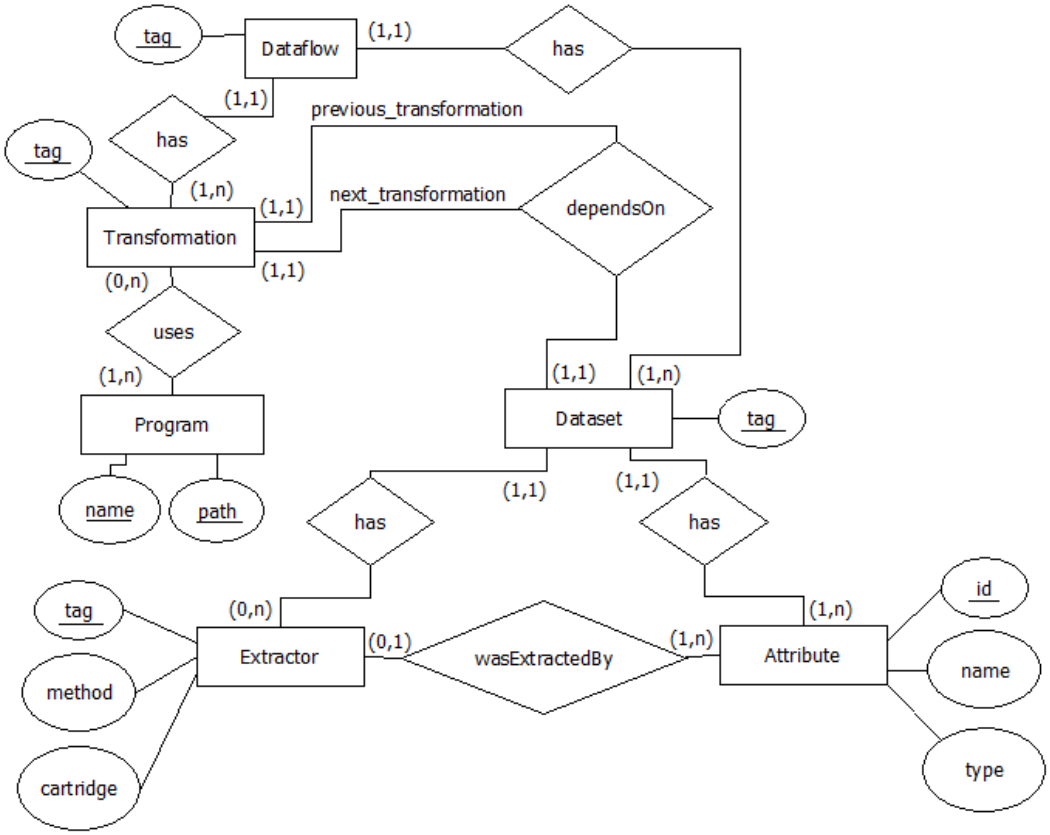


Figura 15. Diagrama ER para a especificação do fluxo de dados.

tag
input_files
lines
occurrences
counts

Figura 16. Exemplo de instâncias da entidade *Dataset* do diagrama ER da Figura 15.

A partir da especificação do fluxo de dados, o próximo passo dessa fase consiste na *modelagem lógica*, que mapeia os conceitos envolvidos no diagrama ER da Figura 15 em um modelo de dados lógico. Ademais, pelo fato da definição de conjuntos de dados seguir uma representação próxima de uma relação e as tecnologias relacionais apresentarem um bom alicerce na gerência de uma grande variedade de conjuntos de dados, a nossa metodologia segue um modelo de dados relacional. Diferentemente da modelagem do fluxo de dados, que assume o trabalho em colaboração do usuário do domínio científico com o especialista em ciência da computação, este passo contempla esforços apenas do segundo tipo de usuário, pois ele tem acesso aos requisitos necessários da simulação computacional. Além disso, em virtude desse diagrama ER não sofrer mudanças com novas especificações de fluxo de dados, o seu modelo de dados lógico é visto de uma forma fixa, ou seja, ele corresponderia ao esquema inicial da base de dados da DfAnalyzer (sem precisar de modificações com novas especificações de fluxo de dados sendo registradas).

Em termos de modelagem, as entidades do diagrama ER para a especificação do fluxo de dados (Figura 15) são mapeadas de uma forma direta em relações, de maneira que cada atributo da entidade seja um campo da relação, como mostrado na Figura 17. Como exceção, os atributos identificadores de uma entidade, que não são do tipo inteiro, consideram a adição de um campo *id* do tipo inteiro para ser a chave primária da relação. Vale ressaltar que essas relações permitem apenas o armazenamento de dados de proveniência (Seção 3.2), isto é, sem os dados de domínio, pois as suas entidades não referenciam os dados científicos que seriam produzidos em tempo de execução pela simulação computacional.

Diferente das entidades, os relacionamentos nesse diagrama ER podem apresentar três comportamentos distintos. Quando temos um relacionamento ternário (*e.g.*, *dependsOn*), esse relacionamento é mapeado para uma relação no nosso modelo de dados lógico. De uma forma mais específica, nós identificamos cada entidade participante (entidades *Transformation* e *Dataset*) e associamos valores para os seus atributos descritivos do relacionamento, de modo que a relação criada (*e.g.*, *data_dependency*) possua um campo identificador com restrição de chave primária (*e.g.*, *id*) e campos para cada entidade participante com restrição de chave estrangeira (*e.g.*, *previous_dt_id*, *next_dt_id* e *ds_id*).

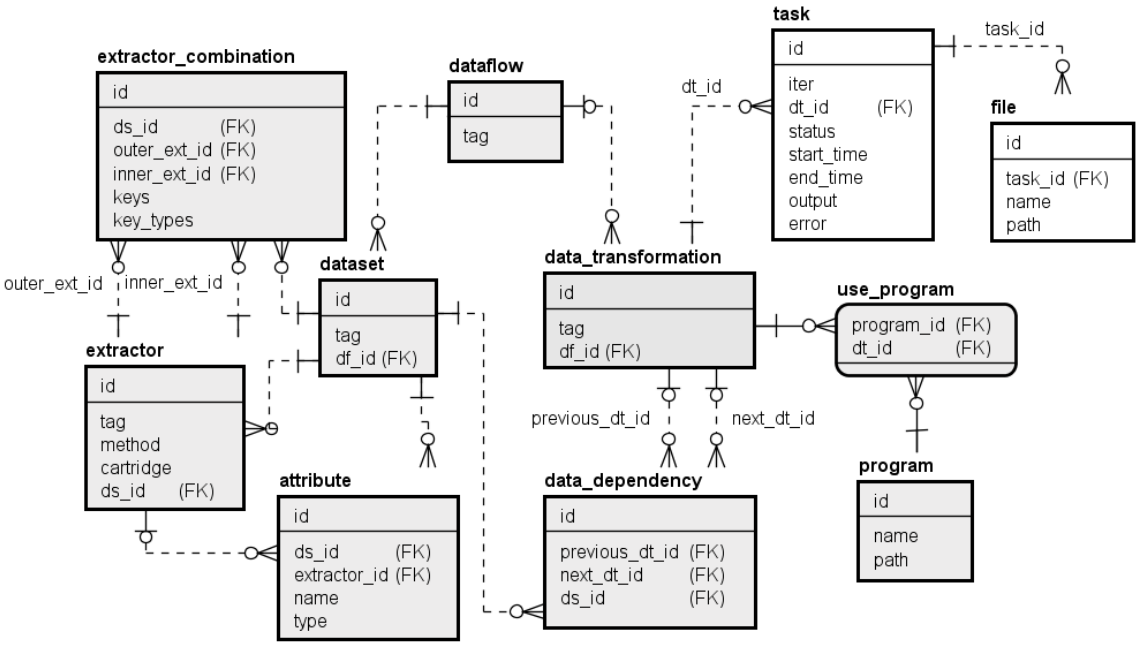


Figura 17. Modelo de dados lógico para os dados de proveniência.

Quando um relacionamento apresenta cardinalidades máximas 1:n (incluindo relacionamentos 1:1), uma restrição de chave estrangeira é adicionada na relação de destino. Por exemplo, no diagrama ER, o relacionamento entre as entidades *Dataflow* e *Transformation* possui cardinalidades máximas 1:n de maneira que um fluxo de dados é composto de um conjunto de transformações de dados. Assim, ao mapearmos esse relacionamento para o modelo de dados lógico, uma restrição de chave estrangeira é criada no campo *df_id* da relação *data_transformation* para o campo *id* da relação *dataflow*.

Por outro lado, quando o relacionamento apresenta cardinalidades máximas n:m, uma nova relação é criada no modelo de dados lógico, como uma relação intermediária. Para isso, restrições de chave estrangeira são criadas de campos da relação intermediária para campos (chave primária) das relações derivadas, ou seja, das entidades envolvidas nesse relacionamento. Por exemplo, o relacionamento entre as entidades *Transformation* e *Program* descreve que uma transformação de dados pode usar um ou mais programas de simulação, assim como um programa pode ser utilizado por nenhuma ou várias transformações. Dessa forma, quando esse relacionamento é mapeado para o modelo de dados lógico, uma relação intermediária *use_program* é criada com dois campos, *program_id* e *dt_id*, que apresentam restrições de chave estrangeira para o campo *id* da relação *program* e para o campo *id* da relação *data_transformation*, respectivamente.

Para permitir o armazenamento de dados de proveniência retrospectiva, o nosso modelo de dados lógico contém duas relações, *task* e *file*, para representar as instâncias das transformações de dados (também conhecidas como tarefas) que foram executadas, assim como os arquivos que foram eventualmente consumidos e produzidos em tempo de execução, respectivamente. Na Figura 17, essas relações são mostradas com o fundo na cor branca. Como a motivação de termos criado essas relações no modelo de dados lógico tem influência direta com a captura de dados científicos, mais discussões sobre esses conceitos são apresentadas na fase de modelagem dos dados de domínio (ou dados científicos).

Apesar de termos um modelo de dados lógico que é utilizado pela DfAnalyzer para armazenar os dados de proveniência na sua base de dados, os programas de simulação ainda não são capazes de capturar os dados de proveniência prospectiva e retrospectiva da simulação computacional. Nesse sentido, o próximo passo dessa fase consiste em realizar ajustes nos programas de simulação a fim de definir a especificação do fluxo de dados e como os dados de proveniência retrospectiva devem ser capturados após a nossa metodologia, ou seja, quando a simulação computacional for executada. Para isso, a DfAnalyzer provê um conjunto de operações para facilitar essas mudanças no código fonte dos programas de simulação, sendo essas operações discutidas em mais detalhes na Subseção 5.3.

Para os dados de proveniência prospectiva, a Figura 18 apresenta um *script* em Python – usando as operações apresentadas na Subseção 5.3.1 –, que seria invocado antes do *script* de contagem de palavras, com o objetivo de armazenar a especificação do fluxo de dados na base de dados da DfAnalyzer. Quanto aos dados de proveniência retrospectiva, essa biblioteca de componentes deve ser capaz de obter informações como quais instâncias da transformação de dados foram executadas, quais são as dependências de dados entre essas instâncias (em função da definição do comportamento de cada transformação de dados) e quais arquivos foram manipulados por elas. Para facilitar esses ajustes, a DfAnalyzer conta com operações específicas para a proveniência retrospectiva, conforme apresentado na Subseção 5.3.3. Do ponto de vista de implementação, o especialista em ciência da computação também não precisa da ajuda do usuário do domínio científico nesse momento, pois ele já tem os requisitos ou o conhecimento necessário da simulação para saber em que trechos do código fonte os ajustes devem ser realizados.

```
dataflow("word_count")
program("GetLines::textFile()", "/root/word_count")
dataset("input_files", ["file"], ["file"])
dataset("lines", ["line"], ["text"])
transformation("get_lines", ["input_files"], ["lines"], ["GetLines::flatMap()"])
program("IdentifyOccurrences::flatMap()", "/root/word_count")
dataset("occurrences", ["word","occurrence"], ["text","numeric"])
transformation("identify_occurrences", ["lines"], ["occurrences"], ["IdentifyOccurrences::flatMap()"])
program("CountWords::reduceByKey()", "/root/word_count")
dataset("counts", None, None)
extractor("odeduplication", "odeduplication", "EXTRACTION", "EXTERNAL_PROGRAM",
        ["word","count], ["text","numeric"])
transformation("count_words", ["occurrences"], ["counts"], ["CountWords::reduceByKey()"])
```

Figura 18. *Script* em Python usando as operações da DfAnalyzer para armazenar na base de dados a especificação do fluxo de dados da simulação de contagem de palavras.

Considerando novamente a especificação do fluxo de dados do *script* em Python para a contagem de palavras (Figura 14), temos dois comportamentos possíveis para as suas transformações de dados. Existem transformações de dados 1:n, em que, para cada elemento de dados de entrada, uma tarefa produz vários elementos de dados de saída, como instâncias das transformações *get_lines* e *identify_occurrences*. Por outro lado, temos transformações de dados n:1, em que, para cada conjunto de elementos de dados de entrada (ou coleção de dados), uma tarefa produz apenas um elemento de dados de saída, como uma instância da transformação *count_words*. Essas informações são essenciais para o especialista em ciência da computação realizar corretamente os ajustes nos programas de simulação e gerenciar as dependências de dados em função das instâncias das transformações de dados.

Em virtude das mudanças realizadas nos programas de simulação, a nossa biblioteca de componentes favorece a execução de diferentes iterações da simulação computacional (semelhante ao cenário de varredura de parâmetros), de modo que os processos de captura e de carga dos dados de proveniência sejam realizados corretamente, garantindo a consistência da nossa base de dados. Consequentemente, por respeitar os requisitos definidos no diagrama ER para a especificação do fluxo de dados, os dados armazenados

na nossa base de dados nesse passo também contemplam os rastros de proveniência. Mais detalhes sobre essa questão são apresentados na Subseção 5.3.3.

5.2.2. Modelagem dos dados de domínio

Na subseção anterior vimos os passos envolvidos na fase de modelagem dos dados de proveniência. Entretanto, o modelo de dados lógico obtido não apresenta os dados científicos de interesse comumente analisados pelo usuário do domínio científico. Nesse sentido, esta subseção apresenta a fase de modelagem dos dados de domínio, assim como a importância de especificarmos os extratores no cenário da análise exploratória de dados científicos. Ademais, os ajustes necessários nos programas de simulação para apoiar essa fase também são apresentados em detalhes.

Retomando a primeira etapa que envolvia o trabalho colaborativo entre o usuário do domínio científico e o especialista em ciência da computação para especificar o fluxo de dados de uma simulação, observamos que, nesse momento, esses usuários já haviam definido os conjuntos de dados e os seus esquemas (ou seja, os seus atributos), assim como os extratores de dados científicos. Entretanto, o mapeamento desses conceitos para um diagrama ER não foi realizado e, consequentemente, o nosso modelo de dados lógico não contempla tais dados do domínio científico.

Dessa forma, a nossa metodologia ainda não havia, de fato, especificado como os dados científicos seriam obtidos ou extraídos de arquivos nessa simulação computacional. Como primeiro passo dessa fase, portanto, uma modelagem conceitual foi realizada a partir de conceitos da nossa especificação do fluxo de dados para compreender os conjuntos de dados, os atributos e os extratores de dados científicos utilizados na simulação computacional de interesse. Como resultado, um segundo diagrama ER (Figura 19) foi criado para descrever quais conjuntos de dados e extratores (com os seus atributos) são manipulados pelas diferentes transformações de dados e como eles podem ser relacionados ao longo da especificação do fluxo de dados.

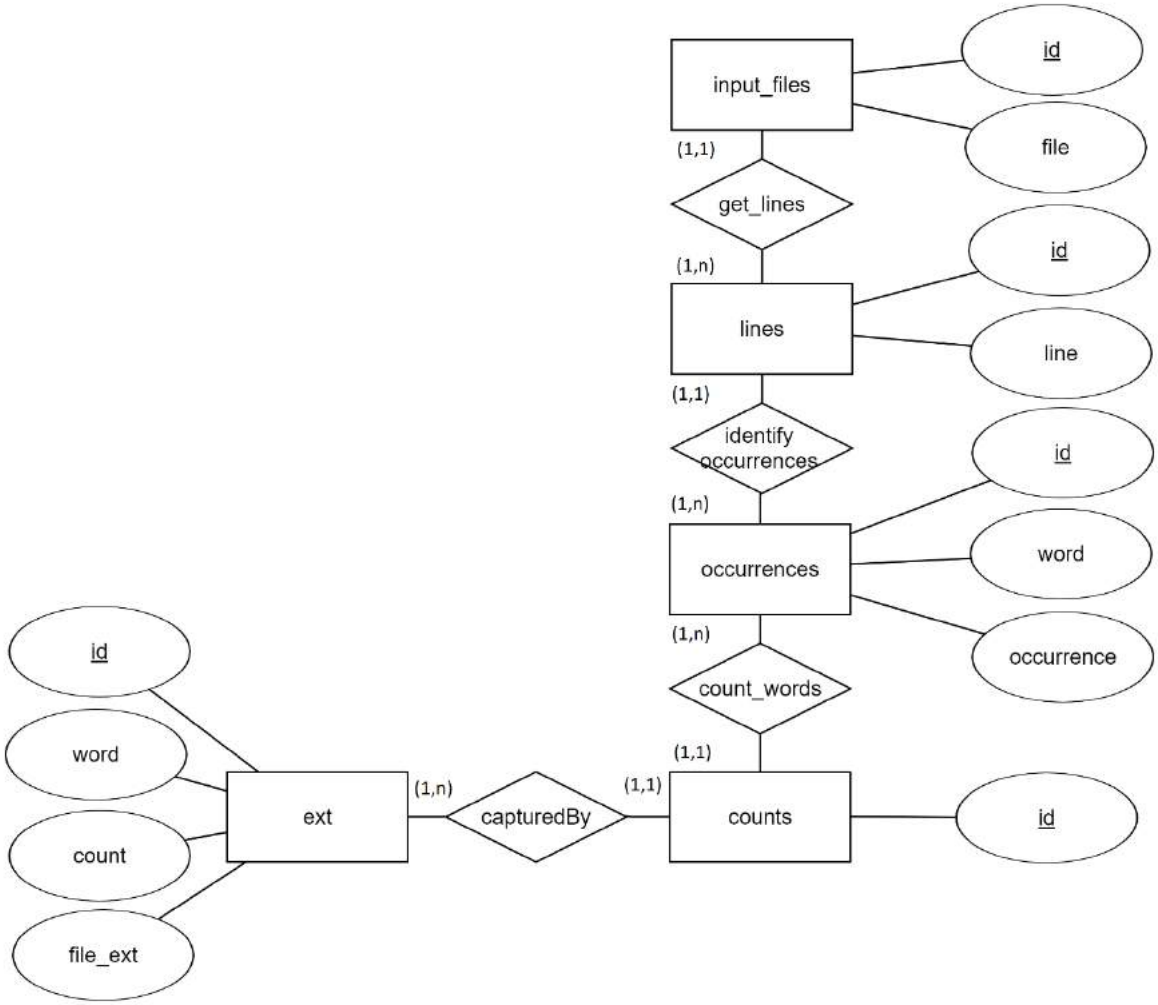


Figura 19. Diagrama ER para os dados de domínio a partir do *script* da Figura 14.

Esse mapeamento é mais intuitivo do ponto de vista do usuário do domínio científico do que o primeiro diagrama, pois esse usuário conhece os atributos de interesse e o modelo computacional responsável por manipular os dados de domínio desses atributos. Basicamente, as entidades nesse diagrama correspondem aos conjuntos de dados e aos extratores de dados, sendo que os atributos das entidades representam os atributos dos conjuntos de dados ou os atributos obtidos pelos extratores, enquanto os relacionamentos descrevem como instâncias de uma transformação de dados (ou tarefas) consomem elementos de dados de um conjunto e produzem elementos de dados em outro conjunto, ou como dados científicos extraídos estão associados aos elementos de dados de um conjunto. Ou seja, os relacionamentos são fundamentais para a descrição das dependências de dados e do processo de extração de dados científicos.

Considerando o *script* para a contagem de palavras da Figura 14, a Figura 20 mostra as instâncias da entidade *occurrences*, que correspondem aos elementos de dados do conjunto de dados *occurrences* para uma execução dessa simulação computacional. Enquanto isso, a Figura 21 mostra instâncias do relacionamento *count_words*, que considera o consumo de vários elementos de dados do conjunto *occurrences* (instâncias da entidade *occurrences*) por uma instância da transformação de dados *count_words* para produzir apenas um elemento de dados no conjunto *counts* (instâncias da entidade *counts*). Por exemplo, a instância (4,2) indica que o quarto elemento de dados do conjunto *occurrences* (atributo *occurrences_id*) foi consumido por uma instância da transformação de dados *count_words* para produzir o segundo elemento de dados do conjunto *counts* (atributo *counts_id*).

id	word	occurrence
1	science	1
2	data	1
3	big	1
4	data	1
5	data	1
6	science	1

Figura 20. Instâncias da entidade *occurrences* do diagrama ER da Figura 19.

occurrences_id	counts_id
1	1
2	2
3	3
4	2
5	2
6	1

Figura 21. Instâncias do relacionamento *count_words* do diagrama ER da Figura 19.

A partir da especificação dos conjuntos de dados e dos extratores, o próximo passo consiste na *modelagem lógica* realizada apenas pelo especialista em ciência da computação. Nesse caso, os conceitos envolvidos no diagrama ER da Figura 19 são mapeados em um modelo de dados lógico. Primeiramente, cada entidade do diagrama ER para os dados de domínio é mapeada em uma nova relação no modelo de dados lógico, a fim de representar um conjunto de dados, em que as tuplas da relação correspondem aos elementos de dados, ou um extrator, em que as tuplas da relação correspondem aos dados científicos extraídos de arquivos. A Figura 22 mostra esse modelo de dados lógico, em que as relações dos

conjuntos de dados são nomeadas com um prefixo "*ds_*" acrescido do rótulo do conjunto de dados (*e.g.*, a relação *ds_input_files* foi criada para o conjunto de dados *input_files*).

Além disso, cada atributo de uma entidade do diagrama ER é representado por um campo na sua respectiva relação. Em termos de mapeamento, se o atributo for do tipo arquivo, então o seu campo adotará o tipo inteiro com uma restrição de chave estrangeira para o campo *id* da relação *file*. Isso evidencia que todos os arquivos manipulados pela nossa solução, mesmo que sejam expressos em um conjunto de dados ou um extrator, são armazenados na relação *file* e, consequentemente, associados à instância da transformação de dados (campo *task_id*) responsável pela sua geração.

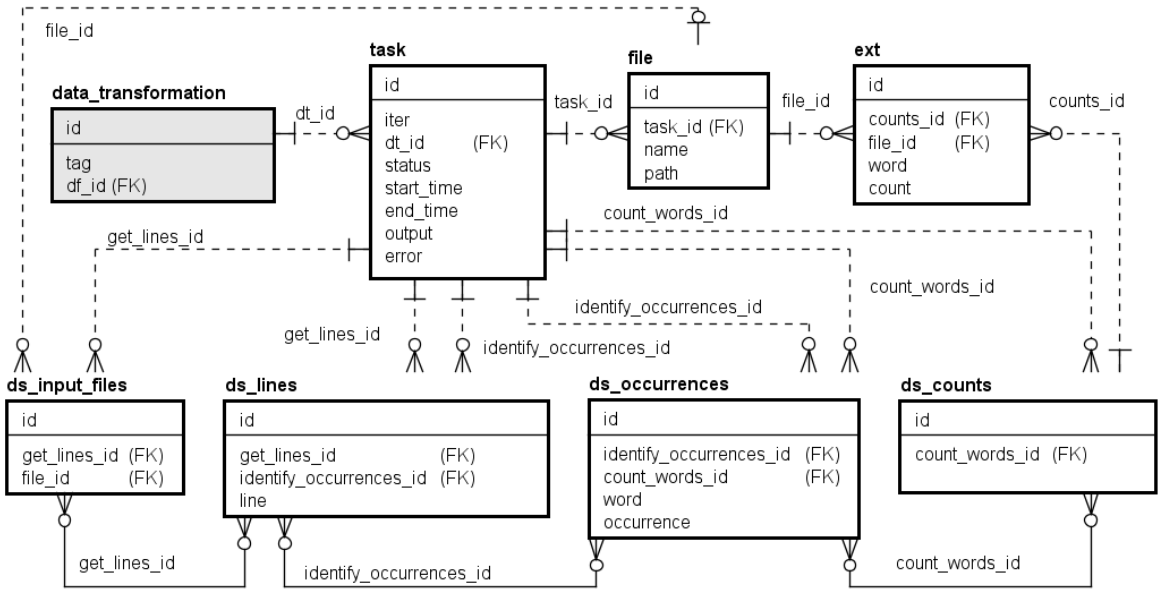


Figura 22. Modelo de dados lógico para os dados de domínio capturados no exemplo da Figura 14.

Uma vez criadas as relações para os conjuntos de dados e os extratores, os relacionamentos do diagrama ER são avaliados. Para os relacionamentos entre dois conjuntos de dados, temos que, semanticamente, uma instância de uma transformação de dados específica foi executada para consumir elementos de dados de um conjunto e produzir elementos de dados em outro conjunto. Assim, um novo campo é criado em cada relação do conjunto de dados para identificar as instâncias dessa transformação envolvidas na manipulação de elementos de dados. Esse novo campo apresenta o rótulo da transformação de dados acrescido do sufixo "*_id*", *e.g.*, os campos *identify_occurrences_id* nas relações *ds_lines* e *ds_occurrences* obtidos do relacionamento *identify_occurrences_id*.

Além da criação do campo para as instâncias da transformação de dados, uma restrição de chave estrangeira é definida desse campo para o campo *id* da relação *task*, a fim de garantir a integridade referencial. Ao mesmo tempo, uma restrição é definida para expressar o relacionamento entre elementos de dados de dois conjuntos de dados distintos em função das instâncias de uma transformação. Esse relacionamento é fundamental na análise exploratória de dados científicos, pois ele permite a análise do rastro dos dados de proveniência no nível físico (Subseção 3.2.4), ou seja, ele favorece a investigação do fluxo de elementos de dados em uma simulação computacional utilizando as instâncias de uma transformação para associar elementos de dados de diferentes conjuntos de dados.

Diante dessa modelagem de dados no nível lógico, a DfAnalyzer cria uma base de dados capaz de armazenar os dados de domínio produzidos pela simulação computacional, utilizando ou não um extrator de dados científicos. Formalmente, o esquema da relação para um conjunto de dados é apresentado na Definição 5.2.3.

Definição 5.2.3. (Esquema da Relação do Conjunto de Dados) Uma relação de dados r de um conjunto de dados s, também denotada por $Relation(s)$, contém um esquema de dados $\mathcal{R} = (a_{id}, A_{transformation}, A_{scientific})$ composto de uma tripla com o atributo a_{id} de identificação do elemento de dados do conjunto, os atributos $A_{transformation}$ referentes aos identificadores das instâncias das transformações de dados que envolveram a produção ou o consumo de tuplas em r, e os atributos $A_{scientific}$ associados às propriedades do domínio científico que se deseja representar em r. Logo, o esquema $\mathcal{R}$ da relação de dados r de um conjunto de dados s é denotado como $Schema(r)$ ou $Schema(Relation(s))$.

Já os relacionamentos entre um conjunto de dados e um extrator visam especificar como os dados científicos capturados por um extrator correspondem a uma coleção de dados no conjunto. Em termos de mapeamento, um campo com o rótulo do conjunto de dados acrescido do sufixo "*_id*" é criado na relação do extrator de dados com restrição de chave estrangeira para o campo *id* da relação do conjunto de dados. Pelo fato do extrator realizar a captura de dados científicos em arquivos, um campo *file_id* é criado na sua relação. Formalmente, o esquema da relação para um extrator de dados é apresentado na Definição 5.2.4.

Definição 5.2.4. (Esquema da Relação do Extrator de Dados Científicos) Uma relação r de um extrator de dados científicos ψ, também denotada por $Relation(\psi)$, contém um esquema de dados

$\mathcal{R} = (a_{id}, a_{dataset}, a_{file}, A_{scientific})$, em que a_{id} é o identificador dos dados científicos capturados, $a_{dataset}$ é o identificador do elemento de dados no conjunto de dados $dataset$, a_{file} é o atributo opcional para o arquivo de dados científicos e $A_{scientific}$ são os atributos do domínio científico. Logo, o esquema $\mathcal{R}$ de uma relação r de um extrator de dados científicos ψ é denotado como $Schema(r)$ ou $Schema(Relation(\psi))$.

Como resultado das modelagens lógicas para os dados de proveniência e os dados de domínio, a DfAnalyzer é capaz de criar uma base de dados relacional que armazena os dados de proveniência e de domínio. Vimos também que o modelo de dados lógico sofrerá mudanças apenas em função da especificação dos dados de domínio representados nos conjuntos de dados ou capturados pelos extratores, uma vez que um novo conjunto de dados (ou extrator) implica uma nova relação e seus relacionamentos no modelo de dados lógico.

Na fase de especificação do fluxo de dados, vimos que os esquemas dos conjuntos de dados apresentam os atributos de domínio, cujos valores são, frequentemente, capturados pelos extratores de dados científicos. Nos casos em que não há o uso de extratores, os dados de domínio são obtidos diretamente de dados estruturados alocados em memória, que podem ser representados como elementos de dados em um determinado conjunto. Todavia, quando os extratores de dados científicos são utilizados, os valores de atributos de interesse são acessados, extraídos, e, eventualmente, indexados de arquivos. Consequentemente, esses dados precisam ser associados a um determinado conjunto.

Em função desses casos, os ajustes nos programas de simulação também consideram esses dois comportamentos, um para a extração de dados científicos e outro para representar os elementos de dados obtidos diretamente dos programas de simulação. Considerando primeiramente a extração de dados científicos, esse comportamento requer a interação entre o usuário do domínio científico e o especialista em ciência da computação para identificarem quais arquivos precisam ser investigados, como extrair os atributos de interesse deles, quais tipos de elementos de dados ele deseja capturar (*e.g.*, filtragem de determinados valores de atributo) e qual programa *ad-hoc* ou ferramenta alternativa, como o FastBit, deveria ser utilizado. Como resultado, o especialista em ciência da computação saberia como invocar o programa ou a ferramenta para extrair os dados científicos de interesse.

Apesar desses passos iniciais, o especialista em ciência da computação ainda não possui as informações necessárias para realizar os ajustes nos programas de simulação. Portanto, o especialista em ciência da computação trabalha com o usuário do domínio científico novamente para definir em que trechos do código fonte do programa de simulação as invocações dos programas de extração precisam ser adicionadas. Por exemplo, a Figura 23 mostra a invocação de um programa *ad-hoc extractProgram* para extrair dados científicos no *script* em Python para contagem de palavras usando Spark. Nesse caso, o arquivo *output.bin* apresenta os dados de domínio presentes no RDD *counts*. Em termos de processamento computacional, o programa *extractProgram* realiza a extração das palavras identificadas e dos seus números de ocorrências. Apesar desse exemplo de extração de dados científicos, vale ressaltar que, em uma implementação usando Spark, a abordagem mais recomendada seria tirar proveito dos elementos de dados alocados em memória pelos RDDs.

```
# get input_files
input_files = sc.textFile("hdfs://../input_files.txt").collect()
occurrences = sc.parallelize([]) # empty RDD

# loop - for each input file
for file in input_files:
    # get lines
    lines = sc.textFile("hdfs://../" + file)

    # get word occurrences in each line
    words = lines.flatMap(lambda line: line.split(" "))
    tempOccurrences = words.map(lambda word: (word, 1))
    occurrences = occurrences.union(tempOccurrences)

# count occurrences of each
wordcounts = occurrences.reduceByKey(lambda  a, b: a + b)

# save output
datasetcounts.saveAsRawDataFile("hdfs://../output.bin")
```

```
extract("./extractProgram", "hdfs://output.bin")
```

Figura 23. Ajuste em um programa de simulação para a extração de dados científicos.

Em outros cenários, os usuários precisam capturar dados científicos de dois ou mais arquivos usando diferentes extratores. Como esses dados extraídos poderiam representar elementos de dados em um conjunto de dados específico, os usuários precisariam definir como os dados científicos obtidos por diferentes extratores devem se relacionar. Diante

dessa circunstância, nós introduzimos o conceito de *combinação de extratores* na Definição 5.2.5, que especifica como os dados científicos capturados por diferentes extratores devem ser relacionados. Em uma perspectiva de apenas dois extratores, descrevemos que essa combinação de extratores corresponde a uma operação relacional de junção, em que os predicados de junção seriam os atributos que apresentam o mesmo nome e tipo nos esquemas das relações de dados obtidas pelos extratores. Logo, ao invés de exigir mais um esforço de especificação pelos usuários, nós inferimos automaticamente os predicados de junção.

Definição 5.2.5. (Combinação de Extratores) Uma combinação de dois extratores ψ_1 e ψ_2, denotada por $CombinedExtractors(\Psi) \mid \Psi = \psi_1 \cup \psi_2$, é uma operação de junção natural entre as relações de dados de cada extrator, denotadas por $r_{\psi_1} = Relation(\psi_1)$ e $r_{\psi_2} = Relation(\psi_2)$. De forma simplicada, $CombinedExtractors(\Psi) = r_{\psi_1} \bowtie_{\xi} r_{\psi_2}$. Caso Ψ tenha apenas um extrator ψ_1, isto é, $\Psi = \{\psi_1\}$, então $CombinedExtractors(\Psi) = r_{\psi_1}$.

Uma vez que os dados científicos tenham sido capturados por extratores ou obtidos diretamente de estruturas alocadas em memória, esses dados precisam ser representados de acordo com o conceito de coleção de dados da nossa abstração de fluxo de dados (Seção 3.1). Ou seja, esses dados científicos precisam ser vistos como um conjunto de elementos de dados, em que cada elemento corresponde a uma sequência de valores de atributos, sendo que os atributos predefinidos fazem parte da especificação do conjunto de dados. De uma forma mais prática, o especialista em ciência da computação, com o auxílio do usuário do domínio científico, precisa ajustar o programa de simulação para ser capaz de capturar esses dados de domínio e armazená-los na nossa base de dados.

Ademais, a DfAnalyzer apresenta uma operação, conhecida como *collection*, para permitir a captura e a carga dos dados de domínio como elementos em um conjunto de dados (mais detalhes na Subseção 5.3.3). Por exemplo, a Figura 24 mostra a especificação da operação *collection* no *script* de contagem de palavras, assumindo que o valor do atributo *word* é obtido diretamente de dados alocados em memória, enquanto o valor do atributo *count* é obtido por um extrator *ext*. No caso, esse extrator obtém os valores do atributo *count* de um arquivo binário, cujo conteúdo corresponde aos dados do RDD *counts*, e os armazena no arquivo *ext.data*. Nessa invocação da operação *collection*, o primeiro argumento corresponderia ao nome da transformação de dados (*i.e.*, *count_words*), o segundo à

iteração dessa transformação de dados (ou à tarefa), o terceiro ao rótulo do conjunto de dados (*i.e.*, *counts*) e o último à uma matriz com os elementos de dados de uma coleção desse conjunto. Vale ressaltar que, ao especificar um extrator de dados, o elemento do conjunto de dados que utilizou esse extrator apresenta um valor de atributo adicional para o arquivo com os dados científicos extraídos (*e.g.*, *ext.data*).

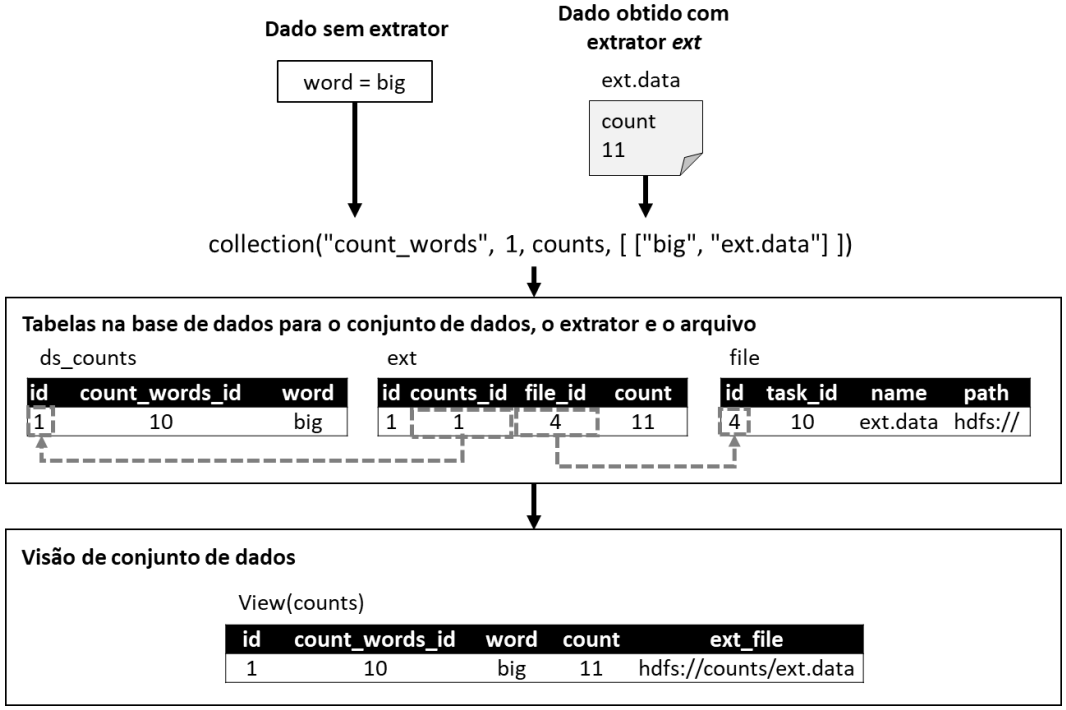


Figura 24. Exemplo de representação de dados de domínio em um conjunto de dados.

Por conseguinte, nós definimos, formalmente, a *visão do conjunto de dados* (Definição 5.2.6), quando extratores de dados científicos podem ser empregados e dados científicos também podem ser capturados sem o uso de extratores. Assim, ao invés de realizar diversas operações de projeção e junção para relacionar dados científicos presentes nas relações do conjunto de dados e dos seus extratores (como no exemplo da Figura 24), o usuário do domínio científico pode, de uma forma mais simples, utilizar apenas a visão do conjunto de dados. Conjuntamente, essas visões respeitam os esquemas das relações do conjunto de dados e dos extratores apresentados nas Definições 5.2.3 e 5.2.4. Mais detalhes sobre a nossa metodologia para a modelagem de consultas são apresentados na Subseção 5.2.3.

Definição 5.2.6. (Visão do Conjunto de Dados) Uma visão de um conjunto de dados s, denotada por $View(s)$, é definida pela junção da relação do conjunto de dados s, considerando os seus

atributos $s.A$, com as relações de dados dos seus extratores $s.\Psi$. De uma forma simplificada, $View(s) = Relation(s) \bowtie_{ds\_id} CombinedExtractors(s.\Psi)$, onde $CombinedExtractors(s.\Psi)$ é uma relação com os elementos de dados capturados pelos extratores $s.\Psi$ e ds_id é o identificador dos elementos no conjunto de dados, correspondendo ao predicado de junção.

5.2.3. Modelagem das consultas

Nas fases anteriores, vimos como uma simulação computacional pode ser representada por um fluxo de dados, assim como os dados de proveniência e do domínio científico podem ser modelados em uma base de dados que segue o nosso modelo de dados lógico. Nesse contexto, os usuários do domínio científico estão frequentemente interessados em realizar análises baseadas apenas na especificação do fluxo de dados (*i.e.*, dados de proveniência prospectiva) ou análises exploratórias dos dados científicos manipulados pela simulação computacional (*i.e.*, dados de proveniência retrospectiva e dados científicos). Como discutido no Capítulo 2, observamos três tipos mais frequentes de análises da segunda categoria, que consideram o acesso:

(i) ao **conteúdo de arquivos** de dados científicos;

(ii) aos **múltiplos arquivos** relacionados pelos programas de simulação; e

(iii) aos **elementos de dados relacionados** por múltiplos arquivos.

Esta fase propõe uma abordagem baseada na virtualização dos dados, a fim de facilitar a modelagem das consultas, integrando os dados pertinentes nessas análises como visões de dados, segundo o nosso modelo de dados lógico. De forma mais específica, essa abordagem introduz um conjunto de definições de visões (RAMAKRISHNAN e GEHRKE, 2003) para apoiar análises baseadas na estrutura do fluxo de dados, nos rastros de proveniência e nos dados científicos capturados de arquivos. Somando-se a isso, a visão em si consiste em uma relação, cujas tuplas não estão explicitamente armazenadas na base de dados, mas podem ser computadas a partir das definições de visões. Nesta subseção, discutimos também como as visões podem ser utilizadas na modelagem de cada tipo de análise mencionado anteriormente.

Para analisar as especificações de fluxo de dados armazenadas na base de dados, três definições de visões foram propostas para consultar o encadeamento das transformações em função dos conjuntos de dados (*dataflows*), os esquemas dos conjuntos

de dados (*datasetSchemas*) e os programas de simulação utilizados por cada transformação de dados (*programs*). A definição da visão *dataflows* realiza uma consulta com a cláusula SELECT aos dados de proveniência prospectiva da nossa base de dados com o intuito de obter as especificações de fluxo de dados, ou seja, as dependências de dados dos fluxos de dados registrados. Como resultado, uma visão, ou uma relação, é apresentada com os rótulos (do termo, em inglês, *tag*) do fluxo de dados, de 2 (duas) transformações de dados (campos *previous_transformation* e *next_transformation*) e de um conjunto de dados (campo *dataset*). Por exemplo, a Figura 25 mostra a visão *dataflows*, assumindo-se que apenas a especificação do fluxo de dados de contagem de palavras (Figura 14) está armazenada na nossa base de dados. Vale enfatizar que, em cenários que a base de dados apresenta mais de uma especificação de fluxo de dados, os usuários precisam selecionar apenas as tuplas correspondentes ao fluxo de dados de interesse, ou seja, com um valor específico para a coluna *dataflow* a partir da visão obtida.

dataflow	previous_transformation	next_transformation	dataset
word_count	null	get_lines	input_files
word_count	get_lines	identify_occurrences	lines
word_count	identify_occurrences	count_words	occurrences
word_count	count_words	null	counts

Figura 25. Visão *dataflows* para a especificação de fluxo de dados da Figura 14.

Além disso, os usuários frequentemente analisam os atributos representados em um determinado conjunto de dados. Com esse propósito, introduzimos a definição de visão *datasetSchemas*, que obtém uma relação com os rótulos dos conjuntos de dados (campo *dataset*), os nomes e os tipos dos atributos manipulados (campos *attribute_name* e *attribute_types*), e os rótulos dos extratores de dados empregados em cada atributo (campo *extractor*), se aplicável. Considerando o *script* para contagem de palavras, a Figura 26 apresenta a visão *datasetSchemas*.

dataset	attribute_name	attribute_type	extractor
input_files	file	file	null
lines	line	text	null
occurrences	word	text	null
occurrences	occurrence	numeric	null
counts	word	text	ext
counts	count	numeric	ext

Figura 26. Visão *datasetSchemas* para o fluxo de dados da Figura 14.

Semelhante à visão *datasetSchemas*, a definição da visão *programs* realiza uma consulta com a cláusula SELECT para obter todos os programas de simulação executados pelas transformações de dados. Mais especificamente, essa visão apresenta os rótulos das transformações de dados (campo *transformation*), acrescidos dos nomes e dos diretórios dos programas de simulação (campos *name* e *path*).

Com a informação de quais transformações de dados foram executadas, quais conjuntos de dados foram manipulados e como as transformações de dados são encadeadas em uma simulação computacional, a especificação do fluxo de dados pode ser representada por um grafo direcionado acíclico. No contexto desta tese, os usuários estão comumente interessados nos conjuntos de dados científicos. Logo, a representação de dados adotada segue a Definição 5.2.7, em que os vértices são os conjuntos de dados e os arcos são as dependências de dados entre os conjuntos em função de uma transformação de dados, que consome dados de um conjunto e produz dados em outro conjunto.

Definição 5.2.7. (Fluxo de Dados na Perspectiva de Grafo) Um fluxo de dados D_F é representado por um grafo direcionado acíclico $G = (S, \Phi')$, em que S são os conjuntos de dados representados como vértices e Φ' são as dependências de dados representadas como arcos. Cada arco $\varphi' \in \Phi'$ é um par ordenado de dois conjuntos de dados (s_1, s_2), correspondendo ao fato de que uma transformação de dados t é responsável pelo consumo do conjunto de dados cabeça s_1 e pela produção do conjunto de dados cauda s_2, de modo que $s_1 \overset{dep}{\longleftarrow} s_2$ em função de t.

Além disso, metadados adicionais podem estar relacionados aos conjuntos e às transformações de dados, como os atributos presentes em cada conjunto de dados (visão *datasetSchemas*) e os programas de simulação executados por cada transformação de dados (visão *programs*). Considerando novamente o *script* de contagem de palavras, mas agora assumindo que as operações nos RDDs (conjunto de dados usando o Spark) de cada transformação de dados foram especificadas, separadamente, em um arquivo com a extensão da linguagem de programação Python, a Figura 27 mostra uma visualização da especificação do seu fluxo de dados na perspectiva de grafos com os metadados adicionais (atributos de cada conjunto e os programas executados por cada transformação).

Somando-se a essas visões de dados de proveniência prospectiva, introduzimos também definições de visões para apoiar os três tipos frequentes de análises exploratórias de dados científicos. Nesse contexto, observamos que essas análises são baseadas em três

conceitos da nossa abstração de fluxo de dados: arquivo, elemento de dados (considerando a extração de dados científicos) e instância da transformação de dados (ou tarefa). No que diz respeito aos arquivos, os usuários estão interessados em saber quais arquivos foram consumidos e produzidos por determinadas instâncias de transformações de dados. Em relação aos conjuntos de dados, os usuários investigam os elementos de dados manipulados pelas instâncias de transformações de dados durante a execução da simulação computacional, sendo que esses elementos podem ser capturados, inclusive, por extratores de dados científicos (Subseção 5.2.2). Portanto, observa-se que os dois primeiros conceitos dependem da definição de instância de uma transformação de dados. Além disso, essas análises podem ser do tipo que acessa o conteúdo de apenas um arquivo de dados científicos (instâncias de apenas uma transformação de dados) ou dos tipos que acessam múltiplos arquivos e elementos de dados relacionados, que envolvem consultas mais complexas para investigar o fluxo de arquivos e de elementos de dados em função de instâncias de múltiplas transformações de dados executadas pela simulação computacional.

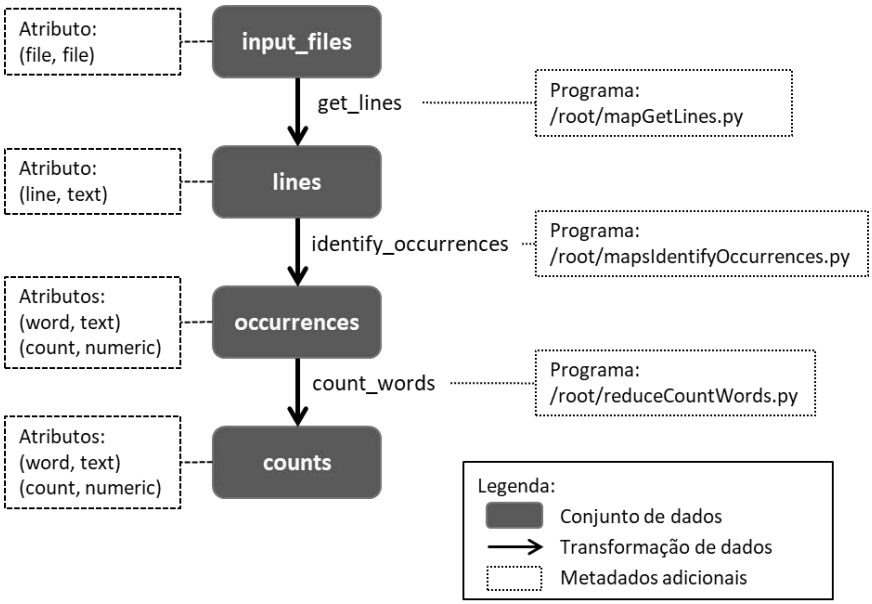


Figura 27. Visualização de uma especificação de fluxo de dados na perspectiva de grafo.

Ao considerar, inicialmente, as análises baseadas em instâncias de apenas uma transformação de dados, uma definiçao de visão foi proposta para consultar os arquivos e os dados científicos produzidos pela simulação computacional, que são representados em um conjunto de dados. Essa visão contém, portanto, tanto os elementos de dados capturados diretamente da simulação computacional (sem extratores), como os dados científicos obtidos pelos extratores. Dessa forma, uma definição de visão é introduzida para cada

conjunto de dados – nomeada com o rótulo do conjunto de dados de interesse –, sendo responsável por processar uma consulta com a cláusula SELECT que relaciona tuplas da relação do conjunto de dados com as tuplas das relações dos extratores utilizados, conforme apresentado na Definição 5.2.6.

Quanto ao esquema dessa visão (ou relação), temos os identificadores do elemento de dados (campo *id*) e das instâncias das transformações de dados envolvidas no consumo ou na produção desse elemento de dados (campos com o nome da transformação de dados, acrescido do sufixo "*_id*"), assim como os atributos de domínio e os arquivos que foram utilizados pelos extratores (como arquivos). No caso em que os atributos de domínio são do tipo arquivo, o valor desse atributo corresponde a combinação do diretório com o nome do arquivo, sendo esta informação obtida pelo processamento de uma consulta na relação *file* a partir do identificador do arquivo presente na relação do conjunto de dados. Por exemplo, a Figura 28 mostra a visão do conjunto de dados *counts* para o problema de contagem de palavras, sendo que o atributo *ext_file* representa o arquivo que teve os seus dados científicos extraídos (ou seja, um atributo específico para cada extrator).

id	count_words_id	word	count	ext_file
1	13	science	2	hdfs://counts/output.bin
2	13	data	3	hdfs://counts/output.bin
3	13	big	1	hdfs://counts/output.bin

Figura 28. Visão *counts* para a execução do *script* da Figura 14.

Em outros cenários, as consultas aos dados científicos podem ser mais complexas, exigindo a análise do fluxo de arquivos e de elementos de dados em função de instâncias de diferentes transformações de dados executadas pela simulação computacional. Assim, para que seja possível realizar esse tipo de análise, os elementos de dados dos conjuntos de interesse precisam estar acessíveis (uso das visões de conjunto de dados), assim como o usuário precisa ter conhecimento do encadeamento dos conjuntos de dados para saber como relacionar os conjuntos de dados relevantes (uso da visão *dataflows*, filtrando apenas as dependências de dados envolvidas no fragmento do fluxo de dados de interesse). Visto que o usuário conheça o encadeamento dos conjuntos e saiba quais conjuntos precisam ter os seus elementos de dados consultados, ele precisa definir quais atributos de junção devem ser utilizados para associar elementos de dados entre dois conjuntos.

Segundo as discussões da Subseção 3.2.4, os rastros de proveniência podem ser de dois níveis, físico e lógico, em que os atributos das instâncias das transformações de dados ou do domínio, respectivamente, são utilizados para associar os elementos de dados presentes nas visões dos conjuntos de dados. De um ponto de vista mais prático, a Figura 29 mostra como os conjuntos de dados podem ser associados utilizando os atributos das tarefas (*e.g.*, campo *get_lines_id* das visões *input_files* e *lines*) ou do domínio (*e.g.*, campo *word* das visões *occurrences* e *counts*) na simulação de contagem de palavras.

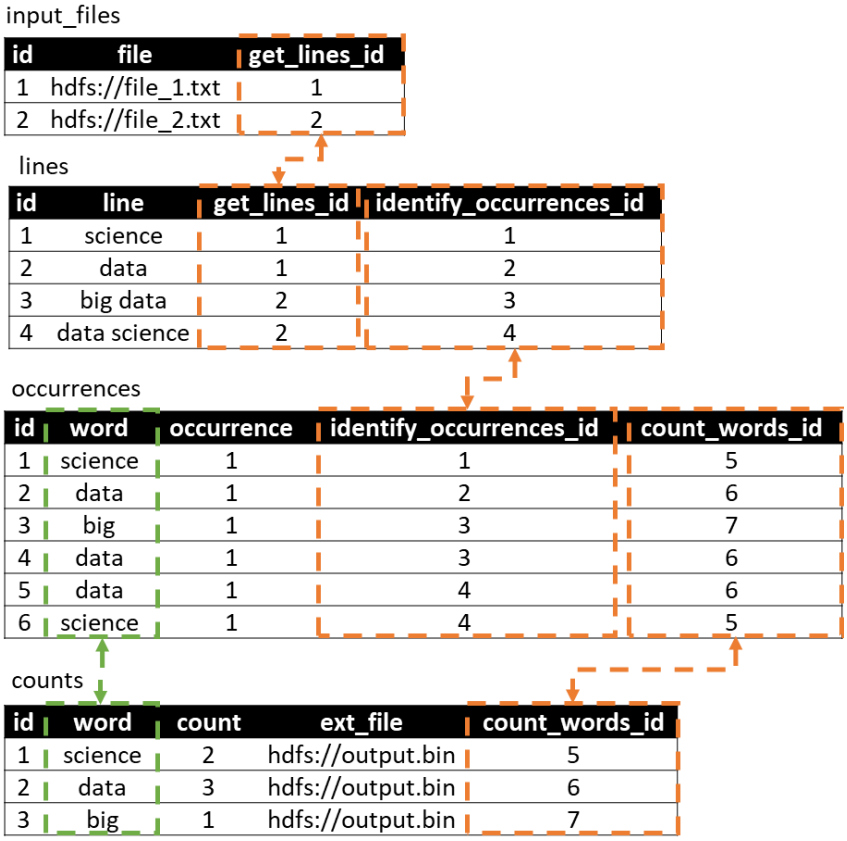


Figura 29. Análise do fluxo de elementos de dados pelos rastros de proveniência nos níveis físico (setas laranjas) e lógico (seta verde).

Além disso, a Figura 30 mostra a consulta na linguagem SQL que retorna os arquivos de entrada (campo *file* na visão *input_files*), as palavras e os seus números totais de ocorrências (campos *word* e *count* na visão *counts*), considerando todos os mapeamentos de atributos apresentados na Figura 29. Vale destacar também que, na nossa metodologia, os arquivos são representados nos conjuntos de dados. Logo, ao apoiarmos a análise do fluxo de elementos de dados – análises de elementos de dados relacionados –, a nossa metodologia também permite a análise do fluxo de arquivos – análises de múltiplos arquivos

–. Como exemplo de atributos do tipo arquivo, temos o campo *file* da visão *input_files*. Do ponto de vista arquitetural, a DfAnalyzer apresenta um componente, conhecido como *Interface de Consulta*, que facilita a geração e o processamento de consultas para a análise do fluxo de elementos de dados e de arquivos, como discutido na Subseção 5.4.

```
SELECT DISTINCT file, counts.word, count
FROM input_files, lines, occurrences, counts
WHERE input_files.get_lines_id = lines.get_lines_id
AND lines.identify_occurrences_id=occurrences.identify_occurrences_id
AND occurrences.count_words_id = counts.count_words_id
AND occurrences.word = counts.word;
```

Figura 30. Consulta para a análise do fluxo de elementos no exemplo de contagem de palavras.

5.3. Operações para a captura e a carga de dados de proveniência e de domínio

Para facilitar as diferentes atividades presentes na nossa metodologia, a DfAnalyzer introduz um conjunto de operações focadas na captura de dados de proveniência prospectiva (*i.e.*, especificação do fluxo de dados) e retrospectiva (*i.e.*, tarefas), na invocação de extratores de dados científicos, e na captura dos dados de domínio produzidos pela simulação computacional. Esta subseção está dividida em três partes com o objetivo de descrever cada categoria dessas operações disponibilizada pela biblioteca de componentes DfAnalyzer.

5.3.1. Operações para a especificação do fluxo de dados

Com o intuito de registrar a especificação do fluxo de dados na base de dados, seguindo o nosso modelo de dados lógico, a DfAnalyzer introduz um conjunto de cinco operações para capturar e armazenar os dados de proveniência prospectiva (Tabela 4). Dessas operações, algumas são mais gerais ao considerar os conceitos de fluxo de dados, como *dataflow*, *dataset* e *transformation*. Por outro lado, existem operações que proveem metadados adicionais, como os programas de simulação executados pelas transformações de dados (*program*) e os extratores utilizados para a captura de dados científicos (*extractor*). Vale destacar que o termo extrator foi utilizado para descrever, de uma forma mais geral, o processo de captura de dados científicos, considerando o acesso, a extração e, eventualmente, a indexação de dados científicos presentes em arquivos.

Como primeiro passo, a operação *dataflow* registra o fluxo de dados, recebendo um rótulo ou nome único para o fluxo de dados representativo da simulação computacional. No exemplo do *script* para contagem de palavras (Figura 14), o rótulo do fluxo de dados é *word_count*. Depois disso, as outras operações podem ser realizadas, evidenciando uma hierarquia para a estrutura desse fluxo de dados. Embora não tenhamos uma operação para os atributos presentes nos conjuntos de dados, a especificação dos atributos está presente na operação *dataset* por meio dos parâmetros *attributes* (nome dos atributos) e *attributesTypes* (tipos de dados armazenados nos atributos, de acordo com a ordem de *attributes*).

Tabela 4. Operações para a especificação do fluxo de dados.

Operação
dataflow(*tag***)**
program(*name, path***)**
dataset(*dataflow, tag, attributes, attributesTypes***)**
extractor(*dataset, tag, method, cartridge, attributes, attributesTypes***)**
transformation(*dataflow, tag, inputDatasets, outputDatasets, programs***)**

Da mesma forma, as dependências de dados entre transformações não são especificadas como operações. Entretanto, discutimos na Seção 3.1 que duas transformações podem apresentar dependências de dados entre si em função de um conjunto de dados, quando elementos de dados desse conjunto são produzidos por uma transformação e consumidos por outra transformação. Assim, por meio da composição de transformações de dados (uso da operação *transformation*), as dependências de dados podem ser inferidas de acordo com os conjuntos de dados manipulados pelas transformações. Segundo o exemplo da Figura 31, podemos afirmar que $t_1 \overset{dep}{\longleftarrow} t_2$, consequentemente, existe uma dependência de dados entre t_1 e t_2 em relação ao conjunto de dados s_2. Além disso, como a dependência de dados contempla propriedades de transitividade, se $\{t_1 \overset{dep}{\longleftarrow} t_2\}$, então $\{s_1 \overset{dep}{\longleftarrow} s_2, s_2 \overset{dep}{\longleftarrow} s_3\} \vDash s_1 \overset{dep}{\longleftarrow} s_3$.

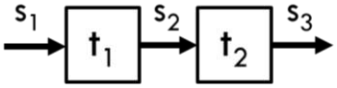


Figura 31. Exemplo de dependência de dados: t_2 depende de t_1.

Como cada relação no modelo de dados inicial – relações para os dados de proveniência prospectiva e as relações *task* e *file* – corresponde à um conceito da abstração de fluxo de dados, a execução de uma operação de proveniência prospectiva (Tabela 4) implica em adicionar tuplas em uma ou mais dessas relações. Além disso, as operações de especificação do fluxo de dados executam consultas de criação de outras relações para armazenar os dados de proveniência retrospectiva e de domínio (*i.e.*, relações dos conjuntos de dados e dos extratores de dados científicos).

No caso das operações *dataflow* e *program*, elas realizam apenas inserções de tuplas nas relações com os seus respectivos nomes. Por outro lado, as operações *dataset*, *extractor* e *transformation* apresentam comportamentos mais complexos. A operação *dataset* registra um conjunto de dados (inserção de tuplas na relação *dataset*) para depois adicionar os atributos presentes nesse conjunto de dados (inserção de tuplas na relação *attribute*), a fim de garantir a integridade referencial no campo *ds_id* da relação *attribute* (Figura 17). Mais especificamente, por meio dos parâmetros *attributes* e *attributesTypes*, operações de inserção de tuplas são realizadas na relação *attribute* para registrar os nomes dos atributos e os seus tipos em um determinado conjunto de dados. Somando-se a isso, a operação *dataset* cria uma relação para o conjunto de dados.

Supondo uma operação *dataset*(‘occurrences’,{word, occurrence},{text,numeric}), essa operação terá que adicionar uma tupla na relação *dataset* com o rótulo (campo *tag*) ‘occurrences’ e duas tuplas na relação *attribute* com os seguintes pares de valores para os campos *name* e *type*: (word, text) e (occurrence, numeric). Ademais, essa operação criará uma relação de dados, nomeada como *ds_occurrences*, para armazenar os elementos de dados capturados nesse conjunto de dados durante a execução da simulação (*i.e.*, dados de domínio). Esse exemplo foi baseado no *script* em Python usando Spark da Figura 14 e o seu modelo de dados para os dados de domínio é mostrado na Figura 22. Ressalta-se também que, nesse cenário, os valores desses campos (*i.e.*, valores de atributos na abstração de fluxo de dados) não são característicos do uso de extratores de dados científicos.

Para a captura de dados científicos, a DfAnalyzer provê outra operação, denominada *extractor*, que especifica quais programas *ad-hoc* ou ferramentas alternativas serão executados para acessar, extrair e, eventualmente, indexar dados científicos presentes em determinados arquivos. Com esse objetivo, ao ser invocada, essa operação insere uma tupla

na relação *extractor* para registrar esse novo extrator. Como os dados capturados por um extrator devem ser representados em um conjunto de dados, a integridade referencial em função da restrição de chave estrangeira no campo *ds_id* precisa ser respeitada. Do mesmo modo, a operação *extractor* registra os atributos capturados pelo extrator ao utilizar os parâmetros *attributes e attributesTypes*, inserindo tuplas na relação *attribute*. Somando-se a isso, para armazenar os dados científicos capturados em tempo de execução, a operação *extractor* cria uma relação com o rótulo do extrator, sendo a especificação do seu esquema apresentada formalmente na Definição 5.2.4.

Entretanto, a maior complexidade da operação *extractor* está no fato dela ser chamada por uma segunda vez (ou posteriormente) para um mesmo conjunto de dados, pois os elementos de dados produzidos pelos dois extratores precisam ser relacionados entre si para representar o conjunto de dados de interesse. Baseado no conceito de junção de dados da abordagem relacional (ELMASRI e NAVATHE, 2010), a nossa metodologia assume que os campos com o mesmo nome e tipo obtidos pelos extratores podem ser associados automaticamente pelo operador relacional de junção (Definição 5.2.5). No modelo de dados lógico proposto, a especificação dessa junção é armazenada na relação *extractor_combination*, que apresenta o identificador do conjunto de dados (*ds_id*), os identificadores dos extratores (*outer_ext_id* e *inner_ext_id*), os atributos de junção (*keys*) e os tipos dos atributos de junção (*key_types*).

Por último, a operação *transformation* exige que o fluxo de dados, os conjuntos de dados, e os programas de simulação já tenham sido especificados (ou seja, já tenham sido registrados a partir da operação *program*), uma vez que os seus nomes (ou rótulos) são passados como parâmetros. Vale destacar que esses nomes e rótulos são únicos na nossa metodologia. Em função da composição das transformações de dados, a operação *transformation* também apresenta um comportamento específico ao ser invocada pela segunda vez (ou posteriormente) manipulando um mesmo conjunto de dados de uma transformação de dados anterior. Esse comportamento contempla a identificação das dependências de dados entre as transformações e o registro dessas dependências de dados na relação *data_dependency*, como discutido anteriormente.

Considerando o fluxo de dados especificado para o *script* da Figura 14, as operações para o armazenamento dos dados de proveniência prospectiva são mostradas na Figura 32.

Essas operações realizam consultas de inserção para adicionar tuplas em relações que representam conceitos da abstração de fluxo de dados e de criação de relações (apenas para as operações *dataset* e *extractor*), para que os dados de proveniência retrospectiva e os dados de domínio sejam, futuramente, armazenados durante a execução da simulação computacional. Na Figura 32 mostramos também quais relações são utilizadas ou criadas para algumas operações no modelo de dados lógico. Somando-se a isso, o *script* original desse exemplo (Figura 14) não apresenta linhas de código para realizar a captura de dados científicos. Por essa razão, esse importante recurso foi adicionado pela DfAnalyzer, sendo discutido em detalhes na Subseção 5.3.2.

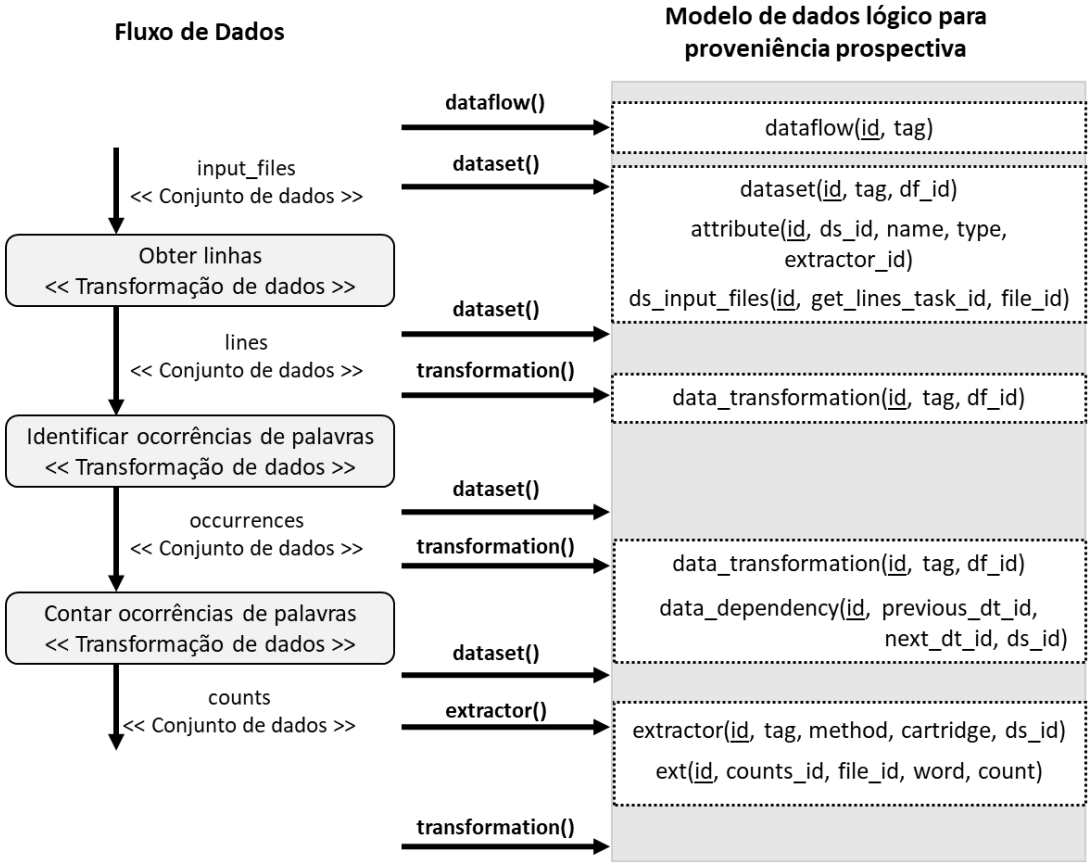


Figura 32. Mapeamento do fluxo de dados para o modelo de dados lógico.

5.3.2. Operação para a captura de dados científicos

Em virtude da necessidade de análises de dados científicos presentes em arquivos, o especialista em ciência da computação, em um trabalho de colaboração com o usuário do domínio científico, frequentemente define extratores de dados científicos na especificação do fluxo de dados. Desse modo, antes de capturar e armazenar os dados de proveniência

retrospectiva e de domínio na nossa base de dados, os dados científicos de interesse precisam ser efetivamente extraídos. Para isso, esses usuários precisam realizar ajustes no código fonte dos seus programas de simulação (*e.g.*, *script* em Python usando Spark) para invocar esses extratores de dados científicos.

Do ponto de vista computacional, um extrator de dados científicos é responsável por acessar o conteúdo presente em arquivos, realizar as análises léxica e sintática desse conteúdo para identificar apenas os dados relevantes, selecionar os conjuntos de dados científicos de interesse, indexar os dados, quando necessário, e prepará-los para serem armazenados na base de dados (Seção 4.3). Assim, pela invocação de um extrator, dados ainda alocados em memória, que necessitam da invocação de tecnologias para o processamento de dados científicos *in situ*, ou armazenados em arquivos, que necessitam da invocação de programas *ad-hoc* ou de ferramentas alternativas, podem ser capturados e armazenados na relação desse extrator (Definição 5.2.4).

Como mencionado, esse processo de captura de dados científicos envolve diferentes etapas. As duas abordagens comumente utilizadas – extração e indexação de dados – compartilham do mesmo comportamento na maioria dessas etapas. Assim, esta tese procurou formalizar o comportamento de cada etapa, inspirando-se em operações da álgebra relacional, para, depois, descrever uma operação genérica para a captura de dados científicos. A principal vantagem de associar essas etapas às operações da álgebra relacional consiste no fato dos resultados, vistos como conjuntos de dados na abstração de fluxo de dados, serem representados como uma relação. Consequentemente, os dados científicos capturados já estariam prontos para serem carregados nas relações dos extratores, segundo o nosso modelo de dados lógico (*e.g.*, relação *ext* da Figura 19). As etapas a serem descritas formalmente são as seguintes:

(1) Acesso aos dados científicos;
(2) Projeção de dados científicos;
(3) Seleção de dados científicos de interesse; e
(4) Indexação de dados científicos.

Com o objetivo de obter os dados científicos presentes em um arquivo manipulado por uma instância de uma transformação de dados, o *acesso aos dados científicos* é caracterizado por duas análises, conhecidas como léxica e sintática, como discutido na

Seção 4.2. Em soluções para a análise de dados científicos, o alfabeto para a análise léxica está normalmente presente em um *catálogo de dados científicos*, que consiste em um conjunto de metadados necessários para a compreensão do alfabeto adotado em cada formato de arquivo ou mesmo em estruturas de dados em memória. A partir dos símbolos gerados na análise léxica, a análise sintática transforma a sequência de caracteres de entrada em uma estrutura de dados conhecida (*e.g.*, número em ponto flutuante). Semelhante à análise léxica, a análise sintática também precisa ter acesso ao catálogo de dados científicos, uma vez que ele contém metadados sobre os atributos e as suas estruturas de dados. Assumimos também que um arquivo pode conter diversos elementos de dados. Assim, inspirados em uma abordagem relacional, esses elementos de dados seriam equivalentes a tuplas em uma relação. Com o intuito de formalizar essa etapa, esta tese define o *acesso aos dados científicos*.

Definição 5.2.8. (Acesso aos Dados Científico) O acesso aos dados científicos em um arquivo φ é uma operação que identifica os elementos de dados e as estruturas de dados dos seus atributos por meio das análises léxica e sintática, segundo metadados presentes em um catálogo de dados científicos ς. Em uma abordagem relacional, cada elemento de dados pode ser representado por uma tupla em uma relação de dados r_{access}. Assim, o acesso aos dados científicos presentes em φ, de acordo com ς, é expresso por $r_{access} \leftarrow Access(\varphi, \varsigma)$.

Em muitos casos, os usuários do domínio científico estão interessados apenas em determinados atributos de interesse. Logo, eles não exigem que todos os atributos presentes nos elementos de dados de um arquivo sejam capturados. Esse comportamento é característico da etapa de *projeção de dados científicos*, sendo descrito por uma fragmentação vertical da relação de dados acessados. Nesta tese definimos formalmente a *projeção de dados científicos*, inspirada no operador relacional de projeção.

Definição 5.2.9. (Projeção de Dados Científicos) A projeção de dados científicos é uma operação de fragmentação vertical da relação de dados acessados r_{access} de um arquivo para obter apenas um subconjunto de atributos A', tal que $A' \subset Schema(r_{access})$. Após essa operação, uma relação $r_{projection}$ é produzida, de forma que $r_{projection} \leftarrow \pi_{A'}(r_{access})$.

Após a captura de apenas alguns atributos de interesse, os usuários do domínio científico definem, em diversas circunstâncias, condições para obterem apenas elementos de dados em regiões de interesse. Logo, eles comumente desejam obter um subconjunto de

todos os elementos presentes em um arquivo, sendo esta etapa definida como *seleção de dados científicos*. Esta tese define formalmente a *seleção de dados científicos*, inspirando-se no operador relacional de seleção.

Definição 5.2.10. (Seleção de Dados Científicos) A seleção de dados científicos é definida pela fragmentação horizontal da relação de dados projetados $r_{projection}$ que produz uma relação $r_{selection}$ apenas com os elementos de dados que atendem a um conjunto de condições de seleção C, de forma que $r_{selection} \leftarrow \sigma_{\mathrm{C}}(r_{projection})$.

Além dessas três etapas, quando as relações de dados apresentam grandes volumes de dados ou determinados valores de atributos precisam representar estruturas de dados mais complexas, técnicas de indexação de dados científicos são empregadas para reduzir o espaço de armazenamento da relação de dados e para prover um acesso eficiente aos dados científicos durante o processamento de consultas. Dessa forma, a etapa de *indexação de dados científicos* (Definição 5.2.11) cria índices para determinados elementos de dados ou tuplas, definindo quais atributos devem ser favorecidos por essa técnica.

Definição 5.2.11. (Indexação de Dados Científicos) A indexação de dados científicos é definida pela execução de uma função f para gerar índices a partir de elementos de dados de uma relação $r_{selection}$ obtidos do arquivo φ. A função f pode apresentar argumentos adicionais que definem especificidades da técnica de indexação aplicada, como o argumento Å que especifica quais atributos devem ser indexados. Como resultado, essa etapa produz uma relação de dados $r_{indexing}$, de forma que $r_{indexing} \leftarrow f(r_{selection}, \mathrm{Å})$.

Embora as etapas para a captura de dados científicos tenham sido apresentadas de uma forma encadeada, apenas a etapa de acesso aos dados científicos é obrigatória. Logo, as outras etapas são vistas como opcionais e podem ter a sua ordem de execução alterada de acordo com o programa de captura de dados científicos. Por uma questão de desempenho, adotamos as mesmas prioridades empregadas pela álgebra relacional para a execução das etapas descritas anteriormente. A partir dessas definições, a DfAnalyzer introduz a operação *extract* (traduzida como *extrair*) para permitir a captura de dados científicos baseadas na invocação de programas *ad-hoc* e de ferramentas alternativas. Embora o seu nome induza a pensar em uma abordagem de extração de dados científicos, na verdade, essa operação permite que ambas as abordagens sejam empregadas, ou seja, tanto a extração, como a indexação de dados científicos.

Em outra perspectiva, essa operação é responsável por invocar um programa *ad-hoc* ρ que realiza o acesso aos dados científicos presentes em um arquivo φ de acordo com um catálogo de dados ς, a projeção dos dados científicos para os atributos de interesse A' e a seleção de valores de atributos segundo um conjunto de condições de seleção C, indexando valores da sequência de atributos $A_{indexed} \mid A_{indexed} \subseteq A'$. Como resultado, a operação *extract* produz uma relação de saída r, em que o seu esquema apresenta um atributo que contém o caminho para os arquivos, representado por a_{φ}, os atributos extraídos desses arquivos ($A_{extracted} \mid A_{extracted} = A' - A_{indexed}$), e os atributos indexados ($A_{indexed}$), isto é, a relação r apresenta o esquema $\mathcal{R} = \{a_{\varphi}, A'\}$, ou $Schema(r) = \{a_{\varphi}, A'\}$. De uma forma geral, $r \leftarrow f(\sigma_{\mathsf{C}}(\pi_{a_{\varphi},A'}(Access(\varphi, \varsigma))), A_{indexed})$ a partir da execução de ρ. No caso da captura de dados ainda alocados em memória, o esquema da relação de saída não possui o atributo φ em função da sua volatilidade.

Formalmente, a operação *extract* é especificada conforme a expressão abaixo. Vale ressaltar que, pelo fato da invocação do programa ρ já conhecer qual catálogo de dados (ς) será necessário na extração de dados científicos, quais condições de seleção serão aplicadas (C) e quais atributos precisam ser extraídos ou indexados, esses argumentos não estão presentes na especificação da operação *extract*. Da mesma maneira, por conhecer o esquema da relação de saída, a DfAnalyzer já conhece os atributos a serem capturados (A).

$$r \leftarrow extract(\rho, \varphi) \tag{5}$$

Assumindo-se novamente o *script* em Python usando Spark para a contagem de palavras (Figura 14), podemos observar que os seus resultados são armazenados em arquivos binários, *e.g.*, *output.bin*. Supondo que os usuários do domínio científico desejem analisar determinados valores de atributos presentes nesses arquivos, como a palavra (atributo *word*) e o seu número de ocorrências em todos os arquivos de entrada (atributo *count*), a nossa abordagem utiliza, portanto, a operação *extract* para capturar esses dados científicos de interesse. A Figura 33 apresenta esse exemplo com a especificação dessa operação, em que o programa *ad-hoc extractRDF* extrai elementos de dados de um arquivo no formato binário para os atributos de interesse (*word* e *count*), quando o número de ocorrências de uma palavra é maior que 10 (condição de seleção *count* > 10).

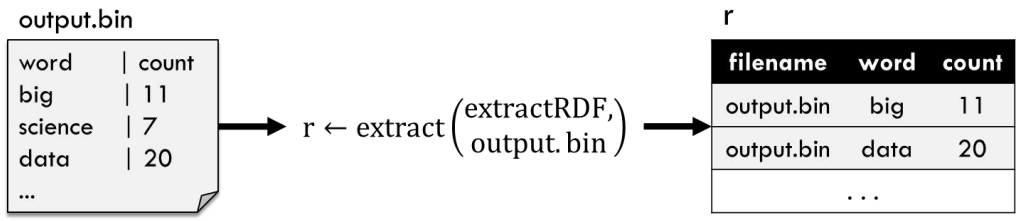


Figura 33. Operação de extração de dados científicos.

Considerando o mesmo exemplo apresentado acima, o usuário pode decidir por indexar os valores dos atributos de interesse, como mostrado na Figura 34. Nesse caso, assume-me que a captura de dados científicos considera a indexação de dados, isto é, a etapa (4). No exemplo da Figura 34, um programa *indexRDF* é utilizado para gerar um arquivo de índices para cada atributo (*i.e.*, *WORD* e *COUNT*), evitando a carga direta dos valores dos atributos na base de dados futuramente. De forma análoga à abordagem de extração de dados, esse extrator com a indexação de dados científico também permite que apenas os elementos de dados relevantes, segundo um conjunto de condições $\mathbb{C} = \{count > 10\}$, sejam indexados. Sendo assim, vale ressaltar que o uso de um programa de indexação também pode gerar índices para apenas alguns atributos presentes no arquivo de dados científicos.

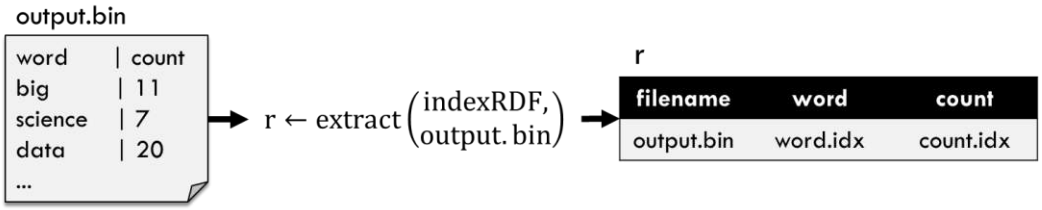


Figura 34. Operação de indexação de dados científicos.

Em cenários reais para apoiar a captura de dados científicos, o usuário do domínio científico precisa realizar ajustes nos seus programas de simulação com a ajuda do especialista em ciência da computação, a fim de que as invocações da operação *extract* sejam adicionadas. No exemplo de contagem de palavras em arquivos, o *script* em Python não permite, originalmente, a captura de dados científicos. Por isso, a inserção de apenas uma linha de código foi realizada para apoiar a extração de dados científicos, como mostrado na Figura 23. Esse exemplo corresponde à operação de captura de dados científicos realizada pelo extrator *ext* da Figura 32, sendo os seus dados armazenados na relação do extrator *ext* de acordo com o nosso modelo de dados lógico.

5.3.3. Operações para a captura de dados de proveniência retrospectiva e de domínio

Com a especificação do fluxo de dados no nosso modelo de dados lógico e os ajustes realizados nos programas de simulação para a captura de dados científicos, os usuários possuem uma visão geral das transformações de dados e dos conjuntos de dados manipulados. Ademais, eles têm conhecimento de quais arquivos são investigados e como os dados científicos foram capturados. Entretanto, a DfAnalyzer ainda não armazena os dados de proveniência retrospectiva e de domínio na nossa base de dados durante a execução da simulação computacional. Para atender a essa questão, a DfAnalyzer introduz outras quatro operações, como mostrado na Tabela 5, que devem ser invocadas em tempo de execução de acordo com ajustes realizados nos programas de simulação pelo usuário do domínio científico em colaboração com o especialista em ciência da computação.

Tabela 5. Operações para a captura de dados de proveniência retrospectiva.

Operação
task(*dataflow*, *transformation*, *iteration*, *status* [, *workspace*, *resource*]**)**
file(*dataflow*, *transformation*, *iteration*, *name*, *path***)**
collection(*dataflow*, *transformation*, *iteration*, *dataset*, *elements***)**
dependency(*dataflow*, *transformation*, *iteration*, *dependentTransformations*, *dependentTasks***)**

Para essas operações, adotamos uma granularidade fina na gerência da proveniência retrospectiva, a fim de permitir, futuramente, a análise dos rastros de proveniência no nível do fluxo de arquivos e dos elementos de dados (Subseção 3.2.4). Ao mesmo tempo, a granularidade dos dados de proveniência está associada aos ajustes realizados nos programas de simulação, fazendo com que a nossa metodologia permita diferentes níveis de granularidade. Diferentemente de outras soluções, *e.g.*, noWorkflow e RDataTracker, a DfAnalyzer não captura os dados de proveniência necessariamente na granularidade mais fina e oferece uma solução agnóstica a uma linguagem de programação. Apesar de informações valiosas, observa-se que a captura em uma granularidade muita fina compromete as análises de dados científicos, tornando complexa, inclusive, a especificação das consultas de interesse (especialmente quando uma grande quantidade de atributos ou dados científicos não são relevantes nas análises). Logo, a nossa abordagem visa oferecer autonomia aos usuários, ao permitir que eles sejam capazes de definir quais dados

científicos são relevantes nas suas análises. Ou seja, eles são capazes de especificar quais atributos devem ser monitorados em cada conjunto de dados.

Nessa fase, a instância de uma transformação de dados (Seção 3.1) ou tarefa apresenta a menor unidade de dados necessária, isto é, menor subconjunto de elementos de dados de entrada (ou coleção de dados), para executar uma transformação de dados. Esses elementos de dados correspondem aos dados científicos manipulados pelos diferentes modelos computacionais que os usuários do domínio científico desejam comumente analisar. Assim, a nossa principal operação é definida como *task* com o objetivo de armazenar os metadados associados à execução de uma instância de uma transformação de dados. Enquanto isso, as outras operações são dependentes de uma tarefa, evidenciando que uma operação diferente de *task* (*e.g.*, *file*) só pode ser executada após a operação *task* da sua tarefa dependente ter finalizado (especificada por alguns argumentos). Em mais detalhes, a operação *task* está associada a um fluxo de dados (argumento *dataflow*, que corresponde ao rótulo do fluxo de dados), uma transformação de dados (argumento *transformation*, que corresponde ao rótulo da transformação de dados) e uma iteração (argumento *iter*).

No cenário de análise exploratória de dados científicos, a iteração corresponde a uma combinação dos parâmetros de entrada (ou dos elementos de dados) para validar uma hipótese científica. Em cenários reais, uma simulação computacional pode apresentar diversas iterações, *e.g.*, varredura de parâmetros (RAICU *et al.*, 2008), de forma que inúmeras invocações da operação *task* são realizadas com diferentes valores para o argumento *iter* (para uma mesma transformação de dados). Como resultado da operação *task*, o usuário pode monitorar o comportamento de cada iteração em uma transformação de dados ao consultar as tuplas armazenadas na relação *task* na nossa base de dados.

Ademais, a operação *task* permite que o estado da execução da tarefa (argumento *status*) seja especificado como executando (*RUNNING*), finalizado (*FINISHED*) ou finalizado com erro (*FINISHED_WITH_ERRORS*). Como argumentos opcionais, os usuários também podem informar o diretório de execução da tarefa (argumento *workspace*) e o recurso computacional responsável pela tarefa (argumento *resource*). Em uma perspectiva geral, portanto, a operação *task* corresponde ao processamento de uma consulta de inserção de tuplas na relação *task* da nossa base de dados.

Na Figura 35, mostramos um exemplo de tarefas executadas para as duas primeiras transformações de dados do *script* em Python usando Spark da Figura 14. Nesse exemplo, apenas uma iteração da simulação computacional foi apresentada, consumindo o arquivo *file_1.txt* como elemento de dados de entrada. Para essa iteração, duas tarefas foram executadas (uma para cada transformação de dados) e, consequentemente, duas invocações da operação *task* foram realizadas. Considerando-se os dados de entrada do conjunto de dados *input_files* e o comportamento de cada transformação, podemos afirmar que essas duas transformações de dados teriam, em uma representação completa da execução dessa simulação, duas iterações com quatro tarefas executadas (duas tarefas para cada iteração).

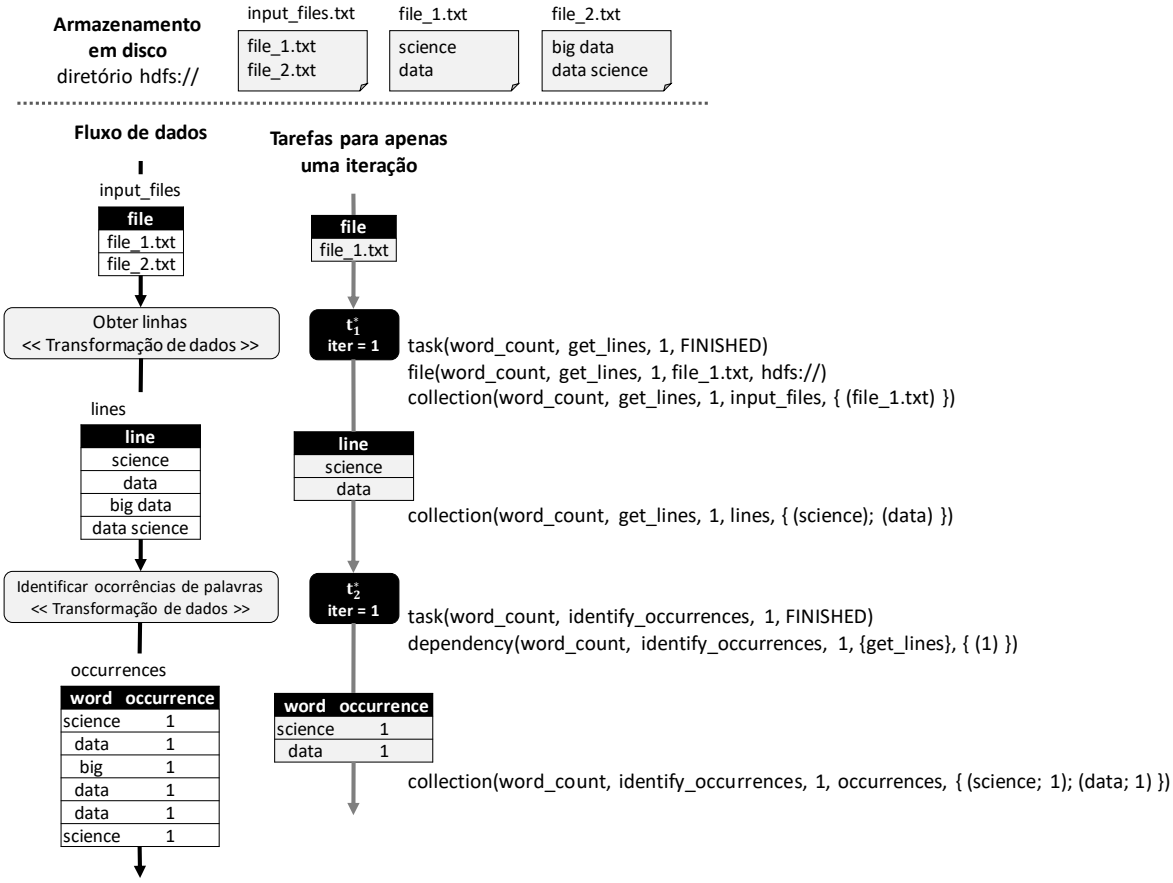


Figura 35. Exemplo de tarefas a partir da execução do *script* em Python da Figura 14.

Outro aspecto importante consiste no monitoramento dos arquivos consumidos e produzidos pelas tarefas. Para esse recurso, a nossa abordagem apresenta a operação *file*, que registra os arquivos manipulados por uma tarefa (argumento *task*) com o seu nome (argumento *name*) e o diretório em que esse arquivo está armazenado (argumento *path*). Para garantir a integridade referencial em função da restrição de chave estrangeira do

atributo *task_id* na relação *file*, a nossa abordagem consulta a relação *task* para identificar a tarefa informada como argumento, que apresenta os rótulos do fluxo de dados e da transformação de dados, assim como a iteração correspondente à tarefa. Caso exista essa tarefa e esse arquivo ainda não exista na relação *file*, então a operação *file* realiza uma consulta de inserção de tuplas na relação *file* para armazenar o arquivo manipulado pela tarefa. Considerando novamente o exemplo da Figura 35, observamos apenas uma invocação da operação *file*, para o arquivo *file_1.txt*. Em uma representação completa da execução dessa simulação computacional para as duas transformações de dados, a operação *file* seria invocada duas vezes, para os arquivos *file_1.txt* e *file_2.txt*.

Todavia, em cenários reais, os usuários do domínio científico estão mais interessados nos dados de domínio para validarem ou refutarem as suas hipóteses científicas. Considerando os extratores de dados e os conjuntos de dados manipulados por cada transformação, a DfAnalyzer introduz uma operação, definida como *collection*, para registrar a coleção de dados (ou os elementos de dados) manipulados por uma tarefa. Como argumentos, essa operação exige a identificação da tarefa (argumentos *dataflow*, *transformation* e *iteration*) e do conjunto de dados (*dataset*), além dos elementos de dados que fazem parte dessa coleção de dados (*elements*).

Conforme definido na Seção 3.1, uma coleção de dados $c = \{e \mid e \in E\}$ é composta de um conjunto de elementos de dados E. Cada elemento de dados $e = (v \mid v \in V)$ é composto de uma sequência de valores V, sendo que a ordem dos seus valores está de acordo com a ordem de especificação dos atributos A no conjunto de dados s ao qual esse elemento de dados pertence, *i.e.*, $e \subseteq s.C$ e $card(e.V) = card(s.A)$. Considerando-se essa definição de coleção de dados, elementos de dados e os seus valores, assumimos que os elementos de dados do argumento *elements* são especificados como uma matriz, em que cada linha dessa matriz corresponde a um elemento de dados e cada célula dessa matriz corresponde ao valor de um atributo para um determinado elemento de dados.

Ao mesmo tempo, essa representação matricial possui semelhanças com a abordagem relacional, pois cada linha da matriz pode ser associada a uma tupla de uma relação de dados, enquanto que cada coluna da matriz corresponde a um atributo ou coluna da relação. Consequentemente, essa estrutura matricial favorece a carga dos dados de domínio em uma base de dados relacional. Tirando proveito dessa característica, a operação

collection realiza consultas na nossa base de dados com o intuito de inserir tuplas (ou elementos de dados) na relação do conjunto de dados (argumento *dataset*). A Figura 35 mostra invocações da operação *collection* para registrar as coleções de dados manipuladas por duas tarefas em um *script* em Python para a contagem de palavras em um arquivo, como $collection(word\_count, get\_lines, 1, input\_files, \{(file\_1.txt)\})$ que registra o elemento de dados do conjunto *input_files* (com o valor *file_1.txt* para o atributo *FILE*) a partir da primeira iteração da transformação de dados *get_lines* do fluxo de dados *word_count*. Nesse exemplo, as invocações dessa operação só consideram dados científicos obtidos diretamente de estruturas de dados alocadas em memória pelo *script*, sem o uso de extratores.

Entretanto, em muitos cenários, os usuários do domínio científico utilizam extratores para acessar, extrair e, eventualmente, indexar dados científicos presentes em arquivos, conforme discutido na Subseção 5.2.2. Nesse contexto, os dados capturados devem ser armazenados em suas relações de acordo com o nosso modelo de dados lógico. No *script* em Python da Figura 14, apresentamos um extrator *ext* para o conjunto de dados *counts*, em que a carga dos dados científicos capturados é realizada pelo operador *collection*. Para isso, nos casos em que alguns atributos são obtidos usando extratores de dados científicos, o argumento *elements* da operação *collection* apresenta os elementos de dados com seus valores de atributos sem e com o uso de extratores.

Quando os valores dos atributos são obtidos pelo uso de um extrator, assume-se que os dados capturados pela operação *extract* são armazenados em arquivos e o caminho para esse arquivo é especificado para o valor dos atributos extraídos. Logo, um elemento de dados do conjunto de dados s na operação *collection*, utilizando os extratores de dados científicos Ψ, deve apresentar valores para os atributos obtidos sem extratores ($s.A$), acrescido de um valor de atributo para cada extrator de dados científicos $\psi_i \in \Psi$, em que esse valor corresponde ao diretório para o arquivo com os dados extraídos por ψ_i. Dessa maneira, a operação *collection* é capaz de armazenar os dados de domínio acessados diretamente nos programas de simulação ou capturados por extratores de dados científicos.

Na Figura 24 mostramos um exemplo de operação para registrar uma coleção de dados do conjunto *counts* produzida por uma tarefa *task* da transformação de dados *count_words*. Nesse caso, o valor do atributo *word* foi obtido diretamente do programa de

simulação, enquanto o valor do atributo *count* foi capturado de um arquivo usando o extrator de dados *ext*. Após a execução do extrator, o dado extraído foi armazenado no arquivo *ext.data*. Portanto, ao especificar a operação *collection*, apenas um elemento de dados foi definido no argumento *elements* com os valores *big* (dado obtido sem extrator) e *ext.data* (diretório para o arquivo com o dado extraído).

Na fase de especificação do fluxo de dados no nosso modelo de dados lógico, discutimos também como as dependências de dados entre transformações foram inferidas e, consequentemente, representadas na nossa base de dados (relação *data_dependency*). De uma forma simplicada, as dependências de dados indicam quais transformações de dados precisam ser executadas, para que outras transformações possam iniciar. Consequentemente, na perspectiva de execução da simulação computacional, uma instância de uma transformação de dados (ou tarefa) não tem conhecimento, a priori, de quais são as instâncias das suas transformações dependentes, ou seja, quais instâncias de uma ou mais transformações de dados precisam ser executadas antes.

Diante dessa questão, a nossa abordagem introduz a operação *dependency* para registrar as dependências de dados entre as instâncias de transformações de dados. Para a sua devida especificação, a operação *dependency* apresenta uma invocação baseada em cinco argumentos, em que os três primeiros contemplam a tarefa atual (argumento *dataflow*, *transformation* e *iteration*), ou seja, a tarefa cujos metadados estão sendo armazenados; o quarto corresponde a uma lista com os rótulos das transformações de dados dependentes da tarefa atual (argumento *transformations*); e o último é uma matriz com os identificadores das iterações das tarefas dependentes (de acordo com a ordem dos elementos na lista *transformations*). Em uma perspectiva formal, essa especificação de dependência de dados permite a gerência dos rastros de proveniência no nível físico, como discutido na Subseção 3.2.4.

Como exemplo, a Figura 36 mostra cinco casos de dependências de dados entre tarefas de até três transformações de dados, dt_1, dt_2 e dt_3. A nomenclatura adotada para uma tarefa t_j^i contempla o identificador i da transformação dt_i, como sobrescrito, e a iteração j, como subscrito. No caso da Figura 36(a), existe apenas uma dependência de dados entre as tarefas t_1^1 e t_1^2 (de modo que $t_1^1 \overset{dep}{\longleftarrow} t_1^2$) e, consequentemente, uma invocação da operação *dependency*: dependency$(t_1^2, \{dt_1\}, \{(1)\})$. Por outro lado, a Figura 36(b) mostra

um caso em que a tarefa t_1^3 da transformação de dados dt_3 depende de dados produzidos pelas tarefas t_1^1 e t_1^2 das transformações dt_1 e dt_2, respectivamente (de modo que $t_1^1 \overset{dep}{\longleftarrow} t_1^3$ e $t_1^2 \overset{dep}{\longleftarrow} t_1^3$). Nessa circunstância, apenas uma invocação da operação *dependency* é necessária: dependency$(t_1^3, \{dt_1; dt_2\}, \{(1,1)\})$.

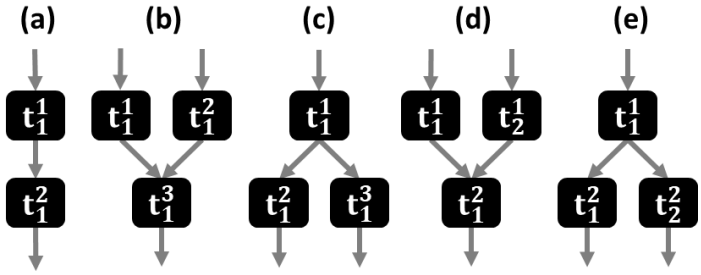


Figura 36. Dependência de dados entre tarefas.

Assim como nos dois primeiros casos, a Figura 36(c) apresenta um exemplo de dependência de dados com apenas uma iteração, em que a tarefa t_1^2 da transformação de dados dt_2 depende de dados produzidos pela tarefa t_1^1, da mesma maneira que a tarefa t_1^3 da transformação dt_3 depende de dados produzidos pela tarefa t_1^1. Logo, duas invocações da operação *dependency* devem ser realizadas, como mostrado a seguir: dependency$(t_1^2, \{dt_1\}, \{(1)\})$ e dependency$(t_1^3, \{dt_1\}, \{(1)\})$.

Já a Figura 36(d) apresenta um caso em que a tarefa t_1^2 da transformação de dados dt_2 depende dos dados produzidos por duas tarefas da transformação de dados dt_1, mas pertencentes a duas iterações diferentes, isto é, t_1^1 e t_2^1. Dessa forma, a invocação da operação *dependency* terá apenas uma transformação de dados especificada em *transformations* (dt_1), mas duas iterações de tarefas dependentes para essa transformação de dados no argumento *dependentTasks* (t_1^1 e t_2^1), como apresentado a seguir: dependency$(t_1^2, \{dt_1\}, \{(1); (2)\})$.

Por último, a Figura 36(e) mostra um cenário em que duas tarefas de diferentes iterações de uma mesma transformação de dados (dt_2), t_1^2 e t_2^2, dependem dos mesmos dados gerados pela tarefa t_1^1 da transformação de dados dt_1, de maneira que duas invocações da operação *dependency* são realizadas: dependency$(t_1^2, \{dt_1\}, \{(1)\})$ e dependency$(t_2^2, \{dt_1\}, \{(1)\})$. Segundo o formalismo de fluxo de dados, uma transformação determina o uso de um ou mais programas de simulação. Consequentemente, tarefas de uma

mesma transformação (de diferentes iterações) consumindo a mesma coleção de dados não representam um comportamento intuitivo, apesar da sua importância em cenários em que as transformações de dados envolvem a execução de modelos computacionais não-determinísticos, pois uma nova iteração (isto é, nova tarefa) pode produzir resultados diferentes a partir de uma mesma coleção de dados de entrada.

Considerando as operações introduzidas nessa subseção, a nossa abordagem favorece a captura de dados de proveniência retrospectiva e de domínio (dados científicos) produzidos pelas simulações computacionais em tempo de execução. Para isso, o usuário do domínio científico trabalha em colaboração com o especialista em ciência da computação para modificar os seus programas de simulação a fim de capturar esses dados no nível de granularidade adequado para as suas análises futuras. Como resultado, a nossa metodologia evita que dados não relevantes sejam armazenados na base de dados ou que dados importantes não estejam disponíveis durante as análises. Ao mesmo tempo, por permitir que os usuários carreguem apenas os dados científicos de interesse, a DfAnalyzer também contribui ao minimizar o custo de captura e de carga dos dados na base de dados.

5.4. Componentes da DfAnalyzer

A DfAnalyzer é composta de três camadas, conhecidas como *Painéis*, *Serviços de Fluxo de Dados* e *Armazenamento*, conforme mostrado na Figura 37. Essas camadas apresentam, no total, seis componentes e uma base de dados para armazenar os dados de proveniência e de domínio (dados científicos), sendo representados como retângulos arredondados com a cor de fundo branca.

A camada Painéis disponibiliza dois componentes com recursos gráficos que favorecem a análise dos fluxos de dados gerados pelas simulações computacionais. Já a camada Serviços de Fluxo de Dados apresenta quatro componentes responsáveis pela extração de dados de proveniência e de dados científicos, a indexação de dados extraídos, assim como a submissão de consultas para a sua base de dados. Por último, a camada Armazenamento é responsável por organizar os arquivos de dados científicos e a base de dados no sistema de arquivos utilizado pelo ambiente computacional. De forma mais detalhada, as subseções seguintes apresentam cada componente dessas camadas.

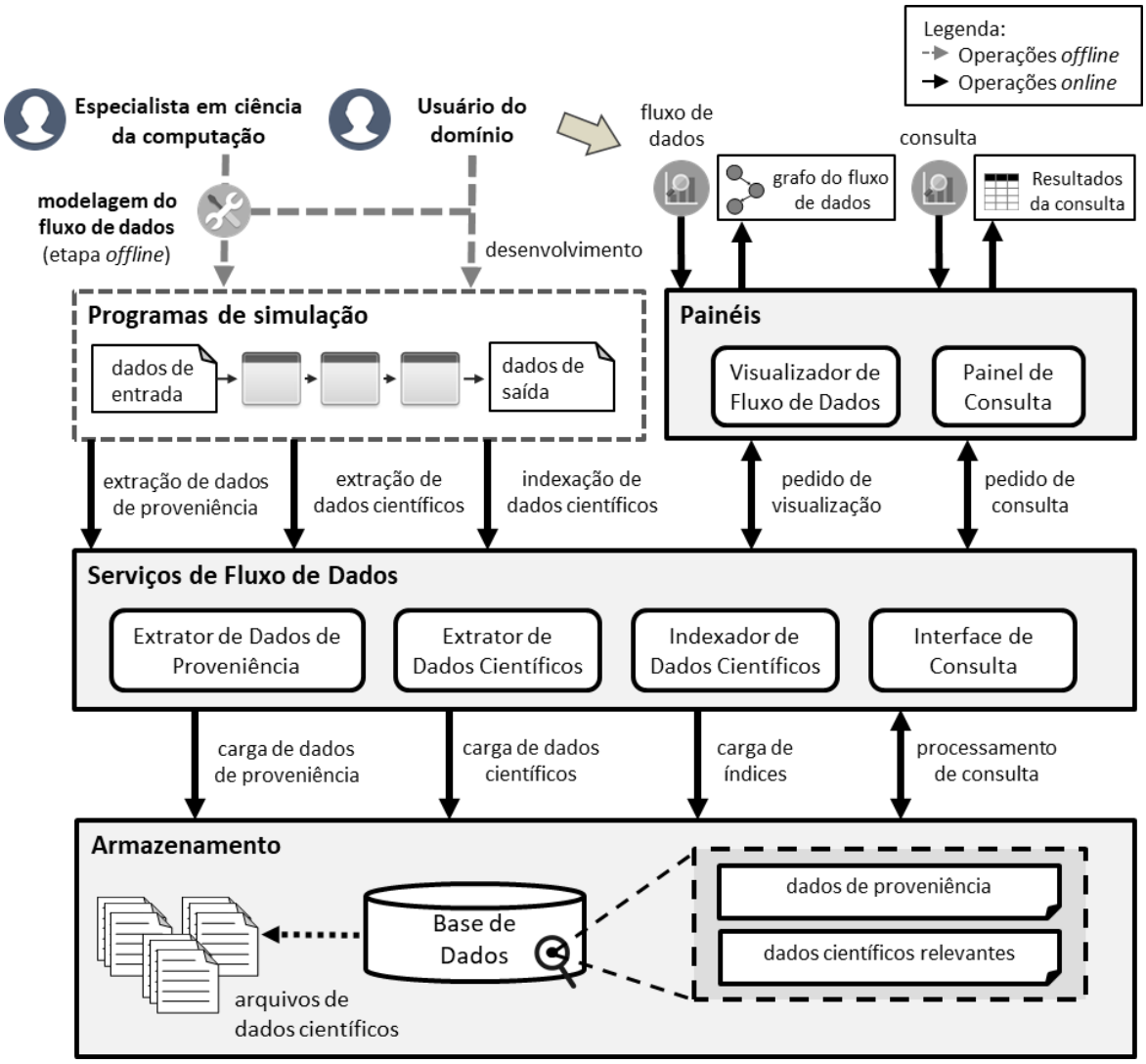


Figura 37. Arquitetura da DfAnalyzer.

5.4.1. Serviços de Fluxo de Dados

O componente *Extrator de Dados de Proveniência*, do termo em inglês *Provenance Data Extractor* (PDE), é responsável por extrair os dados de proveniência prospectiva e retrospectiva presentes nos programas de simulação e representá-los de acordo com o formalismo de fluxo de dados apresentado na Seção 3.1. Para que isso seja possível, os programas de simulação são vistos como caixas cinzas e precisam ser modificados para acessar e capturar os dados de proveniência. Assim, como primeira etapa, o PDE requer a captura dos dados de proveniência prospectiva, que consistem na representação da *especificação do fluxo de dados* que deve ser monitorado durante a execução da simulação computacional. De uma forma mais prática, essa captura dos dados de proveniência

prospectiva é baseada em uma sequência de invocações das operações apresentadas na Tabela 4 da Subseção 5.3.1.

Para que os componentes da DfAnalyzer sejam agnósticos à linguagem de programação adotada pelos programas de simulação, essa biblioteca de componentes foi disponibilizada como um conjunto de serviços RESTful que auxiliam o usuário tanto na captura de dados de proveniência e de domínio, como na análise dos fluxos de dados registrados anteriormente. Para permitir a captura e o armazenamento dos dados de proveniência prospectiva na base de dados da DfAnalyzer, a aplicação RESTful recebe e processa requisições HTTP (URL `/pde/dataflow`) com o método POST em que o conteúdo ou a mensagem dessa requisição deve apresentar esses dados de proveniência.

Ainda em relação ao funcionamento do PDE, diferentes pacotes foram desenvolvidos em linguagens de programação distintas (*e.g.*, Java, Python e C++) para facilitar a captura de dados de proveniência retrospectiva em programas de simulação por meio de um método genérico, conhecido como `capture` (que pode ser traduzido para `capturar`, em português). Esse método realiza a invocação de requisições HTTP para o serviço RESTful do PDE (URL `/pde/task`) a partir do código fonte modificado dos programas de simulação. Esse método é frequentemente invocado duas vezes: antes e após uma tarefa (ou instância de uma transformação de dados).

Antes da execução da tarefa, o método `capture` é invocado para indicar o momento em que a tarefa inicializará, consumindo uma coleção de dados específica (*i.e.*, conjunto de elementos de dados de entrada). Após a execução da tarefa, esse método é invocado novamente para indicar o término da tarefa e capturar a coleção de dados produzida. Assim, a invocação do método `capture` pode ser representada conforme a equação a seguir, em que `fluxo` corresponde ao rótulo do fluxo de dados; `transformação` corresponde ao rótulo da transformação de dados; `iteração` corresponde à iteração da tarefa executada; `estado` representa o estado da tarefa, que pode ser *RUNNING*, *FINISHED* ou *FINISHED_WITH_ERRORS*; e `coleções` corresponde às coleções de dados manipuladas pelos conjuntos de dados. As coleções de dados podem ser inicializadas pelo método `createDataElement` para criar um elemento de dados de acordo com os valores de atributos apresentados pelo usuário. Esses valores de atributo seguem a mesma sequência em que os

atributos do conjunto de dados de interesse (primeiro parâmetro desse método) foram definidos.

```
PDE.capture(<fluxo>, <transformação>, <iteração>, <estado>, <coleções>)
```
(6)

Considerando um programa de simulação real em dinâmica de fluidos computacionais, apresentamos a seguir um exemplo de ajustes no código fonte desse programa, que utiliza a biblioteca libMesh (KIRK *et al.*, 2006), vista como uma caixa preta pela DfAnalyzer. Nesse exemplo, as duas invocações do método `capture` ocorrem antes e após a transformação de dados de interesse. Além disso, essas invocações podem ser realizadas diversas vezes – ou seja, diferentes iterações de tarefas para essa transformação –, pois existe um *loop* que realiza inúmeras iterações não lineares para simular o comportamento do fluido e dos sedimentos em um tanque ou numa bacia sedimentar.

```
for (nli_counter = 0; nli_counter < max_nl_iteractions; ++nli_counter) {
        // provenance - input dataset to transformation flow solver
        PDE->capture("sedimentation", "Flow_Solver", taskIter, "RUNNING", NULL);

        last_nonlinear_soln->zero();
        last_nonlinear_soln->add(*flow_system.solution);

        // Assemble & solve the linear system.
        flow_system.solve();

        // Compute the difference between this solution and the last
        // nonlinear iterate.
        last_nonlinear_soln->add(-1., *flow_system.solution);

        // Close the vector before computing its norm
        last_nonlinear_soln->close();

        // Compute the l2 norm of the difference
        const Real norm_delta = last_nonlinear_soln->l2_norm();
        const Real u_norm = flow_system.solution->l2_norm();

        // How many iterations were required to solve the linear system?
        this->linear_iteractions += flow_system.n_linear_iterations();

        // provenance – output dataset to transformation flow solver
        dataElement = PDE->createDataElement("oflow_solver", t_step, dt,
                                    time, r_step, nli_counter,
                                    linear_iteractions,
                                    flow_system.final_linear_residual(),
                                    norm_delta, norm_delta / u_norm,
                                    !diverged);
        PDE->capture("sedimentation", "Flow_Solver", taskIter,
                     "FINISHED", dataElement);
}
```
(7)

Apesar de não ser apresentado nesse exemplo, as coleções de dados também podem envolver conteúdo específico presente em arquivos ou estruturas de dados alocadas em

memória. Nesse cenário, a DfAnalyzer possui dois componentes para apoiar a extração e a indexação de dados científicos presentes em arquivos, conhecidos como *Extrator de Dados Científicos* (do termo em inglês *Raw Data Extractor* – RDE) e *Indexador de Dados Científicos* (do termo em inglês *Raw Data Indexer* – RDI), respectivamente.

Pelo fato de seguir a arquitetura ARMFUL, as implementações de algoritmos para a extração de dados científicos usando o RDE são representadas por cartuchos, que precisam ser especificados pelo usuário. Nesta tese, desenvolveu-se dois cartuchos para o RDE, que extraem (a) dados científicos usando programas *ad-hoc* (SILVA *et al.*, 2017b); e (b) dados científicos ainda alocados em memória por meio da biblioteca ParaView Catalyst (SILVA *et al.*, 2016a), como mostrado na Figura 38. O cartucho (a) caracteriza-se por invocar programas externos, que são responsáveis por acessar os dados científicos, realizar as análises léxica e sintática, e extrair apenas os dados de interesse.

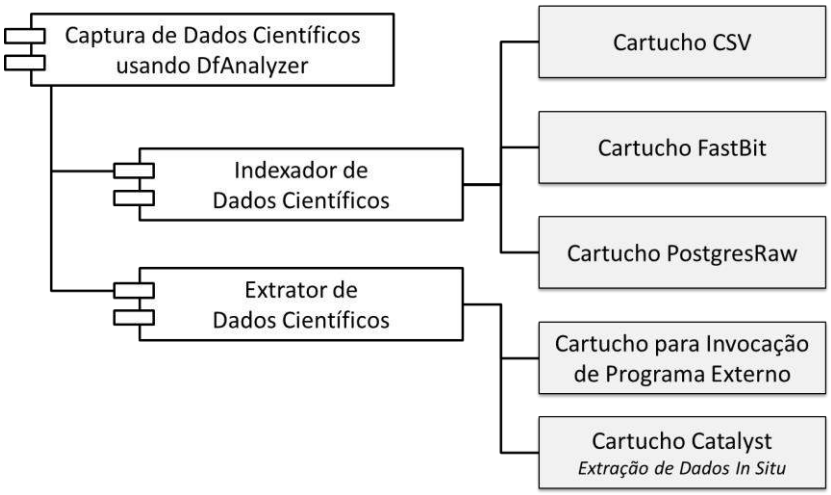


Figura 38. Cartuchos para os componentes de extração e de indexação de dados científicos.

Já o cartucho (b) consiste em uma integração do RDE com o ParaView Catalyst. Para isso, os programas de simulação, que desejam realizar a extração de dados *in situ*, precisam implementar um adaptador Catalyst, conforme especificado em Ayachit *et al.* (2015). Vale ressaltar que esse adaptador requer um mapeamento das estruturas de dados adotadas pelo programa de simulação para o modelo de dados do VTK (VTK, 2009). Uma vez implementado esse adaptador, o Catalyst pode extrair dados a partir de *scripts* na linguagem de programação Python, que implementam *pipelines* de coprocessamento. Para facilitar o desenvolvimento desses *scripts* em Python, o ParaView conta com um recurso de geração automática desses *scripts* por meio da sua interface gráfica.

Mais especificamente, o RDE permite que dados científicos de interesse sejam acessados e extraídos de fontes de dados por meio do uso do método `extract` (que pode ser traduzido como `extrair` para o português), em que o usuário especifica o rótulo do extrator que possuirá os dados científicos capturados (`extrator`), o cartucho do RDE utilizado (`cartucho`) e o caminho do arquivo ou *script* responsável por realizar a extração dos dados científicos de interesse (`arquivo`). Vale ressaltar que os atributos a serem extraídos e as condições para filtrar os elementos de dados de interesse (*i.e.*, condições de seleção) são determinados pelo programa externo ou pelo *script* em Python para o ParaView Catalyst. No caso do cartucho Catalyst, além de extrair dados científicos, ele permite a geração de visualizações de dados. Essa especificação do método `extract` permite a invocação da operação com o mesmo nome apresentada na Subseção 5.3.2, ajustando apenas a invocação do programa de extração de acordo com o cartucho solicitado.

```
RDE.extract(<extrator>, <cartucho>, <arquivo>)
```
(8)

Ainda em relação ao Catalyst, o RDE possui outros três métodos específicos para esse cartucho, sendo os seguintes: `inicializar`, `atualizar` e `finalizar`. O método `inicializar` é responsável por registrar novos *pipelines* (*scripts* em Python) a serem utilizados pelo adaptador do Catalyst. Sendo assim, esse método recebe como argumento apenas o caminho para o *script* em Python. Já o método `atualizar` deve ser invocado todas as vezes que a estrutura de dados do programa de simulação sofre alterações e precisa ser atualizada de acordo com o modelo de dados do VTK para o Catalyst. Por último, o método `finalizar` é invocado para terminar a execução dos *pipelines* registrados e, consequentemente, a extração de dados científicos. O método `finalizar` é importante também para liberar espaço alocado em memória pelas estruturas de dados do VTK ou mesmo pela manutenção da instância do Catalyst.

Somando-se a isso, existem cenários em que as estruturas de dados ou o volume de dados não favorecem a extração de dados científicos em função do custo de carga dos dados em uma base de dados. Nessas circunstâncias, técnicas de indexação são comumente aplicadas para permitir o acesso, de forma eficiente, aos dados científicos. Assim, de forma complementar à extração de dados científicos, o componente RDI foi proposto para gerar índices para os dados científicos acessados. Nesse sentido, três cartuchos foram desenvolvidos: *CSV*, *FastBit* e *PostgresRaw*. O cartucho *CSV* gera índices posicionais para

os dados científicos acessados em arquivos no formato CSV, semelhante à implementação do NoDB (ALAGIANNIS *et al.*, 2015). Já o cartucho *FastBit* consiste no uso da ferramenta FastBit (WU *et al.*, 2009) para gerar índices *bitmaps* para os dados científicos acessados. E, por último, o cartucho *PostgresRaw* (ALAGIANNIS *et al.*, 2015) consiste no uso da ferramenta alternativa com o mesmo nome para gerar índices posicionais.

Já o componente RDI conta com o método `index` (do termo, em português, `indexar`) e os mesmos argumentos do método `extract`, acrescido apenas dos atributos a serem indexados (argument `atributos`) e de propriedades extras (argumento `extra`) para os algoritmos aplicados pelos seus cartuchos. Essas propriedades extras devem ser informadas no formato de invocação por linha de comando adotado por essas ferramentas. Por exemplo, o cartucho FastBit permite definir quais tipos de codificação, de compressão e de precisão podem ser adotados ao gerar os índices.

```
RDI.index(<extrator>, <cartucho>, <arquivo>, <atributos>, <extra>)
```
(9)

No caso do FastBit, o seu cartucho foi resultado de um trabalho em colaboração com o M.Sc. José Vitor Leite, em que se utilizou essa ferramenta alternativa para gerar índices do tipo mapa de *bits* para dados científicos em uma simulação de dinâmica de fluidos computacionais (SILVA *et al.*, 2017b). Mais especificamente, esse trabalho considerou a exploração das diferentes combinações de algoritmos providos pelo FastBit de acordo com a estrutura dos dados produzidos por essa simulação computacional.

Além da sintaxe contemplar a indexação de dados científicos, o processo de acesso a esses dados por meio dos índices gerados foi implementado para cada cartucho, sendo representado pelo método `access` (do termo, em português, `acessar`). Esse método requer como argumentos o cartucho utilizado no processo de indexação (`cartucho`), os atributos a serem acessados (`atributos`), os índices (`índices`) a serem utilizados para acessar os dados científicos, e o caminho para o arquivo de dados científicos (`arquivo`).

```
RDI.access(<cartucho>, <atributos>, <índices>, <arquivo>)
```
(10)

Após o processo de captura de dados de proveniência e dos dados científicos utilizando os componentes PDE, RDE e RDI, esses componentes realizam a carga dos dados capturados em uma base de dados utilizando o SGBD relacional orientado a coluna MonetDB (BONCZ *et al.*, 2008). O MonetDB foi escolhido pelo fato de apresentar algoritmos eficientes de indexação tardia dos dados, favorecendo a carga de grande volume

de dados (ideal para o cenário de extração em larga escala); de apoiar a gerência de arquivos de dados científicos por meio da análise sintática de dados científicos; de realizar a carga parcial de dados científicos a partir de arquivos; e de apresentar um mecanismo eficiente de processamento de consultas.

A Figura 39 apresenta o diagrama de sequência para essas duas etapas: *offline* e *online*. Na etapa *offline*, o usuário do domínio científico precisa modelar o fluxo de dados de acordo com as transformações, os conjuntos de dados e as suas dependências de dados envolvidas na simulação computacional. Logo, essa etapa é manual, uma vez que exige desse usuário ajustes nos códigos fontes dos programas de simulação para invocar os diferentes componentes da DfAnalyzer, como o PDE, RDE e RDI. Enquanto isso, a etapa *online* lida com a extração dos dados de proveniência e dos dados científicos durante a execução dos programas de simulação modificados. Somando-se à extração, essa etapa automática também contempla a carga dos dados extraídos na base de dados da nossa biblioteca de componentes por meio do envio de requisições para os serviços RESTful da DfAnalyzer.

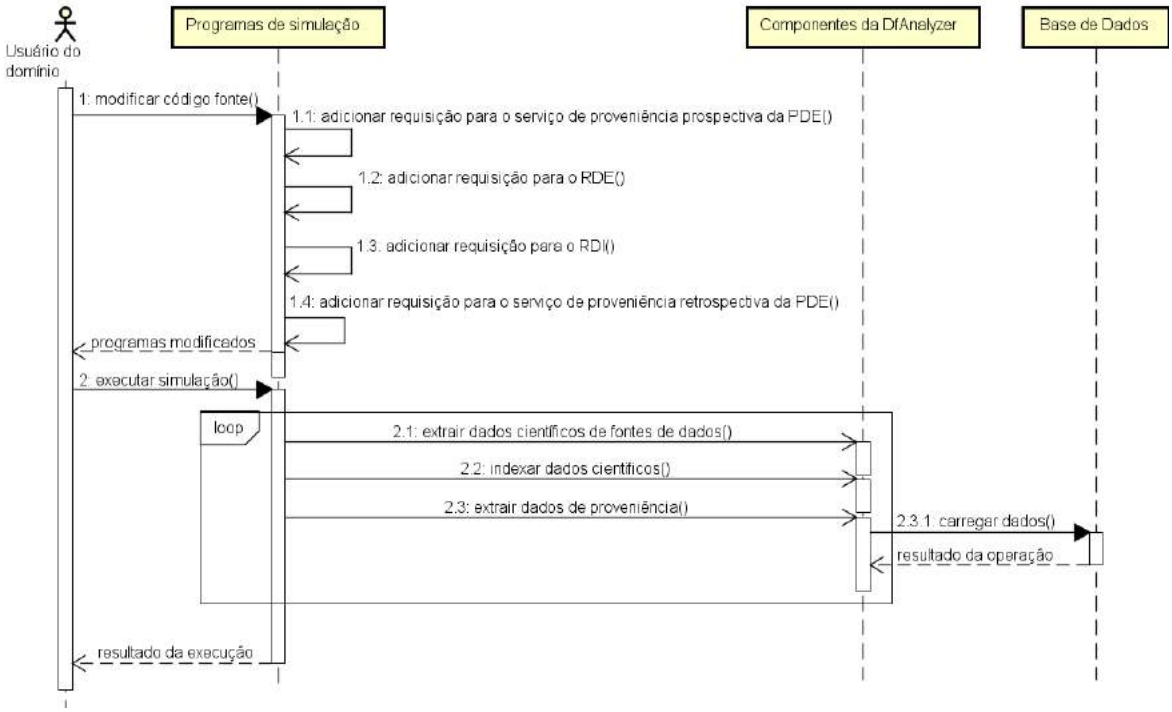


Figura 39. Diagrama de sequência para (1) a modelagem do fluxo de dados (etapa *offline*) e (2) a extração de dados durante a execução da simulação computacional (etapa *online*).

Além desses componentes, a DfAnalyzer apresenta o componente *Interface de Consulta* (do termo em inglês *Query Interface* e da sigla QI), que auxilia os usuários a submeterem consultas para a base de dados e obtém os resultados das consultas. A DfAnalyzer permite o processamento de consultas por meio do desenvolvimento da consulta em uma sintaxe disponibilizada pelo componente QI com os seguintes argumentos: (i) nome e identificador do fluxo de dados registrado na base de dados que será analisado, (ii) os conjuntos de dados que serão usados como origem (*i.e.*, conjuntos de dados de entrada) na busca de um fragmento do fluxo de dados, (iii) os conjuntos de dados que serão usados como destino, (iv) o tipo de mapeamento de atributos (físico, lógico ou híbrido, conforme discutido na Subseção 3.2.4), (v) os atributos a serem projetados (*i.e.*, valores de atributos a serem obtidos após o processamento da consulta), (vi) as seleções de atributos a serem consideradas, semelhante ao processo de filtragem de dados, (vii) os conjuntos de dados que serão considerados nos caminhos possíveis do fragmento do fluxo de dados da consulta, e (viii) os conjuntos de dados que serão desconsiderados nos caminhos possíveis. Nesse caso, as características do QI são mais próximas de um pré-processador de consultas, pois o mesmo não modificou as funcionalidades nativas dos SGBD utilizados ou dos processadores de consultas das ferramentas alternativas, como o FastBit.

Semelhante ao PDE, as funcionalidades do QI foram providas por meio de um serviço RESTful, em que os usuários submetem requisições HTTP para a URL `/query_interface/{dataflow_tag}/{dataflow_id}` com o método POST acrescido da especificação da consulta (como uma mensagem), conforme apresentado na Tabela 6. Essa URL considera os primeiros argumentos do QI, que consistem no nome e no identificador do fluxo de dados (`dataflow_tag` e `dataflow_id`). Enquanto isso, o restante da especificação da consulta deve estar presente na mensagem da requisição. Uma vez submetida a especificação da consulta para o QI, ele é responsável por processar a consulta automaticamente e retornar os resultados da consulta por meio de um ponteiro para um arquivo no formato CSV. Vale ressaltar que o desenvolvimento do QI e os seus resultados experimentais (com exceção do seu serviço RESTful) foram obtidos por meio da orientação dos projetos de graduação do Thiago Barroso Perrotta e da Débora Barbosa Pina.

Tabela 6. Métodos do QI para a especificação da consulta.

Conceito	Métodos do QI para a mensagem de uma requisição HTTP	Informação adicional
Mapeamento de atributos	**mapping(***tipo***)**	tipo = PHYSICAL, LOGICAL ou HYBRID
Conjuntos de dados de origem	**source(***datasetTags***)**	Conjuntos de dados de origem do fragmento do fluxo de dados a ser analisado
Conjuntos de dados de destino	**target(***datasetTags***)**	Conjuntos de dados de destino do fragmento do fluxo de dados a ser analisado
Projeção de dados	**projection(***atributos***)**	*atributos* consiste na sequência de atributos cujos valores são obtidos após o processamento da consulta
Seleção de dados	**selection(***condições***)**	*condições* representa as seleções ou filtros realizados no processamento da consulta
Inclusão de conjuntos de dados	**include(***datasetTags***)**	*datasetTags* contempla os rótulos dos conjuntos de dados que devem ser obrigatoriamente considerados no fragmento do fluxo de dados a ser analisado
Exclusão de conjuntos de dados	**exclude(***datasetTags***)**	*datasetTags* contempla os rótulos dos conjuntos de dados que devem ser obrigatoriamente excluídos do fragmento do fluxo de dados a ser analisado

Como exemplo de uso do QI em uma simulação de sedimentação (mais detalhes na Subseção 6.1.4), os usuários do domínio de dinâmica de fluidos computacionais comumente desejam analisar a concentração média de sedimentos em uma linha de pontos localizados no fundo de uma bacia sedimentar. A Figura 40 apresenta o fluxo de dados dessa simulação, em que os retângulos representam as transformações de dados e as setas correspondem aos conjuntos de dados. Diante dessa especificação do fluxo de dados, o usuário pode informar ao componente QI que precisa executar uma consulta em que o conjunto de dados de origem é o *osolversimulationtransport*; os conjuntos de dados de destino são *oline2extraction* e *omeshwriter*, pois além dos elementos de dados científicos, os usuários estariam interessados nos arquivos de dados científicos gerados; o tipo de mapeamento de atributos é físico, ou seja, utiliza os identificadores das tarefas para realizar as junções entre os conjuntos de dados; e os dados projetados contemplam o tempo da simulação computacional (atributo *time* no conjunto de dados *osolversimulationtransport*), a média da concentração de sedimentos (atributo *d* no conjunto de dados *oline2extraction*) e o ponteiro para os arquivos HDF5 gerados (atributo *hdf5* no conjunto de dados *omeshwriter*). Dessa

forma, as análises de dados permitidas pelo QI podem relacionar dados de proveniência e elementos de dados científicos, assim como referências para arquivos (*i.e.*, o caminho para os arquivos de dados científicos ou de visualização). Outras análises ao fluxo de dados podem ser realizadas sem o componente QI, em que os usuários utilizam as visões dos conjuntos de dados para relacionar elementos de dados em diferentes conjuntos em função dos identificadores das tarefas (atributos com o sufixo *task_id*), como discutido na Subseção 5.2.3 da nossa metodologia.

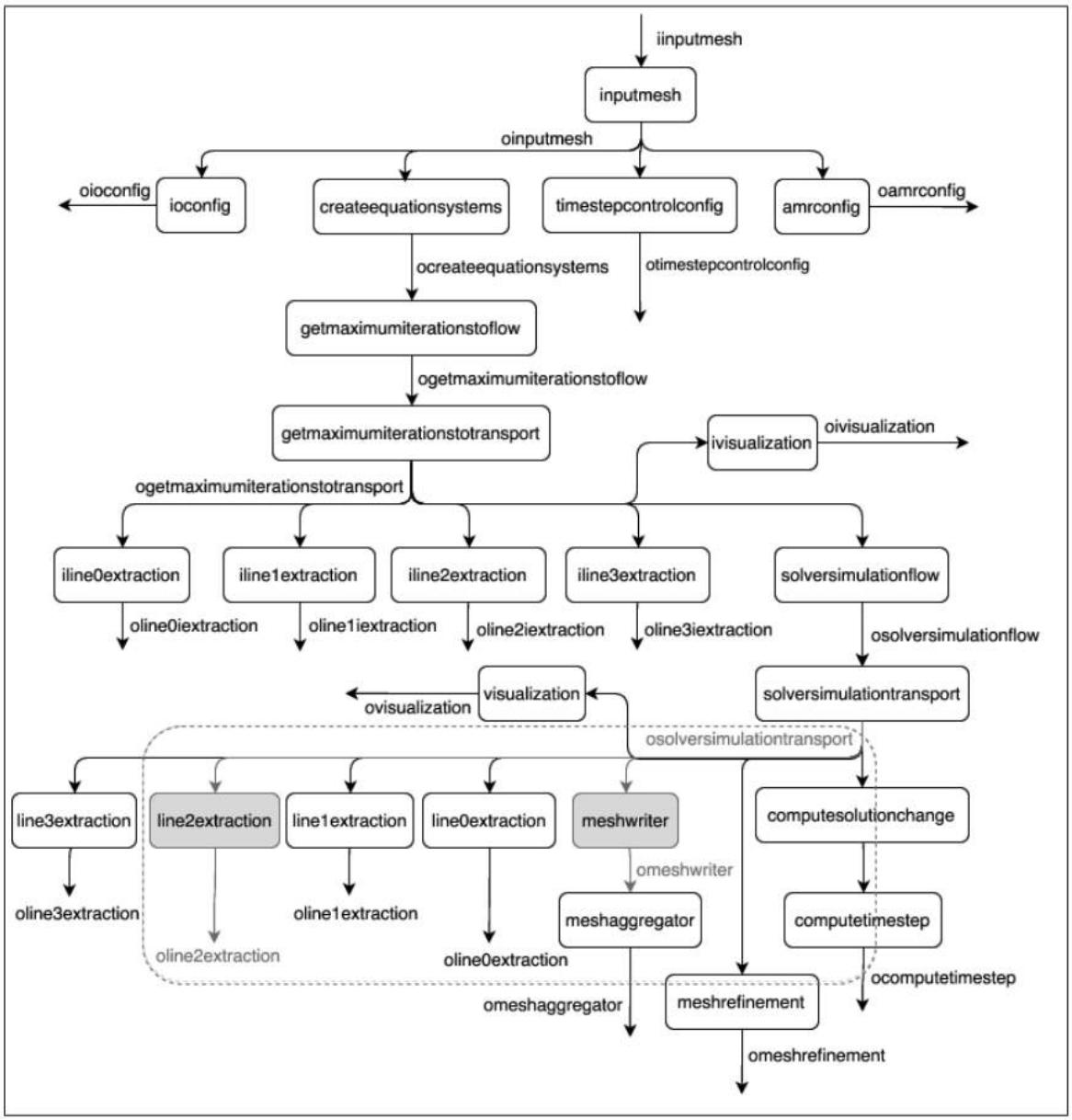


Figura 40. Fragmento do fluxo de dados analisado em uma simulação real de dinâmica de fluidos computacionais.

5.4.2. Armazenamento

Outra camada da nossa solução consiste no armazenamento de dados. Especificamente, essa camada contempla o armazenamento dos dados científicos produzidos pelas simulações computacionais em arquivos e uma base de dados, que integra os dados de proveniência, os dados científicos extraídos de arquivos e os índices para o conteúdo de arquivos de dados científicos. Em termos de infraestrutura, o MonetDB é instanciado em um nó computacional diferente dos nós computacionais destinados à execução da simulação, a fim de evitar gargalos em termos de desempenho, particularmente associados às operações de leitura e escrita em disco. Entretanto, os serviços RESTful da DfAnalyzer são instanciados no mesmo recurso computacional do MonetDB. Ao mesmo tempo, questões relacionadas à tolerância e à recuperação em casos de falha da base de dados não fazem parte do escopo desta tese. Logo, esta tese assume um cenário de alta disponibilidade dos recursos utilizados pela DfAnalyzer, sem considerar ocorrências de falhas no nível de hardware. Por outro lado, os nossos experimentos consideraram uma análise de desempenho, em que medimos a sobrecarga em termos de tempo ao integrar a DfAnalyzer em uma simulação computacional em larga escala. Esses resultados são apresentados no capítulo de avaliação experimental.

5.4.3. Painéis

Somando-se aos componentes para os serviços de fluxo de dados e a infraestrutura para o armazenamento dos dados, a DfAnalyzer conta com uma camada chamada de *Painéis* para prover recursos gráficos capazes de facilitar a análise do fluxo de dados. Nesse sentido, a DfAnalyzer disponibiliza o componente *Visualizador de Fluxo de Dados*, do termo em inglês *Dataflow Viewer* (DfViewer). Esse componente permite que os usuários consultem as especificações de fluxo de dados já registradas na base de dados por meio de uma interface gráfica, que lista todos os fluxos de dados. Assim, o usuário pode escolher o fluxo de dados de interesse e uma visualização da especificação desse fluxo de dados é apresentada na perspectiva de conjuntos de dados, conforme a Figura 41. Ademais, os usuários são capazes de clicar nos conjuntos de dados para terem acesso aos seus esquemas, isto é, aos nomes e aos tipos dos atributos pertencentes ao conjunto de dados selecionado.

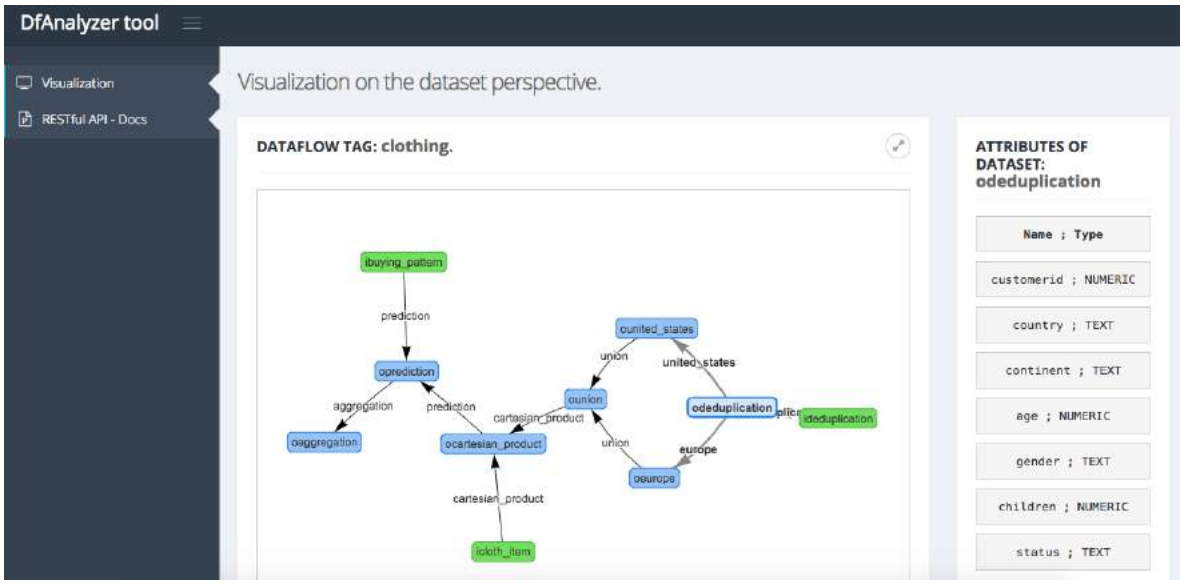


Figura 41. Visualização de uma especificação de fluxo de dados usando o DfViewer.

Ainda no quesito análise de fluxo de dados, outro componente presente nessa camada consiste no *Painel de Consulta* (do termo em inglês *Query Dashboard*). A DfAnalyzer permite o processamento de consultas por meio da edição e submissão da consulta pela interface gráfica do *Painel de Consulta* e da invocação do componente QI com os seguintes argumentos: (i) os conjuntos de dados que serão usados na análise de um fragmento de dados, (ii) o tipo de mapeamento de atributos (físico, lógico ou híbrido, conforme discutido na Subseção 3.2.4), (iii) os atributos a serem projetados (*i.e.*, valores de atributos a serem obtidos após o processamento da consulta), e (iv) as seleções de atributos a serem consideradas, semelhante ao processo de filtragem de dados. Nesse caso, esse componente utiliza esses argumentos para especificar uma consulta a ser executada pelo componente QI, considerando os dados de proveniência e os dados de domínio extraídos pelos componentes PDE, RDE e RDI. A Figura 42 apresenta uma visualização da interface do Painel de Consulta sendo utilizado em uma simulação de dinâmica de fluidos computacionais.

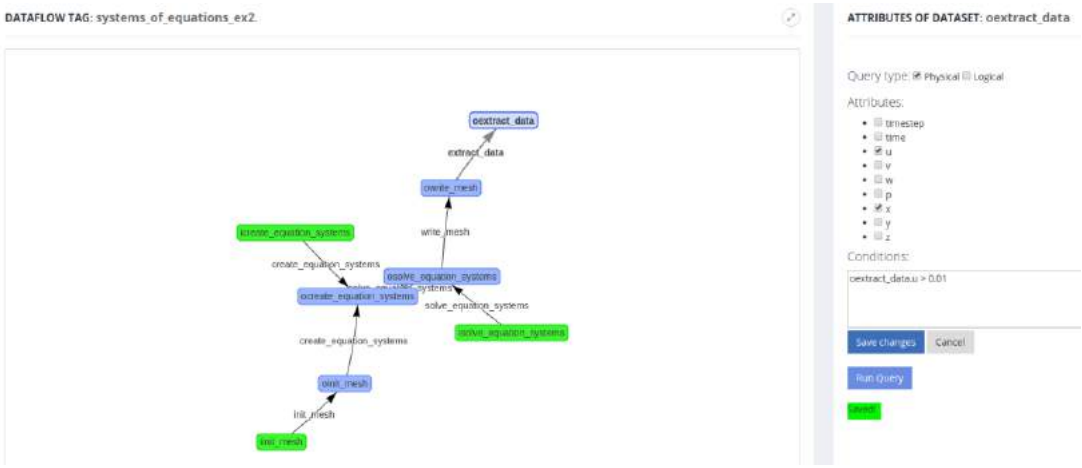


Figura 42. Interface gráfica para a especificação de consulta.

Capítulo 6- Avaliação experimental

Este capítulo apresenta os resultados obtidos utilizando as implementações da arquitetura ARMFUL, o SCC-A e a DfAnalyzer, para apoiar a captura e a análise de dados científicos. Para isso, quatro simulações computacionais foram modeladas (Seção 6.1) e executadas em três ambientes de Processamento de Alto Desempenho (PAD) (Seção 6.1.5), a fim de gerenciar o fluxo de dados gerado em tempo de execução. Os experimentos (Seções 6.3 e 6.3.6) realizados tiveram o objetivo de analisar os seguintes pontos: (i) custo da extração de dados científicos, variando-se ou não a carga de trabalho; (ii) custo de indexação dos dados científicos; (iii) custo de carga dos dados em uma base de dados usando um Sistema de Gerência de Banco de Dados (SGBD); (iv) custo da captura dos dados científicos, considerando uma abordagem de processamento de dados *in situ*; e (v) apresentação do apoio ao processamento de consultas na análise exploratória de dados científicos. Esses resultados experimentais foram publicados em diferentes artigos, conforme descrito na conclusão desta tese.

6.1. Simulações computacionais

Esta seção descreve as simulações computacionais modeladas e executadas nesta tese. Mais especificamente, apresenta-se quatro simulações computacionais executadas usando o SCC-A e a DfAnalyzer, alternadamente, de acordo com a natureza da aplicação científica utilizada. No caso das simulações computacionais de astronomia (Montage) e de óleo e gás (SyntheticOil&Gas e EdgeCFD), o SCC-A foi utilizado em ambientes de *cluster* computacional. Enquanto isso, a DfAnalyzer foi empregada na simulação computacional baseada no solucionador libMesh-sedimentation, uma vez que tal programa de simulação já utilizava uma biblioteca para apoiar a execução paralela. O Sistema de Gerência de *Workflow* Científico (SGWfC) SCC-A não pôde ser utilizado nesta última simulação pelo fato de não permitir a invocação de tarefas (execução de atividades do *workflow*) em múltiplos nós computacionais.

6.1.1. Montage

A partir do uso de um conjunto de ferramentas conhecido como Montage (JACOB *et al.*, 2009), o *workflow* científico com esse mesmo nome provê aos astrônomos serviços para a construção de mosaicos personalizados no formato de arquivo FITS (do termo em

inglês *Flexible Image Transport System*) (HANISCH *et al.*, 2001). No que diz respeito à geração desses mosaicos, os programas implementados no Montage permitem analisar diferentes sistemas de coordenadas, tamanhos de imagens arbitrários, rotações e todos os mapeamentos de projeções do sistema de coordenadas mundial, conhecido como *World Coordinate System*. Nessa simulação foram consideradas diferentes regiões do espaço no processamento computacional usando os programas do Montage a fim de gerar mosaicos personalizados.

Quanto à sua composição, o *workflow* científico Montage possui nove atividades, conforme mostrado na Figura 43. Considerando o seu processamento computacional, a primeira transformação de dados (*List FITS*) extrai diferentes arquivos no formato FITS de um arquivo compactado obtido de um repositório externo da astronomia, *Two Micron All Sky Survey*[15] (2MASS) desenvolvido pela NASA/IPAC. Cada arquivo no formato FITS descompactado apresenta mais de vinte tipos de dados específicos do domínio ou atributos. A segunda transformação de dados (*Projection*) realiza o cálculo das projeções de acordo com as referências fornecidas pelos arquivos no formato FITS, a fim de que todas as regiões do espaço sejam projetadas em um mesmo plano.

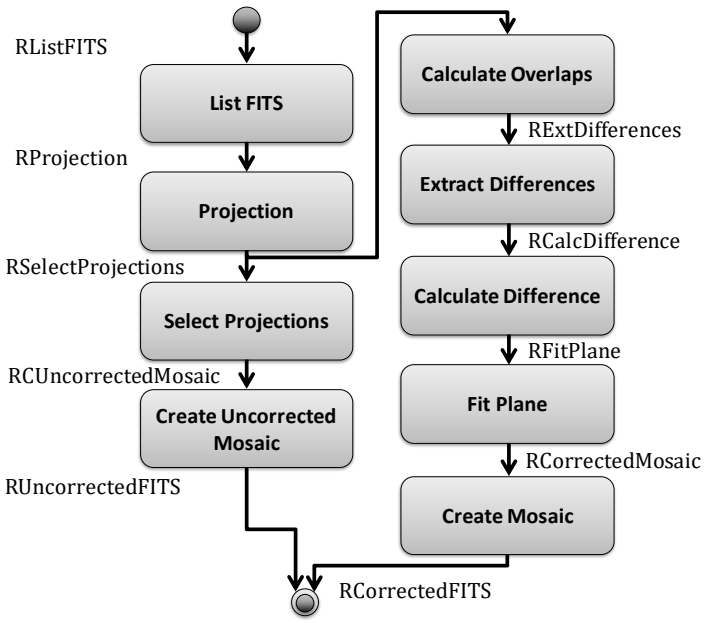


Figura 43. *Workflow* científico Montage.

[15] http://irsa.ipac.caltech.edu/Missions/2mass.html

Por conseguinte, a transformação de dados *Select Projections* une todos os arquivos no formato FITS após as projeções em um determinado plano, que atendem à mesma construção de mosaico. A transformação de dados *Create Uncorrected Mosaic* lida com a criação do mosaico sem correções, ou seja, sem considerar as correções de cor e as interferências de sobreposição de regiões do espaço (presentes nos arquivos de formato FITS) a partir do conjunto de dados de saída da transformação de dados *Select Projections*. Como resultado, essa transformação de dados apresenta uma execução consideravelmente mais rápida e gera como resultado uma imagem no formato JPG para representar o mosaico sem correções. Ainda nesse *workflow* científico, as outras transformações de dados ponderam as interferências das sobreposições de imagens e as correções de cor, a fim de gerar um arquivo no formato JPG como saída que corresponde ao mosaico personalizado com correções.

6.1.2. *SyntheticOil&Gas*

A segunda simulação computacional é baseada em um *workflow* científico sintético que reproduz o comportamento de uma execução de modelos reais no domínio de Óleo e Gás, chamada de *SyntheticOil&Gas* (OGASAWARA *et al.*, 2011). Esse *workflow* científico foi desenvolvido por meio do *Scientific Workflow Benchmark* (SWB) (CHIRIGATI *et al.*, 2012), um gerador de *workflow*s científicos a serem usados com *benchmarks.* SWB é capaz de gerar *workflow*s sintéticos baseado em especificações de transformações de dados que determinam o comportamento esperado de uma simulação computacional real. Esse *workflow* científico sintético é mostrado na Figura 44.

Em relação à composição desse *workflow* científico, a primeira transformação de dados (conhecida como *A*) realiza a fragmentação dos dados de entrada de acordo com um fator (termo em inglês *split factor*) (CHIRIGATI *et al.*, 2012). Esse fator corresponde ao número de elementos de dados produzidos na saída após o consumo de um elemento de dados pela transformação de dados. Nesta tese, considerou-se um valor fixo de *split factor* igual a 240. Na sequência, o *workflow* científico conta com duas transformações de dados, sendo conhecidas como *B* e *C*, em que, para cada elemento de dados de entrada, apenas um elemento de dados de saída é produzido.

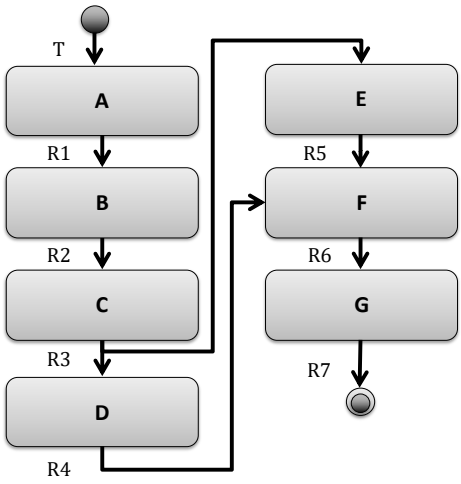


Figura 44. *Workflow* científico sintético de Óleo e Gás.

Após a execução de *C*, duas outras transformações de dados, conhecidas como *D* e *E*, são executadas em paralelo e apresentam o comportamento de um filtro. Para essas transformações de dados, considerou-se que as seletividades dos filtros (CHIRIGATI *et al.*, 2012) seriam iguais a 60% e 80%, respectivamente. Após a execução de *D* e E, a transformação de dados *F* une os elementos de dados dos conjuntos de dados R4 e R5. Por último, a transformação de dados *G* agrupa conjuntos de elementos de dados de acordo com um atributo de agregação.

6.1.3. EdgeCFD

Outra simulação computacional modelada nesta tese consiste em uma aplicação científica que visa analisar o fluxo de fluidos incompressíveis em um problema de cavidade. Essa simulação usa o programa *EdgeCFD* e, consequentemente, o *workflow* científico modelado apresenta o mesmo nome. O EdgeCFD é um programa de simulação de elementos finitos em Fortran90, em que o seu núcleo (*kernel*) da solução computacional consiste em um preditor multicorretor totalmente implícito para o esquema de integração temporal como descrito por Elias e Coutinho (2007). A regra do trapezoidal generalizada é empregada na discretização do tempo, com um procedimento de passos de tempo adaptativo baseado em um controlador proporcional-integral-derivativo (PID).

As não linearidades devido ao termo convectivo na equação de Navier-Stokes (descreve o movimento do fluxo viscoso) são tratadas pelo algoritmo Newton-GMRES inexato (GUERRA *et al.*, 2016). Na solução desse algoritmo, no começo das iterações não-

lineares em cada passo de tempo, o algoritmo calcula passos por meio de tolerâncias mais elevadas, produzindo passos não-lineares mais rápidos. Conforme as iterações progridem em direção à solução, o método não-linear inexato se adapta de acordo com as tolerâncias GMRES para alcançar a acurácia desejada. A tolerância, cujo sistema linearizado é resolvido, conhecido como termo de força, apresenta um papel importante no desempenho numérico desse método. Para melhorar a robustez desses métodos, eles precisam ser empregados com estratégias de globalização, que contemplam procedimentos auxiliares que aumentam a convergência da solução, quando boas aproximações iniciais para a solução não estão disponíveis. O EdgeCFD apresenta quatro alternativas para a definição desse termo de força, ou seja, quatro valores diferentes para o atributo de entrada dessa propriedade.

Como motivação para a escolha desta simulação computacional, esta tese considerou o grande volume de dados manipulados por esse *workflow*. Para facilitar a carga desses dados em um repositório externo ou uma base de dados, esta tese considerou o uso de ferramentas alternativas, como a biblioteca ParaView (AYACHIT, 2015) e a ferramenta FastBit (WU *et al.*, 2009), que realizam o acesso aos dados científicos a partir de arquivos com formatos heterogêneos e a geração de índices, respectivamente.

A Figura 45 mostra o *workflow* EdgeCFD. Nesse *workflow*, apenas as atividades *EdgeCFD Pre* e *EdgeCFD Solver* usam os programas de simulação binários do EdgeCFD (*i.e.*, programas *edgecfdPre* e *edgecfdSolver*, respectivamente), enquanto as outras atividades executam os programas *uncompressDataset*, *preProcessing*, e *setSolverConfiguration* para lidar com conjuntos de dados compactados, arquivos de configuração para executar os binários do EdgeCFD, e dados científicos armazenados em arquivos nos formatos XDMF e HDF5[16], respectivamente.

A atividade *EdgeCFD Pre* consome um conjunto de dados de entrada (*i.e.*, malha representada pelo atributo *MESH*) e define as propriedades da malha (*i.e.*, fragmentação para permitir o processamento paralelo e a reordenação nodal para melhorar a localidade dos dados) a serem empregadas no solucionador de dinâmica de fluidos computacionais (do termo, em inglês, *Computational Fluid Dynamics* – sigla CFD). Esse solucionador é

[16] https://support.hdfgroup.org/HDF5

executado na atividade *EdgeCFD Solver*. Dessa forma, a atividade *EdgeCFD Solver* consome os arquivos gerados pela atividade *EdgeCFD Pre* (após serem organizados pela atividade *Set Solver Configuration*) e calcula o comportamento de um fluido em uma malha específica ao longo do intervalo de tempo predefinido para essa simulação computacional.

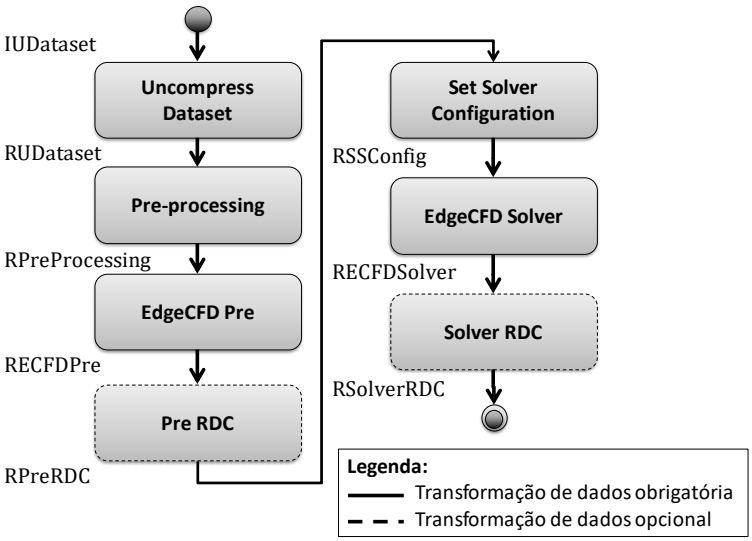


Figura 45. *Workflow* científico EdgeCFD.

Como resultado, esse solucionador produz diferentes arquivos nos formatos XDMF e HDF5, que apresentam informações sobre a pressão, a velocidade e outras propriedades do fluido nesse intervalo de tempo. Mais especificamente, os arquivos HDF5 contém valores para os atributos de domínio (*e.g.*, velocidade do fluido) e os arquivos XDMF apresentam ponteiros para indicar cada arquivo no formato HDF5 para os diferentes passos de tempo dessa simulação. Por último, as atividades *Pre RDC* e *Solver RDC* são atividades com apoio à captura de dados científicos em arquivos (do termo, em inglês, *Raw Data Capture* – sigla RDC), sendo que a estratégia de extração de dados científicos e de geração de índices podem variar nessas atividades.

Um exemplo de análise de dados científicos com dados extraídos ou indexados contempla uma linha de pontos para o problema de cavidade, considerando as condições de contorno e o número de Reynolds (um número adimensional de caracterização do fluxo do fluido), conforme mostrado na Figura 46. Essa região específica de interesse coleta os componentes de velocidade a serem comparados com soluções de referência existentes (LO *et al.*, 2005), a fim de controlar a acurácia da solução para um conjunto de parâmetros

específicos. Como resultado, a Figura 46(b) mostra as velocidades horizontal e vertical representadas pelos atributos V_x e V_z, respectivamente.

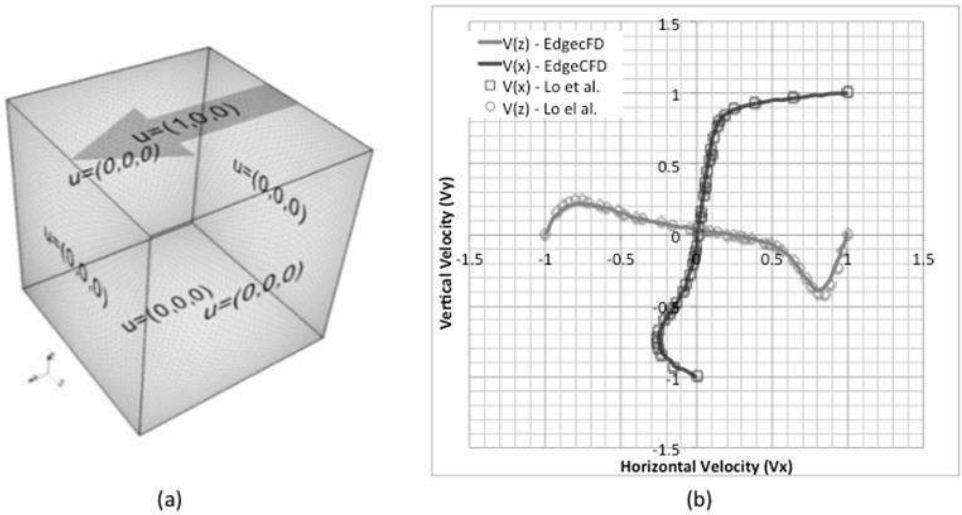


Figura 46. *Workflow* EdgeCFD: (a) condições de contorno e (b) validação do fluxo da cavidade para número de Reynolds Re=1000.

Além disso, os usuários podem variar os valores dos atributos de entrada. A execução dessas múltiplas combinações de valores para os atributos de entrada descreve um cenário de varredura de parâmetros (RAICU *et al.*, 2008), em que os usuários avaliam as suas hipóteses científicas a partir de ajustes ou "perturbações" nos dados de entrada. Nessa simulação computacional, os usuários precisam, em geral, variar o tamanho da malha, a densidade do fluido, a viscosidade do fluido, e os parâmetros do solucionador, como o termo de força e os parâmetros para a estratégia de *backtracking*.

O domínio físico assumido nessa simulação é discretizado por tetraedros, sendo que, ao aumentar o tamanho da malha, o número de vértices e de tetraedros também deve aumentar. Cada tetraedro nessa representação consiste de quatro números inteiros, em que três deles mapeiam os vértices e o remanescente serve como identificador do tetraedro. Já a especificação da malha, representada em arquivos no formato HDF5, contempla as coordenadas em três dimensões de todos os pontos e as suas conectividades. Os resultados esperados são a pressão do fluido, a viscosidade, a velocidade do fluido nos três componentes (*x*, *y* e *z*), entre outras propriedades em cada passo de tempo, também armazenados nos arquivos HDF5. Os experimentos conduzidos para esse *workflow* consideraram três tipos de malhas de entrada: grossa, média e fina. A *malha grossa* possui

32.109 vértices e 128.436 tetraedros; a *malha média* possui 257.852 vértices e 1.031.408 tetraedros; e a *malha fina* possui 1.145.569 vértices e 4.583.276 tetraedros.

6.1.4. libMesh-sedimentation

A quarta simulação computacional considerou a execução de um solucionador, conhecido como libMesh-sedimentation[17], que simula as correntes turbidíticas tipicamente encontradas em processos geológicos. Esse solucionador utiliza a libMesh, uma biblioteca de código aberto para elementos finitos, que permite modificar o refinamento de malhas para serem mais finas ou mais grossas de forma adaptativa (do termo, em inglês, *Adaptive Mesh Refinement and Coarsing* – sigla AMR/C). Essa estratégia é largamente utilizada em diversas aplicações de multifísica. Os sedimentos transportados em função do movimento do fluido são descritos por um arcabouço Euleriano, em que os resultados dos modelos matemáticos a partir da equação Navier-Stokes incompressível para o fluido são combinados com uma equação de transporte dominada por advecção para o cálculo da concentração de sedimentos.

O libMesh-sedimentation emprega um método de elementos finitos multi-escala variacional (CAMATA *et al.*, 2018, SILVA *et al.*, 2016a, 2017a), em que uma abordagem escalonada é usada para evoluir em relação ao tempo as equações de fluido-sedimento acopladas. Esse solucionador também permite a adaptatividade no tempo (AHMED e JOHN, 2015, VALLI *et al.*, 2005). Contudo, a descoberta de quais dados precisam ser analisados e decidir por ajustes finos na simulação pode ser difícil, uma vez que os dados estão frequentemente espalhados em diferentes arquivos e desconectados do passo de tempo correspondente.

A Figura 47 mostra as principais transformações de dados da simulação computacional utilizando o libMesh-sedimentation. A transformação *Configuração do AMR/C* é responsável por consumir uma malha de entrada e definir as configurações a serem aplicadas para o refinamento da malha adaptativa. A transformação *Configuração da Simulação* define propriedades mais gerais do solucionador, como o tempo máximo da simulação e os níveis de tolerância linear e não linear.

[17] https://bitbucket.org/nacadhpc4e/libmesh-sedimentation

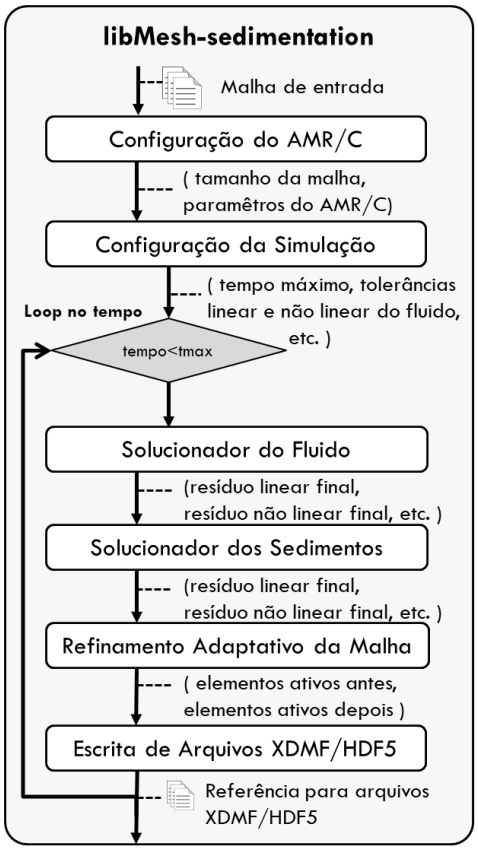


Figura 47. Simulação computacional usando o solucionador libMesh-sedimentation.

Já as transformações de dados subsequentes estão presentes dentro de um *loop* que itera a simulação computacional no tempo, ou seja, um passo de tempo avança em cada iteração desse *loop* até alcançar o tempo máximo definido. Nesse sentido, a primeira transformação de dados dentro desse *loop*, *Reconfigurar Parâmetros de Simulação*, apenas apresenta processamento computacional, quando os usuários realizam ajustes nos parâmetros de entrada do solucionador libMesh-sedimentation. Em seguida, o solucionador progride para calcular o comportamento esperado do fluido e dos sedimentos em um passo de tempo específico nas transformações de dados *Solucionador do Fluido* e *Solucionador dos Sedimentos*, respectivamente, considerando as configurações mais recentes dos parâmetros de entrada da simulação.

Após essa etapa, a transformação *Refinamento Adaptativo da Malha* avalia se a malha precisa sofrer refinamento e/ou desrefinamento de acordo com os parâmetros de tolerância de entrada definidos e a estimativa de erro calculada, e a transformação *Escrita de Arquivos XDMF e HDF5* realiza o armazenamento dos dados científicos, obtidos pela execução do solucionador, em disco rígido. Vale ressaltar que a escrita dos arquivos só

acontece em intervalos grandes de tempo (*e.g.*, 5.000 passos de tempo) para evitar um alto custo de armazenamento de dados em disco rígido e uma sobrecarga em termos de tempo, ao executar essa simulação em larga escala.

No que diz respeito ao domínio dessa simulação computacional, dois estudos foram realizados: um tanque de sedimentação proposto por *de Rooij e Dalziel* (DE ROOIJ e DALZIEL, 2009) e um tanque real de batimetria. O tanque *de Rooij e Dalziel* é baseado em um tanque que inicializa a simulação com os sedimentos separados fisicamente do fluido. Como condições de contorno, considera-se uma caixa retangular com dimensões 20 x 2 x 2 (Figura 48). O bloqueio, em que o fluido está inicialmente em repouso, tem altura 2 e comprimento 0,75. Nessa simulação assumiu-se que $Gr = 10^6$, $Sc = 1$ e $u_s = 0{,}02$ (velocidade de sedimentação adimensional). Nossa malha de entrada é estruturada e hexaedra com espaçamento da grade igual a 0,1, sendo que dois refinamentos iniciais são aplicados, totalizando 4.608.000 de hexaedros. O estimador de erro de Kelly é calculado para a velocidade e a concentração de sedimentos, e usado para conduzir o refinamento adaptativo da malha. Os sistemas de equações lineares de Navier-Stokes e para a concentração de sedimentos são resolvidos em paralelo pelos métodos Block-Jacobi + GMREL(35) com pré-condicionamento local ILU(0). A tolerância do GMREL é definida para 10^{-6}. Para o solucionador não-linear, a tolerância é 10^{-3} e o tamanho de cada passo no tempo é de 0,005. Os arquivos nos formatos XDMF e HDF5 são escritos a cada 50 passos de tempo.

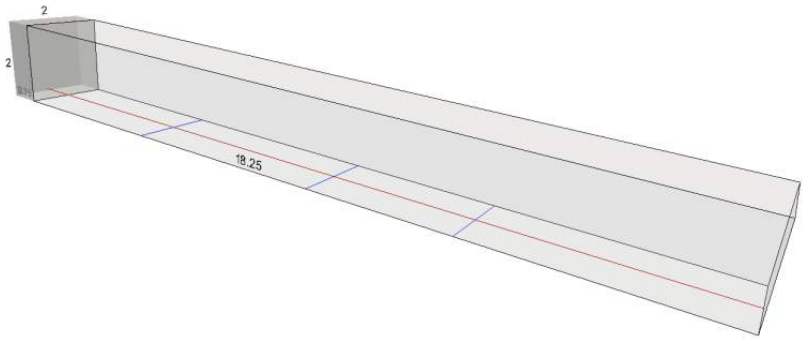


Figura 48. Tanque de sedimentação proposto por *de Rooij e Dalziel*.

Outro estudo de caso considera um tanque real de batimetria, conforme o cenário físico mostrado na Figura 49. O mapa de cor dessa figura representa a batimetria em um cenário real, em que as regiões mais fundas apresentam a cor azul escuro. Nesse cenário, uma mistura diluída dos sedimentos é continuamente injetada no canal de forma que os

sedimentos são lançados no tanque em um ângulo de 45°. Observa-se, inclusive, que o canal está localizado próximo de uma região do domínio. Esse problema é configurado, portanto, por um canal com comprimento L_c = 2,5, largura W_c = 0,2 e altura H_c = 1,5, e um tanque com comprimento L_T = 14, largura W_T = 12 e altura H_T = 1,5.

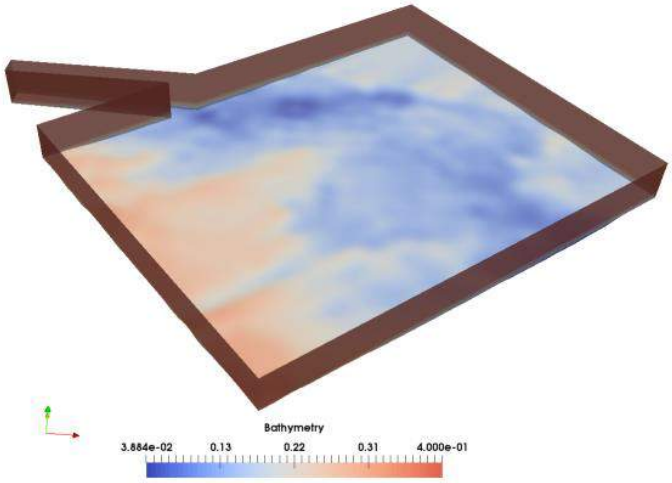


Figura 49. Tanque real de batimetria: cenário físico.

A discretização espacial emprega uma malha desestruturada fixada com 7.674.812 tetraedros e 1.418.934 nós. A mistura monodispersa é considerada com velocidade de sedimentação adimensional u_s = 5,6651 x 10^{-3}, Gr = 10^6 e Sc = 1,0. O passo de tempo é fixado como t = 5 x 10^{-3} e a análise é realizada a cada 100 unidades adimensionais de tempo. Os parâmetros do solucionador linear e não-linear são os mesmos do estudo de caso anterior. As condições de contorno de velocidade de não penetração e não escoamento são definidas para as paredes do canal e as paredes do tanque que estão em contato com o canal. As outras paredes apresentam condições que não investigam o escoamento (do termo, em inglês, *free slip condition*). Os sedimentos são injetados no canal com uma velocidade de magnitude de 0,1.

6.1.5. Multifísica

Além das simulações computacionais já descritas, esta tese também considera uma aplicação de CSE da área de multifísica que simula as interações entre dois fluidos num sistema inicialmente instável no qual a massa é constante e os fluidos encontram-se inicialmente misturados. O objetivo da aplicação é descobrir a partir de qual instante de tempo os dois fluidos passam a ficar completamente separados. Esse sistema pode ser

modelado utilizando a equação de Cahn-Hilliard que descreve um processo binário de separação de fluidos e pode ser resolvida utilizando métodos numéricos, como métodos de elementos finitos. Essa aplicação de CSE utiliza as ferramentas disponibilizadas pelo projeto FEniCS (ALNÆS *et al.*, 2015), que permite modelar esse sistema físico utilizando a equação de Cahn-Hilliard e os métodos de elementos finitos para resolvê-la. Do ponto de vista de uso dessa simulação computacional, esta tese considerou a execução em um ambiente local, ou seja, em um computador pessoal com um processador Intel Core i7-5500U e 8GB de memória RAM usando o sistema operacional Ubuntu 16.04 LTS.

6.2. Ambientes de Processamento de Alto Desempenho (PAD)

As simulações computacionais apresentadas foram executadas em três ambientes de Processamento de Alto Desempenho (PAD). Em mais detalhes, nossos experimentos consideraram dois *clusters* computacionais, Uranus e LoboC, do Núcleo Avançado de Computação de Alto Desempenho[18] (NACAD) localizado na COPPE / Universidade Federal do Rio de Janeiro (UFRJ); um *cluster* computacional, Stampede[19], do Centro de Computação Avançada do Texas (do termo, em inglês, *Texas Advanced Computing Center* e da sigla TACC) localizado na Universidade do Texas; e uma grade computacional, conhecida como Grid5000, cujos recursos computacionais estão espalhados em diversas cidades da França. Mais detalhes sobre o uso dos recursos computacionais desses ambientes são apresentados nas subseções a seguir.

6.2.1. *Cluster* Uranus

Os experimentos utilizando os *workflows* Montage (Subseção 6.1.1) e sintético para aplicações de Óleo e Gás (Subseção 6.1.2) foram conduzidos no *cluster* de nome Uranus, pertencente ao NACAD da COPPE/UFRJ. O Uranus consiste de uma máquina SGI Altix ICE 8400 com 128 CPUs Intel Xeon (640 *cores* no total) separados em duas filas de escalonamento de jobs. Esse *cluster* conta com 1,28 TB de memória RAM distribuída, 72 TB de armazenamento em disco através da SGI InfiniteStorage NAS e com rede Infiniband QDR, DDR e Gigabit. O sistema operacional é o Suse Linux Enterprise Server (SLES)

[18] www.nacad.ufrj.br

[19] https://www.tacc.utexas.edu/systems/stampede

associado ao SGI Performance Suite. Nos experimentos desta tese utilizou-se apenas a fila de 64 CPUs Quad Core Intel Xeon X5355 (Clovertown), que contém 256 *cores* por uma questão de disponibilidade.

Para utilizar o SCC-A, a base de dados de proveniência usando o SGBD PostgreSQL foi instanciada em um nó computacional dedicado para armazenar os dados de proveniência e os dados científicos. Esse nó computacional com a base de dados possuía 8 núcleos de processamento ou processadores. O SCC-A conecta o nó mestre (ou nó central) com o nó da base de dados para obter as tarefas (ou ativações) prontas para serem executadas e para gerenciar os dados de proveniência. No código fonte do SCC-A, a conexão entre o SGWfC e o SGBD é realizada por um *driver* JDBC para o PostgreSQL.

Ademais, a extração de dados científicos corresponde a um recurso opcional na modelagem do *workflow* Montage utilizando o SCC-A. Mais especificamente, a Subseção 6.3.1 apresenta resultados experimentais obtidos ao executar o *workflow* científico Montage com e sem extração de dados científicos. Nesses experimentos, destaca-se como os dados científicos extraídos e carregados em uma base de dados favorecem análises focadas no conteúdo dos arquivos de dados científicos ou no próprio fluxo de elementos de dados, além da sobrecarga em termos de tempo de processamento ao extrair dados científicos durante a execução paralela desse *workflow*.

Já o *workflow* SyntheticOil&Gas permite tanto a extração como a indexação de dados científicos presentes em arquivos gerados em tempo de execução. Tais arquivos seguem formatos com dados binários comumente adotados por aplicações no domínio de Óleo & Gás. Diante dessa configuração, os experimentos com esse *workflow* analisam os custos em termos de tempo para a extração, a indexação e a carga dos dados científicos.

6.2.2. Grade computacional Grid5000

Enquanto os experimentos iniciais para a extração de dados científicos foram realizados usando o *cluster* Uranus, os resultados preliminares para a indexação posicional de dados científicos foram obtidos a partir de execuções no Grid 5000[20]. Esse ambiente de PAD é composto de diversos *clusters* distribuídos por regiões da França. Em cada região,

[20] www.grid5000.fr

o Grid 5000 apresenta um sistema de arquivos compartilhado para todos os *clusters*. A especificação do *cluster* utilizado em nossos experimentos está descrita na Tabela 7. Como o Grid 5000 funciona por meio da virtualização dos recursos computacionais, utilizou-se um arquivo TAR compactado com binários para uma distribuição Linux genérica de 64 bits. No que diz respeito ao repositório de dados de proveniência, tanto na Uranus, quanto no Grid 5000, instalou-se uma versão binária do PostgreSQL Server 9.4 e utilizou-se alguns *scripts* desenvolvidos pelo nosso grupo de pesquisa para gerenciar as instâncias do SGBD.

Tabela 7. Recursos computacionais utilizados do Grid 5000.

Cluster / Região	Processador	#Nós	#Processadores por nó	Total de processadores	Memória por nó	Rede
Suno / Sophia	Intel Xeon X5520 2.26GHz/8MB	45	8	360	32 GB	Gigabit Ethernet

6.2.3. *Cluster* Stampede

O *cluster* computacional Stampede foi utilizado nos experimentos com o *workflow* EdgeCFD na primeira instanciação da ARMFUL, o SCC-A (Seção 4.7). O Stampede consiste de um *cluster* Linux da Dell baseado em mais de 6.400 nós servidores Dell PowerEdge, cada um com 2 processadores Intel Xeon E5 (Sandy Bridge) e um coprocessadores Intel Xeon Phi (arquitetura MIC), pertencente ao Centro de Computação Avançado do Texas (do termo, em inglês, *Texas Advanced Computing Center* e da sigla TACC) e localizado na Universidade do Texas. Cada nó computacional contém 32GB de memória RAM com um adicional de 8GB de memória em cada cartão do coprocessador Xeon Phi. Os nós computacionais são interconectados com a tecnologia Mellanox FDR InfiniBand em dois níveis (processadores e folhas) em uma topologia *fat-tree*. Todos os experimentos realizados nesse *cluster* não consideraram os coprocessadores Intel Xeon Phi.

Considerando as condições de contorno da Figura 46(a), os experimentos realizados com o SCC-A utilizaram a API na linguagem de programação Python da ferramenta ParaView para acessar e extrair dados científicos de uma região de interesse a partir de arquivos nos formatos XDMF e HDF5, após a execução da atividade *EdgeCFD Solver*. Conforme a especificação dos operadores Raw e RawI implementados nas atividades *Pre RDC* e *Solver RDC*, o usuário deve especificar o atributo que contém o caminho para o arquivo de dados científicos, assim como os atributos a serem extraídos do arquivo. Os resultados dessa operação são armazenados em arquivos no formato CSV.

Para utilizar o SCC-A, a base de dados de proveniência usando o SGBD PostgreSQL foi instanciada em um dos nós computacionais disponíveis para armazenar os dados de proveniência e os dados científicos nesse SGBD, exclusivamente. Esse nó computacional com a base de dados usou 16 núcleos de processamento ou processadores. O SCC-A requer uma conexão estável do nó mestre (ou nó central) com o nó da base de dados para obter as tarefas (ou ativações) prontas para serem executadas e para gerenciar os dados de proveniência.

6.2.4. Cluster LoboC

O *cluster* computacional Lobo Carneiro[21], também conhecido como LoboC, foi utilizado nos experimentos envolvendo a simulação computacional baseada no solucionador libMesh-sedimentation, na segunda instanciação da ARMFUL, que consistiu no desenvolvimento da biblioteca de componentes DfAnalyzer (Capítulo 5). O LoboC consiste de uma *cluster* SGI ICE-X Linux com 504 CPUs Intel Xeon E5-2670v3 (Haswell), totalizando 6.048 processadores, pertencente ao Núcleo Avançado de Computação de Alto Desempenho (NACAD) localizado na COPPE / UFRJ. Cada nó computacional contém 24 processadores (adicional de mais 24 processadores com *Hyper-Threading*) com 64GB de memória RAM. Os nós computacionais são interconectados com a tecnologia InfiniBand FDR – 56 Gbs (Hypercube). Além disso, esse *cluster* computacional apresenta uma partição para armazenamento de dados usando o sistema de arquivo paralelo Intel Lustre, com a capacidade de armazenamento de 500 TB de dados.

Para utilizar a DfAnalyzer, a base de dados de proveniência usando o SGBD MonetDB foi instanciada em um nó computacional dedicado para armazenar os dados de proveniência e os dados científicos nesse SGBD. Esse nó computacional com a base de dados possuía 24 processadores. A DfAnalyzer consome as requisições HTTP com os dados de proveniência e os dados científicos por meio dos seus serviços RESTful. Em seguida, por meio da validação desses dados, a DfAnalyzer estabelece uma conexão com a base de dados do MonetDB por meio do *driver* JDBC para carregar os dados extraídos.

Considerando o solucionador libMesh-sedimentation, a Figura 50 mostra a integração do libMesh-sedimentation com o cartucho do Extrator de Dados Científicos

[21] http://www.nacad.ufrj.br/pt/recursos/sgiicex

(RDE) da DfAnalyzer que utiliza a ferramenta ParaView Catalyst (Subseção 5.4). Esse cartucho permite o apoio à extração e à visualização de dados científicos *in situ*. Nesse cenário, a invocação do Catalyst consiste em *scripts* na linguagem de programação Python, que implementam *pipelines* para extrair e visualizar dados científicos presentes em cada ponto da malha.

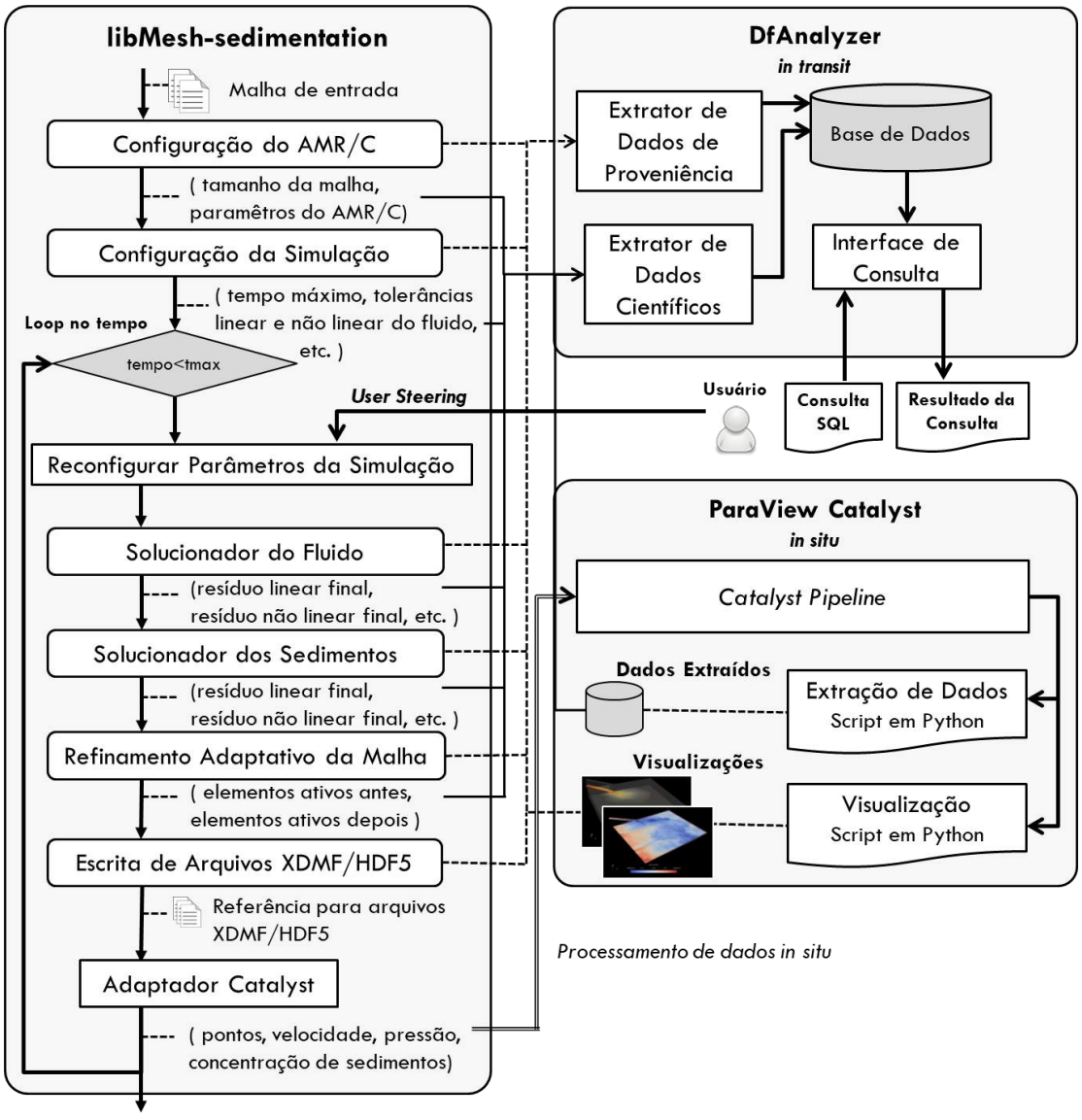


Figura 50. libMesh-sedimentation usando DfAnalyzer e ParaView Catalyst.

Mais especificamente, as invocações para o Extrator de Dados de Proveniência (PDE) estão presentes no código fonte do libMesh-sedimentation para capturar dados de proveniência a cada iteração (como apresentado na Subseção 5.4). Do mesmo modo, as

invocações para o componente Extrator de Dados Científicos (RDE) também estão presentes ao executar o cartucho do ParaView Catalyst.

Ademais, a Interface de Consultas (QI) fornece recursos para que os usuários submetam consultas em tempo de execução e visualizem fluxos de dados já armazenados na base de dados. Consequentemente, por meio da análise dos resultados das consultas processadas, os usuários podem identificar possíveis intervenções nos parâmetros de entrada do solucionador. Por esse motivo, o libMesh-sedimentation foi modificado para permitir adaptações nos parâmetros de entrada durante a execução da simulação.

No tanque *de Rooij e Dalziel*, um programa na linguagem de programação Python foi criado usando a interface de usuário do ParaView, em que uma sequência de filtros do ParaView foi aplicada para gerar o perfil de concentração dos sedimentos no plano x-z obtido para y = 1,0. Dessa forma, essa abordagem controla a frequência com que se deve invocar o adaptador do Catalyst da DfAnalyzer e escrever os dados científicos em arquivos. Nesse experimento, essas frequências apresentaram valores idênticos. A Figura 51 mostra o estado da simulação computacional com os perfis da concentração de sedimentos gerados pelo Catalyst em t = 10 e t = 20. Vale enfatizar que o processo de extração considerou a captura de dados científicos da malha em quatro linhas de interesse do usuário, conforme apresentado pelas linhas em vermelho e azul da Figura 48.

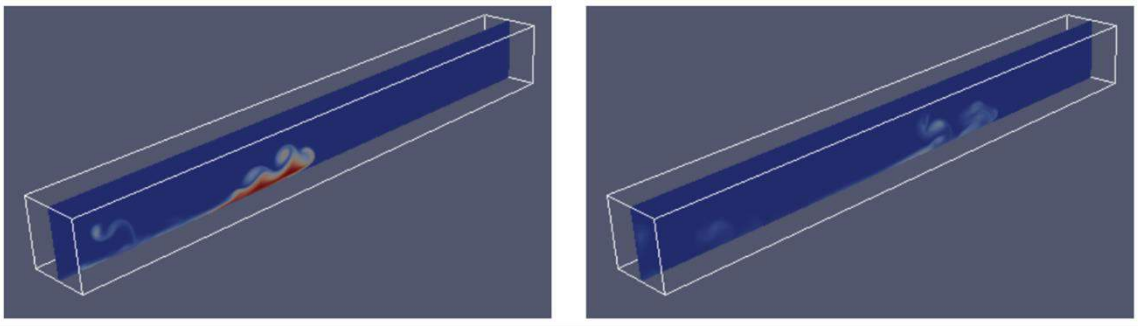


Figura 51. Perfis de concentração dos sedimentos em (a) t = 10 e (b) t = 20 no tanque de *de Rooij* e *Dalziel*.

Já no tanque real de batimetria, um *script* do ParaView Catalyst foi desenvolvido com a opção de prover visualizações quanto ao mapa de depósito durante a execução da simulação computacional, sendo que a frequência de invocação do Catalyst é 50 vezes maior que a frequência para a escrita de arquivos de dados científicos. Semelhante ao estudo

de caso anterior, o tanque real de batimetria também considera a extração de quatro linhas de interesse da malha pelo adaptador Catalyst ao longo da execução da simulação computacional.

6.3. Avaliação de custos

Nesta tese realizamos seis avaliações experimentais com o propósito de analisar o comportamento da ARMFUL em execuções reais, analisando especificamente os seguintes custos:

- **Custo da extração de dados científicos:** Medida de custo para extrair dados científicos usando o *workflow* científico Montage no SGWfC SCC-A. O custo de extração consiste no tempo decorrido no processamento do arquivo de dados científicos, no acesso ao conteúdo do arquivo, na indexação dos dados obtidos e no armazenamento dos dados em uma base de dados de proveniência;
- **Cargas de trabalho para a extração de dados científicos:** Avaliação do custo da extração de dados científicos usando diferentes cargas de trabalho nos *workflows* SyntheticOil&Gas e EdgeCFD usando o SGWfC SCC-A. Nessa avaliação consideramos duas análises, uma baseada em diferentes tempos de execução para as transformações de dados de acordo com uma função gama (CHIRIGATI *et al.*, 2012) e outra variando-se o tamanho dos arquivos de dados científicos manipulados pelo *workflow* científico. A primeira análise considerou apenas o *workflow* científico sintético, enquanto que a segunda análise contemplou os dois *workflows*;
- **Custo da indexação de dados científicos:** Avaliação do custo em termos de tempo ao aplicar técnicas de indexação de dados científicos nos *workflows* SyntheticOil&Gas e EdgeCFD. As técnicas de indexação aplicadas nesses *workflows* científicos utilizaram os cartuchos desenvolvidos na arquitetura RDC para o SCC-A;
- **Custo da carga de dados científicos:** Avaliação do custo em termos de tempo ao realizar a carga dos dados científicos extraídos ou dos índices gerados em uma base de dados relacional. Nesse experimento considerou-se o *workflow* EdgeCFD usando o SCC-A; e

- **Custo da captura de dados científicos em uma abordagem *in situ*:** Análise do custo da captura dos dados científicos e dos dados de proveniência em termos do tempo e do volume de dados a serem armazenados, além da análise do custo da carga dos dados em uma base de dados. Esse experimento contempla a execução da simulação computacional baseada no solucionador libMesh-sedimentation usando a biblioteca de componentes DfAnalyzer para apoiar a processamento, a análise e a visualização de dados *in situ*.
- **Custo de captura de dados científicos usando a DfAnalyzer e o noWorkflow:** Análise do custo em termos de tempo para a captura dos dados científicos e dos dados de proveniência durante a execução de simulações computacionais. Nessa análise, a proposta é comparar a sobrecarga de tempo de uma instância da ARMFUL, a DfAnalyzer, com o noWorkflow, ou seja, outro sistema que apoia a captura de dados de proveniência.

6.3.1. Custo da extração de dados científicos

Nessa primeira análise modelamos e executamos duas versões do *workflow* científico Montage com 1.585 arquivos de entrada no formato FITS baixados a partir do repositório 2MASS. A primeira versão desse *workflow* científico é modelada sem extratores, ou seja, o SCC-A gerencia apenas o fluxo de arquivos e apresenta o comportamento da versão original do SCC. Já a segunda versão realiza a extração de dados científicos em cada transformação de dados. Consequentemente, nessa versão há a gerência do fluxo de dados no nível lógico, ao contrário da primeira versão que se dedica apenas ao nível físico. Como a proposta dessa avaliação é identificar o custo da extração de dados científicos, comparou-se os tempos de execução para as duas versões do *workflow* científico. Cada uma das versões foi executada três vezes, sendo que consideramos a média de todos os tempos como o valor do tempo de execução a ser comparado.

No que diz respeito aos aspectos da execução desse *workflow* científico, ele é executado em aproximadamente 2 horas e 30 minutos usando 96 *cores* do *cluster* Uranus, gerencia em torno de 166.000 arquivos e necessita de mais de 500GB para armazenamento desses arquivos em disco. A Figura 52 mostra os resultados obtidos para essa avaliação experimental. O *workflow* científico Montage sem extração de dados científicos executou em 137,85 minutos, enquanto que o *workflow* científico com extração de dados científicos

executou em 143,15 minutos. Portanto, o acesso, a extração e a indexação de dados científicos presentes em arquivos numa base de dados apresentam um acréscimo de aproximadamente 3,84% no tempo total de execução desse *workflow* científico.

A partir desses resultados, também observamos que a extração de dados científicos corresponde a apenas 3,70% do tempo de execução da segunda versão do *workflow* científico Montage (5,30 minutos). Esse custo representa o tempo de processamento envolvido em invocar os programas de extração dos dados científicos, em acessar os dados científicos em arquivos e em gerar os índices no SGBD. Ou seja, esse custo corresponde à execução das atividades do *workflow* regidas pelo operador algébrico Raw. De forma análoga, se for adicionada uma ferramenta alternativa, como o FastBit, na arquitetura RDC acoplada ao SCC-A para realizar a indexação dos dados (*i.e.*, uso do operador algébrico RawI), ao invés de programas desenvolvidos pelo nosso grupo, espera-se que esse custo de indexação seja semelhante ao da extração de dados científicos. Ao mesmo tempo, a extração e a indexação de dados científicos também apresentam potenciais analíticos, uma vez que permite o processamento de consultas pelo usuário para analisar os dados científicos capturados ou mesmo o fluxo de dados nos níveis físico e lógico.

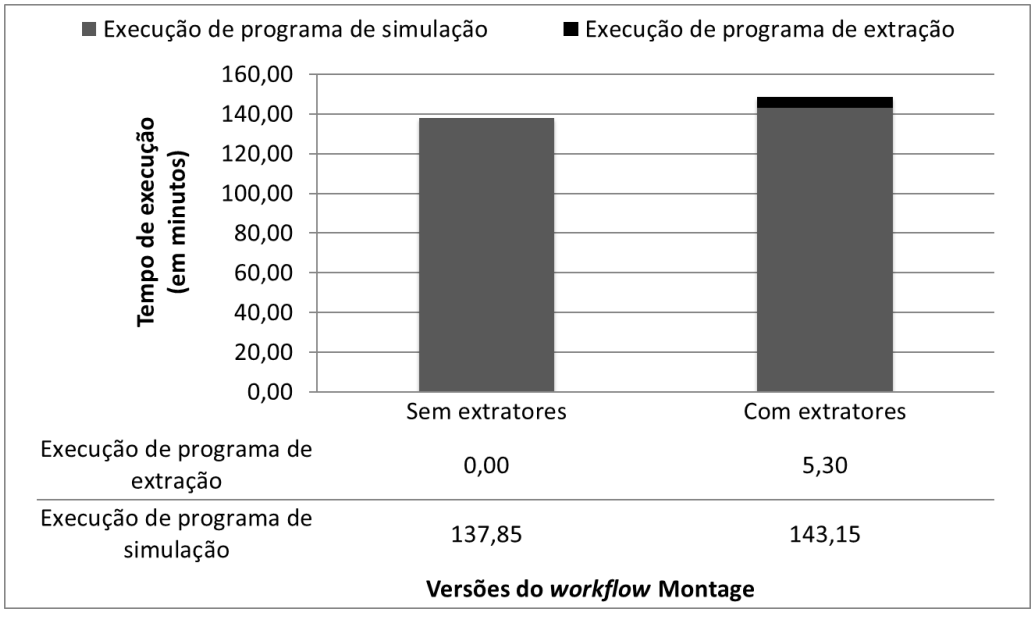


Figura 52. Custo da extração de dados científicos usando o *workflow* Montage.

Considerando as análises exploratórias sobre as simulações computacionais, uma das principais dificuldades encontradas consiste no ajuste fino das configurações durante a geração de fluxos de dados, como a definição de outros atributos e programas a serem

invocados pelas transformações de dados. Nessas circunstâncias, os usuários comumente necessitam modificar os valores dos atributos ou o algoritmo de uma determinada transformação de dados, para que os resultados apresentem a qualidade desejada ou que os programas de simulação não falhem em tempo de execução.

Para o *workflow* científico Montage, a modificação de valores de atributos pode significar o aumento da precisão das imagens finais ou da complexidade computacional do algoritmo, por exemplo, de projeção de imagens. Com isso, aumenta-se o tempo de execução do *workflow* científico e, consequentemente, o tempo de extração de dados científicos também pode aumentar, pois os arquivos de dados científicos podem apresentar mais resultados relevantes armazenados (*i.e.*, maior volume de dados). Na Subseção 6.3.2 avaliamos a influência da carga de trabalho no custo da extração de dados científicos, variando-se o tempo médio de execução das transformações de dados e o volume de dados armazenados em arquivos de dados científicos.

6.3.2. Cargas de trabalho para a extração de dados científicos

Nesse experimento utilizamos o *workflow* SyntheticOil&Gas, a fim de avaliar o custo da extração de dados científicos ao modificar a complexidade de executar uma transformação de dados e o volume de dados armazenados em arquivos de dados científicos. Como discutido na Subseção 6.1.2, os fatores de fragmentação dos dados e de filtro foram fixados para esse *workflow* científico. A primeira análise desse experimento avaliou o custo da extração de dados científicos ao variar o tempo médio de execução das transformações de dados, sendo esse tempo determinado pelo fator de custo da transformação de dados, do termo em inglês *Data Transformation Cost Factor* (DTCF). De acordo com o valor de DTCF informado, os tempos de execução das transformações de dados seguem uma distribuição gama $\Gamma(\kappa, \theta)$, em que $\kappa = 2^{\mathrm{DTCF}}$ e $\theta = 1$, para $\mathrm{DTCF} \geq 0$.

A Tabela 8 apresenta os resultados obtidos para os valores de DTCF iguais a 5, 7 e 8 usando 96 *cores* no *cluster* Uranus. As medidas apresentadas nessa tabela foram baseadas em três execuções desse *workflow* científico, sendo todas elas submetidas nas mesmas condições. Consideramos também um nível de confiança maior que 95% para o tempo de execução do *workflow* científico. Nessa análise, o tempo de extração dos dados científicos pode ser compreendido pelo tempo de processamento computacional para executar os programas de extração e carregar os dados científicos na base de proveniência. Somando-

se a isso, fixou-se nessa análise o tamanho dos arquivos de dados científicos, consequentemente, a extração de dados científicos apresentou tempos semelhantes em todas as configurações.

De forma análoga, podemos observar na Tabela 8 que DTCF = 5 apresentou o maior custo percentual de extração de dados científicos (2,42%), pois o tempo de execução do *workflow* científico é o menor de todas as configurações consideradas (*i.e.*, menor contribuição percentual no tempo médio de execução do *workflow* científico). Entretanto, ao aumentar o valor de DTCF (*i.e.*, aumentar o tempo médio de execução do *workflow* científico), observamos que a extração de dados científicos apresenta uma menor sobrecarga percentual, pois o processamento computacional necessário para a extração de dados científicos continua semelhante (*i.e.*, mesmos dados de saída, ou seja, mesmo volume de dados científicos armazenados em arquivos a serem analisados). No cenário de simulações computacionais em larga escala, que levam muito tempo executando modelos computacionais complexos (*i.e.*, transformações de dados com DTCF > 7), pode-se destacar que o tempo de extração é menor que 0,50% do tempo total de execução do *workflow* científico.

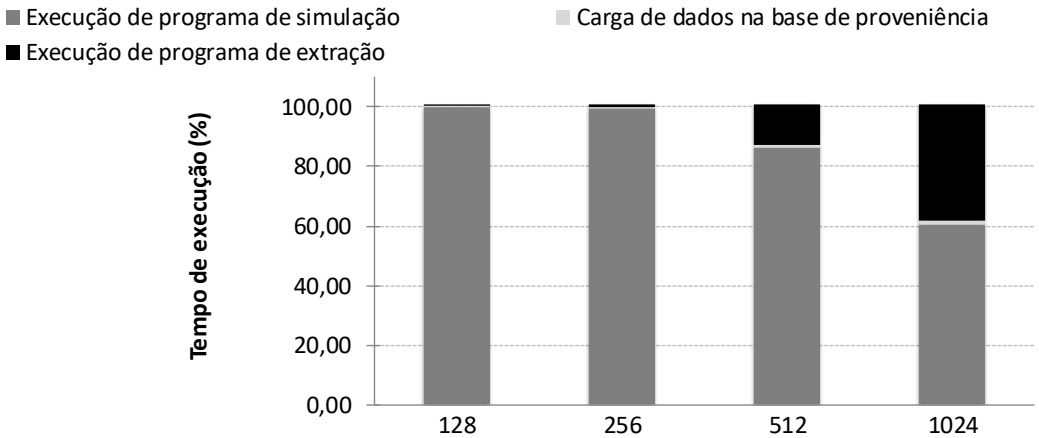
Tabela 8. *Workflow* SyntheticOil&Gas considerando diferentes custos para as transformações de dados usando 96 *cores*.

Tempo médio de execução do *workflow* (DTCF)	Média do tempo de execução de uma transformação de dados (em segundos)	Média do tempo de execução do *workflow* científico (em minutos)	Desvio padrão para o tempo de execução do *workflow* científico (em minutos)	Tempo de extração de dados científicos (em minutos)	Tempo de extração de dados científicos (%)
5	32	40,89	2,39	0,010	2,42
7	128	122,79	8,53	0,006	0,41
8	256	242,31	9,36	0,007	0,22

A segunda análise lida com uma avaliação da extração de dados científicos ao variar o tamanho dos arquivos de dados científicos. Nessa análise definimos um valor fixo de DTCF = 6 e variamos o volume de dados armazenados em arquivos de dados científicos segundo o fator de tamanho do arquivo, conhecido pelo termo em inglês *File Size Factor* (FSF). De acordo com o valor de FSF definido, o tamanho dos arquivos de dados científicos em megabytes (∂) muda conforme a seguinte equação: $\partial = 2^{\mathrm{FSF}}$, para $\mathrm{FSF} \geq 0$. Considerou-se os valores 7, 8, 9 e 10 para FSF usando o *cluster* Uranus com 64 *cores*. A

Figura 53 mostra a contribuição percentual do tempo de execução dos programas de simulação (*i.e.*, barras de cor cinza escuro) e do tempo de extração dos dados científicos, que é composto de dois aspectos: o tempo para invocar os programas de extração (*i.e.*, barras de cor preta) e o tempo para carregar os dados na base de proveniência (*i.e.*, barras de cor cinza claro).

Pelos resultados da Figura 53, observamos que a carga de dados na base de proveniência é desprezível em comparação com as contribuições percentuais para executar o *workflow* científico e invocar os programas de extração de dados científicos. Para a extração de dados científicos, o seu custo está diretamente associado ao tamanho dos arquivos (*i.e.*, quantidade de dados científicos a serem extraídos) e ao programa de extração empregado. Desse modo, ao aumentar o tamanho dos arquivos, observamos um aumento considerável na contribuição percentual da extração de dados. Na nossa abordagem, os dados extraídos foram definidos pelo próprio usuário. Nesse caso foi realizada a seleção de todos os atributos. Ao mesmo tempo, o próprio usuário poderia tanto aumentar como reduzir o processamento computacional envolvido na extração, de acordo com a quantidade de atributos a serem acessados. Além disso, a definição dessa quantidade de atributos está diretamente relacionada às consultas que esse usuário realizará futuramente.



	128	256	512	1024
Execução de programa de extração	0,11	0,66	13,77	39,35
Carga de dados na base de proveniência	0,34	0,61	0,89	1,05
Execução de programa de simulação	99,89	99,34	86,23	60,65

Tamanho dos arquivos de dados científicos (MB)

Figura 53. *Workflow* SyntheticOil&Gas considerando diferentes tamanhos para os arquivos de dados científicos.

Considerando-se um cenário de larga escala, ou seja, que um grande volume de dados científicos é armazenado em arquivos, os usuários devem extrair apenas os elementos de dados relevantes para apoiar as análises da simulação computacional. Para isso, condições de filtro (*i.e.*, seleção de elementos de dados) precisam ser definidas para que o custo da extração de dados não seja alto, assim como o custo de carregar os dados de proveniência e os dados científicos na base de dados. Nesse sentido, esse experimento também contemplou a execução do *workflow* EdgeCFD usando o SCC-A com 240 *cores* no *cluster* Stampede para investigar o custo em termos de tempo da extração de dados científicos, da execução das atividades do *workflow* e da carga de dados na base de dados, ao variar a estratégia de extração de dados.

A execução desse *workflow* considerou 4.800 combinações de valores de atributos de entrada (*e.g.*, atributos de termo de força e de viscosidade do fluido), que manipulou 86.400 arquivos no formato HDF5 e XDMF e necessitou de aproximadamente 1,09 TB de espaço de armazenamento em disco. Como conjunto de dados de entrada, considerou-se a malha fina para esse *workflow*, conforme discutido na Subseção 6.1.3. Para comparar os custos em termos de tempo associados ao armazenamento do conteúdo em arquivos de dados científicos em uma base de dados, calculou-se a média do tempo de execução a partir de três execuções de cada versão do *workflow*.

Ademais, duas abordagens foram propostas para a extração de dados científicos: parcial e total. Os resultados para cada abordagem são apresentados na Tabela 9, assim como o custo em termos do tempo de executar o *workflow* EdgeCFD sem a extração de dados científicos. Pelos resultados obtidos, pode-se observar que ao aumentar o volume de dados extraídos de arquivos, aumentou-se o tempo para acessar os dados científicos nos arquivos (*i.e.*, tempo de execução do programa de extração). Consequentemente, pelo fato de aumentar o número de elementos de dados extraídos, o custo de carga dos dados no SGBD também aumentou. Portanto, assim como observado no experimento com o *workflow* SyntheticOil&Gas, os resultados com o *workflow* EdgeCFD apontaram que os elementos de dados relevantes para o usuário de domínio em suas análises têm fator determinante na sobrecarga da captura de dados científicos, uma vez que determina o volume de dados extraídos dos arquivos e um esforço em termos de processamento computacional para carregar os dados na base de dados de proveniência.

Tabela 9. *Workflow* EdgeCFD considerando a extração parcial e total de dados científicos.

Versão do *workflow*	Tempo total de execução em minutos				
	Execução das atividades do *workflow*	Extração de dados científicos	Carga de dados no SGBD	*Workflow*	Desvio padrão para o tempo de execução do *workflow*
sem extração	37,22	0,00	2,58	39,80	1,35
extração parcial	36,72	2,36	16,52	55,60	3,04
extração total	41,03	6,21	84,36	131,60	3,66

Outra opção consiste em o usuário não realizar a extração de dados científicos. Logo, os usuários limitariam as suas consultas ao fluxo de arquivos e realizariam apenas análises dos dados científicos de forma isolada (*i.e.*, investigação de apenas um arquivo por vez). Mesmo assim, os usuários ainda dependeriam dos programas de extração (*e.g.*, ferramentas alternativas como FastBit) para realizar as análises dos dados científicos, o que implica dizer que o custo de extração ainda existiria e os usuários não tirariam proveito do paralelismo apoiado pelo SCC-A.

Concomitantemente, esse seria um processo manual, enquanto que a implementação da ARMFUL exigiria apenas informações como quais arquivos precisam ser investigados e quais atributos de interesse precisam ser capturados. Portanto, a execução do *workflow* científico usando o SCC-A apresenta vantagens quanto ao potencial analítico provido com a extração de dados científicos com a ressalva de que os usuários devem definir quais coleções de dados (*i.e.*, conjunto de elementos de dados) eles têm interesse em capturar para apoiar consultas durante ou após a execução. Com a extração parcial de apenas os elementos de dados de interesse, os usuários podem reduzir esse custo de carga de dados na base de proveniência (*i.e.*, dependendo da quantidade de dados a serem extraídos) e tirar proveito do paralelismo.

6.3.3. Custo da indexação de dados científicos

Além dos experimentos apresentados nas seções anteriores, realizamos uma avaliação experimental para identificar a sobrecarga de duas abordagens diferentes para a captura de dados científicos, baseadas (i) apenas na extração e carga de dados científicos (chamado de carga de dados); e (ii) na captura e indexação de dados científicos (chamado de indexação de dados). Nesse experimento, considerou-se um *workflow* científico

composto de apenas uma transformação de dados regida pelo operador algébrico SplitMap, que simula o comportamento de um programa de simulação no domínio de Óleo e Gás. Semelhante aos experimentos anteriores, assumiu-se dois fatores para a carga de trabalho: DTCF (tempo de execução do programa de simulação) e FSF (tamanho dos arquivos de dados científicos). O fator DTCF apresentou um valor fixo igual a 6, que corresponde a um tempo médio de 64 segundos para a execução do programa de simulação. Por outro lado, o valor de FSF apresentou diferentes configurações para a análise da escalabilidade de cada abordagem na captura de dados científicos. Os valores de FSF foram os seguintes: 10, 11, 12, 13 e 14.

Os arquivos de dados científicos utilizados nesse experimento seguem o formato CSV, sendo que cada linha de um arquivo contém 4 colunas. Nesse sentido, a primeira linha apresenta o cabeçalho (nomes dos atributos) e as linhas subsequentes possuem os valores assumidos pelos atributos. Como entrada desse fluxo de dados, considerou-se um conjunto de dados composto de 256 elementos de dados de entrada. Consequentemente, para a execução do *workflow* científico com FSF igual a 10, espera-se que 256 arquivos no formato CSV sejam manipulados, produzindo 262.144 elementos de dados no conjunto de saída dessa transformação de dados.

A Figura 54, a Figura 55 e a Figura 56 mostram os resultados obtidos ao executar esse *workflow* científico no *cluster* Suno do Grid 5000 com 32 *cores*. Na Figura 54 mostramos o custo em termos de tempo, em segundos, para capturar os dados científicos usando esse *workflow* científico sintético para as duas abordagens. Em todas as configurações, observamos que o custo de indexação dos dados é menor que o custo de carga dos dados. Além disso, o custo da captura de dados aumenta de acordo com que o tamanho do arquivo de dados científicos também aumenta. Tal comportamento é esperado pelo fato dos arquivos apresentarem mais conteúdo a ser acessado, extraído e, eventualmente, indexado. Ao mesmo tempo, esse *workflow* científico considera uma extração de dados científicos total, ou seja, todos os atributos presentes nos arquivos de formato CSV são acessados e extraídos.

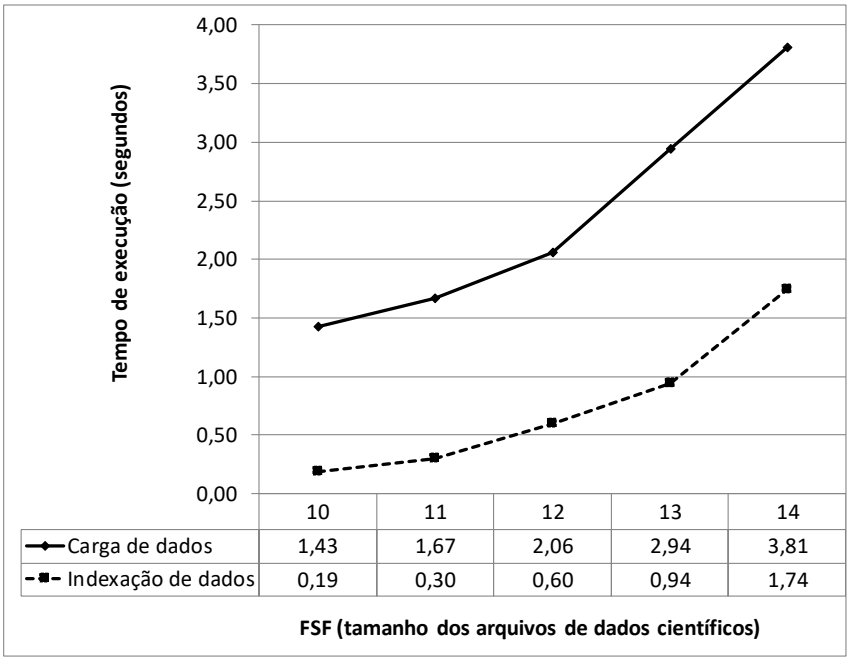


Figura 54. Comparação dos tempos para executar programa de extração nas abordagens de carga e indexação de dados.

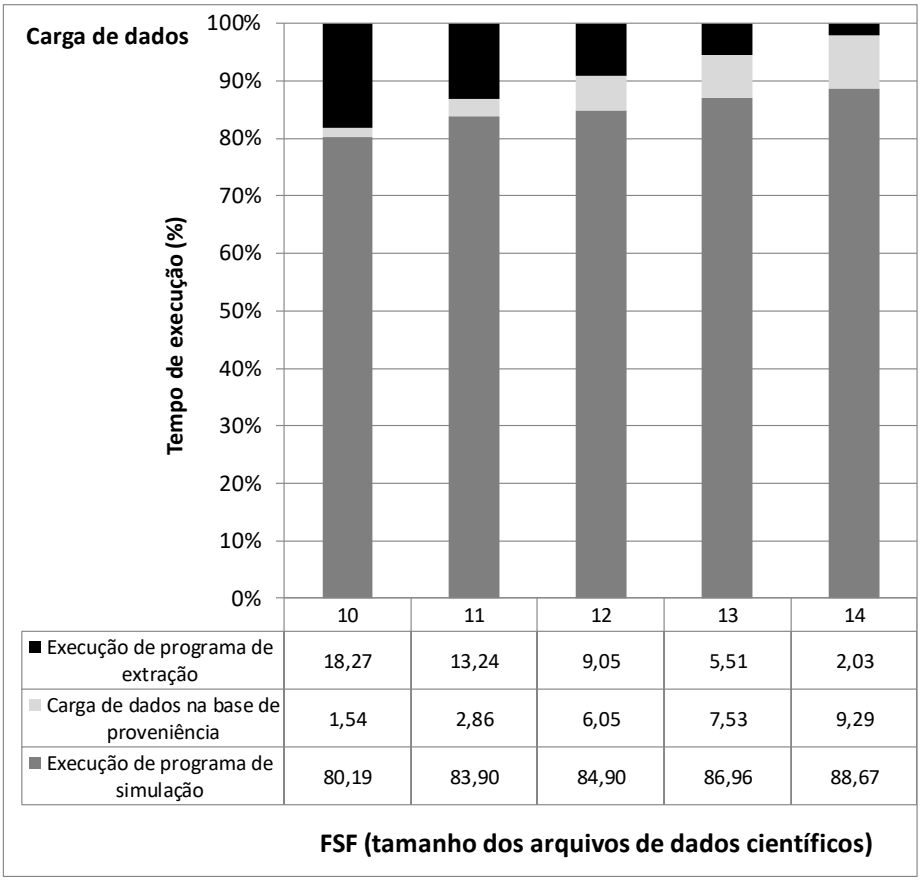


Figura 55. *Workflow* sintético com programa de extração baseado na carga de dados.

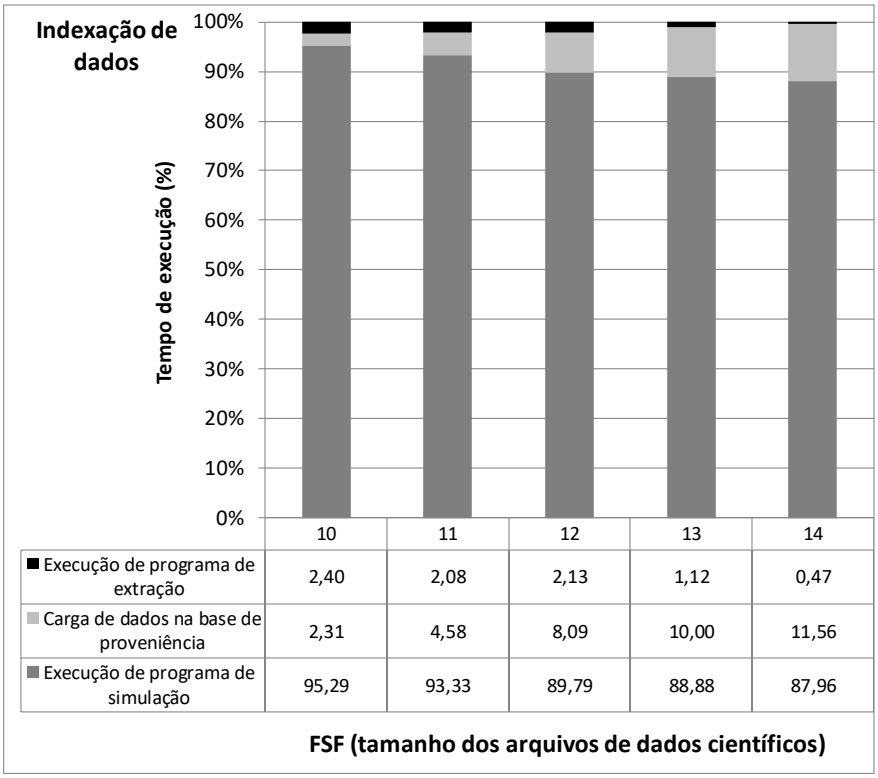


Figura 56. *Workflow* sintético com programas de extração baseados na indexação de dados.

Já a Figura 55 e a Figura 56 mostram que, ao aumentar o tamanho dos arquivos de dados científicos, o tempo de execução necessário para capturar os dados científicos apresenta uma redução da sua representatividade percentual em relação ao tempo total de execução do *workflow* científico. Esse comportamento pode ser observado por dois fatores. Primeiramente, ao capturar mais dados científicos, maior a sobrecarga para armazenar os dados científicos em um determinado repositório (*i.e.*, base de proveniência). Outro fator que justifica a redução percentual da extração de dados é o custo de gerar os arquivos no formato CSV durante a execução do *workflow* científico. Esse custo está adicionado ao custo de executar o programa de simulação, uma vez que tais programas normalmente são os responsáveis pela geração dos arquivos específicos do domínio no cenário científico real.

Ademais, podemos destacar que ambas as abordagens para o acesso aos dados científicos apresentam uma redução do tempo gasto para capturar os dados científicos ao considerar o aumento do tamanho dos arquivos em termos de proporção, como apresentado em mais detalhes em SILVA *et al.* (2017b). Por exemplo, ao duplicar o tamanho dos arquivos (de FSF igual a 10 para 11), o tempo de extração aumentou 16,78% e o tempo de indexação aumentou 57,89%, enquanto que, em um caso linear, o aumento esperado seria

de 100%. Entretanto, o custo de armazenamento dos dados na base de proveniência aumentou significativamente em função da seleção de todos os elementos de dados. Por último, podemos destacar que a abordagem de indexação poderia apresentar resultados melhores se, ao invés de indexar cada valor de atributo no arquivo de formato CSV, a indexação fosse baseada em cada linha (ou elemento de dados) do arquivo no formato CSV. Assim, ao realizar uma consulta, o processador de consultas teria que utilizar o índice e identificar o atributo de interesse nos arquivos de dados científicos.

Outra evidência desse experimento contempla o espaço de armazenamento da base de dados de proveniência. Ao utilizar a abordagem de indexação de dados científicos, a base de proveniência consumiu mais espaço de armazenamento (aproximadamente 665MB para a configuração de FSF=14) que a abordagem de carga de dados (aproximadamente 255MB para a configuração de FSF=14).

Outra análise foi realizada a partir da execução do *workflow* EdgeCFD usando o SCC-A no *cluster* Stampede com 240 *cores* para investigar o custo da indexação dos dados científicos em arquivos nos formatos XDMF e HDF5, ao variar a carga de trabalho, ou seja, o volume de dados armazenados nesses arquivos. Mais especificamente, essa análise considera o custo da indexação em termos de tempo sequencial ao variar o número de vértices na malha de entrada (*i.e.*, tamanho da malha), assim como o algoritmo de indexação dos dados científicos (índices posicionais e *bitmap*).

A Figura 57 apresenta os resultados obtidos para as quatro versões do *workflow* (*baseline*, *carga*, *idx-posicional*, *idx-bitmap*), sendo que se calculou o tempo médio de geração dos índices a partir de três execuções para cada versão do *workflow*. Vale ressaltar também que a versão *baseline* do *workflow* EdgeCFD não realiza a extração, a indexação, nem a carga de dados em um SGBD usando o SCC-A, isto é, ela consiste em uma versão sem apoio à captura de dados científicos. Enquanto isso, a versão *carga* considera a extração de dados científicos de arquivos nos formatos XDMF e HDF5 e a sua carga na base de dados do SCC-A.

Ao aplicar algoritmos de indexação, o custo em termos de tempo para gerar os índices internos no SGBD deve ser agregado ao custo para gerar os índices de acesso aos dados científicos em arquivos. Além disso, como a versão *baseline* do *workflow* não extrai dados científicos de arquivos, ela também não gera índices via SGBD e, dessa forma, o

tempo sequencial é igual a zero para indexar dados científicos. Já a versão *carga* do *workflow* apresenta o maior tempo sequencial para indexar dados científicos, quando comparado com os algoritmos de indexação. Nesse caso, essa abordagem é aproximadamente duas vezes (2x) mais lenta que o algoritmo de indexação posicional, quando se considera malhas mais finas, semelhante ao cenário real de simulações computacionais de CFD. Sendo assim, observa-se uma redução de 64,95% e 46,39% na sobrecarga de tempo para a geração de índices, quando se aplicou os algoritmos de indexação posicional e *bitmap*, respectivamente, ao invés de uma abordagem de extração e carga dos dados científicos em uma base de dados.

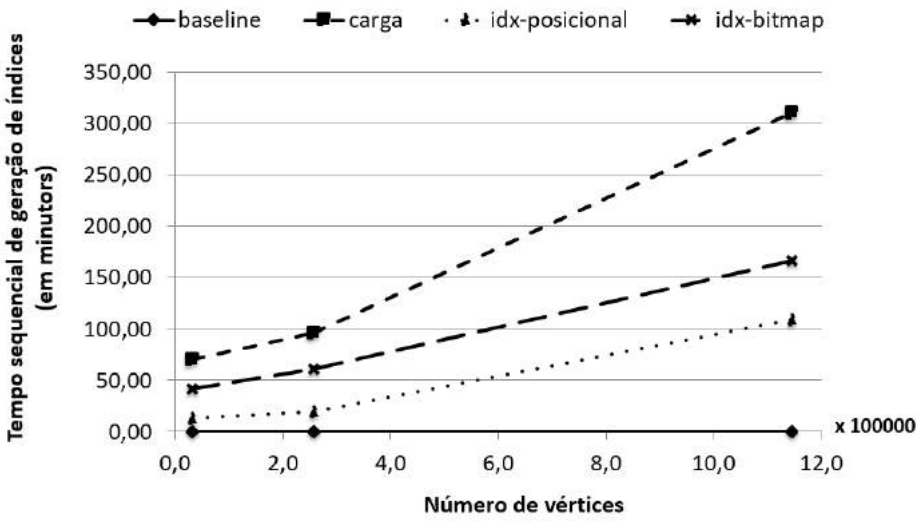


Figura 57. Tempo sequencial de geração dos índices com diferentes cargas de trabalho.

Ao analisar exclusivamente os algoritmos de indexação, conclui-se que o algoritmo de indexação bitmap, que utiliza a ferramenta FastBit, apresenta um desempenho em termos de tempo pior que o algoritmo de indexação posicional. Em função dos diferentes valores assumidos pelos atributos indexados (*i.e.*, domínio grande de valores possíveis para cada atributo) em arquivos de dados científicos, o algoritmo de indexação *bitmap* gerou um mapa de *bits* esparso que consiste em uma matriz com muitas colunas e diferentes valores zerados, que representam a não ocorrência de um determinado valor para o atributo representativo desse mapa de *bits* (mais detalhes na Seção 2.1).

Portanto, esse algoritmo de indexação demanda mais tempo para gerar os índices e requer mais espaço em disco. Por exemplo, o algoritmo de indexação posicional requer 4,4GB a menos que o algoritmo de indexação *bitmap* com malhas finas ao considerar o

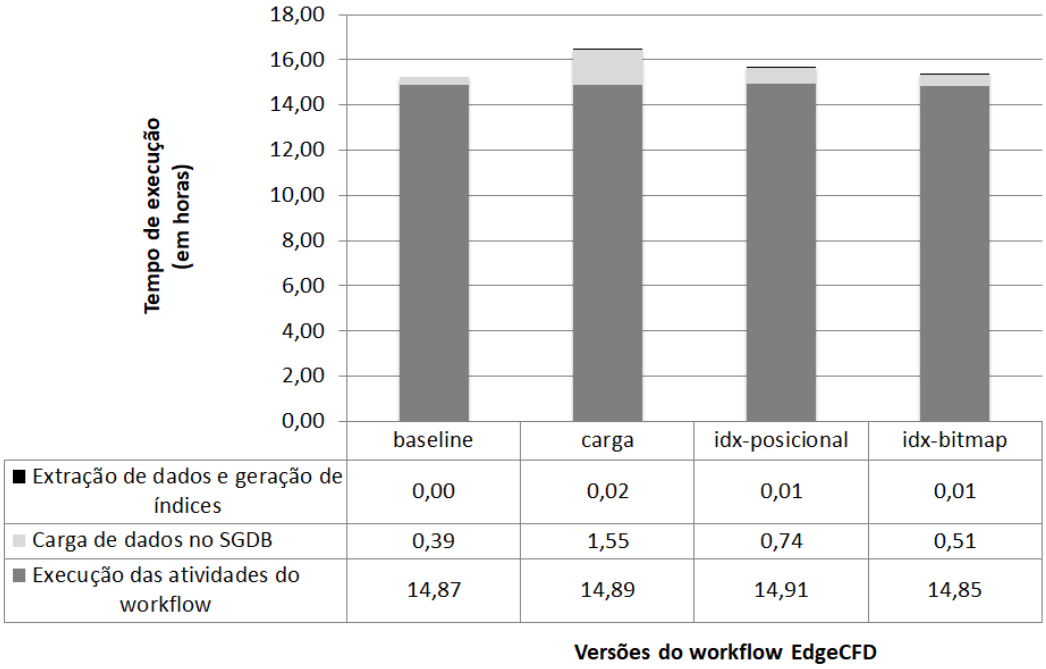
armazenamento em disco da base de dados de proveniência. Embora a ferramenta FastBit apresente recursos para permitir que os usuários modifiquem as configurações do algoritmo de indexação, os experimentos realizados não consideraram essas configurações avançadas do FastBit. Nesse caso, o custo de indexação do FastBit poderia ser reduzido, caso o mapa de *bits* fosse gerado em função de intervalos de valores, ou seja, baseados em inequidades. Como resultado de um trabalho de colaboração, a dissertação do M.Sc. José Vitor Leite apresenta avaliações mais detalhadas sobre o uso dos algoritmos do FastBit ao executar esse *workflow* científico.

6.3.4. Custo da carga de dados

Dependendo da abordagem de extração ou indexação de dados científicos adotada, o custo em termos de tempo para carregar os dados em um repositório externo ou uma base de dados pode variar. Nesse sentido, o custo da carga de dados foi avaliado a partir da execução do *workflow* EdgeCFD usando o SCC-A no *cluster* Stampede com 240 *cores*, ao variar a abordagem de captura de dados científicos. Nesse experimento considerou-se as seguintes abordagens: *baseline* (sem a captura de dados científicos), *carga* (extração e carga de dados em um SGBD, sendo que a geração de índices é realizada internamente pelo SGBD), *idx-posicional* (indexação posicional), *idx-bitmap* (indexação usando o FastBit para a geração de mapas de *bits*). Esse *workflow* foi executado com malhas finas de entrada para representar um cenário real no domínio de CFD. Vale ressaltar que apenas parte dos dados científicos foram capturados dos arquivos gerados, ou seja, consiste em uma abordagem de extração parcial dos dados.

A Figura 58 mostra os custos de executar as atividades do *workflow*, de capturar os dados científicos por meio da extração ou indexação dos dados, e de carregar os dados no SGBD para cada versão do *workflow*. Nesse caso, observou-se que o custo em termos de tempo de carregar apenas os dados de proveniência prospectiva e retrospectiva foi igual a, aproximadamente, 23,40 minutos (versão *baseline* do *workflow* que não realiza extração/indexação de dados científicos). Com esses resultados, observou-se que a versão *carga* do *workflow* requer 72,20 minutos a mais que a versão *baseline*, e 24,20 minutos com relação à pior versão que aplica uma técnica de indexação de dados científicos, *i.e.*, *idx-posicional*.

Além disso, a versão *carga* do *workflow* apresenta uma sobrecarga em termos de tempo para armazenar os dados científicos igual a 69,75 minutos, se desconsiderarmos o custo de armazenamento dos dados de proveniência, que pode ser observado na versão *baseline* do *workflow* (que não realiza a extração ou a indexação de dados científicos). Em relação às abordagens de indexação posicional e *bitmap*, essa sobrecarga foi de 20,91 e 7,37 minutos, respectivamente. Ao mesmo tempo, o custo de executar, no pior caso, o algoritmo de indexação é de aproximadamente 79 segundos, o que representa 0,13% do tempo de execução do *workflow*. Sendo assim, a carga de dados científicos em um SGBD pode apresentar uma sobrecarga considerável na análise exploratória de dados científicos, mesmo em cenários com a extração parcial dos dados científicos. Por outro lado, o uso de técnicas de indexação de dados científicos pode reduzir as sobrecargas em termos do tempo de execução do *workflow*.



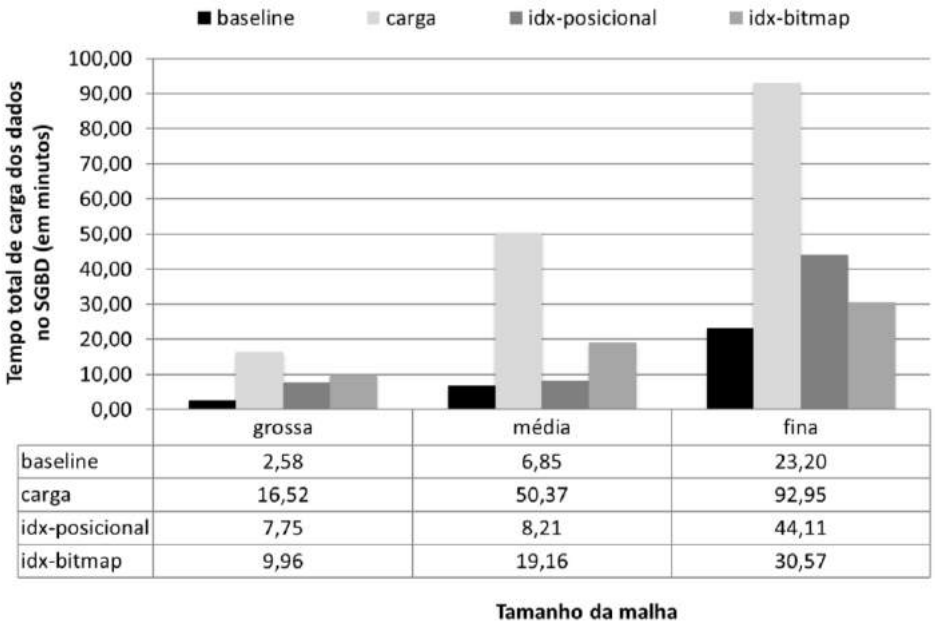
	baseline	carga	idx-posicional	idx-bitmap
Extração de dados e geração de índices	0,00	0,02	0,01	0,01
Carga de dados no SGDB	0,39	1,55	0,74	0,51
Execução das atividades do workflow	14,87	14,89	14,91	14,85

Figura 58. Carga de dados no SGBD do SCC-A usando o *workflow* EdgeCFD.

Outra avaliação realizada baseou-se na investigação do custo de carga dos dados com diferentes cargas de trabalho ainda utilizando o *workflow* EdgeCFD. Nesse caso, os três tipos de malhas foram considerados, variando-se o número de vértices na malha: grossa, média e fina. Logo, para cada tamanho da malha, analisou-se o custo de carregar os dados na base de dados de proveniência, variando-se o apoio à extração de dados científicos e o uso de um algoritmo de indexação (posicional ou *bitmap*).

A Figura 59 apresenta o custo para as quatros versões do *workflow* EdgeCFD. Primeiramente, observou-se que o custo da carga de dados aumenta de acordo que o tamanho da malha também aumenta, já que mais dados de proveniência, de execução e científicos devem ser armazenados na base de dados do SCC-A. Em todas as versões da malha, a versão *carga* do *workflow* apresenta a maior sobrecarga na carga de dados, uma vez que ela acessa o conteúdo nos arquivos de dados científicos, converte tais dados em estruturas de dados específicas e carrega os dados estruturados na base de dados de proveniência integrada.

Por outro lado, as versões do *workflow* baseadas em algoritmos de indexação apresentam estruturas de dados otimizadas para referenciar os dados científicos, como discutido na Seção 2.1. Logo, os resultados experimentais mostram que houve uma redução de 52,55% na sobrecarga de tempo para a carga de dados no SGBD, quando se aplicou a indexação posicional (aproximadamente 44,11 minutos) nesse *workflow* com malhas finas de entrada, em comparação com a versão *carga* do *workflow*. Nessa mesma configuração, a versão *carga* do *workflow* apresentou uma sobrecarga de 92,95 minutos.



	grossa	média	fina
baseline	2,58	6,85	23,20
carga	16,52	50,37	92,95
idx-posicional	7,75	8,21	44,11
idx-bitmap	9,96	19,16	30,57

Figura 59. Custo da carga de dados com diferentes cargas de trabalho.

Considerando a versão *idx-bitmap* do *workflow* que utiliza a ferramenta FastBit, apenas o caminho para o arquivo de índice gerado (*i.e.*, catálogo que pode ser utilizado no processamento de consulta) é carregado na base de dado para permitir a redução do tempo de execução da carga de dados e do espaço em disco necessário para a base de dados, como

apresentado na Tabela 10. Apesar da redução da sobrecarga ao aplicar a indexação *bitmap*, essa abordagem ainda apresenta um custo em termos de tempo superior para gerar os índices em relação à sobrecarga introduzida pelo algoritmo de indexação posicional. Somando-se a isso, a versão *idx-bitmap* do *workflow* contempla outra sobrecarga relacionada à consulta dos dados científicos, uma vez que ela precisa acessar o caminho dos arquivos de índices armazenados na base de dados e usá-los no processador de consultas da ferramenta FastBit para acessar os dados científicos presentes nos arquivos. Diferente dessa proposta, a versão *idx-posicional* do *workflow* utiliza diretamente os índices posicionais gerados e armazenados na base de dados para acessar os dados científicos em arquivos.

Tabela 10. Armazenamento da base de dados do SCC-A usando o *workflow* EdgeCFD.

Versão do *workflow*	Espaço de armazenamento em GB		
	Malhas grossas	**Malhas médias**	**Malhas finas**
baseline	2,88	2,98	3,20
carga	4,80	5,76	6,40
idx-posicional	5,44	5,56	5,73
idx-bitmap	3,01	3,05	3,20

6.3.5. Custo da captura de dados em uma abordagem *in situ*

Essa avaliação experimental analisou o custo da captura de dados em uma abordagem de processamento de dados *in situ* ao executar uma simulação computacional baseada no solucionador libMesh-sedimentation. Mais especificamente, analisou-se o custo em termos de tempo de realizar a captura de dados científicos ainda alocados na memória utilizando o componente RDE da DfAnalyzer, de executar os componentes da biblioteca libMesh e de capturar dados de proveniência usando a DfAnalyzer. Nesse sentido, dois estudos de caso foram realizados nessa simulação computacional, conforme apresentado na Subseção 6.1.4: um tanque de sedimentação de Rooij e Dalziel (DE ROOIJ e DALZIEL, 2009) e um tanque real de batimetria. Ambos estudos de caso utilizaram 480 *cores* do *cluster* LoboC.

A Tabela 11 mostra o tempo de execução de cada etapa dessa simulação computacional. O tempo de execução necessário para utilizar as ferramentas de captura e análise de dados foi de 6,41% do tempo total de execução da simulação computacional, sendo que 6,33% corresponde à extração e à visualização de dados científicos em uma abordagem *in situ*, e o restante (0,08%) é oriundo da captura de dados de proveniência

retrospectiva. A instrumentação do código fonte demonstra que o custo computacional em termos de tempo para executar o adaptador Catalyst depende da quantidade de filtros aplicados nos *scripts* em Python e da frequência de invocação do Catalyst. Mais específico do Catalyst, a extração de dados demanda mais tempo de execução que a visualização. Além disso, o maior custo relacionado ao solucionador consiste no uso de outros componentes do libMesh com 33,67% do tempo total da execução da simulação, seguido da solução não-linear das equações do fluido, que requer aproximadamente 32,67%, e do refinamento e desrefinamento da malha (AMR/C) com 20,70%.

Tabela 11. Tempo de execução para as diferentes contribuições da simulação de sedimentação baseada no tanque proposto por de Rooij e Dalziel.

Contribuição em termos de tempo	Tempo de execução (em segundos)	Custo por invocação	Tempo de execução (%)
Solucionador do Fluido	16.203,71	0,87	32,67%
Solucionador do Sedimento	2.797,36	0,15	5,64%
AMR/C	10.268,32	17,11	20,70%
Escrita de dados em arquivos XDMF/HDF5	453,96	4,93	0,92%
Extração e visualização de dados *in situ*	3.137,24	33,73	6,33%
Proveniência (DfAnalyzer)	38,47	0,01	0,08%
Outros componentes do libMesh	16.598,00	-	33,67%
Total	62.171,00		

Apesar do custo em termos de tempo para extrair e visualizar os dados *in situ* ser maior que a escrita de dados científicos em arquivos nos formatos XDMF e HDF5, as operações em disco podem ser uma limitação em função da capacidade de armazenamento de dados nessa infraestrutura. Vale ressaltar que o intervalo em número de passos de tempo para armazenar os dados em disco é grande o suficiente para não apresentar gargalos em termos de desempenho, conforme discutido na Subseção 6.1.4. Enquanto isso, as invocações do Catalyst são feitas em uma frequência 50 vezes maior, para que dados parciais, não armazenados em arquivos, sejam extraídos em tempo de execução. A Tabela 12 mostra o espaço em disco necessário para armazenar os dados científicos extraídos, as visualizações (*i.e.*, imagens no formato PNG), os dados de proveniência e os arquivos de dados científicos produzidos pela execução dessa simulação. A partir desses resultados, ressalta-se que a base de dados de proveniência requer apenas 2,96% do espaço em disco

necessário para armazenar os arquivos de dados científicos, enquanto que os arquivos de visualização usam apenas 2,48% desse espaço.

Tabela 12. Armazenamento de dados para o tanque de sedimentação de Rooij e Dalziel.

Tipo de dados	Espaço de armazenamento (em MB)	Número de arquivos
Arquivos de visualização	248,00	62
Dados extraídos	2,48	248
Proveniência (DfAnalyzer)	296,01	75.696
Dados científicos armazenados em arquivos XDMF/HDF5	10.000,00	44.254

Já o tanque real de batimetria consiste em uma simulação computacional mais complexa no domínio de CFD, requisitando mais que o dobro do tempo de execução com 480 *cores* no *cluster* LoboC, quando comparado com o tanque do primeiro estudo de caso. A Tabela 13 mostra o tempo de processamento de cada contribuição. Além disso, o uso da DfAnalyzer acrescenta uma baixa sobrecarga em relação ao custo de execução da simulação computacional, representando aproximadamente 1,84% do tempo de execução dessa simulação. Tal comportamento está relacionado ao fato dos modelos computacionais requisitarem mais tempo de processamento para avaliar o comportamento do fluido e dos sedimentos no tanque de interesse.

Tabela 13. Tempo de execução para as diferentes contribuições da simulação de sedimentação baseada no tanque real de batimetria.

Contribuição em termos de tempo	Tempo de execução (em segundos)	Tempo de execução (%)
Solucionador do Fluido	75.523,49	50,71%
Solucionador do Sedimento	28.000,50	19,58%
Escrita de dados em arquivos XDMF/HDF5	421,23	0,29%
Extração e visualização de dados *in situ*	2.175,16	1,52%
Proveniência (DfAnalyzer)	451,70	0,32%
Total	143.029,00	

No que diz respeito ao armazenamento de dados, a Tabela 14 mostra o espaço em disco necessário para armazenar os dados de proveniência, os dados científicos e os arquivos nos formatos XDMF e HDF5. Nesse cenário, os arquivos com dados científicos representam a maior contribuição em termos de espaço de armazenamento nessa simulação computacional, sendo que menos de 3% do espaço em disco são referentes aos dados analíticos, como os dados de proveniência e os arquivos de visualização. Entretanto, caso

houvessem limitações em termos da capacidade de armazenamento em disco, o intervalo de escrita em arquivos poderia ser aumentado e, como resultado, o volume de dados armazenados em disco diminuiria. Ao mesmo tempo, o intervalo de tempo para a extração e a visualização de dados *in situ* usando o adaptador Catalyst poderia ser mantido, a fim de que resultados parciais fossem obtidos em tempo de execução. Assim, dados não armazenados em arquivos no disco poderiam ser parcialmente consultados na base de dados da DfAnalyzer.

Tabela 14. Armazenamento de dados em disco para o tanque real de batimetria.

Tipo de dados	Espaço de armazenamento (em GB)	Dados científicos (%)
Arquivos de visualização	0,28	1,21%
Proveniência (DfAnalyzer)	0,38	1,60%
Dados científicos armazenados em arquivos XDMF/HDF5	23,44	-

6.3.6. Custo de captura de dados científicos usando a DfAnalyzer e o noWorkflow

Outro experimento conduzido no contexto desta tese considerou uma análise comparativa do custo de captura de dados científicos usando a DfAnalyzer e o noWorkflow ao executar a simulação computacional de multifísica usando as ferramentas do projeto FEniCS. Mais especificamente, analisou-se o tempo de execução da simulação (i) sem a captura de dados de proveniência, conhecida como a versão *baseline*, (ii) com a captura de dados de proveniência usando a DfAnalyzer e (iii) com a captura de dados de proveniência usando o noWorkflow.

A Figura 60 mostra esses tempos de execução em segundos. Pelos resultados experimentais, destaca-se que o noWorkflow apresenta uma sobrecarga em termos de tempo para capturar os dados de proveniência significativamente superior à DfAnalyzer (aproximadamente 29x mais lento). Além disso, a sobrecarga do noWorkflow é impraticável em um cenário que envolva o acompanhamento da execução de simulações computacionais, pois a sua sobrecarga é aproximadamente 13x maior que o tempo de executar a aplicação de CSE sem captura de dados de proveniência.

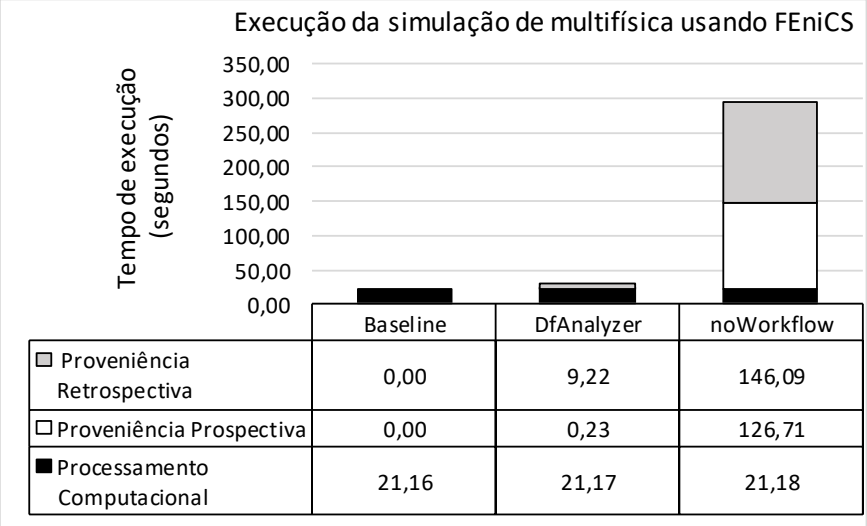


Figura 60. Comparação do custo de captura de dados de proveniência usando a DfAnalyzer e o noWorkflow na simulação de multifísica.

Ao mesmo tempo, a sobrecarga da DfAnalyzer também é significativa, representando quase em 50% de sobrecarga de tempo em relação ao tempo de execução da versão *baseline*. Vale destacar que, pelo fato desse experimento ter sido executado em um ambiente local, os componentes do projeto FEniCS competiram diretamente pelos mesmos recursos computacionais que deveriam ser dedicados à DfAnalyzer para a carga dos dados extraídos. Tal comportamento não ocorreria em um ambiente de PAD, uma vez que dedicamos recursos computacionais para a DfAnalyzer diferentes dos utilizados pelas aplicações de CSE.

Ademais, o custo de armazenamento de dados usando a DfAnalyzer foi de 1,0 MB de dados em nossa base de dados relacional, enquanto que o noWorkflow demandou aproximadamente 123,6 MB de dados. Diante desses valores, pode-se destacar que a menor sobrecarga da DfAnalyzer está relacionada ao fato de especificarmos quais dados científicos devem ser capturados e a sua vantagem analítica por disponibilizar apenas dados científicos ou valores de atributos que são representativos para as consultas de interesse dos usuários do domínio. Por outro lado, a DfAnalyzer exige um esforço a mais na etapa *offline*, ou seja, durante a modelagem das simulações computacionais para apoiarem a captura dos dados de proveniência e dos dados científicos.

6.4. Apoio ao processamento de consultas para a análise de dados científicos

Diferentemente dos experimentos anteriores, que apresentaram execuções das simulações computacionais modeladas para capturar dados específicos do domínio a partir de arquivos de dados científicos ou de dados ainda alocados em memória, essa seção apresenta o potencial analítico da arquitetura ARMFUL e das suas implementações por meio do processamento de consultas. Nesse sentido, as seguintes simulações computacionais foram consideradas: Montage, EdgeCFD e libMesh-sedimentation. Basicamente, executamos diversas consultas com o propósito de demonstrar a abrangência da nossa solução na análise exploratória de dados científicos, considerando os três tipos de consultas discutidos no Capítulo 2.

MONTAGE. As primeiras consultas realizadas foram baseadas na análise de dados científicos provenientes de arquivos produzidos por programas de simulação no domínio da astronomia, considerando a execução do *workflow* Montage usando o SCC-A. Para a extração de dados científicos, considerou-se o uso do operador algébrico Raw. A primeira consulta, mostrada na Figura 61, favorece a análise do conteúdo específico do domínio presente em um arquivo de dados científicos (Seção 2.1). Nessa consulta, obtém-se os valores dos atributos *CRVAL1* e *CRVAL2*, quando o valor de *CRVAL1* é maior que 210 (Figura 62). Nesse caso, os dados científicos são extraídos de arquivos no formato FITS, que são gerados por um programa de projeção e carregados na tabela *RProjection* da base de dados do SCC-A. A tabela *RProjection*, dessa forma, representa uma visão reduzida do conteúdo original presente nesses arquivos.

```
SELECT CRVAL1, CRVAL2
FROM RProjection
WHERE CRVAL1 > 210.00;
```

Figura 61. Consulta para analisar o conteúdo de um arquivo específico do domínio.

De uma forma alternativa, esse tipo de consulta também poderia ser obtido sem utilizar o SCC-A, por meio do uso de ferramentas alternativas como o FastQuery e RAW. Entretanto, a nossa abordagem apresenta algumas vantagens em relação a essas soluções alternativas. Primeiramente, as consultas podem selecionar os atributos presentes nos arquivos de formato FITS durante a execução da transformação de dados que utiliza o

programa de simulação para a projeção de imagens. Essa característica pode auxiliar o usuário a analisar se a projeção está sendo realizada da forma apropriada, identificar potenciais anomalias, ou observar ajustes finos a serem realizados nas configurações. Assim, o usuário pode, inclusive, observar a necessidade de interrupção da própria execução do *workflow* científico. Outra característica importante consiste em evitar a análise sintática do arquivo no formato FITS após a sua geração para saber como deve ser o acesso aos atributos, uma vez que contamos com um catálogo para os formatos de arquivos de dados científicos.

CRVAL1	CRVAL2
246.401554	-27.1663622
246.5329709	-26.4132476
246.5329737	-26.682692
246.1365759	-27.1608872
246.5329764	-26.9521364
245.8708342	-27.1609012
...	...

Figura 62. Resultados da consulta da Figura 61.

Somando-se a isso, na nossa abordagem, os atributos selecionados podem ser capturados e armazenados em uma base de proveniência integrada (*i.e.*, de acordo com o modelo de dados PROV-Df apresentado na Seção 4.4), enquanto eles estão sendo escritos no arquivo. Nessa perspectiva, diferentemente das soluções existentes para a análise exploratória de dados científicos, a nossa abordagem considera a gerência do fluxo de arquivos e do fluxo de elementos de dados, integrando dados de proveniência, de execução e científicos em uma mesma base de dados.

A Figura 63 mostra a segunda consulta realizada, que tem o objetivo de selecionar os arquivos no formato HDU que apresentam *CRVAL1* maior que 210 e *CRVAL2* menor que 60, dado que a transformação de dados *Projection* produza, pelo menos, um resultado de projeção. O propósito dessa consulta é permitir que os usuários verifiquem, em tempo de execução, se as condições ocorrem para qualquer um dos elementos de dados consumidos pela transformação de dados *Projection*. Esse tipo de consulta se enquadraria nas análises da Seção 2.2, pois necessita rastrear o fluxo de arquivos no programa de simulação para a projeção de imagens astronômicas. Ao mesmo tempo, essa consulta

também requer uma análise de dados científicos por conta dos atributos presentes na condição de seleção.

```
SELECT p.CRVAL1, p.CRVAL2, sp.HDU_AREA, sp.HDU_FILE
FROM RProjection p, RSelectProjections sp
WHERE sp.MOSAIC_ID = p.MOSAIC_ID
AND sp.ewkfid = p.ewkfid AND sp.CNTR = p.CNTR
AND p.CRVAL1 > 210.00 AND p.CRVAL2 < 60.00
AND p.ewkfid = 1;
```

Figura 63. Consulta para rastrear os arquivos baseando-se em um critério do domínio.

A terceira consulta consiste em identificar todas as transformações lineares (*i.e.*, seleção dos atributos *FA* e *FB* presentes na relação *RFitPlane*) obtidas a partir de cálculos pelos programas de simulação para a projeção de imagens astronômicas. Como ponto de partida do fluxo de dados, considerou-se os repositórios de entrada para a geração de mosaicos, *i.e.*, atributo *REPOSITORY* na relação *RListFITS*. A partir dessas informações, a consulta contempla a análise das transformações lineares, considerando o fluxo de elementos de dados ao longo das transformações de dados com correções (*i.e.*, cor e interferências de sobreposição de regiões do espaço) na geração de mosaicos personalizados. As transformações de dados e as tabelas envolvidas nessa consulta são mostradas na Figura 64. Dessa maneira, a consulta rastreia os fluxos de arquivos e de elementos de dados ao longo das transformações de dados.

Por exemplo, o fluxo de elementos de dados entre as transformações *List FITS* e *Projection* é definido pelo atributo *MOSAIC_ID*, que relaciona o repositório de origem do mosaico (atributo *REPOSITORY*) com os arquivos no formato FITS obtidos do repositório de origem. O comportamento da transformação de dados *List FITS* é característico do operador algébrico SplitMap. Na sequência, os atributos *MOSAIC_ID* e *CNTR* são os responsáveis por permitir a rastreabilidade do fluxo de elemento de dados para as transformações de dados *Projection* e *Calculate Overlaps*. Do mesmo modo, o uso do atributo *MOSAIC_ID* se estende para outros casos de rastreabilidade do fluxo de elementos de dados. Por exemplo, a última etapa utiliza o atributo *MOSAIC_ID* para unir todos os arquivos no formato FITS projetados (*i.e.*, atributo *TNAME*) e as transformações lineares,

a fim de gerar o mosaico personalizado em um arquivo no formato JPG (*i.e.*, atributo *MOSAIC_JPG*). Vale ressaltar que essa consulta atende às análises das Seções 2.2 e 2.3.

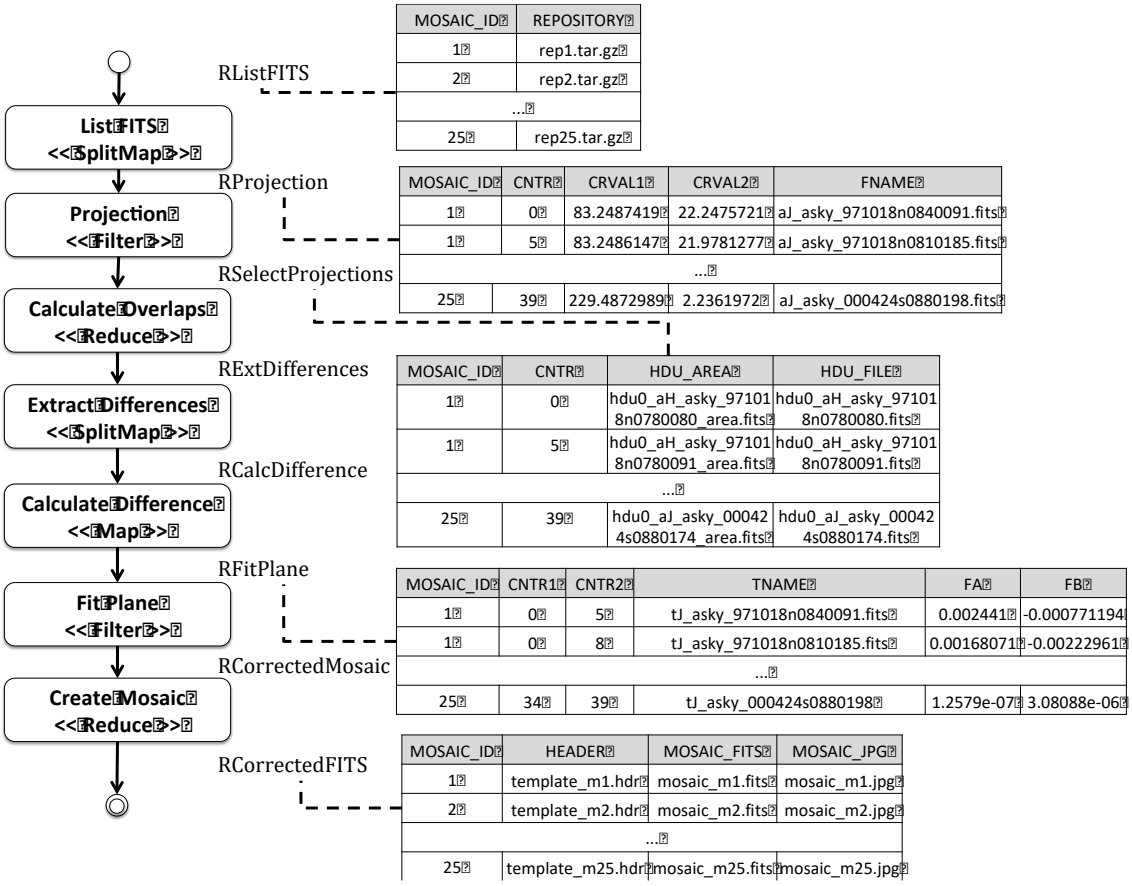


Figura 64. Consulta para analisar o fluxo de elementos de dados no *workflow* Montage.

EDGECFD. A segunda simulação computacional considera o *workflow* EdgeCFD executado no SCC-A, em que se aplicou técnicas de extração e de indexação de dados científicos. Em mais detalhes, as três consultas apresentadas nessa simulação consideram a análise do fluxo de elementos de dados, relacionando dados de proveniência, de execução e científicos, por meio da captura de dados científicos. Por último, uma análise de desempenho em termos de tempo de processamento é apresentada para cada uma dessas consultas ao realizar (i) a carga; (ii) a indexação posicional; e (iii) a indexação *bitmap* dos dados científicos armazenados em arquivos no formato XDMF e HDF5. A abordagem de carga caracteriza-se pelo processo de acesso e extração dos dados científicos de arquivos, seguido da carga dos elementos de dados capturados diretamente na base de dados. Nesse caso, a base de dados do SCC-A.

A primeira consulta, denominada FLUXO_DE_ELEMENTOS (Figura 65), analisa múltiplos valores de dados estruturados presentes em diferentes malhas após a execução do solucionador EdgeCFD com a média da velocidade na coordenada *x* (atributo *VELOCITY_0* na relação *RSolverRDC* da Figura 45) maior que 0,40 e menor que 0,50, considerando diferentes valores para o parâmetro do solucionador inexato de Newton (atributo *ISOLVER*). Essa consulta também relaciona as malhas em função das propriedades do fluido, como os atributos de viscosidade (*VISC*) e de densidade (*DENS*), que foram definidas com valores específicos (*e.g.*, 0,001 e 1,00, respectivamente). Portanto, essa consulta contempla a análise do fluxo de elementos de dados ao longo das diferentes transformações de dados dessa simulação, como *Pre RDC* e *Set Solver Configuration*.

```
SELECT rprerdc.ISOLVER, rsolvrdc.timestep,
       AVERAGE(rsolvrdc.velocity_0) as vx
FROM RPreRDC rprerdc, RSSConfig rconf, RECFDSolver rsolv,
       RSolverRDC rsolvrdc
WHERE rprerdc.nextActivationID = rconf.previousActivationID
AND rconf.nextActivationID = rsolv.previousActivationID
AND rsolv.nextActivationID = rsolvrdc.previousActivationID
AND vx>0.40 AND vx<0.50
AND rprerdc.DENS=1.00 AND rprerdc.VISC=0.001
GROUP BY rprerdc.ISOLVER, rsolvrdc.timestep;
```

Figura 65. FLUXO_DE_ELEMENTOS – Análise da velocidade do fluido na coordenada *x* variando o método inexato de Newton para o solucionador.

Diante dessa descrição, os usuários podem processar essa consulta para investigar o comportamento do solucionador ao longo da simulação computacional, considerando diferentes combinações de valores para os atributos, como a viscosidade e a densidade do fluido. A Figura 66 apresenta as tabelas da base de dados relacional do SCC-A que apresenta os dados específicos do domínio analisados pela consulta FLUXO_DE_ELEMENTOS. Nesse sentido, as tabelas *RPreRDC* e *RSolverRDC* apresentam os dados científicos extraídos de arquivos usando o operador algébrico Raw (apenas extração de dados) ou RawI (extração e indexação dos dados), conforme apresentado na Subseção 4.7.2. Enquanto isso, as setas tracejadas representam os atributos utilizados para relacionar tuplas de duas tabelas. No caso dessa consulta, apenas atributos associados aos identificadores da execução dos programas de simulação (ou tarefas) foram utilizados, como os atributos *previoustaskid* e

nexttaskid. Para que pudéssemos relacionar corretamente esses identificadores pelas relações com dados de domínio, nós consultamos a especificação do fluxo de dados (ou a estrutura do *workflow* científico modelado) na nossa base de dados.

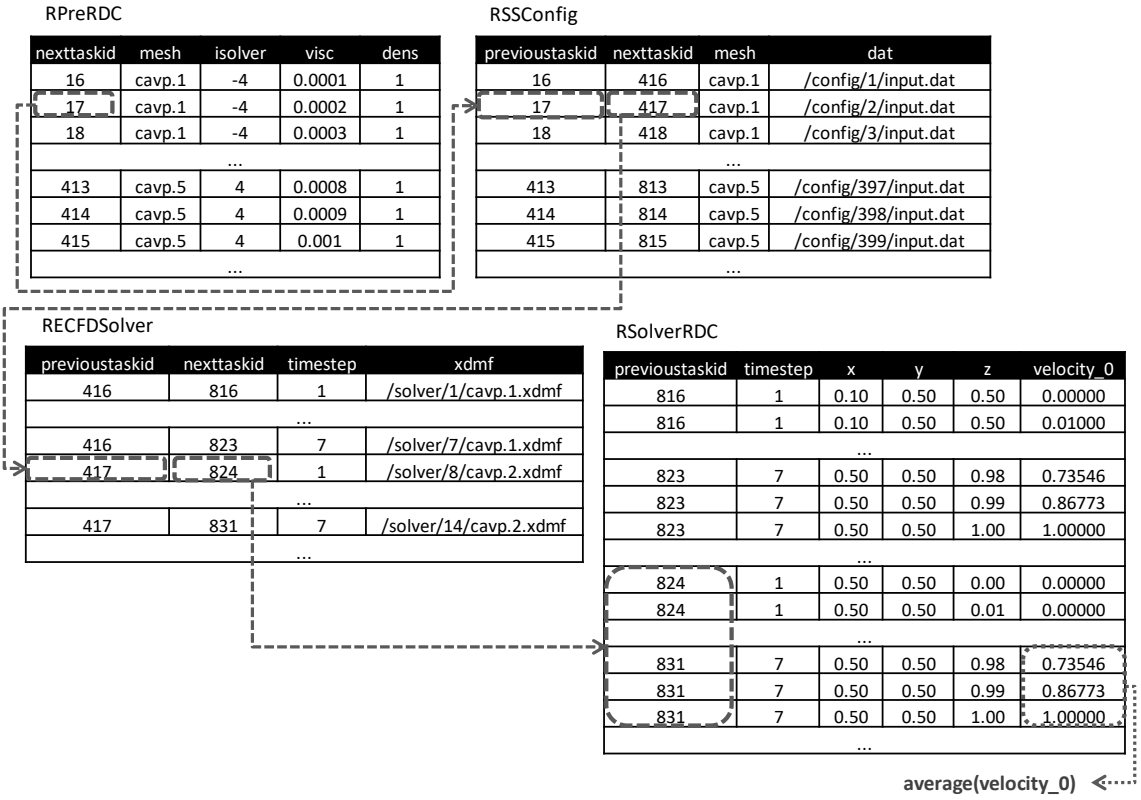


Figura 66. Tabelas analisadas na consulta FLUXO_DE_ELEMENTOS.

Já a segunda consulta, denominada FLUXO_DE_DADOS (Figura 67), investiga dois valores escalares que descrevem a convergência do solucionador de CFD (atributos na relação *RSolverRDC*) considerando valores distintos para a viscosidade do fluido, enquanto mantemos a densidade do fluido fixa, uma configuração específica para o solucionador inexato de Newton e um tempo predefinido para o solucionador (*i.e.*, passo de tempo representado pelo atributo *TIME*). Diferentemente da consulta anterior, essa consulta permite a análise do fluxo de elementos de dados relacionando-os por meio de atributos de domínio, como o número de iterações lineares (atributo *ITER*) e o resíduo (atributo *RESIDUALS*). Ao mesmo tempo, os resultados dessa consulta (Figura 68) representam uma análise pontual do comportamento do solucionador (*i.e.*, em um momento específico da simulação computacional), fixando-se o valor do atributo *TIME*.

```
SELECT rprerdc.VISC, rsolvrdc.ITER, rsolvrdc.RESIDUALS
FROM RPreRDC rprerdc, RSSConfig rconf, RECFDSolver rsolv,
       RSolverRDC rsolvrdc
WHERE rprerdc.nextActivationID = rconf.previousActivationID
AND rconf.nextActivationID = rsolv.previousActivationID
AND rsolv.nextActivationID = rsolvrdc.previousActivationID
AND rprerdc.DENS = 1 AND rprerdc.ISOLVER = 3
AND rsolvrdc.TIME=1.5;
```

Figura 67. FLUXO_DE_DADOS – Análise da convergência da simulação de CFD ao fixar determinados valores de atributos de domínio.

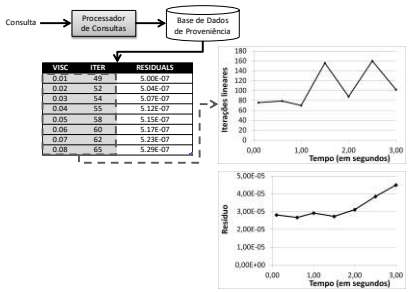


Figura 68. Resultado da consulta FLUXO_DE_DADOS a partir da análise do número de iterações lineares e do resíduo.

A terceira consulta, denominada DESEMPENHO (Figura 69), analisa o tempo de execução do solucionador para algumas transformações de dados. Essa análise se assemelha à consulta FLUXO_DE_DADOS pelo fato de considerar valores distintos para a viscosidade do fluido, enquanto valores fixos são assumidos para a densidade do fluido, o tempo de execução do solucionador (atributo *TIME*) e a configuração do solucionador. Por outro lado, essa consulta relaciona os dados de domínio com o tempo de execução da atividade *Solver RDC*. Em termos analítico, esse tipo de consulta é comumente realizado pelos usuários para monitorar o desempenho do solucionador de CFD de acordo com as propriedades do fluido ou mesmo a ocorrência de erros.

Apesar do resultado de cada consulta não variar ao modificar a abordagem de captura de dados científicos nos arquivos XDMF e HDF5, o tempo de processamento de cada consulta pode variar em termos de desempenho. Por exemplo, na carga de dados científicos, os valores dos atributos acessados e extraídos dos arquivos de dados científicos são carregados diretamente na base de dados. Assim, ao executar uma consulta, o processador de consultas precisa apenas acessar os valores dos atributos já armazenados na

base de dados. Por outro lado, se fosse aplicada a indexação posicional, os índices gerados seriam armazenados na base de dados. Assim, ao submeter uma consulta, o processador de consultas teria que acessar os índices posicionais armazenados na base de dados em um primeiro momento para, depois, capturar os elementos de dados nos arquivos XDMF e HDF5 de uma forma eficiente usando os índices acessados.

```
SELECT rprerdc.VISC, a.elapsedTime
FROM RPreRDC rprerdc, RSSConfig rconf, RECFDSolver rsolv,
       RSolverRDC rsolvrdc, activation a
WHERE rprerdc.nextActivationID = rconf.previousActivationID
AND rconf.nextActivationID = rsolv.previousActivationID
AND rsolv.nextActivationID = rsolvrdc.previousActivationID
AND rsolvrdc.previousActivationID = a.activationID
AND rprerdc.DENS = 1 AND rprerdc.ISOLVER = 3;
```

Figura 69. DESEMPENHO – Análise de desempenho do solucionador de CFD ao fixar a densidade do fluido e a configuração do solucionador inexato de Newton.

Considerando o cenário de indexação de dados científicos, desenvolveu-se um pré-processador de consultas capaz de consultar os índices gerados na base de dados do SCC-A, acessar os elementos de dados presentes em arquivos por meio dos índices e realizar a junção dos dados com outros resultados obtidos nesse processo. Em relação ao acesso aos elementos de dados em arquivos, utilizou-se os índices obtidos da base de dados do SCC-A e invocou-se um processo inverso ao da indexação de dados científicos, ou seja, de acesso aos dados por meio dos índices. No caso da indexação posicional, desenvolveu-se um método de acesso baseando-se na implementação do RAW (KARPATHIOTAKIS *et al.*, 2014). Para a indexação *bitmap*, utilizou-se o mecanismo de processamento de consultas da própria ferramenta FastBit. Além disso, a etapa de junção dos resultados foi responsável por relacionar os dados científicos presentes nos arquivos com os dados de outros atributos projetados na especificação da consulta, que não estavam presentes nos arquivos de dados científicos (ou seja, que não foram indexados).

Baseando-se nessa abordagem, a Tabela 15 apresenta o tempo de processamento de cada uma das consultas para o *workflow* EdgeCFD, variando-se a abordagem de captura dos dados científicos. Esse tempo de processamento engloba o custo de acessar os dados na base de dados do SCC-A e do repositório externo (*e.g.*, usando o processador de consultas

da ferramenta FastBit), e de combinar os resultados obtidos na etapa anterior. Pelos resultados dessa tabela, observa-se que a abordagem de carga dos dados apresentou o melhor desempenho, pois não apresenta a sobrecarga de utilizar os índices gerados para depois acessar os elementos de dados em arquivos, como acontece nas abordagens de indexação posicional (*idx-posicional*) e *bitmap* (*idx-bitmap*).

Tabela 15. Tempo de processamento de consultas de acordo a abordagem de extração e indexação de dados científicos.

Versão do *workflow*	Tempo de processamento da consulta (em minutos)		
	FLUXO_DE_ELEMENTOS	**FLUXO_DE_DADOS**	**DESEMPENHO**
carga	0,28	4,39	3,88
indexação *bitmap*	0,51	41,89	41,04
indexação posicional	0,56	45,74	44,81

Por outro lado, a sobrecarga de tempo para processar as consultas usando os algoritmos de indexação (aproximadamente 45,74 minutos no pior caso) é compensada com a sobrecarga de tempo para executar os programas de extração de dados científicos; gerar os índices; e carregar os dados em uma base de dados. Nesse cenário, essa diferença da sobrecarga de tempo (entre a abordagem de carga de dados e de indexação *bitmap*) é de aproximadamente 192,84 minutos para malhas finas, conforme apresentado nas Subseções 6.3.3 e 6.3.4. Além disso, técnicas de processamento de consultas de forma adaptativa, como apresentado no RAW (KARPATHIOTAKIS *et al.*, 2014), poderiam ser empregadas para reduzir o tempo de processamento das consultas, quando não fosse a sua primeira submissão para o pré-processador de consultas.

SEDIMENTAÇÃO. Somando-se ao potencial de análise do fluxo de dados, conforme apresentado nas consultas anteriores, a abordagem proposta nesta tese também apoia o processamento de consultas que visam monitorar, depurar e intervir nas simulações computacionais em tempo de execução. Na simulação de sedimentação utilizando o solucionador libMesh-sedimentation, os usuários estão tipicamente interessados em monitorar o progresso do seu modelo computacional, depurar o desempenho da sua aplicação e, eventualmente, realizar intervenções na execução da simulação por meio de ajustes nos parâmetros de entrada. Sendo assim, as consultas apresentadas para esta simulação consideram essas quatro propriedades: monitoramento, depuração, intervenção, e análise de dados científicos. Vale enfatizar que, apesar de apresentarmos as consultas na

linguagem SQL, as mesmas foram obtidas a partir da sua especificação utilizando o componente QI da DfAnalyzer (Capítulo 5). Ademais, todas as consultas foram executadas 10 vezes, desconsiderando-se uma primeira execução para tirar proveito de dados alocados em memória, e nenhuma delas executou em mais de 400 milissegundos.

No cenário de monitoramento do fluxo de dados, os usuários de domínio frequentemente submetem a consulta mostrada na Figura 70, que analisa quando a concentração de sedimentos depositados no fundo do tanque (*i.e.*, malha) é diferente de zero. Para isso, dados de uma linha posicionada no fundo da malha são extraídos utilizando o cartucho de extração de dados *in situ* da DfAnalyzer. Portanto, ressalta-se a necessidade de conhecermos a malha antes da execução da simulação, para que os algoritmos de extração sejam desenvolvidos na linguagem Python para o ParaView Catalyst e, consequentemente, sejam capazes de obter os dados científicos de interesse em tempo de execução. Para essa consulta, utilizou-se a malha referente ao tanque real de batimetria (Subseção 6.1.4) e alguns elementos de dados obtidos, como resultado do processamento dessa consulta, são mostrados na Figura 71.

```
SELECT time_step, x, y, z, d
FROM line_extraction_1
WHERE simulation_id = 1
AND d <> 0;
```

Figura 70. Monitoramento da concentração de sedimentos no tanque de batimetria.

time_step	x	y	z	d
1	0.000	1.000	0.000	2.00E-04
1	0.180	1.000	0.000	2.00E-04
1	0.360	1.000	0.000	2.00E-04
1	0.540	1.000	0.000	2.00E-04
1	0.720	1.000	0.000	1.99E-04
1	0.900	1.000	0.000	1.19E-04
1	1.080	1.000	0.000	3.04E-08
...	...	...	...	...

Figura 71. Resultado da consulta de monitoramento da concentração de sedimentos.

Outra consulta importante, ainda nessa simulação computacional, envolve a depuração do desempenho do solucionador libMesh-sedimentation em tempo de execução, conforme mostrado na Figura 72. Nesse caso, os usuários tipicamente analisam os

elementos de dados de saída após a convergência do solucionador (condição *converged=true* na consulta em SQL) para o tanque de sedimentação proposto por *de Rooij* e *Dalziel* (Subseção 6.1.4). Do ponto de vista dos parâmetros de interesse, a consulta retorna os valores dos resíduos finais (linear e não-linear) para o fluido e os sedimentos após a convergência do solucionador para cada passo de tempo, como mostrado na Figura 73. Para favorecer as análises visuais pelos usuários, essa consulta também retorna o arquivo no formato PNG, que consiste em uma visualização de dados para a análise da concentração de sedimentos na malha ao longo da execução da simulação computacional.

```
SELECT f.time_step,
       f.final_linear_residual, f.final_non_linear_residual,
       s.final_linear_residual, s.final_non_linear_residual,
       v.png
FROM solver_simulation_fluid as f,
     solver_simulation_sediments as s,
     visualization as v
WHERE f.converged = true AND s.converged = true
AND f.simulation_id = s.simulation_id
AND f.simulation_id = v.simulation_id
AND f.time_step = s.time_step;
```

Figura 72. Consulta de depuração do solucionador libMesh-sedimentation.

time step	Fluido		Sedimentos		Visualização
	linear residual	nonlinear residual	linear residual	nonlinear residual	png
1000	0.0168372	0.0053668	0.000226965	0.00000000	/viz/img_1000.png
2000	0.0643741	1.75431e-06	0.000109237	0.00186711	/viz/img_2000.png
3000	0.1270304	0.0027324	0.000230409	3.265e-06	/viz/img_3000.png
...	...	...	...	...	...

Figura 73. Resultado da consulta de depuração do solucionador libMesh-sedimentation.

Além dessas análises, os usuários também necessitam monitorar o valor máximo da concentração de sedimentos na região do fundo do tanque para identificar casos de anomalia do solucionador. Um caso de anomalia consiste na identificação de uma concentração de sedimentos maior que um valor máximo permitido pelo modelo computacional (*e.g.*, valor máximo igual a 1,00), em função dos parâmetros de entrada definidos para a execução da simulação computacional. A especificação dessa consulta é mostrada na Figura 74 e o seu

resultado (Figura 75) demonstra que o solucionador gerou resultados nos quais o fundo do tanque possui concentrações acima de 1,00.

```
SELECT f.time_step, t.model_name, amr.c_fraction,
       fconfig. nonlinear_tolerance,
       sconfig. nonlinear_tolerance,
       f.final_linear_residual, f.final_non_linear_residual,
       s.final_linear_residual, s.final_non_linear_residual,
       max(ext.d)
FROM time_step_control as t,
     amr_config as amr,
     get_maximum_iterations_to_flow as fconfig,
     get_maximum_iterations_to_sediments as sconfig,
     solver_simulation_fluid as f,
     solver_simulation_sediments as s,
     line_extraction_1 as ext
WHERE t.simulation_id = amr.simulation_id
AND amr.simulation_id = fconfig.simulation_id
AND fconfig.simulation_id = sconfig.simulation_id
AND sconfig.simulation_id = f.simulation_id
AND f.simulation_id = s.simulation_id
AND s.simulation_id = ext.simulation_id;
```

Figura 74. Consulta para apoiar intervenções nos parâmetros de entrada do solucionador.

time step	Parâmetros de entrada				Valor máximo da concentração de sedimentos
	model name	coarse fraction	flow nonlinear tolerance	sediment nonlinear tolerance	max(d)
0	PC11	0,001	0,0001	0,0001	1,0000
1000	PC11	0,001	0,0001	0,0001	1,0000
2000	PC11	0,001	0,0001	0,0001	1,0002
3000	PC11	0,001	0,0001	0,0001	1,0002
...					

Figura 75. Resultado da consulta para permitir intervenções nos parâmetros de entrada.

Assim, de acordo com a anomalia detectada durante a execução do solucionador, o usuário pode realizar ajustes nos valores dos parâmetros de entrada ou no próprio solucionador, a fim de que essa concentração de sedimentos não ultrapasse o valor de 1,00 em nenhum ponto da malha. Ao mesmo tempo, vale ressaltar que os ajustes nos valores dos

parâmetros de entrada exigem que o solucionador libMesh-sedimentation tenha apoio a adaptações em tempo de execução da simulação, como mostrado na Subseção 6.2.4.

Capítulo 7 - Conclusão

Simulações computacionais em larga escala são caracterizadas pela manipulação de grandes volumes de dados pelos seus modelos computacionais. Para validarem ou refutarem uma hipótese científica, os usuários comumente precisam analisar os dados científicos produzidos por esses modelos computacionais, que são armazenados em diferentes arquivos. Em um cenário tradicional, os usuários do domínio científico frequentemente desenvolvem programas *ad-hoc* para acessar e extrair determinados dados científicos dos arquivos de interesse ou de dados ainda alocados em memória. Em casos mais complexos, eles ainda precisam relacionar elementos de dados presentes em múltiplos arquivos, sem nenhum apoio computacional centrado em dados. Portanto, essa abordagem *ad-hoc* caracteriza-se por ser custosa do ponto de vista de desenvolvimento, propensa a erros, de difícil reaproveitação, sendo específica para cada análise realizada pelos usuários.

Nesta tese analisamos as abordagens existentes para apoiar a captura e a análise exploratória de dados científicos. Foram identificadas diversas limitações nessas soluções, principalmente no que diz respeito à análise de dados durante a execução e ao relacionamento entre dados dentro de arquivos distintos. Outro problema relacionado está na ausência de apoio à consulta aos dados de proveniência, também durante a execução e junto com os demais dados científicos, o que limita a gerência dos rastros dos dados de proveniência. Finalmente, um dos desafios ao apresentar soluções analíticas de dados de domínio e dados de proveniência está ligado ao desempenho computacional. As aplicações alvo desta tese caracterizam-se por serem em larga escala, demandando forte uso de recursos computacionais ainda que em ambientes de processamento de alto desempenho. Tais aplicações não podem ser penalizadas por soluções de análise de dados que, em muitos casos, incorram em sobrecargas ainda que da ordem de 5% no tempo de execução do *workflow*.

Como parte da solução, apresentamos uma abstração para a representação de fluxo de dados baseada em proveniência de dados para apoiar a análise do fluxo de arquivos e de elementos de dados em simulações computacionais em larga escala. Mais especificamente, foi proposto um formalismo para a proveniência de fluxo de dados, em conformidade com o padrão de dados de proveniência segundo o W3C. Para fazer uso dessas abstrações de dados, uma abordagem foi proposta para permitir a captura e a análise de dados científicos

durante a execução paralela de aplicações científicas. Tal abordagem foi implementada em um sistema de *workflows* científicos (SCC-A) e como uma biblioteca de componentes para a análise de fluxo de dados em simulações computacionais (DfAnalyzer). A solução caracteriza-se por ser assíncrona, utilizando recursos de processamento e armazenamento que não competem com a simulação computacional. Essa abordagem permitiu o não comprometimento do desempenho.

Nos experimentos realizados com o SCC-A, observou-se que a carga de dados científicos pode apresentar um gargalo em termos de desempenho ao empregar a extração total dos dados científicos presentes em arquivos. Por isso, a extração parcial e algoritmos de indexação foram aplicados em outros experimentos para destacar a viabilidade de apoiar a análise de dados científicos em tempo de execução, a partir da redução do custo de carga dos dados em nossa base de dados. As implementações dessa abordagem utilizam soluções existentes para a extração e a indexação de dados científicos, como o FastBit e o ParaView Catalyst. Já no cenário de processamento de dados *in situ*, os resultados experimentais usando a DfAnalyzer destacaram as vantagens do acesso e da captura de dados científicos ainda alocados em memória, além de enfatizar a baixa sobrecarga da nossa proposta para gerenciar os dados de proveniência e de domínio (aproximadamente 0,32% em uma simulação real).

Os resultados obtidos destacaram que:

(i) nos cenários reais, o custo da extração de dados científicos representa um valor percentual pequeno (aproximadamente 3,7% no *workflow* Montage usando o SCC-A) em relação ao custo de processamento dos programas de simulação;

(ii) a sobrecarga do processo de extração de dados científicos tende a diminuir na proporção que a complexidade dos modelos computacionais aumenta (*i.e.*, maior tempo de processamento computacional);

(iii) o custo de captura dos dados científicos depende diretamente da quantidade de dados obtidos dos arquivos e dos algoritmos utilizados (ou ferramentas alternativas, *e.g.* FastBit) no acesso, na extração e, eventualmente, na indexação dos dados científicos;

(iv) a indexação de dados científicos apresentou-se como uma alternativa vantajosa na captura de dados científicos, ao reduzir o custo envolvido nos algoritmos de extração de dados. Por outro lado, tal abordagem deve ser bem empregada em estruturas de dados específicas, a fim de reduzir o volume de dados a serem carregados na base de dados de proveniência. No cenário científico, se estruturas de dados mais complexas, como malhas, fossem indexadas, menor seria essa sobrecarga de armazenamento de dados. Os experimentos usando o *workflow* EdgeCFD comprovam exatamente essa redução do custo de carga dos dados ao aplicar técnicas de indexação em malhas; e

(v) o processamento de dados *in situ* tira proveito de dados ainda alocados em memória. Em nossos experimentos, utilizou-se o cartucho do ParaView Catalyst do componente RDE da DfAnalyzer para apoiar a captura de dados científicos *in situ* de uma malha em uma simulação de dinâmica de fluidos computacionais. Em um cenário de larga escala, a sobrecarga para a captura de dados científicos foi de apenas 1,52% em relação ao tempo de execução da simulação, enquanto a gerência dos dados de proveniência apresentou uma sobrecarga de 0,32%.

Tais experimentos nos permitem concluir que o uso da abordagem proposta não compromete o desempenho das aplicações científicas em larga escala. Do ponto de vista do poder analítico, as implementações da ARMFUL, SCC-A e DfAnalyzer são capazes de apoiar os três tipos de consultas para a análise exploratória de dados científicos (Capítulo 2) em todas as simulações computacionais consideradas nos experimentos conduzidos nesta tese. Mais especificamente, consultas foram desenvolvidas em quatro simulações computacionais para apoiar a análise do conteúdo de arquivos de dados científicos, dos arquivos de dados científicos relacionados pelos programas de simulação (ou transformações de dados) e dos elementos de dados relacionados pelas transformações.

7.1. Contribuições Alcançadas

Nesta tese apresentamos uma visão geral do cenário das análises exploratórias dos dados científicos, discutindo os tipos de consultas comumente realizados pelos usuários e as suas limitações.

Como contribuições destacamos:

(i) Formalização do fluxo de dados para representar uma simulação computacional como uma sequência de transformações de dados e acompanhar a propagação dos dados de domínio (como conjuntos de dados) ao longo dos programas de simulação em uma granularidade fina (*i.e.*, múltiplos arquivos ou elementos de dados relacionados). Essa definição norteou a gerência do fluxo de dados que é o principal recurso para apoiar a consistência, a reprodutibilidade e o armazenamento de dados de proveniência em uma granularidade fina para consultas.

(ii) Modelo de dados de proveniência PROV-Df, compatível com as recomendações do PROV W3C, para representar os dados de proveniência e os dados de domínio envolvidos no fluxo de dados tanto no nível físico, como no lógico.

(iii) Metodologia para identificar o fluxo de dados em simulações computacionais, a partir da extensão de uma metodologia já existente que permitia o apoio à captura de dados de proveniência em aplicações computacionais, conhecida como PrIMe.

(iv) A arquitetura ARMFUL que detalha como prover a captura e a análise de dados científicos, considerando técnicas de captura de dados científicos e uma interface para o processamento de consultas. Cartuchos foram desenvolvidos nos componentes de extração e de indexação de dados científicos, permitindo o uso de ferramentas alternativas, como o FastBit, o PostgresRaw e o ParaView Catalyst.

(v) Implementações dessa arquitetura em um SGWfC, o SCC-A, e como uma biblioteca de componentes, a DfAnalyzer.

Com contribuições secundárias, uma interface gráfica foi desenvolvida como um componente da DfAnalyzer para permitir a análise do fluxo de arquivos e de elementos de dados durante a execução de simulações, sem que seja necessário o uso de linguagens de consulta declarativas, como a SQL.

7.2. Trabalhos futuros

A abordagem proposta nesta tese para a captura e a análise exploratória de dados científicos baseada na abstração de fluxo de dados pode ser avaliada como um passo inicial na investigação do grande volume de dados manipulados pelas simulações computacionais, conforme apresentado em Atkinson *et al.* (2017) no contexto de *workflows* científicos. Somando-se aos avanços realizados nesta tese, ainda existem desafios relacionados ao processamento de consultas e à visualização de dados científicos (e do fluxo de dados) em tempo de execução. Em relação ao processamento de consultas, pode-se considerar a proposta de um sistema de consultas mais eficiente e que assuma uma solução agnóstica para a estratégia de indexação empregada. Já para a visualização de dados, destaca-se as dificuldades de análise de fluxos de dados complexos produzidos pelas simulações computacionais em larga escala, que apresentam diferentes caminhos possíveis em função da execução de centenas ou milhares de tarefas por transformação de dados, relacionando arquivos e elementos de dados.

Ainda nesse cenário de fluxo de dados complexos, pode-se enfatizar a importância das operações de projeção, seleção e agregação dos elementos de dados ao longo do fluxo de dados, a fim de que os usuários possam investigar apenas regiões de interesse da sua simulação computacional, como discutido no Capítulo 3. Entretanto, ainda existem desafios quanto ao uso de técnicas de aprendizado de máquinas para apoiar a identificação de padrões em execuções de simulações computacionais, tendo como objetivo informar ao usuário do domínio científico eventuais erros ou comportamentos científicos de interesse, como discutimos em (OCAÑA *et al.*, 2015).

Do ponto de vista da abordagem proposta, podemos enaltecer as contribuições de monitoramento, depuração e análise de dados científicos proporcionadas pelo SCC-A e pela DfAnalyzer. Apesar desses resultados analíticos que favorecem a intervenção durante a execução da simulação computacional, existem oportunidades no que diz respeito à gerência dos dados científicos e das suas intervenções (FERREIRA DA SILVA *et al.*, 2017), que poderiam ser explorados, inclusive, com as implementações da ARMFUL (DEELMAN *et al.*, 2017, SOUZA *et al.*, 2016, 2017). Nesse sentido, a dissertação de mestrado do Luciano Silva Leite propõe a captura de dados de proveniência e dados científicos em simulações computacionais usando a DfAnalyzer sem o pré-processamento

manual e *offline* de modelagem do fluxo de dados. Com relação ao SCC-A e à DfAnalyzer, ainda existem oportunidades de uso de sistemas de gerência de banco de dados paralelos e distribuídos, ou mesmo baseados no processamento de dados em memória para reduzir a sobrecarga de operações em disco durante a execução de simulações em larga escala.

Em termos de infraestrutura, a alocação do SGBD MonetDB no mesmo recurso computacional que a aplicação RESTful da DfAnalyzer pode apresentar gargalos em termos de desempenho em simulações computacionais mais complexas. Como a própria complexidade das simulações computacionais tem apresentado um crescimento considerável a cada ano, pode ser que a infraestrutura assumida para essa primeira implementação da DfAnalyzer não se apresente como a mais adequada em cenários futuros, incluindo questões como a tolerância e a recuperação em casos de falha. Além disso, estudos mais aprimorados quando à análise do comportamento de CPU e de operações de entrada e saída na base de dados do MonetDB, ou mesmo na aplicação RESTful, poderiam ser conduzidos em trabalhos futuros.

Referências bibliográficas

AHMED, N., JOHN, V., 2015, "Adaptive time step control for higher order variational time discretizations applied to convection–diffusion–reaction equations", *Computer Methods in Applied Mechanics and Engineering*, v. 285 (Mar.), pp. 83–101.

ALAGIANNIS, I., BOROVICA-GAJIC, R., BRANCO, M., *et al.*, 2015, "NoDB: efficient query execution on raw data files", *Communications of the ACM*, v. 58, n. 12 (Nov.), pp. 112–121.

ALNÆS, M., BLECHTA, J., HAKE, J., *et al.*, 2015, *The FEniCS Project Version 1.5*, University Library Heidelberg. Disponível em: http://journals.ub.uni-heidelberg.de/index.php/ans/article/view/20553.

ALPER, P., BELHAJJAME, K., GOBLE, C. A., *et al.*, 2015, "LabelFlow: Exploiting Workflow Provenance to Surface Scientific Data Provenance", In: LUDÄSCHER, B., PLALE, B. [eds.] (eds), *Provenance and Annotation of Data and Processes*, chapter 8628, Cham: Springer International Publishing, pp. 84–96.

ASSUNCAO, L., GONCALVES, C., CUNHA, J. C., 2012, "Autonomic Activities in the Execution of Scientific Workflows: Evaluation of the AWARD Framework". In: IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing (UIC/ATC), pp. 423–430, Fukuoka.

ATKINSON, M., GESING, S., MONTAGNAT, J., *et al.*, 2017, "Scientific workflows: Past, present and future", *Future Generation Computer Systems*, v. 75 (Oct.), pp. 216–227.

AYACHIT, U., 2015, *The ParaView guide: updated for ParaView version 4.3*. Full color version ed. Los Alamos, Kitware.

AYACHIT, U., BAUER, A., DUQUE, E. P. N., *et al.*, 2016, "Performance Analysis, Design Considerations, and Applications of Extreme-scale in Situ Infrastructures". In: *Supercomputing conference*, pp. 79:1–79:12, Piscataway, NJ, USA.

AYACHIT, U., BAUER, A., GEVECI, B., *et al.*, 2015, "ParaView Catalyst: Enabling In Situ Data Analysis and Visualization". In: *In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, pp. 25–29

BAUER, A. C., ABBASI, H., AHRENS, J., *et al.*, 2016, "In Situ Methods, Infrastructures, and Applications on High Performance Computing Platforms", *Computer Graphics Forum*, v. 35, n. 3 (Jun.), pp. 577–597.

BENNETT, J. C., ABBASI, H., BREMER, P.-T., *et al.*, 2012, "Combining in-situ and in-transit processing to enable extreme-scale scientific analysis". In: *Supercomputing Conference*, pp. 1–9, Salt Lake City, UT, USA.

BERNHOLDT, D., DUBEY, A., HEROUX, M., *et al.*, 2017, "Improving Reproducibility Through Better Software Practices". In: *SIAM Conference on Computational Science and Engineering*, Atlanta, GA, USA.

BIRSAN, D., 2005, "On plug-ins and extensible architectures", *Queue*, v. 3, n. 2, pp. 40–46.

BLANAS, S., WU, K., BYNA, S., *et al.*, 2014, "Parallel data analysis directly on scientific file formats". In: *ACM SIGMOD International Conference on Management of Data*, pp. 385–396, Snowbird.

BONCZ, P. A., KERSTEN, M. L., MANEGOLD, S., 2008, "Breaking the memory wall in MonetDB", *Communications of the ACM*, v. 51, n. 12 (Dec.), pp. 77.

BOWERS, S., MCPHILLIPS, T., RIDDLE, S., *et al.*, 2008, "Kepler/pPOD: Scientific Workflow and Provenance Support for Assembling the Tree of Life". In: *International Provenance and Annotation Workshop (IPAW)*, pp. 70–77, Salt Lake City, UT, USA.

BROWN, P. G., 2010, "Overview of SciDB: large scale array storage, processing and analysis". *ACM SIGMOD International Conference on Management of data*, pp. 963–968, Indianapolis, IN, USA.

CALLAHAN, S. P., FREIRE, J., SANTOS, E., *et al.*, 2006, "VisTrails: Visualization Meets Data Management". In: *ACM SIGMOD International Conference on Management of Data*, pp. 745–747, New York, NY, USA.

CAMATA, J. J., SILVA, V., VALDURIEZ, P., *et al.*, 2018, "In situ visualization and data analysis for turbidity currents simulation", *Computers & Geosciences*, v. 110, pp. 23–31.

CHIRIGATI, F., SILVA, V., OGASAWARA, E., *et al.*, 2012, "Evaluating Parameter Sweep Workflows in High Performance Computing". In: *International Workshop on Scalable Workflow Enactment Engines and Technologies (SWEET'12)*, pp. 10, Scottsdale, AZ, EUA.

CHOU, J., WU, K., PRABHAT, 2011, "FastQuery: A Parallel Indexing System for Scientific Data". In: *CLUSTER*, pp. 455–464, Austin, TX, USA.

COSTA, F., SILVA, V., DE OLIVEIRA, D., *et al.*, 2013, "Capturing and Querying Workflow Runtime Provenance with PROV: A Practical Approach". In: *Joint EDBT/ICDT 2013 Workshops*, pp. 282–289, New York, NY, USA.

DE OLIVEIRA, D., OGASAWARA, E., BAIÃO, F., *et al.*, 2010, "SciCumulus: A Lightweight Cloud Middleware to Explore Many Task Computing Paradigm in Scientific Workflows". In: *International Conference on Cloud Computing*, pp. 378–385, Washington, DC, USA.

DE OLIVEIRA, D., SILVA, V., MATTOSO, M., 2015, "How Much Domain Data Should Be in Provenance Databases?". In: *Workshop on Theory and Practice of Provenance (TaPP)*, Edinburgh, Scotland.

DE ROOIJ, F., DALZIEL, S. B., 2009, "Time- and Space-Resolved Measurements of Deposition under Turbidity Currents", In: MCCAFFREY, W., KNELLER, B., PEAKALL, J. [eds.] (eds), *Particulate Gravity Currents*, Oxford, UK: Blackwell Publishing Ltd., pp. 207–215.

DEELMAN, E., GANNON, D., SHIELDS, M., *et al.*, 2009, "Workflows and e-Science: An overview of workflow system features and capabilities", *Future Generation Computer Systems*, v. 25, n. 5, pp. 528–540.

DEELMAN, E., PETERKA, T., ALTINTAS, I., *et al.*, 2017, "The future of scientific workflows", *The International Journal of High Performance Computing Applications* (Apr.), pp. 1–7.

DEELMAN, E., VAHI, K., JUVE, G., *et al.*, 2015, "Pegasus, a workflow management system for science automation", *Future Generation Computer Systems*, v. 46, pp. 17–35.

DIAS, J., GUERRA, G., ROCHINHA, F., *et al.*, 2015, "Data-centric iteration in dynamic workflows", *Future Generation Computer Systems*, v. 46, pp. 114–126.

DONG, B., BYNA, S., WU, K., 2013, "SDS: a framework for scientific data services". In: *Parallel Data Storage Workshop (PDSW'13)*, pp. 27–32, Denver, Colorado.

ELIAS, R. N., COUTINHO, A. L. G. A., 2007, "Stabilized edge-based finite element simulation of free-surface flows", *International Journal for Numerical Methods in Fluids*, v. 54, n. 6–8, pp. 965–993.

ELMASRI, R., NAVATHE, S., 2010, *Fundamentals of Database Systems*. 6 ed. Addison-Wesley.

FERREIRA DA SILVA, R., FILGUEIRA, R., PIETRI, I., *et al.*, 2017, "A characterization of workflow management systems for extreme-scale applications", *Future Generation Computer Systems*, v. 75 (Oct.), pp. 228–238.

FREIRE, J., KOOP, D., SANTOS, E., *et al.*, 2008, "Provenance for Computational Tasks: A Survey", *Computing in Science and Engineering*, v. 10, n. 3, pp. 11–21.

GIL, Y., RATNAKAR, V., KIM, J., *et al.*, 2011, "Wings: Intelligent Workflow-Based Design of Computational Experiments", *IEEE Intelligent Systems*, v. 26, n. 1 (Jan.), pp. 62–72.

GONÇALVES, B., PORTO, F., 2014, "γ-DB: managing scientific hypotheses as uncertain data", *VLDB Endowment*, v. 7, n. 11 (Jul.), pp. 959–962.

GUERRA, G. M., ZIO, S., CAMATA, J. J., *et al.*, 2016, "Uncertainty quantification in numerical simulation of particle-laden flows", *Computational Geosciences*, v. 20, n. 1 (Feb.), pp. 265–281.

HANISCH, R. J., FARRIS, A., GREISEN, E. W., *et al.*, 2001, "Definition of the Flexible Image Transport System (FITS)", *Astronomy and Astrophysics*, v. 376, n. 1 (Sep.), pp. 359–380.

HENDRIX, V., FOX, J., GHOSHAL, D., *et al.*, 2016, "Tigres Workflow Library: Supporting Scientific Pipelines on HPC Systems". In: *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 146–155, Cartagena, Colômbia.

HEUSER, C. A., 2009, *Projeto de Banco de Dados*. 6a edição ed. Editora Bookman.

HONG, Z., YU, C., WANG, J., *et al.*, 2016, "AQUAdexIM: highly efficient in-memory indexing and querying of astronomy time series images", *Experimental Astronomy*, v. 42, n. 3 (Dec.), pp. 387–405.

HONG, Z., YU, C., XIA, R., *et al.*, 2015, "AQUAdex: A Highly Efficient Indexing and Retrieving Method for Astronomical Big Data of Time Series Images", In: WANG,

G., ZOMAYA, A., MARTINEZ, G., *et al.* [eds.] (eds), *Algorithms and Architectures for Parallel Processing*, , chapter 9529, Cham: Springer International Publishing, pp. 92–105.

IDEAS productivity. Disponível em: https://ideas-productivity.org.

IKEDA, R., DAS SARMA, A., WIDOM, J., 2013, "Logical provenance in data-oriented workflows?". In: *IEEE International Conference on Data Engineering (ICDE)*, pp. 877–888, Brisbane, QLD, Australia.

IKEDA, R., WIDOM, J., 2010, "Panda: A System for Provenance and Data". In: *IEEE Data Engineering Bulletin, Special Issue on Data Provenance*, v. 33, n. 3 (Sep.), pp. 42–49.

JACOB, J. C., KATZ, D. S., BERRIMAN, G. B., *et al.*, 2009, "Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking", *International Journal of Computational Science and Engineering (IJCSE)*, v. 4, n. 2, pp. 73–87.

KARPATHIOTAKIS, M., BRANCO, M., ALAGIANNIS, I., *et al.*, 2014, "Adaptive Query Processing on RAW Data", *VLDB Endowment*, v. 7, n. 12, pp. 1119–1130.

KIM, J., ABBASI, H., CHACON, L., *et al.*, 2011, "Parallel in situ indexing for data-intensive computing". In: *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pp. 65–72, Providence, Rl, USA.

KIRK, B. S., PETERSON, J. W., STOGNER, R. H., *et al.*, 2006, "libMesh : a C++ library for parallel adaptive mesh refinement/coarsening simulations", *Engineering with Computers*, v. 22, n. 3–4 (Dec.), pp. 237–254.

LASLUISA, S., ZHANG, F., JIN, T., *et al.*, 2015, "In-situ feature-based objects tracking for data-intensive scientific and enterprise analytics workflows", *Cluster Computing*, v. 18, n. 1 (Mar.), pp. 29–40.

LERNER, B. S., BOOSE, E. R., 2015, "RDataTracker and DDG Explorer", In: LUDÄSCHER, B., PLALE, B. [eds.] (eds), *Provenance and Annotation of Data and Processes*, , chapter 8628, Cham: Springer International Publishing, pp. 288–290.

LIU, Q., LOGAN, J., TIAN, Y., *et al.*, 2014, "Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks", *Concurrency and Computation: Practice and Experience*, v. 26, n. 7 (May.), pp. 1453–1473.

LO, D. C., MURUGESAN, K., YOUNG, D. L., 2005, "Numerical solution of three-dimensional velocity-vorticity Navier-Stokes equations by finite difference method", *International Journal for Numerical Methods in Fluids*, v. 47, n. 12 (Apr.), pp. 1469–1487.

LUSTOSA, H., 2015, *Managing Numerical Simulation Data Using a Multidimensional Array Representation*. Dissertação de Mestrado.

MA, B., SHOSHANI, A., SIM, A., *et al.*, 2012, "Efficient Attribute-Based Data Access in Astronomy Analysis". In: *Supercomputing conference*, pp. 562–571, Salt Lake City, UT, USA.

MATTOSO, M., DIAS, J., OCAÑA, K. A. C. S., *et al.*, 2015, "Dynamic steering of HPC scientific workflows: A survey", *Future Generation Computer Systems*, v. 46 (May.), pp. 100–113.

MATTOSO, M., OCAÑA, K., HORTA, F., *et al.*, 2013, "User-steering of HPC workflows: state-of-the-art and future directions". In: *ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies (SWEET)*, pp. 1–6, New York, NY, USA.

MATTOSO, M., WERNER, C., TRAVASSOS, G. H., *et al.*, 2010, "Towards supporting the life cycle of large scale scientific experiments", *International Journal of Business Process Integration and Management*, v. 5, n. 1, pp. 79–92.

MCPHILLIPS, T., SONG, T., KOLISNIK, T., *et al.*, 2015, "YesWorkflow: A User-Oriented, Language-Independent Tool for Recovering Workflow Information from Scripts", *International Journal of Digital Curation*, v. 10, n. 1 (Feb.)

MOREAU, L., GROTH, P., 2013, *Provenance: an introduction to PROV*. Morgan & Claypool Publishers.

MUNROE, S., MILES, S., MOREAU, L., *et al.*, 2006, "PrIMe: a software engineering methodology for developing provenance-aware applications". In: *International workshop on Software engineering and middleware*, pp. 39–46, Portland, USA.

MURTA, L., BRAGANHOLO, V., CHIRIGATI, F., *et al.*, 2014, "noWorkflow: Capturing and Analyzing Provenance of Scripts". In: *International Workshop on Provenance Annotation (IPAW)*, pp. 1–12

NAJJAR, W. A., LEE, E. A., GAO, G. R., 1999, "Advances in the dataflow computational model", *Parallel Computing*, v. 25, n. 13–14, pp. 1907–1929.

OCAÑA, K., DE OLIVEIRA, D., SILVA, V., *et al.*, 2015, "Data Analytics in Bioinformatics: Data Science in Practice for Genomics Analysis Workflows". In: *IEEE International Conference on eScience*, Munich, Germany.

OCAÑA, K., OLIVEIRA, D. DE, OGASAWARA, E., *et al.*, 2011, "SciPhy: A Cloud-Based Workflow for Phylogenetic Analysis of Drug Targets in Protozoan Genomes". In: *Advances in Bioinformatics and Computational Biology*, pp. 66–70, Angra dos Reis, Brazil.

OGASAWARA, E., DIAS, J., OLIVEIRA, D., *et al.*, 2011, "An Algebraic Approach for Data-Centric Scientific Workflows", In: *International Conference on Very Large Data Bases*, v. 4, n. 12, pp. 1328–1339.

OGASAWARA, E., DIAS, J., SILVA, V., *et al.*, 2013, "Chiron: A Parallel Engine for Algebraic Scientific Workflows", Concurrency and Computation: Practice and Experience, v. 25, n. 16, pp. 2327–2341.

OLDFIELD, R. A., MORELAND, K., FABIAN, N., *et al.*, 2014, "Evaluation of methods to integrate analysis into a large-scale shock physics code". In: *Supercomputing conference*, pp. 83–92

OLIVEIRA, D., COSTA, F., SILVA, V., *et al.*, 2014, "Debugging Scientific Workflows with Provenance: Achievements and Lessons Learned". In: *Simpósio Brasileiro de Banco de Dados*, Curitiba, Paraná.

OLIVEIRA, D. DE, OCAÑA, K. A. C. S., BAIÃO, F., *et al.*, 2012, "A Provenance-based Adaptive Scheduling Heuristic for Parallel Scientific Workflows in Clouds", *Journal of Grid Computing*, v. 10, n. 3, pp. 521–552.

OLIVEIRA, W., OLIVEIRA, D. D., BRAGANHOLO, V., 2018, "Provenance Analytics for Workflow-Based Computational Experiments: A Survey", *ACM Computing Surveys*, v. 51, n. 3 (May.), pp. 1–25.

ÖZSU, M. T., VALDURIEZ, P., 2011, *Principles of Distributed Database Systems*. 3 ed. New York, Springer.

PENNISI, E., 2011, "Will Computers Crash Genomics?", *Science*, v. 331, n. 6018 (Feb.), pp. 666–668.

RAICU, I., FOSTER, I. T., YONG ZHAO, 2008, "Many-task computing for grids and supercomputers". In: *Workshop on Many-Task Computing on Clouds, Grids, and Supercomputers (MTAGS)*, pp. 1–11, Austin, Texas, USA.

RAMAKRISHNAN, R., GEHRKE, J., 2003, *Database management systems*. Third edition, international edition ed. New York, McGraw-Hill.

RUBEL, O., LORING, B., VAY, J.-L., *et al.*, 2016, "WarpIV: In Situ Visualization and Analysis of Ion Accelerator Simulations", *IEEE Computer Graphics and Applications*, v. 36, n. 3 (May.), pp. 22–35.

RÜDE, U., WILLCOX, K., MCINNES, L. C., *et al.*, 2016, "Research and Education in Computational Science and Engineering", *CoRR*, v. abs/1610.02608

SILVA, V., CAMATA, J., MATTOSO, M., *et al.*, 2017a, "In Situ Data Steering with Provenance Data.". In: *Minisymposium 19 Applications and Computational Strategies for Finite Element Computations using LibMesh, SIAM Conference on Computational Science and Engineering*, Atlanta, GA, USA.

SILVA, V., CAMATA, J., DE OLIVEIRA, D., *et al.*, 2016a, "In Situ Data Steering on Sedimentation Simulation with Provenance Data". In: *Poster session of Supercomputing conference*, Salt Lake City, UT, USA.

SILVA, V., DE OLIVEIRA, D., VALDURIEZ, P., *et al.*, 2018, "DfAnalyzer: Runtime Dataflow Analysis of Scientific Applications using Provenance". In: *International Conference on Very Large Data Bases*, Rio de Janeiro, Brazil.

SILVA, V., LEITE, J., CAMATA, J., *et al.*, 2017b, "Raw Data Queries during Data-intensive Parallel Workflow Execution", *Special Issue on Workflows for Data-Driven Research in the Future Generation Computer Systems Journal*, v. 75, p. 402-422.

SILVA, V., NEVES, L., SOUZA, R., *et al.*, 2016b, "Integrating domain-data steering with code-profiling tools to debug data-intensive workflows". In: *Workshop on Workflows in Support of Large-Scale Science (WORKS)*, Salt Lake City, Utah, USA.

SILVA, V., DE OLIVEIRA, D., VALDURIEZ, P., *et al.*, 2016c, "Analyzing related raw data files through dataflows", Concurrency and Computation: Practice and Experience, v. 28, n. 8, pp. 2528–2545.

SILVA, V., OLIVEIRA, D., MATTOSO, M., 2014, "SciCumulus 2.0: Um Sistema de Gerência de Workflows Científicos para Nuvens Orientado a Fluxo de Dados". In: *Sessão de Demos do Simpósio Brasileiro de Banco de Dados*, Curitiba, Paraná.

SOUZA, R., SILVA, V., CAMATA, J., *et al.*, 2017, "Tracking of Online Parameter Fine-tuning in Scientific Workflows". In: *Workshop on Workflows in Support of Large-Scale Science (WORKS'17)*, Denver, CO, USA.

SOUZA, R., SILVA, V., COUTINHO, A. L. G. A., *et al.*, 2016, "Online Input Data Reduction in Scientific Workflows". In: *Workshop on Workflows in Support of Large-Scale Science (WORKS'16)*, Salt Lake City, Utah, USA.

SOUZA, R., SILVA, V., NEVES, L., *et al.*, 2015, "Monitoramento de Desempenho usando Dados de Proveniência e de Domínio durante a Execução de Aplicações Científicas". In: *Workshop em Desempenho de Sistemas Computacionais e de Comunicação*, Recife, PE.

TIAN, Y., ALAGIANNIS, I., LIAROU, E., *et al.*, 2014, "DiNoDB: Efficient Large-Scale Raw Data Analytics". In: *Data4U*, pp. 1–6.

VAHI, K., RYNGE, M., JUVE, G., *et al.*, 2013, "Rethinking Data Management for Big Data Scientific Workflows". In: *Workshop on Big Data and Science: Infrastructure and Services, BigData Conference*, pp. 27–35, Silicon Valley, CA, USA.

VALLI, A. M. P., CAREY, G. F., COUTINHO, A. L. G. A., 2005, "Control strategies for timestep selection in finite element simulation of incompressible flows and coupled reaction-convection-diffusion processes", *International Journal for Numerical Methods in Fluids*, v. 47, n. 3 (Jan.), pp. 201–231.

VTK, 2009, *Visualization Toolkit*, http://www.vtk.org.

W.F.J., B., 2007, "Human-in-the-loop'simulation: the right tool for port design". In: *Port Technology International*, pp. 48–50.

WU, K., AHERN, S., BETHEL, E. W., *et al.*, 2009, "FastBit: interactively searching massive data", *Journal of Physics: Conference Series*, v. 180 (Jul.), pp. 12-53.

ZAHARIA, M., CHOWDHURY, M., FRANKLIN, M. J., *et al.*, 2010, "Spark: cluster computing with working sets". In: *USENIX conference on Hot topics in cloud computing*, pp. 10–17, Boston, MA, USA.

ZHAO, Y., RAICU, I., T. FOSTER, I., *et al.*, 2008, "Realizing Fast, Scalable and Reliable Scientific Computations in Grid Environments", *Grid computing research progress*, New York: Nova Science Publishers, pp. 341.