Universidade Federal
do Rio de Janeiro
Escola Politécnica

DFA-LIB-PYTHON: UMA BIBLIOTECA PARA A EXTRAÇÃO DE DADOS CIENTÍFICOS USANDO A DFANALYZER

Vinícius Silva Campos

Projeto de Graduação apresentado ao Curso de Engenharia de Computação e Informação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientadores: Marta Lima de Queirós Mattoso
Vítor Silva Sousa

Rio de Janeiro
Junho de 2018

DFA-LIB-PYTHON: UMA BIBLIOTECA PARA A EXTRAÇÃO DE DADOS CIENTÍFICOS USANDO A DFANALYZER

Vinícius Silva Campos

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE COMPUTAÇÃO E INFORMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE COMPUTAÇÃO E INFORMAÇÃO.

Examinada por:

_____

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

_____

Vítor Silva Sousa, M.Sc.

_____

Prof. Alvaro Luiz Gayoso de Azeredo Coutinho, D. Sc.

_____

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

RIO DE JANEIRO, RJ – BRASIL

Junho de 2018

Campos, Vinícius Silva

DfA-lib-Python: Uma biblioteca para a extração de dados científicos usando a DfAnalyzer / Vinícius Silva Campos – Rio de Janeiro: UFRJ/ Escola Politécnica, 2018.

X, 35 p.: il.; 29,7cm.

Orientadores: Marta Lima de Queirós Mattoso e Vítor Silva Sousa

Projeto de Graduação – UFRJ/ Escola Politécnica/ Curso de Engenharia de Computação e Informação, 2018.

Referências Bibliográficas: p. 34-35.

1. Simulações Computacionais 2. Extração de Dados de Proveniência 3. Extração de Dados Científicos 4. Fluxo de Dados I. Mattoso, Marta Lima de Queirós *et al*. II Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia de Computação e Informação. III Título.

Agradecimentos

Agradeço aos meus pais pelo amor incondicional e por todo o apoio fornecido ao longo desses anos.

Agradeço à professora Marta Mattoso e ao Vítor Silva, meus orientadores, pela disposição e motivação em orientar esse projeto e pelas excelentes oportunidades oferecidas ao longo da minha graduação.

Agradeço a todos os meus amigos, que me apoiaram ao longo dessa jornada, responsáveis por lembranças e histórias maravilhosas obtidas.

Resumo do Projeto de Graduação apresentação à Escola Politécnica/ UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Computação e Informação.

DfA-lib-Python: Uma Biblioteca para a Extração de Dados Científicos usando a DfAnalyzer

Vinícius Silva Campos

Junho/2018

Orientadores: Marta Lima de Queirós Mattoso
Vítor Silva Sousa

Curso: Engenharia de Computação e Informação

Com o avanço da ciência da computação, experimentos científicos são comumente baseados em simulações computacionais para validar seus modelos matemáticos e físicos. Em função do aumento da complexidade desses modelos computacionais, surgiu a necessidade pelo uso de ferramentas que favoreçam a criação e a execução desses modelos computacionais. Nesse cenário, aplicações de Ciência Computacional e Engenharia (CSE) surgem com o objetivo de fornecer ferramentas que auxiliem no processo de modelagem de simulações computacionais de diversos domínios científicos. Pelo fato dessas simulações demandarem muito tempo de execução, os usuários do domínio científico necessitam realizar as suas análises durante a execução, a fim de anteciparem a investigação de determinados comportamentos científicos e, consequentemente, serem capazes de ajustar determinados parâmetros de simulação ou mesmo de interromper uma determinada execução. Para isso, destaca-se a importância de permitir tanto a captura como a análise do fluxo de dados, por exemplo, por meio de dados de proveniência e dados científicos, ao longo da execução das simulações computacionais. Esta monografia propõe uma biblioteca, conhecida como DfA-lib-Python, para permitir a captura de dados de proveniência e de dados científicos usando a DfAnalyzer.

Palavras-Chave: Simulações Computacionais, Extração de Dados de Proveniência, Extração de Dados Científicos, Fluxo de Dados.

Abstract of Undergraduate Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Computer and Information Engineer.

DfA-lib-Python: A Library for Extraction of Scientific Data using the DfAnalyzer

Vinícius Silva Campos

Junho/2018

Advisors: Marta Lima de Queirós Mattoso
Vítor Silva Sousa

Major: Computer and Information Engineering

With the advancement of computer science, scientific experiments are commonly based on computational simulations to validate their mathematical and physical models. Due to the increase in the complexity of these computational models, the need arose for the use of tools that favor the creation and execution of these computational models. In this scenario, applications of Computational Science and Engineering (CSE) emerge with the objective of providing tools that assist in the process of modeling computational simulations of various scientific domains. Because these simulations require a long execution time, users in the scientific domain need to carry out their analyses during execution in order to anticipate the investigation of certain scientific behaviors and, consequently, be able to adjust certain parameters of simulation or even to interrupt execution. For this, the importance of allowing both the capture and the analysis of the dataflow, for example, by means of provenance data and scientific data, during the execution of the computational simulations, stands out. This work proposes a library, known as DfA-lib-Python, to allow the capture of provenance data and scientific data with DfAnalyzer.

Keywords: Computational Simulation, Extraction of Provenance Data, Extraction of Scientific Data, Dataflow.

Sumário

Lista de Figuras

Lista de Tabelas

1. Introdução

Desde a sua criação, os computadores têm sido utilizados para auxiliar o avanço do conhecimento científico. A função inicial do computador era a de realizar cálculos de forma precisa. Essa funcionalidade permitiu que procedimentos, como a resolução de equações diferenciais para problemas de engenharia, por exemplo, pudessem ser realizadas por meio da utilização de métodos numéricos implementados e executados em um computador. Com o passar do tempo, houve o crescimento do poder computacional disponível para realização desses experimentos, fazendo com que modelos matemáticos cada vez mais complexos pudessem ser transpostos em modelos computacionais, instanciados e executados.

Diante desse cenário, esses experimentos científicos, em geral, têm como objetivo confirmar ou refutar uma hipótese científica por meio da criação e execução de modelos computacionais encadeados que simulam eventos da natureza. Esse encadeamento de modelos matemáticos com o apoio computacional (ou de programas) é denominado de simulação computacional. Para auxiliar na representação desse encadeamento, abstrações de fluxo de dados podem ser utilizadas para registar os dados consumidos e produzidos nas diferentes etapas da simulação e as dependências desses dados em função dos programas de simulação [1].

Ademais, com o aumento da complexidade desses modelos e da quantidade de dados manipulados pelas simulações, tornou-se necessário o uso de ambientes com alto poder de processamento computacional com o intuito de tirar proveito de técnicas de paralelismo para viabilizar a execução das simulações computacionais. Esses recursos computacionais que permitem a paralelização e a distribuição de dados são conhecidos como ambientes de Processamento de Alto Desempenho (PAD) [2].

Além da abstração de fluxo de dados, uma etapa de modelagem de dados é importante para descrever como os dados devem ser registrados em um repositório externo ou, como no contexto desta monografia, em uma base de dados relacional. Especificamente, a nossa abordagem envolve tanto a abstração de fluxo de dados, como a captura e o registro de dados de proveniência e de dados científicos em uma base de dados, que são produzidos pelos programas simulação. Vale ressaltar que os dados científicos no cenário de simulações computacionais são frequentemente armazenados em arquivos.

Somando-se a essas questões, esta monografia considera uma solução que também apoie a análise de dados durante a execução das simulações computacionais, assumindo mecanismos eficientes para apoiar a captura dos dados de proveniência e científicos. Nesse sentido, a DfAnalyzer [3] foi proposta pelo nosso grupo de pesquisa como uma biblioteca de componentes que permite tanto a captura, como a análise de dados científicos, durante a execução de simulações computacionais.

Entretanto, a sua implementação consiste apenas na disponibilização de serviços RESTful de forma assíncrona em que os especialistas em ciência da computação precisam interagir com os desenvolvedores da aplicação para permitir a captura dos dados relevantes. Nesse caso, chamadas de sistema usando serviços para o envio de requisições HTTP são necessárias, ou seja, uso de chamadas externas (RPC). Assim, cada requisição apresenta uma estrutura própria de mensagem e que não pode ser validada, a priori, pelos usuários.

Diante dessa questão, esta monografia propõe o desenvolvimento de uma biblioteca em Python que permita o uso da DfAnalyzer para a extração de dados de proveniência e de dados científicos produzidos durante a execução da simulação. Para isso, os programas de simulação em Python precisam ter os seus códigos fontes instrumentados. Essa instrumentação ocorre com o uso de classes e métodos para cada conceito envolvido na nossa abstração de fluxo de dados. Além disso, diante de práticas cotidianas usando esses métodos da nossa biblioteca, os usuários do domínio científico poderiam, com o tempo, ter mais facilidade em utilizar essa biblioteca da DfAnalyzer em Python sem o auxílio do especialista em ciência da computação. Ao mesmo tempo, essa captura oferece recursos para a análise do fluxo de dados em simulações computacionais, como a qualidade do resultado obtido e o fluxo de arquivos gerado pela sua execução.

Além desta introdução, esta monografia é composta de mais seis capítulos. No Capítulo 2 são apresentados os principais conceitos que serão utilizados ao longo desta monografia. No Capítulo 3, são apresentados trabalhos relacionados, ou seja, trabalhos que também permitem a extração de dados científicos. O Capítulo 4 apresenta a DfAnalyzer, uma biblioteca de componentes que permite o monitoramento, a depuração e a análise do fluxo de dados de aplicações científicas em tempo de execução, com apoio a ambientes de PAD [3]. O Capítulo 5 apresenta a solução desenvolvida nesta monografia: Uma biblioteca Python para a extração de dados de proveniência e de dados científicos utilizando a DfAnalyzer. No Capítulo 6, são apresentados os resultados

experimentais que compararam o desempenho computacional da captura de dados de proveniência entre a solução desenvolvida nesta monografia e o noWorkflow [4] em duas simulações computacionais. Essas simulações correspondem a aplicações reais, sendo a primeira uma aplicação na área de meteorologia e a outra na área de multifísica. No Capítulo 7 são apresentadas as conclusões.

2. Referencial Teórico

Este capítulo apresenta o referencial teórico com os principais conceitos discutidos nesta monografia, a fim de facilitar o seu entendimento. Nesse sentido, são descritos os ambientes de Processamento de Alto Desempenho (PAD) e as aplicações de Ciência Computacional e Engenharia (CSE), assim como são apresentados os conceitos de fluxo de dados, proveniência de dados e os tipos de gerência de fluxo de dados.

2.1. Ambientes de Processamento de Alto Desempenho (PAD)

Ambientes de Processamento de Alto Desempenho (PAD) são caracterizados principalmente pela execução de tarefas paralelas, permitindo um melhor desempenho do processamento das simulações computacionais. A paralelização de uma aplicação está relacionada à separação dessas tarefas e a distribuição das mesmas para a execução em diferentes recursos computacionais. Em cenários de simulações computacionais em larga escala, ambientes de PAD têm sido utilizados para apoiar o paralelismo no processamento do grande volume de dados envolvido. Dentre os Ambientes de PAD que atendem à essas necessidades estão os *clusters*, as grades e as nuvens computacionais, que permitem que os usuários do domínio científico tratem problemas em escalas maiores, o que não seria possível utilizando ferramentas convencionais.

Cluster é um ambiente que relaciona dois ou mais computadores independentes para que estes trabalhem de maneira conjunta no processamento de uma tarefa. Os computadores dividem entre si as atividades de processamento e executam este trabalho de maneira simultânea. Grade computacional é um modelo capaz de alcançar uma alta taxa de processamento dividindo as tarefas entre as diversas máquinas, podendo ser em rede local ou de longa distância. Esses processos podem ser executados no momento em que as máquinas estão sendo utilizadas pelo usuário, assim evitando o desperdício de processamento da máquina utilizada. Nuvens computacionais dedicam-se à utilização de recursos computacionais alocados dinamicamente de acordo com a necessidade dos usuários, tirando proveito da elasticidade e da escalabilidade desse ambiente.

2.2. Aplicações de Ciência Computacional e Engenharia (CSE)

Em [5] Ciência Computacional e Engenharia é definida como sendo um campo multidisciplinar presente na interseção de matemática e estatística, ciência da computação, e outras disciplinas fundamentais das ciências e engenharia. O foco da CSE está na integração do conhecimento e as metodologias dessas diversas áreas e no desenvolvimento de novas ideias em suas interfaces [5]. Dessa forma, esse campo auxilia no processo de modelagem computacional de problemas de diversas áreas da ciência tais como química, genética e dinâmica dos fluidos. As instanciações desses modelos computacionais também são conhecidas como Aplicações CSE e são um tipo de simulação computacional. Ademais, ambientes de PAD são comumente utilizados para executar esse tipo de aplicações devido à complexidade de seus modelos computacionais e à necessidade de realização de experimentos em larga escala

2.3. Fluxo de dados

Um fluxo de dados pode ser definido como o encadeamento de transformações de dados que geram, consomem e propagam conjuntos de dados. Cada conjunto de dados é composto por conjuntos de elementos de dados. Uma transformação de dados consome dados de um ou mais conjuntos de dados de entrada e produz um ou mais conjuntos de dados de saída [6], conforme apresentado na figura 2.1. Nas subseções a seguir são definidos os conceitos relativos a fluxo de dados.

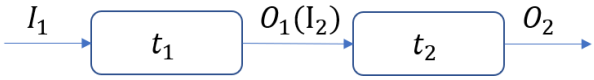

Figura 2.1 - Exemplo de fluxo de dados com transformações de dados $\boldsymbol{t_1}$ e $\boldsymbol{t_2}$, conjunto de dados de entrada $\boldsymbol{I_1}$ e conjuntos de dados de saída $\boldsymbol{O_1}$ e $\boldsymbol{O_2}$.

2.3.1. Atributo de dados

Um atributo de dados é um descritor de características de elementos de dados presentes em um conjunto de dados. Para realizar isso um atributo de dados possui duas propriedades, sendo elas: nome e tipo. O nome serve para identificar elementos de dados presentes num conjunto de dados e, portanto, deve possuir um valor único. O tipo representa a natureza daquele dado podendo assumir diversos valores, dentre eles temos:

texto, número, arquivo, etc. Na Tabela 2.1 são apresentados exemplos de atributos de dados.

Tabela 2.1– Exemplos de atributo de dados

Nome do atributo	Tipo do atributo
Nome	Texto
Idade	Número

2.3.2. Elemento de dados

Um elemento de dados é composto por um conjunto de valores pertinentes a um conjunto de atributos previamente definidos [6]. Portanto, o número de valores disponíveis em cada elemento de dados deve ser igual ao tamanho do conjunto de atributos ao qual faz referência.

2.3.3. Conjunto de dados

Um conjunto de dados é formado por um conjunto de elementos de dados $E = \{e_1,\ e_2, \dots, e_x\}$, sendo x o número de elementos de dados. Cada elemento de dados possui uma sequência de atributos predefinidos $A = \{a_1, a_2, \dots, a_y\}$, sendo y o número de atributos de dados [6].

2.3.4. Transformação de dados

Uma transformação de dados é definida pelo consumo de um ou mais conjuntos de dados de entrada e pela produção de um ou mais conjuntos de dados de saída. Na figura 2.2 temos representado um exemplo de transformação de dados.

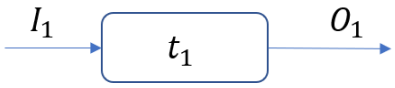

Figura 2.2 – Exemplo de transformação de dados $\boldsymbol{t_1}$ com conjunto de dados de entrada $\boldsymbol{I_1}$e conjunto de dados de saída $\boldsymbol{O_1}$.

2.3.5. Dependência de dados

Sejam duas transformações de dados $t_1 e\ t_2$e um conjunto de dados de saída $o_1$de $t_1$dizemos que $t_2$possui uma dependência de dados em relação a $t_1$se e somente se $o_1$for conjunto de dados de entrada de t_2. Na Figura 2.1 está representado um exemplo de dependência de dados no qual o conjunto de dados de saída da primeira transformação de

dados é utilizado como conjunto de dados de entrada para a segunda transformação de dados.

2.3.6. Exemplo

Para exemplificar os conceitos de fluxo de dados apresentados, vamos utilizar um fragmento de um fluxo de dados de uma aplicação real de multifísica, conforme apresentado na Figura 2.3.

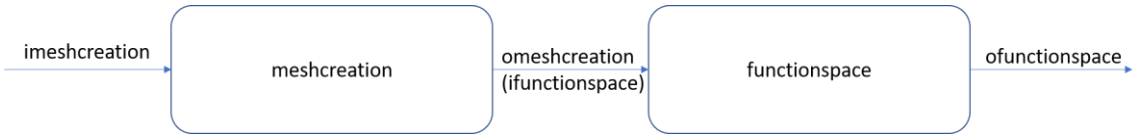

Figura 2.3 – Fluxo de dados de uma aplicação real de multifísica.

2.4. Dados de Proveniência

Um dos pilares do método científico é a reprodutibilidade do experimento que, no contexto computacional, seria a capacidade de assegurar quais etapas foram executadas e quais dados foram gerados, consumidos e propagados em cada etapa da simulação computacional. Nesse contexto, torna-se desejável capturar os dados gerados, consumidos e propagados ao longo da execução da simulação computacional. Esses dados são conhecidos como dados de proveniência.

2.4.1. Definição

Dados de proveniência possuem informações referentes à origem e ao destino de um dado. A captura de dados de proveniência é importante, pois possibilita que os usuários do domínio científico realizem análises, durante ou após a execução, para examinar a qualidade de um determinado resultado obtido, rastrear o fluxo de dados envolvido na simulação computacional ou mesmo monitorar comportamentos inesperados em tempo de execução [7]. Essas análises favorecem a reprodutibilidade e corroboram para a aceitação da simulação computacional.

Ademais, os dados de proveniência podem ser utilizados para facilitar o processo de tomada de decisão por parte de usuários do domínio científico durante ou após a execução com o intuito de obter melhores resultados e reduzir o tempo total de execução [8].

2.4.2. Tipos de dados de proveniência

Os dados de proveniência são divididos nas categorias proveniência prospectiva e proveniência retrospectiva. A proveniência prospectiva é obtida antes mesmo do início da execução da simulação computacional e é responsável por especificar qual será o fluxo de dados ao longo da execução da simulação computacional por meio da especificação de suas transformações de dados e das dependências de dados entre elas.

A proveniência retrospectiva é o produto da execução da simulação computacional contendo informações sobre os dados gerados, consumidos e propagados ao longo da execução [9]. Dentre esses dados temos dados de desempenho, domínio, arquivos consumidos, etc.

Existem diversos sistemas que apoiam a captura e o registro desses dados de proveniência. Dentre esses sistemas podemos citar noWorkflow [4], YesWorkflow [10], SPADE [11] e DfAnalyzer [3]. Além disso, a captura e o registro dos dados de proveniência pode ser realizada de forma *online* e *offline*. Na captura *offline*, o armazenamento dos dados de proveniência ocorre após ao término da execução da simulação computacional. Portanto, o usuário do domínio científico está impedido de utilizar dos dados de proveniência para tomar decisões que impactem a execução corrente como realizar ajustes finos visando reduzir o tempo total de execução e melhorar a qualidade do resultado final [8]. Na captura online, por sua vez, o armazenamento dos dados de proveniência ocorre durante o processo de execução possibilitando que o usuário do domínio científico realize análises sobre os dados gerados ao longo do experimento que favoreçam o processo de tomada de decisão ao longo da execução da simulação computacional [8].

Ademais, é importante salientar que a captura de dados de proveniência *online* pode gerar uma perda de eficiência devido à concorrência por recursos computacionais entre a simulação científica e o sistema responsável pela captura dos dados de proveniência.

3. Trabalhos Relacionados

O objetivo dessa monografia consiste no desenvolvimento de uma biblioteca feita em Python para permitir a extração de dados de proveniência e de dados científicos de simulações computacionais utilizando a ferramenta DfAnalyzer (introduzida no Capítulo 4). Neste capítulo são apresentadas ferramentas que possibilitam essa captura de dados de proveniência. As soluções existentes podem ser separadas entre as que não demandam a instrumentação dos programas científicos e as que demandam.

As soluções que não demandam instrumentação do código fonte das simulações computacionais costumam realizar a captura de dados de proveniência de forma não intrusiva, buscando obter informações por meio da utilização de técnicas de Engenharia de Software, como análise de árvore de abstração sintática, reflexão e perfilhamento, para coletar diferentes tipos de proveniência sem requerer um sistema de controle de versão e uma instrumentação do ambiente como é o caso do noWorkflow [4]. Porém, a granularidade dos dados de proveniência capturados costuma não ser personalizável dificultando ou até mesmo inviabilizando a realização de algumas análises sobre esses dados.

As soluções que demandam instrumentação do código fonte da simulação computacional costumam definir modelos e sintaxes para a realização de anotações como o YesWorkflow [10] ou prover um *middleware* que possibilite a extração e o registro de dados de proveniência como o SPADE [11]. Essas soluções têm como desvantagem o aspecto intrusivo gerado pela necessidade de instrumentação do código fonte da simulação computacional e o fato de ser suscetível a erro humano no momento da instrumentação. Dentre as vantagens está a capacidade de adequar a granularidade dos dados de proveniência de acordo com as análises que deverão ser realizadas posteriormente.

O noWorkflow é uma ferramenta cujo objetivo é permitir a captura de dados de proveniência de forma não intrusiva em *scripts* desenvolvidos na linguagem de programação Python [4]. Os dados de proveniência capturados por essa ferramenta são armazenados em duas bases distintas. Uma delas é uma base de conteúdo, presente no diretório de arquivos comum, na qual são armazenadas as referências brutas tais como as bibliotecas utilizadas (direta ou indiretamente) pelo *script* principal, as versões de arquivos antes e depois de serem modificados, definição das funções utilizadas, etc.

Já a outra base armazena dados que referenciam os registros armazenados na base de conteúdo, dentre eles estão: nome de funções, parâmetros utilizados, variáveis globais, chamada de funções, configurações de ambiente, etc. Além da captura dos dados de proveniência essa ferramenta também permite a análise sobre a base relacional gerada pela solução durante a execução do *script*, por meio da especificação consultas em SQL. Além disso, permite exportar regras relativas aos dados registrados na base relacional para especificar e executar consultas em PROLOG.

Todavia, essa ferramenta também apresenta limitações, as análises sobre os dados de proveniência só podem ser realizadas de forma *offline*, ou seja, após o término da execução do experimento; a ferramenta não permite a especificação da proveniência prospectiva; e o usuário do domínio científico não possui pleno controle sobre quais dados serão capturados e sobre qual será a granularidade desses dados podendo inviabilizar a realização de algumas análises. Isso ocorre, pois essa ferramenta permite que o fluxo de execução seja capturado com apenas dois níveis de granularidade. O modo de captura padrão coleta ativações de funções, variáveis globais, parâmetros de funções e valores de retorno. O outro modo disponibilizado pela ferramenta é o *Tracer* que além das informações capturadas no modo padrão também captura atribuição de variáveis, definição de loops e outras dependências entre variáveis [12].

O YesWorkflow é uma ferramenta que especifica um modelo e uma sintaxe para a definição de comentários que serão introduzidos ao código fonte das simulações computacionais para serem posteriormente analisados possibilitando a extração de dados de proveniência e de dados científicos[10]. Ao contrário do noWorkflow, o YesWorkflow demanda instrumentação por parte do usuário do domínio científico sendo possível especificar a proveniência prospectiva. Além disso, como comentários são ignorados pelos compiladores essa abordagem torna-se agnóstica de linguagem de programação favorecendo a sua adoção.

O SPADE é um *middleware open source* para captura de dados de proveniência [11]. Essa solução fornece um conjunto de interfaces que podem ser utilizadas para armazenar e realizar consultas sobre dados de proveniência. Ao contrário do noWorkflow e do YesWorkflow o SPADE registra os dados de proveniência utilizando um modelo de dados específico, o W3C-PROV. Com isso, os dados de proveniência capturados por diferentes aplicações científicas são instâncias do mesmo modelo. Destarte, as melhorias realizadas nos mecanismos de captura e registro dos dados de proveniência impactariam

a base de usuários (usuários do domínio científico) como um todo. Isso atrelado ao aspecto *open source* da solução favorece a melhoria contínua da solução permitindo que passe a tornar esse processo de captura e registro cada vez mais eficiente.

4. DfAnalyzer

Simulações computacionais têm demandado a execução de modelos computacionais cada vez mais complexos e gerado um grande volume de dados necessitando de mais poder de processamento computacional. Por conta disso, mesmo ao serem executadas em ambientes de PAD, costumam demandar bastante tempo. Além disso, simulações computacionais comumente têm seu fluxo de dados gerado por meio da escrita de arquivos em disco. Em ambientes de PAD isso pode gerar dificuldades para os usuários do domínio científico realizarem o monitoramento e as análises sobre o fluxo de dados gerado pela execução da simulação computacional, pois o sistema de arquivos utilizado costuma ser o distribuído dificultando a reconstrução e a análise desses fluxos de dados.

Nesse sentido, a DfAnalyzer [3] foi proposta pelo nosso grupo de pesquisa como uma biblioteca de componentes que permite tanto a captura, como a análise de dados científicos, durante a execução de simulações computacionais. A DfAnalyzer possui uma arquitetura modularizada na qual os componentes são separados, segundo os serviços (funcionalidades) que fornecem, nas seguintes camadas: extração de dados de proveniência e dados científicos, armazenamento de dados e análise de dados científicos.

Os dados de proveniência e científicos extraídos são armazenados, em tempo de execução, em uma base de dados conhecida como DfDB presente no SGBD (Sistema de Gerência de Banco de Dados) MonetDB [13]. Esses dados podem ser relativos à performance, ao domínio da aplicação, ponteiros para arquivos, etc. A utilização de um SGBD como forma de armazenamento desses dados de proveniência favorece a realização de análises e visualizações. Essas análises podem ser realizadas por meio da execução de consultas SQL.

Contudo, para que o usuário do domínio científico possa realizar essas análises sem precisar especificar consultas SQL, a DfAnalyzer disponibiliza o componente QI (*Query Interface*). Ademais, a DfAnalyzer também dispõe de um módulo chamado DfViewer que permite a visualização da especificação do fluxo de dados em forma de um grafo direcionado acíclico (DAG). Nas subseções a seguir serão apresentados os componentes pertencentes a cada uma das categorias apresentadas anteriormente.

4.1. Extração de dados de proveniência e de dados científicos

Essa camada permite a extração de dados de proveniência e de dados científicos por meio da instrumentação da simulação computacional. Esses dados podem ser extraídos de variáveis alocadas em memória ou pela execução de extratores e indexadores de dados científicos que podem ser programas externos (ou ferramentas alternativas, como FastBit [14]) cuja função é a de extrair ou indexar dados de domínio presentes em arquivos de formatos específicos. A DfAnalyzer possui três componentes que permitem desempenhar essas funções, sendo eles: PDE (*Provenance Data Extractor*), RDE (*Raw Data Extractor*) e RDI (*Raw Data Indexer*). Esses componentes estão descritos detalhadamente nas subseções a seguir.

Os dados extraídos podem ser armazenados na DfDB por meio de chamadas HTTP à API RESTful da DfAnalyzer. Essa API possui *endpoints* para realizar a inserção desses dados definidos como entidades do modelo de fluxo de dados apresentado na Seção 2.3. Dessa forma, o usuário do domínio científico deveria construir essas entidades (podendo ser classes se utilizar o paradigma de programação orientado a objetos) podendo restringir a adesão por parte da comunidade científica não contribuindo com a intenção da DfAnalyzer de ser uma ferramenta intuitiva.

Além disso, cada requisição apresenta uma estrutura própria de mensagem e que não pode ser validada a priori, pelos usuários dificultando o processo de depuração durante a instrumentação da simulação computacional. Desta forma, a DfAnalyzer poderia disponibilizar bibliotecas que já forneçam essa estrutura e a integração com a API RESTful da DfAnalyzer.

Com essas bibliotecas o usuário do domínio científico poderia realizar chamadas RPC (*Remote Procedure Call*) para realizar a captura de dados de proveniência ou, caso seus programas científicos estejam escritos nas linguagens de programação dessas bibliotecas, poderia importá-las como dependências e realizar as operações necessárias. Os componentes dessa camada são: PDE (*Provenance Data Extractor*), RDE (*Raw Data Extractor*) e RDI (*Raw Data Indexer*). Esses componentes sãoapresentados detalhadamente nas subseções a seguir.

4.1.1. *Provenance Data Extractor* (PDE)

O PDE é o componente responsável pela extração de dados de proveniência. Esse componente fornece um conjunto de classes baseadas no modelo conceitual de fluxo de

dados apresentado na Seção 2.3. O usuário do domínio computacional pode modelar o fluxo de dados da aplicação por meio da instanciação e preenchimento dessas classes. A extração desses dados é feita por meio da instrumentação da simulação computacional por meio da utilização das bibliotecas disponibilizadas e os dados de proveniência capturados são armazenados na DfDB. Nessa versão da DfAnalyzer, as instrumentações do código da aplicação científica exigem a adição de requisições HTTP à API RESTful da DfAnalyzer para registrar esses dados de proveniência na base de dados DfDB.

4.1.2. *Raw Data Extractor* (RDE)

O RDE é responsável pela extração de dados de domínio presentes em arquivos. O componente permite que o usuário do domínio científico defina arquivos dos quais deseja extrair dados de domínio, um tipo de cartucho que será utilizado para realizar a extração (podendo ser, por exemplo, um programa capaz de extrair os dados do formato de arquivo escolhido) e os atributos de interesse para serem utilizados na etapa de filtragem [6]. O usuário do domínio científico deve invocar este componente com os parâmetros listados anteriormente obtendo como retorno os dados extraídos já filtrados pelos atributos fornecidos. Portanto, esse componente é extensível visto que, para suportar a extração de um tipo de arquivo novo basta que o usuário do domínio científico utilize o cartucho do tipo programa fornecendo o código necessário para extrair os dados desse novo tipo de arquivo.

4.1.3. *Raw Data Indexer* (RDI)

O RDI é o componente responsável pela indexação de dados de domínio presentes em arquivos. O componente permite que o usuário do domínio científico especifique um tipo de cartucho, o arquivo que deseja indexar e os atributos presentes nesse arquivo para os quais deseja criar os índices. O tipo de cartucho é responsável por indicar qual algoritmo de indexação deverá ser utilizado. A indexação dos dados de domínio pode impactar no tempo de processamento, pois ocorre nos mesmos nós computacionais nos quais a simulação computacional está sendo executada. Porém, a indexação desses dados torna-se mais vantajosa com o crescimento da complexidade da estrutura dos dados de domínio que são acessados mais frequentemente. Dessa forma, ao indexar esses dados facilitamos o acesso direto contribuindo para a redução do tempo de execução e favorecendo o gerenciamento dos metadados associados ao fluxo de dados [6].

4.2. Armazenamento de dados

Essa camada é a responsável por armazenar os dados de proveniência de forma a facilitar a recuperação de informação para realização de análises e visualizações posteriormente. Dentre os elementos pertencentes a essa camada estão os arquivos de dados científicos, os quais podem ter os dados extraídos ou indexados conforme mostrado nas subseções 4.1.2 e 4.1.3, respectivamente. Nessa camada também se encontra a base de dados DfDB na qual os dados de proveniência extraídos por meio do PDE são registrados. Sobre essa base de dados podem ser realizadas análises por meio da execução de consultas especificadas na linguagem SQL (geradas pelo *Query Interface*, conforme apresentado na Seção 4.3) e também podem ser geradas visualizações sobre o fluxo de dados utilizando o *DfViewer*, conforme apresentado na Seção 4.3.

4.3. Análise de dados científicos

Essa camada é responsável por permitir as análises sobre os dados de proveniência capturados utilizando o PDE e os dados científicos extraídos/indexados pelos componentes RDE/RDI. A amplitude das análises possíveis depende diretamente da granularidade dos dados de proveniência capturados ao longo da execução da simulação computacional. Portanto, o usuário do domínio científico deve instrumentar o código fonte de sua simulação computacional tendo em mente quais análises ele deseja realizar. As análises possíveis contemplam tanto a execução de consultas especificadas sem a necessidade de utilização de linguagens declarativas como o SQL, quanto a análise por meio da geração de visualizações sobre o fluxo de dados gerado ao longo da execução da simulação computacional. Essa camada é composta pelos componentes *Query Interface* (QI) e DfViewer. Mais detalhes sobre os recursos analíticos podem ser vistos em [3].

5. DfA-lib-Python: Uma biblioteca para a extração de dados científicos usando a DfAnalyzer

Neste capítulo é apresentado o processo de desenvolvimento da biblioteca em Python para a extração de dados de proveniência utilizando a DfAnalyzer. Com essa biblioteca o usuário do domínio científico pode realizar a extração de dados de proveniência de simulações computacionais escritas em Python por meio de sua instrumentação e representar os dados extraídos segundo o modelo de dados apoiado pela DfAnalyzer. Durante o processo de instrumentação são apenas adicionadas linhas referentes a captura dos dados de proveniência e de dados científicos desejados sem que sejam modificadas as especificações já presentes no código fonte da simulação computacional. Posteriormente, esses dados são armazenados em uma base de dados usando o SGBD MonetDB [13] por meio de requisições HTTP à API RESTful da DfAnalyzer, possibilitando a realização de análises durante a execução da simulação. Ao mesmo tempo, essas análises são importantes, pois auxiliam o usuário do domínio científico a validar os resultados obtidos favorecendo a reprodutibilidade das suas simulações.

Portanto, essa biblioteca permite capturar o fluxo de dados de simulações computacionais escritas em Python, respeitando-se o formalismo da nossa abstração e disponibilizando invocações para a API RESTful da DfAnalyzer, de forma que o usuário do domínio científico não conheça, a priori, as assinaturas de cada conceito (da abstração de fluxo de dados) em função dos métodos desenvolvidos em classes na linguagem de programação Python. Destarte, a solução desenvolvida funciona como uma interface entre a aplicação científica de domínio e a API RESTful da DfAnalyzer. A documentação da biblioteca desenvolvida está disponível em [15].

5.1. Tecnologias

A biblioteca foi desenvolvida na linguagem Python na versão *3.5.2* utilizando o paradigma de programação de orientação a objetos. Esse paradigma foi escolhido, pois além de ser amplamente utilizado na academia e na indústria, ele também favorece o processo de modelagem de entidades de dados. O sistema de gerência de pacotes utilizado foi o *pip* [16]. A ferramenta *virtualenv* [17] foi utilizada para isolar o ambiente Python.

O sistema de gerência de versão distribuído utilizado foi o Git [18]. O editor de texto utilizado foi o VisualStudio Code [19]. A biblioteca *requests* [20] foi utilizada para realizar as requisições HTTP para a API RESTful da DfAnalyzer. Somando-se a isso, foi utilizado o framework de testes *pytest* [21] para criação, execução e gerenciamento dos testes unitários criados para as classes que compõe a biblioteca desenvolvida.

5.2. Captura de dados de proveniência

A captura dos dados de proveniência pode ser feita por meio da utilização dos componentes disponibilizados por esta biblioteca (PDE, RDE e RDI). O PDE tem como responsabilidade permitir a captura dos dados relativos à especificação do fluxo de dados e gerados ao longo da execução da simulação computacional. O RDE e o RDI permitem a extração e a indexação de dados científicos em arquivos do domínio, respectivamente. Esses módulos são apresentados detalhadamente nas seções a seguir.

Os dados científicos extraídos são utilizados para preencher as entidades relativas ao modelo de dados de proveniência seguido pela DfAnalyzer. Essas entidades foram modeladas em classes e podem ser utilizadas através da importação desta biblioteca. Essas classes podem ser divididas em duas categorias referentes ao tipo de proveniência que permitem capturar, sendo essas prospectiva e retrospectiva. Porém, existem duas classes que são utilizadas em ambas as categorias: *ProvenanceObject* e *Dependency*. *ProvenanceObject* é a classe responsável por definir as propriedades e métodos comuns a todos objetos de proveniência sendo, consequentemente, herdada pelas demais classes. A classe *Dependency* representa a dependência de dados em nível lógico e físico podendo ser utilizada pela proveniência prospectiva e retrospectiva. As demais classes pertencentes aos tipos de proveniência suportados serão abordadas nas seções a seguir.

A proveniência prospectiva é responsável por especificar qual será o fluxo de dados ao longo da simulação computacional. Portanto, sua especificação pode ser feita antes do início da execução do experimento científico. O modelo de dados de proveniência prospectiva utilizado pela DfAnalyzer é baseado nos conceitos de fluxos de dados apresentados na Seção 2.3. Na Tabela 5.1 são apresentadas as classes criadas para capturar dados de proveniência prospectiva junto às suas respectivas funções.

Tabela 5.1 – Nomes e definições das classes criadas na DfA-lib-Python para permitir a captura de dados de proveniência prospectiva.

Nome	Definição
Attribute	Atributo de dados
AttributeType	Enumerador dos tipos de atributos de dados possíveis
Set	Conjunto de dados
SetType	Enumerador dos tipos de conjuntos de dados possíveis (entrada e saída)
Program	Programa científico
File	Arquivo em disco
Extractor	Extrator de dados
ExtractorCartridge	Enumerador dos cartuchos suportados pelo extrator
ExtractorExtension	Enumerador das extensões de arquivos suportadas pelo extrator
Transformation	Transformação de dados
Dataflow	Fluxo de dados

A classe Attribute permite que o usuário do domínio científico especifique um descritor para uma característica de um dado de domínio fornecendo um tipo (representado pela classe AttributeType) e um nome para esse atributo. Para representar um conjunto de dados o usuário do domínio científico pode instanciar e preencher a classe Set que recebe uma *tag*, um tipo (SetType), atributos (Attribute) aos quais os valores dos elementos de dados presentes no conjunto de dados fazem referência, opcionalmente extratores (Extractor) responsáveis por extrair esses atributos e as possíveis dependências de dados desse conjunto de dados. Além disso, ao longo da execução das simulações computacionais são comumente realizadas escritas e leitura de arquivos e são executados programas externos. O usuário do domínio científico pode registrar essas etapas usando as classes File e Program, respectivamente. A classe File recebe como parâmetros o diretório do arquivo modificado (lido ou escrito) e o nome desse arquivo. A classe Program também recebe os parâmetros nome e diretório, responsáveis por especificar qual foi o programa utilizado. Ademais, o usuário do domínio científico pode realizar extrações de dados de interesse de arquivos de domínio. Para registrar essa etapa pode utilizar a classe Extractor que recebe como parâmetros uma *tag*, o cartucho utilizado (uma instância da classe ExtractorCartridge) e a extensão do arquivo ao qual o extrator foi utilizado (uma instância da classe ExtractorExtension). Ademais, O usuário do domínio

científico pode especificar uma transformação de dados utilizando a classe Transformation que recebe como parâmetros uma *tag* e uma lista de conjuntos de dados. A especificação do fluxo de dados de execução pode ser especificada com a classe Dataflow que possui uma *tag* e uma lista de transformações de dados. As especificações e os exemplos de uso da biblioteca desenvolvida para a captura de proveniência prospectiva são apresentadas mais detalhadamente em [15].

A proveniência retrospectiva é o produto da execução da simulação computacional contendo informações sobre os dados gerados, consumidos e propagados ao longo da execução [9]. Esse tipo de proveniência é capturado por meio da instrumentação dos programas científicos. O modelo de dados de proveniência retrospectiva utilizado pela DfAnalyzer é baseado nos conceitos de fluxos de dados apresentados na Seção 2.3. Na Tabela 5.2 são apresentadas as classes criadas para capturar os dados de proveniência retrospectiva junto à suas respectivas funções.

Tabela 5.2 – Nomes e definições das classes criadas na DfA-lib-Python para permitir a captura de dados de proveniência retrospectiva.

Nome	Definição
Element	Elemento de dados
DataSet	Conjunto de dados
MethodType	Enumerador dos tipos de atividade computacional realizada
Performance	Enumerador com os tipos de conjuntos de dados possíveis (entrada e saída)
Task	Tarefa de um fluxo de dados (instanciação de uma transformação de dados)
TaskStatus	Enumerador com todos os possíveis estados de uma tarefa

O usuário do domínio científico pode especificar elementos de dados usando a classe Element que recebe como parâmetros uma lista de valores relativos à dados de domínio. A classe DataSet representa um conjunto de dados instanciado, ou seja, uma lista de elementos de dados gerados, consumidos e propagados ao longo da execução da simulação computacional. Essa classe possui como parâmetros uma *tag* e uma lista de elemento de dados. Além disso, dados sobre o desempenho podem ser capturados utilizando a classe Performance que possui como parâmetros obrigatórios os tempos de início e de fim de uma execução e como parâmetros opcionais o método medido (e.g. instrumentação, indexação, representado mapeados na classe MethodType) e uma descrição para a medição feita. Ademais, A classe Task representa uma transformação de

dados instanciada, ou seja, uma transformação de dados que possui informações relativas a execução da simulação computacional como os conjuntos de dados consumidos e produzidos e informações de performance. Essa classe possui como parâmetros um identificador (id), a tag do fluxo de dados ao qual pertence (dataflow_tag) e a tag da transformação de dados que representa (transformation_tag). Essa classe ao ser instanciada pode assumir valores distintos para seus estados (representados TaskStatus) e ter dados de desempenho atrelados à sua execução (representados pela classe Performance). As especificações e os exemplos de uso da biblioteca desenvolvida para a captura de proveniência retrospectiva são apresentadas mais detalhadamente em [15].

5.3. Captura de dados científicos

Já o processo de captura de dados científicos pode ser baseado em duas abordagens, como visto na apresentação da DfAnalyzer: extração ou indexação dos dados. Nesse sentido, o componente RDE é responsável pela extração de dados de domínio presentes em arquivos. A classe *RawDataExtractor* foi criada para desempenhar essa tarefa. Para utilizá-la o usuário do domínio científico deverá informar o tipo de arquivo que quer extrair, uma linha de comando a ser executada e os atributos que deseja filtrar. A documentação detalhada dessa classe está disponível em [15].

Enquanto isso, o RDI é responsável pela indexação de dados de domínio presentes em arquivos. A classe *RawDataIndexer* foi criada para desempenhar essa tarefa. Para utilizá-la, o usuário do domínio científico deverá informar o tipo de arquivo que quer extrair, o nome do extrator que foi utilizado, o diretório absoluto no qual o arquivo com os dados extraídos se encontra, os atributos que deseja indexar e os seus tipos (*e.g.*, ponto flutuante). A documentação detalhada dessa classe está disponível em [15].

5.4. Exemplo de instrumentação

Nesta seção apresenta-se um fragmento de uma aplicação real na área de multifísica. Esse fragmento possui a especificação de uma etapa do fluxo de dados (proveniência prospectiva) e os dados gerados pela sua execução (proveniência retrospectiva). A Figura 5.1 e a Figura 5.2 apresentam fragmentos de uma aplicação multifísica utilizados para capturar proveniência prospectiva e retrospectiva, respectivamente.

```
# Prospective provenance capture
dataflow_tag = "fenics-df"
df = Dataflow(dataflow_tag)
# Transformation MeshCreation
tf1 = Transformation("MeshCreation")
tf1_input = Set("iMeshCreation", SetType.INPUT,
    [Attribute("HEIGHT", AttributeType.NUMERIC),
    Attribute("WIDTH", AttributeType.NUMERIC)])
tf1_output = Set("oMeshCreation", SetType.OUTPUT,
    [Attribute("VERTICES", AttributeType.NUMERIC),
    Attribute("CELLS", AttributeType.NUMERIC)])
tf1.set_sets([tf1_input, tf1_output])
df.add_transformation(tf1)
```

Figura 5.1 – Fragmento de uma aplicação de multifísica para captura de proveniência prospectiva.

```
################################################################
# Mesh Creation
################################################################
retrospective_provenance_start = datetime.datetime.now()
t1 = Task(1, dataflow_tag, "MeshCreation")
t1_input = DataSet("iMeshCreation", [Element([96, 96])])
t1.add_dataset(t1_input)
t1.begin()
#---------------------------------------------------------------
# Create mesh
mesh = UnitSquareMesh(96, 96)
#---------------------------------------------------------------
t1_output= DataSet("oMeshCreation", [Element([mesh.num_vertices(), mesh.num_cells()])])
t1.add_dataset(t1_output)
t1.end()
```

Figura 5.2 – Fragmento de uma aplicação de multifísica para captura de proveniência retrospectiva.

6. Avaliação Experimental

Neste capítulo são apresentadas duas simulações computacionais realizadas para comparar o custo da extração de dados de proveniência e de dados científicos durante a execução de aplicações de CSE escritas em Python utilizando a DfA-lib-Python e o noWorkflow.

6.1. Simulações Computacionais

As duas simulações computacionais apresentadas nesse capítulo são compostas por aplicações de CSE escritas em Python das áreas de meteorologia e multifísica. Em ambas as simulações computacionais foram criadas três versões para cada uma das aplicações de CSE, sendo elas: *baseline*, noWorkflow e DfA-lib-Python. A versão *baseline* contém a aplicação original sem modificações e será utilizada como base de comparação para as versões executadas utilizando a DfA-lib-Python e o noWorkflow.

As versões que utilizam a DfA-lib-Python englobam as aplicações de CSE originais instrumentadas para permitir a extração de dados de proveniência e de dados científicos usando a biblioteca de componentes DfAnalyzer. Essas versões foram avaliadas segundo o tempo de execução e o custo de armazenamento em disco. Cada versão teve seu tempo de execução separado nos tempos de processamento computacional, captura de proveniência prospectiva e captura de proveniência retrospectiva.

O processamento computacional representa o tempo da execução da aplicação de CSE desconsiderando o tempo adicional gerado pela extração de dados de proveniência. Os tempos de captura de dados de proveniência prospectiva e retrospectiva são os tempos de execução para a extração de dados de proveniência antes e durante a execução da aplicação de CSE, respectivamente.

Os resultados apresentados para as versões de ambos os experimentos representam a média de cinco execuções. O noWorkflow não permite a especificação de proveniências prospectiva. Porém, executa uma etapa antes da execução do script principal na qual captura a dependência dos módulos utilizados. Portanto, para essa avaliação experimental, essa etapa foi considerada como captura de proveniência prospectiva.

As especificações do ambiente computacional no qual os experimentos foram realizados são as listadas a seguir:

- Sistema Operacional: Ubuntu 16.04 LTS
- Processador: Intel Core i7-5500U
- Memória RAM: 8GB

6.2. Aplicação de CSE de meteorologia

Nesta seção é apresentada uma aplicação de CSE de meteorologia utilizando as ferramentas DfA-lib-Python e noWorkflow para permitir a extração de dados de proveniência e de dados científicos. Essa aplicação de CSE foi executada nas versões *baseline*, noWorkflow e DfA-lib-Python. A versão noWorkflow foi executada utilizando as configurações padrão e *Tracer* (apresentadas no Capítulo 3).

Essa aplicação de CSE é sequencial e possui como entrada um conjunto de dados contendo a temperatura e a precipitação para um determinado intervalo de dias e prevê a temperatura e pressão para dias próximos no futuro. A aplicação foi originalmente apresentada em [4] e seu código foi obtido em [22], porém, o conjunto de dados de entrada foi ampliado para possibilitar as avaliações de desempenho realizadas.

Foram adicionadas etapas de pré-processamento na aplicação de CSE para formatar o novo conjunto de dados de entrada de acordo com o original. Essa etapa de pré-processamento é composta por três funções, sendo duas delas responsáveis pela carga dos dados referentes à temperatura e precipitação em memória e a outra por colocar os dados no formato adequado e realizar a filtragem para remover erros de medição (valores negativos para temperatura e precipitação, por exemplo). Após essa etapa existem dois conjuntos de entrada cada um com 3.467 registros.

Na Tabela 6.1 são apresentados os tempos médios dessas execuções para cada uma das versões dos sistemas envolvidos na comparação de desempenho. A partir desses tempos foi calculado o custo em termos de tempo para a extração de dados de proveniência pelas ferramentas noWorkflow (nas duas configurações apresentadas) e a DfA-lib-Python. Esse custo é calculado como sendo a razão entre o tempo de execução da solução e o tempo sem a extração de dados de proveniência e de dados científicos (*baseline*). Os custos calculados estão mostrados na Figura 6.1.

Tabela 6.1 – Tempos médios de execução para cada uma das versões da aplicação CSE de meteorologia.

Versões	Tempo médio de execução (s)
Baseline	3,21
DfA-lib-Python	6,08
noWorkflow	133,91
noWorkflow-Tracer	831,72

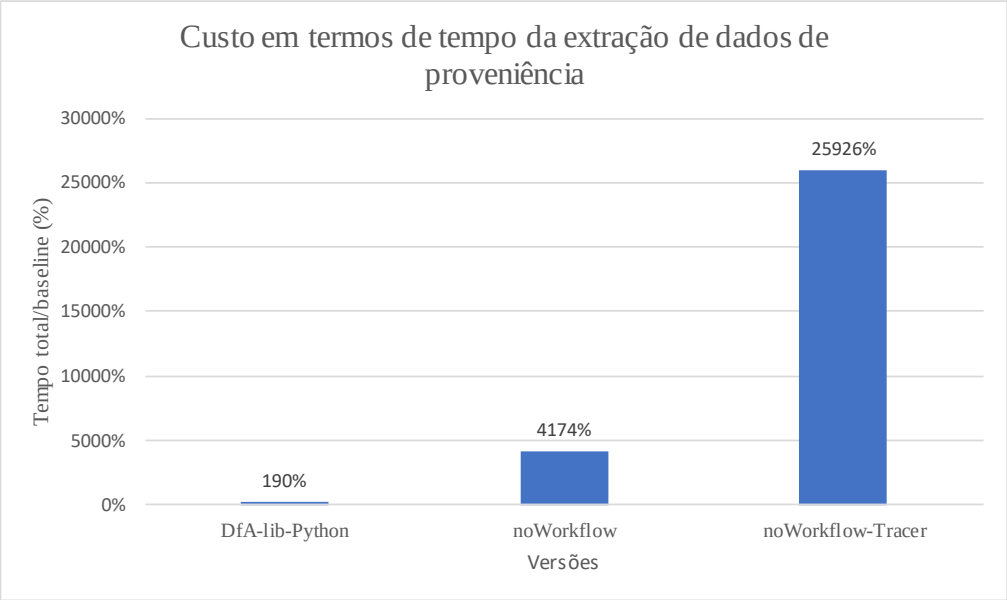

Figura 6.1 – Custo em termos de tempo da extração de dados de proveniência da aplicação de CSE de meteorologia ao utilizar a DfA-lib-Python e o noWorkflow nas configurações padrão e *Tracer*.

O custo em termos de tempo da extração de dados de proveniência e de dados científicos contempla os tempos de captura das proveniências prospectiva e retrospectiva. Portanto, torna-se desejável saber qual dessas etapas onera mais o tempo total de execução de cada versão da aplicação CSE. Na Tabela 6.2 são mostrados os tempos médios da captura de dados de proveniência prospectiva e retrospectiva separadamente para as versões da aplicação CSE. Na Figura 6.2 é mostrada uma comparação relativa entre o tempo de captura para cada tipo de proveniência e o tempo total utilizado para extrair esses dados nas versões da aplicação CSE.

Tabela 6.2 – Tempos médios da captura de dados de proveniência prospectiva e retrospectiva para as versões da aplicação CSE de meteorologia.

Versões	Proveniência Prospectiva	Proveniência Retrospectiva
DfA-lib-Python	0,23	2,64
noWorkflow	108,39	22,31
noWorkflow-tracer	101,95	726,56

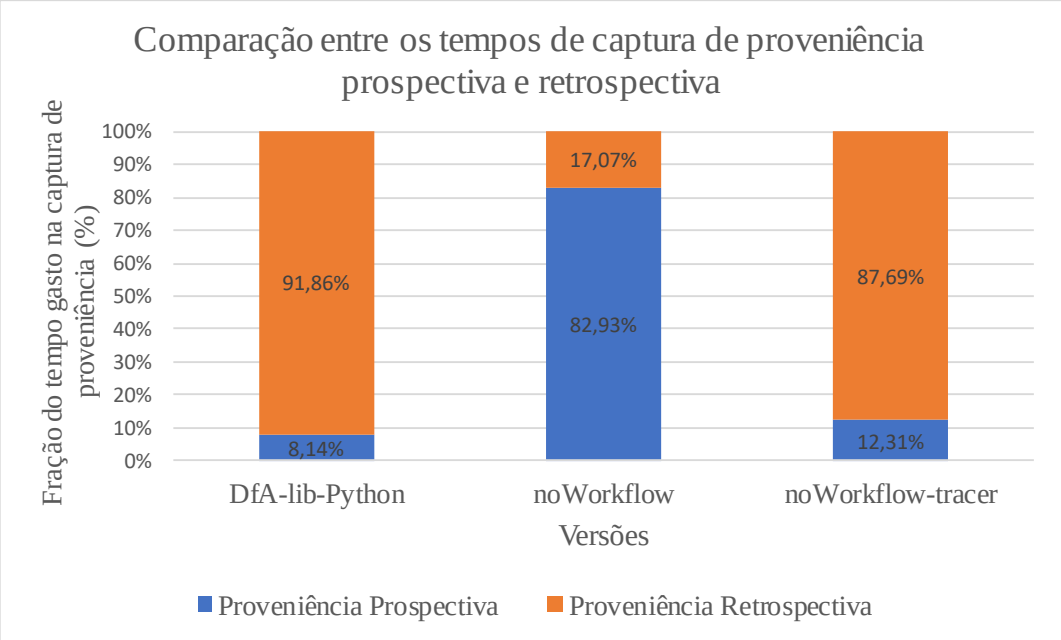

Figura 6.2 - Comparação entre os tempos de captura de proveniência prospectiva e retrospectiva para as versões da aplicação CSE de meteorologia.

Observando a Tabela 6.2 e a Figura 6.2 pode-se notar que a captura de dados de proveniência prospectiva onera mais o tempo de execução para a versão que utiliza o noWorkflow em seu modo de execução padrão enquanto para a configuração *Tracer* do noWorkflow e para a DfA-lib-Python temos a captura de dados de proveniência retrospectiva representando a maior sobrecarga. Na Figura 6.3 é apresentada uma comparação entre os custos de armazenamento em disco das versões da aplicação CSE de meteorologia.

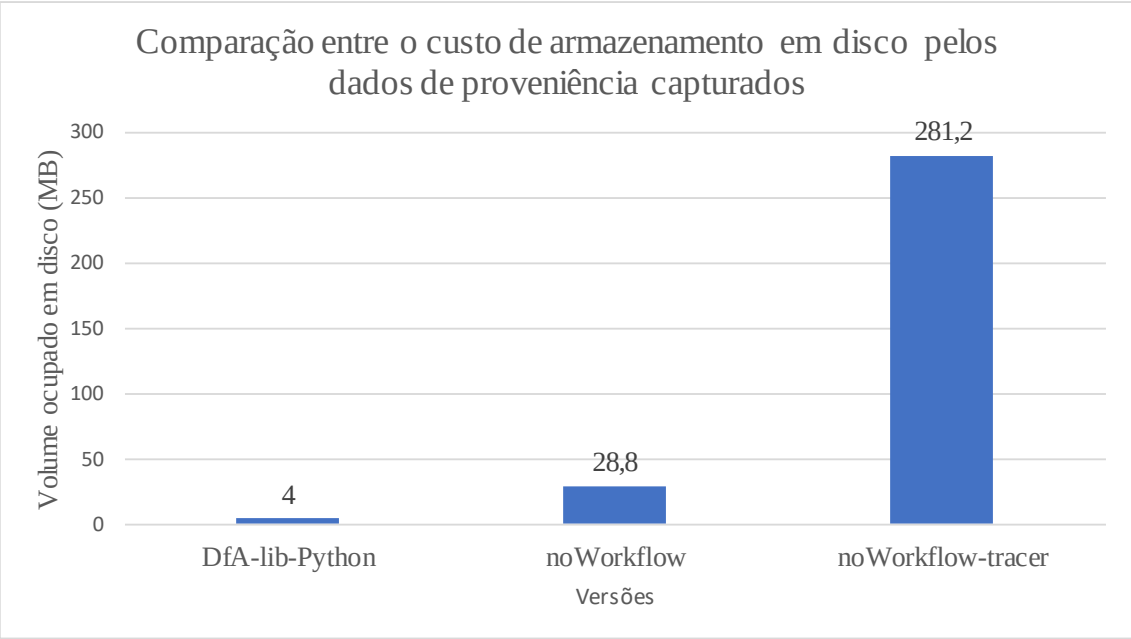

Figura 6.3 – Comparação entre o custo de armazenamento em disco (MB) pelos dados de proveniência capturados por cada uma das versões do experimento da aplicação CSE de meteorologia.

6.3. Aplicação de CSE de multifísica

Equações diferenciais parciais (EDP) são equações diferenciais com funções multivariadas desconhecidas e suas derivadas parciais. Diversos sistemas e fenômenos físicos podem ser descritos por essa classe de equações. Esses sistemas podem ser resolvidos analiticamente ou através de métodos numéricos. Esses métodos costumam ser modelados e executados computacionalmente.

Existem diversas ferramentas computacionais que fornecem suporte em modelagem e resolução de equações diferenciais parciais através de métodos numéricos. Dentre essas ferramentas está o projeto FEniCS, que consiste em uma plataforma de código aberto para solução computacional de equações diferenciais parciais [23]. Essa plataforma fornece interfaces de alto nível nas linguagens de programação *C++* e Python com as quais usuários do domínio científico podem modelar e resolver sistemas de equações diferenciais parciais.

A simulação computacional realizada consiste na execução de uma aplicação de CSE da área de multifísica que representa interação 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 é comumente utilizada para descrever um processo binário de separação de fluidos e pode ser resolvida utilizando métodos numéricos como o método dos elementos finitos. Essa aplicação de CSE utiliza a plataforma FEniCS [23] para modelar esse sistema físico utilizando a equação de Cahn-Hilliard e usa o métodos de elementos finitos para resolvê-la. Inicialmente a versão noWorkflow foi executada utilizando as configurações padrão e *Tracer*. Porém, devido à incompatibilidade entre dependências do noWorkflow nessa configuração e do FEniCS, apenas a configuração padrão foi executada sem erros. Na Figura 6.4 está apresentado o fluxo de dados dessa aplicação CSE. A Tabela 6.3 mostra os tempos médios dessas execuções para cada uma das versões desse experimento.

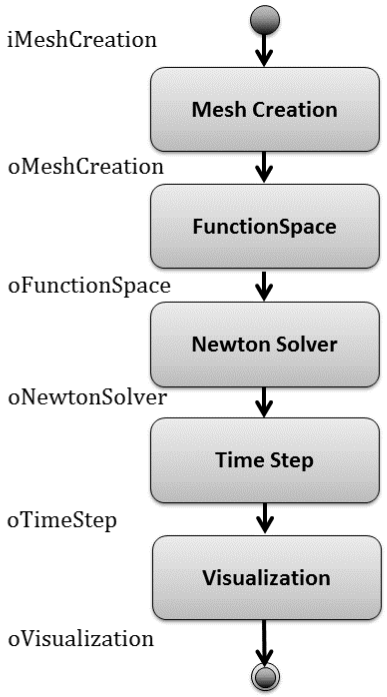

Figura 6.4 - Fluxo de dados da aplicação de CSE de multifísica.

Tabela 6.3 – Tempos médios de execução para cada uma das versões da aplicação de CSE da área de multifísica.

Versões	Tempo médio de execução (s)
Baseline	21,16
DfA-lib-Python	30,61
noWorkflow	293,97

A partir desses tempos foi calculado o custo em termos de tempo da extração de dados de proveniência pelas ferramentas noWorkflow e a DfA-lib-Python. Esse custo é calculado como sendo a razão entre o tempo de execução da solução e o tempo sem a extração de dados de proveniência e de dados científicos(*Baseline*). Os custos calculados estão apresentados na Figura 6.5.

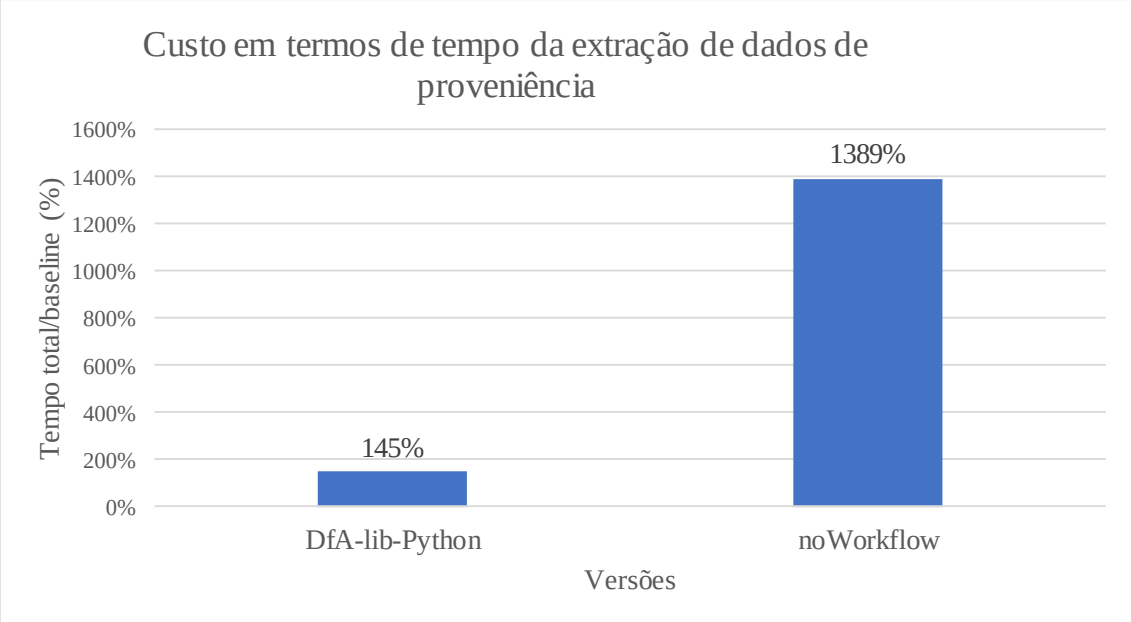

Figura 6.5 – Custo em termos de tempo da extração de dados de proveniência da aplicação de CSE de multifísica ao utilizar a DfA-lib-Python e o noWorkflow com a configuração padrão.

multifísicaAlém disso, torna-se desejável saber qual dessas etapas oneram mais o tempo total de execução de cada versão do experimento. Na Tabela 6.4 são mostrados os tempos médios da captura de dados de proveniência prospectiva e retrospectiva para as versões do experimento. Na Figura 6.6 são mostradas comparações entre o tempo de captura para cada tipo de dados de proveniência e o tempo total utilizado para extrair esses dados nas versões DfA-lib-Python e noWorkflow.

Tabela 6.4 – Tempos médios da captura de dados de proveniência prospectiva e retrospectiva para as versões da aplicação CSE de multifísica.

Versões	Proveniência Prospectiva (s)	Proveniência Retrospectiva (s)
DfA-lib-Python	0,23	9,22
noWorkflow	126,71	146,09

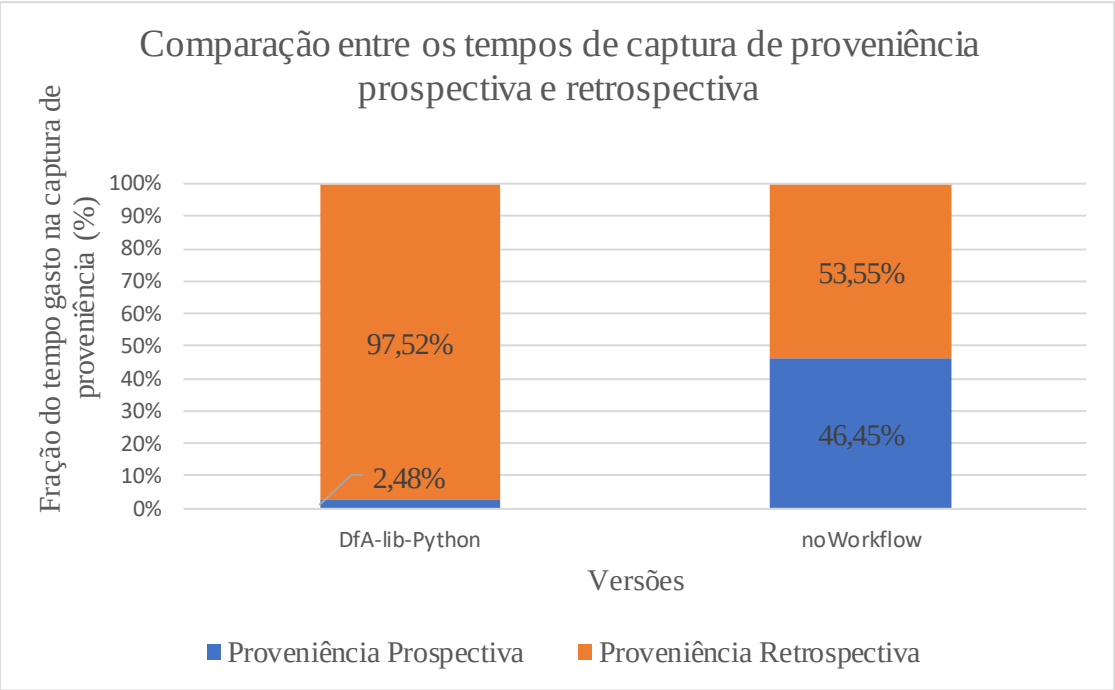

Figura 6.6 – Comparação entre os tempos de captura dos dados de proveniência para as versões da aplicação CSE de multifísica.

Observando a Tabela 6.4 e a Figura 6.6 pode-se notar que, nesse experimento, a captura de dados de proveniência retrospectiva onera mais o tempo total de execução para ambas as versões do experimento. A Tabela 6.5 mostra uma comparação entre os custos de armazenamento em disco pelos dados de proveniência extraídos pelas diferentes versões dessa aplicação CSE.

Tabela 6.5 - Comparação entre o custo de armazenamento em disco (MB) pelos dados de proveniência capturados por cada uma das versões da aplicação CSE de multifísica.

Versões	Volume ocupado (MB)
DfA-lib-Python	1
noWorkflow	123,6

6.4. Análise dos resultados obtidos

Nessa seção são apresentadas análises a respeito dos resultados obtidos pela execução dos dois experimentos descritos nas seções anteriores. A primeira análise diz respeito aos tempos de execução totais obtidos pela execução de ambos experimentos para as versões com DfA-lib-Python e noWorkflow. Conforme apresentado na seção de trabalho relacionados, o noWorkflow não permite especificar quais dados de proveniência serão

extraídos, possibilitando que apenas dois níveis de granularidade para o fluxo de execução possam ser escolhidos. Além disso, essa solução realiza a cópia de arquivos a cada modificação realizada para a base de conteúdo. Essas propriedades davam indícios de que as versões executadas utilizando essa ferramenta deveriam ter um custo gerado pela extração de dados de proveniência maior que o da versão instrumentada com o DfA-lib-Python. Para o experimento da aplicação CSE de meteorologia devemos analisar a diferença de custo entre a versão do noWorkflow padrão e a que utiliza o *Tracer*. Além de variáveis globais, ativações de funções, parâmetros de funções e valores de retorno, ao utilizar o modo *Tracer* o noWorkflow também captura dados referentes à atribuição de variáveis, definição de loops e dependências entre variáveis. Esse processo é mais custoso, pois demanda que a solução armazene um estado global da execução da simulação e verifique as mudanças geradas ao estado pela execução de cada comando, o que justifica que demore mais tempo que a configuração padrão.

A segunda análise diz respeito a proporção entre os tempos de captura de dados de proveniência prospectiva e retrospectiva para cada uma das versões dos experimentos. Os dados de proveniência prospectiva especificam o fluxo de execução da simulação computacional por meio da representação de suas transformações de dados e de suas dependências de dados [9]. Portanto, sua captura ocorre antes do início da execução da simulação computacional não sendo impactada pelo volume de dados gerado, consumido e propagado ao longo da execução. Destarte, é esperado que o custo em termos de tempo para a captura de proveniência prospectiva entre diferentes simulações computacionais esteja na mesma ordem de grandeza. Todavia, a proveniência retrospectiva é o produto da execução da simulação computacional contendo informações sobre os dados gerados, consumidos e propagados ao longo da execução [9]. Ademais, o custo em termos de tempo gerado pela captura de dados de proveniência retrospectiva cresce com o aumento da complexidade do modelo computacional e com o crescimento do volume de dados consumidos e produzidos pela simulação computacional. A proporção obtida para as versões que utilizam a DfA-lib-Python em ambos os experimentos era esperada, pois a proveniência prospectiva capturada representa apenas a especificação do fluxo de dados da simulação computacional enquanto a retrospectiva contém os dados gerados ao longo da execução. As proporções observadas para as versões que utilizam o noWorkflow em ambos experimentos são avaliadas separadamente. No experimento da aplicação de CSE de meteorologia verificamos que os tempos de captura de proveniência prospectiva em

ambas as configurações do noWorkflow foram bastante similares. Isso era esperado, pois a execução utilizando a configuração de *Tracer* altera apenas a granularidade do fluxo de dados extraído ao longo da execução. Todavia, para a configuração padrão o tempo de captura de dados de proveniência prospectiva foi muito superior ao da retrospectiva. Isso pode ter ocorrido devido ao tamanho do conjunto de dados de entrada utilizado. Contudo, a execução utilizando a configuração *Tracer* demorou cerca de seis vezes mais para capturar dados de proveniência retrospectiva que a prospectiva. Isso provavelmente ocorreu pelos motivos listados no parágrafo anterior que impactaram no tempo total dessa execução. Já no experimento da aplicação de CSE de multifísica o tempo de captura de dados de proveniência retrospectiva pelo noWorkflow foi superior com relação ao da prospectiva. Isso provavelmente ocorreu devido à complexidade do modelo computacional utilizado que gera uma maior quantidade de chamada de funções.

A terceira análise é feita sobre o volume ocupado em disco pelas bases que contêm os dados de proveniência capturados para cada versão dos experimentos. Era esperado que o volume ocupado pelas bases geradas pelas versões do noWorkflow (conteúdo e relacional) para ambos experimentos fosse significativamente maior que a DfDB gerada pelas versões do DfA-lib-Python porque devido a instrumentação temos uma captura mais seletiva dos dados gerados ao longo da execução. Ademais, é natural esperar que o volume ocupado pela versão do noWorkflow executada com a configuração de *Tracer* seja maior, uma vez que, essa configuração possui uma granularidade do fluxo de dados mais fina.

7. Conclusão

Atualmente, experimentos científicos são comumente baseados em simulações computacionais para validar seus modelos matemáticos e físicos. Em função do aumento da complexidade desses modelos computacionais, surgiu a necessidade pelo uso de ferramentas que favoreçam a criação e a execução desses modelos computacionais. Nesse cenário, aplicações de Ciência Computacional e Engenharia (CSE) surgem com o objetivo de fornecer ferramentas que auxiliem no processo de modelagem de simulações computacionais de diversos domínios científicos. Pelo fato dessas simulações demandarem muito tempo de execução, os usuários do domínio científico necessitam realizar as suas análises durante a execução, a fim de anteciparem a investigação de determinados comportamentos científicos e, consequentemente, serem capazes de ajustar determinados parâmetros de simulação ou mesmo de interromper uma determinada execução. Para isso, destaca-se a importância de permitir tanto a captura como a análise do fluxo de dados, por exemplo, por meio de dados de proveniência e dados científicos, ao longo da execução das simulações computacionais.

Nesta monografia foi proposta e implementada a DfA-lib-Python, uma biblioteca em Python que permite a extração de dados de proveniência e de dados científicos usando a DfAnalyzer. A linguagem de programação Python foi escolhida devido a sua crescente adesão pela academia, reforçado pelo crescente número de ferramentas que fornecem APIs de alto nível com alto desempenho para serem integradas com bibliotecas consagradas desenvolvidas em linguagens de programação de mais baixo nível como C e FORTRAN. Dentre essas ferramentas podemos citar o projeto FEniCS [23] utilizado na aplicação de CSE de multifísica.

A biblioteca implementada nesta monografia foi avaliada comparativamente com o noWorkflow [22] quanto ao custo em termos de tempo de extração de dados de proveniência e o custo de armazenamento em disco através da instrumentação e execução de aplicações de CSE nos domínios de meteorologia e multifísica. Apesar do custo em termos de tempo de extração de dados de proveniência para as versões das aplicações de CSE que utilizaram a DfA-lib-Python terem sido consideráveis são comparativamente inferiores aos obtidos para as versões que utilizaram o noWorkflow. Além disso, a integração com a DfAnalyzer permite a realização de análises sobre os dados de proveniência extraídos em tempo de execução. O custo de armazenamento em disco pelos

dados de proveniência capturados observado para as versões das aplicações de CSE que utilizaram a DfA-lib-Python foi consideravelmente menor que o observado para as versões do noWorkflow. Portanto, os resultados apresentados para os experimentos realizados apontam que a solução desenvolvida é uma opção viável a ser utilizada para realizar a extração de dados de proveniência e de dados científicos de aplicações científicas escritas em *Python*. Além disso, a ferramenta desenvolvida possui a vantagem de poder ser utilizados em aplicações paralelas o que a torna possível utilizá-la em ambientes de PAD.

Referências Bibliográficas

[1] V. Silva, D. de Oliveira, P. Valduriez, e M. Mattoso, “Analyzing related raw data files through dataflows”, *CCPE*, vol. 28, nº 8, p. 2528–2545, 2016.

[2] E. Ogasawara, J. Dias, D. Oliveira, F. Porto, P. Valduriez, e M. Mattoso, “An Algebraic Approach for Data-Centric Scientific Workflows”, *PVLDB*, vol. 4, nº 12, p. 1328–1339, 2011.

[3] V. Silva, D. De Oliveira, P. Valduriez, e M. Mattoso, “DfAnalyzer: Runtime Dataflow Analysis of Scientific Applications using Provenance”, in *International Conference on Very Large Data Bases*, Rio de Janeiro, Brazil, 2018.

[4] L. Murta, V. Braganholo, F. Chirigati, D. Koop, e J. Freire, “noWorkflow: Capturing and Analyzing Provenance of Scripts”, in *International Workshop on Provenance Annotation (IPAW)*, 2014, p. 1–12.

[5] U. Rüde *et al.*, “Research and Education in Computational Science and Engineering”, *CoRR*, vol. abs/1610.02608, 2016.

[6] V. Silva *et al.*, “Raw data queries during data-intensive parallel workflow execution”, *FGCS*, vol. 75, p. 402–422, 2017.

[7] F. Costa *et al.*, “Capturing and Querying Workflow Runtime Provenance with PROV: A Practical Approach”, in *EDBT/ICDT Workshops*, New York, NY, USA, 2013, p. 282–289.

[8] M. Mattoso *et al.*, “Dynamic steering of HPC scientific workflows: A survey”, *FGCS*, vol. 46, p. 100–113, maio 2015.

[9] C. Lim, S. Lu, A. Chebotko, e F. Fotouhi, “Prospective and Retrospective Provenance Collection in Scientific Workflow Environments”, in *2010 IEEE International Conference on Services Computing*, 2010, p. 449–456.

[10] T. McPhillips *et al.*, “YesWorkflow: A User-Oriented, Language-Independent Tool for Recovering Workflow Information from Scripts”, *International Journal of Digital Curation*, vol. 10, nº 1, fev. 2015.

[11] “Scaling SPADE to ‘Big Provenance’”, in *8th USENIX Workshop on the Theory and Practice of Provenance (TaPP 16)*, Washington, D.C., 2016.

[12] J. F. Pimentel, L. Murta, V. Braganholo, e J. Freire, “noWorkflow: a tool for collecting, analyzing, and managing provenance from python scripts”, *Proceedings of the VLDB Endowment*, vol. 10, nº 12, p. 1841–1844, ago. 2017.

[13] “The column-store pioneer | MonetDB”. [Online]. Disponível em: https://www.monetdb.org/Home. [Acessado: 09-jun-2018].

[14] K. Wu *et al.*, “FastBit: interactively searching massive data”, *Journal of Physics: Conference Series*, vol. 180, p. 012053, jul. 2009.

[15] “Documentation of dfa-lib-python — Main Page documentation”. [Online]. Disponível em: https://dfa-lib-python-docs.herokuapp.com/index.html. [Acessado: 09-jun-2018].

[16] “pip”, *PyPI*. [Online]. Disponível em: https://pypi.org/project/pip/. [Acessado: 09-jun-2018].

[17] “Virtualenv — virtualenv 16.0.0 documentation”. [Online]. Disponível em: https://virtualenv.pypa.io/en/stable/. [Acessado: 09-jun-2018].

[18] “Git”. [Online]. Disponível em: https://git-scm.com/. [Acessado: 09-jun-2018].

[19] “Visual Studio Code - Code Editing. Redefined”. [Online]. Disponível em: http://code.visualstudio.com/. [Acessado: 09-jun-2018].

[20] “Requests: HTTP for Humans — Requests 2.18.4 documentation”. [Online]. Disponível em: http://docs.python-requests.org/en/master/. [Acessado: 09-jun-2018].
[21] “pytest: helps you write better programs — pytest documentation”. [Online]. Disponível em: https://docs.pytest.org/en/latest/. [Acessado: 09-jun-2018].
[22] *noworkflow: Supporting infrastructure to run scientific experiments without a scientific workflow management system*. Grupo de Evolução e Manutenção de Software (GEMS), 2018.
[23] “FEniCS Project”. [Online]. Disponível em: https://fenicsproject.org/. [Acessado: 16-jun-2018].