CAPTURA DE PROVENIÊNCIA ASSÍNCRONA EM SIMULAÇÕES COMPUTACIONAIS

Luciano Silva Leite

Dissertação de Mestrado 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 Mestre em Engenharia de Sistemas e Computação.

Orientadora: Marta Lima de Queirós Mattoso

Rio de Janeiro
Setembro de 2018

CAPTURA DE PROVENIÊNCIA ASSÍNCRONA EM SIMULAÇÕES COMPUTACIONAIS

Luciano Silva Leite

DISSERTAÇÃO 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 MESTRE 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. Paulo de Figueiredo Pires, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
SETEMBRO DE 2018

Leite, Luciano Silva

Captura de Proveniência Assíncrona em Simulações Computacionais / Luciano Silva Leite. – Rio de Janeiro: UFRJ/COPPE, 2018.

XI, 64 p.: il.; 29,7 cm.

Orientador: Marta Lima de Queirós Mattoso

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2018.

Referências Bibliográficas: p. 62-64.

1. Proveniência de dados. 2. Simulação computacional. 3. Captura de dados I. Mattoso, Marta Lima de Queirós. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Aos meus pais.

Agradecimentos

Aos meus pais Mirian e José pelo esforço depositados na minha educação e pelo apoio que sempre deram.

À professora Marta Mattoso e ao Vítor Silva pela generosidade, pelas ideias e críticas na construção deste trabalho.

Aos professores Álvaro Coutinho, José Camata pelas contribuições.

Aos professores Daniel Oliveira e Paulo Pires pela participação na banca.

À Mara Prata e a Solange Santos por toda a ajuda com as questões administrativas.

Ao Victor pela paciência.

Ao Raphael e a todos os amigos que ajudaram, torceram e contribuíram para o sucesso deste trabalho.

Agradeço.

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

CAPTURA DE PROVENIÊNCIA ASSÍNCRONA EM SIMULAÇÕES COMPUTACIONAIS

Luciano Silva Leite

Setembro/2018

Orientadora: Marta Lima de Queirós Mattoso

Programa: Engenharia de Sistemas e Computação

Simulações computacionais em larga escala são experimentos computacionais cada vez com mais processamento de dados. Usuários e desenvolvedores deste tipo de simulação geralmente realizam análises sobre dados científicos durante a execução da simulação. Esta não é uma tarefa trivial, já que as simulações em larga escala costumam ser executadas em ambientes de processamento de alto desempenho e produzir grande volume de dados. Soluções existentes, como o DfAnalyzer, fazem uso de dados de proveniência para auxiliar esta análise com muito sucesso. No entanto, esses sistemas possuem abordagens síncronas de coleta de dados, o que dificulta a sua instalação e, principalmente, interfere no desempenho da simulação computacional. Esta dissertação propõe uma abordagem assíncrona de coleta de dados de proveniência com o objetivo de disponibilizar dados científicos para consulta durante a execução da simulação sem muito impacto no seu tempo de execução. Para validar as estratégias propostas, foi desenvolvida a ferramenta Asynchronous Dataflow Analyzer. A implementação realizada estende o DfAnalyzer para adotar o assincronismo proposto e simplifica a configuração do sistema por meio da flexibilização da gerência da proveniência prospectiva. Os resultados experimentais, com uma simulação de processos de sedimentação de solos, mostram que a ferramenta é capaz de atender as necessidades de análises de dados dos usuários de simulações computacionais com sobrecargas inferiores a ferramentas existentes.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

ASYNCHRONOUS PROVENANCE GATHERING IN COMPUTER SIMULATIONS

Luciano Silva Leite

September/2018

Advisor: Marta Lima de Queirós Mattoso

Department: Systems and Computer Engineering

Large-scale computational simulations are computational experiments increasingly more processing intensive. Users and developers of this type of simulation generally analyze data during simulation execution. This is not a trivial task since large-scale simulations are often performed in high-performance processing environments and can produce a large volume of data. Existing solutions, as DfAnalyzer, use provenance data to assist analysis with success. However, these systems use synchronous approaches to gather data that makes difficult to set up it and, mainly, interferes in the performance of the computational simulation. This dissertation proposes an approach to asynchronously collect provenance data making it available for analysis during the execution of the simulation with the least possible delay. In order to evaluate the proposed strategies, a tool, Asynchronous Dataflow Analyzer. This implementation extends DfAnalyzer to use the proposed asynchronous approach and to simplify the configuration process by making the prospective provenance definition process more flexible. The experimental results, with a soils sedimentation simulation, show that the tool is able to meet the needs of users of large-scale computational simulations with lower overloads than similar tools.

Índice

Índice de Figuras

Índice de Tabelas

Capítulo 1 - Introdução

As aplicações de Ciência Computacional e Engenharia (CSE, do inglês *Computational Science and Engineering*) são, em geral, baseadas em modelos e simulações computacionais (RÜDE *et al.*, 2016). Na condução de pesquisas científicas, é comum que o refinamento dos modelos computacionais e o uso de simulações mais complexas sejam vetores para produção de resultados mais precisos e de novas conclusões científicas.

Experimentos computacionais de larga escala levam muito tempo em execução e podem gerar grandes volumes de dados (MA *et al.*, 2012). Conforme as simulações se tornam mais complexas, o uso de recursos computacionais de maneira eficiente e a capacidade de realizar análises de grandes quantidades de dados rapidamente tem grande impacto na produção de conhecimento científico. Essas necessidades permeiam a rotina da pesquisa científica em diversos ramos incluindo astrofísica, física de altas energias, ciências dos materiais, dinâmica de fluidos, modelagem climática e ciências biológicas (MA *et al.*, 2012).

Simulações computacionais consistem em um encadeamento de programas científicos que podem ter de recursos de paralelismo. Neste encadeamento, os dados produzidos por um programa são usados como dado de entrada para outro programa científico caracterizando uma relação de dependência de dados entre os programas.

Neste contexto, diversas tecnologias surgiram para facilitar adoção de técnicas de paralelismo no desenvolvimento das simulações computacionais para que usem recursos de processamento de alto desempenho (PAD) de maneira eficiente. Dentre eles, o uso de paradigmas de paralelismo como o MPI (*Message Passing Interface*) e o uso de bibliotecas de componentes científicos se destacam como soluções muito populares e presentes nas simulações computacionais.

Simulações computacionais contam com um encadeamento complexo de programas, recursos de paralelismo e componentes externos. Com isso, o trabalho de compor o experimento, que inclui a escolha de parâmetros para diversos programas e componentes, pode ser custoso e propenso a erros. Por isso, é importante que o usuário consiga monitorar resultados parciais durante a execução da simulação para que possa

identificar problemas rapidamente e intervir no fluxo de execução quando necessário. A análise exploratória de resultados e a busca por erros também podem ser tarefas árduas devido ao grande volume de dados e a dificuldade de relacionar os resultados parciais em meio ao complexo encadeamento de programas.

É comum os usuários produzirem soluções *ad hoc* para realizar análise exploratória dos resultados de seus experimentos o que acarreta dificuldades como o alto esforço de desenvolvimento e problemas de desempenho computacional. Soluções de monitoramento e análise de dados precisam ser ajustadas com frequência devido à natureza iterativa do desenvolvimento de aplicações científicas em que alterações na simulação implicam em ajustes na solução de análise.

Ao longo da realização de um experimento frequentemente precisa-se validar as suas hipóteses científicas e analisar comportamentos específicos, a partir de resultados parciais ou finais produzidos pelas suas simulações. Os principais dados envolvidos nesta análise de resultados correspondem aos dados de domínio, ou seja, dados produzidos na execução do experimento cujas propriedades sejam relevantes para a pesquisa científica que está sendo realizada. Dados de domínio podem ser armazenados em arquivos com formatos heterogêneos que variam de acordo com o domínio da aplicação. Como exemplos de formatos de arquivos pode-se citar o FITS (GREISEN & CALABRETTA, 2002) para o domínio da astronomia, assim como o NetCDF (UNIDATA, 2016) e o HDF5 (THE HDF GROUP, 1997) para Dinâmica de Fluidos Computacionais (CFD).

Como cada formato de arquivos de dados científicos tem suas particularidades e definições próprias, os procedimentos para realizar a análise de dados varia bastante segundo o domínio científico e representa uma tarefa não trivial e propensa a erros em função da heterogeneidade dos formatos adotados. Cada formato de arquivo tem um padrão difundido na comunidade que estuda o domínio e diversas ferramentas e bibliotecas para auxiliar a análise e a manipulação. Para realizar a análise dos resultados, o usuário precisa escolher as ferramentas ou as bibliotecas, assim como adaptar os seus programas de simulação para utilizá-las.

Além dos dados em formatos científicos, a execução de programas de simulação também pode produzir dados que podem ser armazenados e consultados através de um Sistema de Gerência de Banco de Dados (SGBD). Os programas da simulação produzem

também arquivos de *log* de execução que podem ser úteis para a extração de informações relevantes na análise da simulação. Esses arquivos de *log* podem informar, sobre erros em alguma das etapas da simulação ou indicar algum comportamento inesperado. Os arquivos de *log* contêm ainda metadados que podem ser extraídos desses arquivos a fim de complementar a análise dos arquivos de dados científicos. Como as simulações tendem a serem complexas e gerar grande volume de dados, a análise e a depuração dos arquivos de *log* também compreendem uma tarefa custosa e propensa a erros.

1.1 - Análise exploratória de dados científicos

Realizar a análise dos resultados de experimentos científicos sem ferramentas apropriadas se torna uma árdua tarefa de inspeção de um grande número de arquivos de dados científicos, dados estruturados e arquivos de *log* de execução. Devido ao grande volume de dados, é comum haver dificuldades em atividades operacionais, como identificar arquivos, produzir ou configurar ferramentas analíticas que atendam às necessidades de cada simulação e reconstruir o fluxo de dados dos resultados, isto é, que conjuntos de dados de entrada e resultados parciais produziram o resultado final. Esse último tipo de análise envolve monitorar os relacionamentos entre elementos de dados manipulados por diferentes programas de simulação, mantendo-se um rastro dos seus consumos e produções.

A análise de arquivos de dados científicos integrada com a análise de metadados da execução também são necessárias ao longo da execução da simulação. A análise destes dados permite que se monitore a execução da simulação e se avalie a ocorrência de possíveis erros ou inconsistências. Esta possibilidade de avaliação de resultados parciais permite que simulações com falhas sejam interrompidas ou ajustadas. A indisponibilidade da análise destes recursos implica em avaliar os resultados apenas após a execução. O que, em casos de erros, exige uma nova execução da simulação com ajustes. Considerando que simulações computacionais levam muito tempo para serem concluídas mesmo em ambientes de PAD, a análise em tempo de execução ajuda a produzir conclusões mais rapidamente e a usar os recursos computacionais de uma forma mais eficiente ao agilizar a identificação de erros e de resultados indesejáveis.

1.2 - Definição do problema

Existem diversas ferramentas para atender às necessidades da análise exploratória de dados científicos em experimentos computacionais em larga escala. Para a análise de arquivos de dados científicos, por exemplo, temos NoDB (ALAGIANNIS *et al.*, 2015) e RAW (KARPATHIOTAKIS *et al.*, 2014) que trabalham diretamente com dados científicos armazenados em formatos de arquivo específicos. Contudo, as suas abordagens não contemplam a análise do fluxo de dados (*dataflow*), dificultando a realização de análises que necessitem dados de proveniência da simulação. A abstração do fluxo de dados é relevante no contexto das simulações por que permite traçar a proveniência de dados e arquivos, ou seja, todo o histórico de um resultado em relação à quais dados de entrada o produziram. Conhecer detalhadamente o fluxo de dados permite entender os dados produzidos por cada etapa da simulação e relacioná-los com os resultados finais além de garantir a reprodutibilidade do experimento. A análise do fluxo de dados de um experimento pode ser feita de diversas maneiras: desde a visualização do fluxo de dados como um diagrama ou a execução de consultas construídas em SQL.

Experimentos computacionais incluem etapas de análise que buscam por atributos de interesse e arquivos que originaram algum resultado científico específico. Para avaliar os resultados da simulação é importante o acesso aos resultados parciais em cada etapa que faz parte do experimento. Assim, é importante capturar e armazenar dados de proveniência de cada etapa da simulação. Tais dados gerados ao longo da simulação permitem representar o fluxo de dados ou o conjunto de dependências entre os dados de domínio consumidos e produzidos em cada etapa da simulação. Acessar estes dados é essencial para as consultas sobre a reconstrução dos dados e processos que fazem parte do histórico de um resultado.

Nesse sentido, diversos Sistemas de Gerência de Workflows Científicos (SGWfC), por exemplo, Pegasus (DEELMAN *et al.*, 2007) e SciCumulus (OLIVEIRA *et al.*, 2010) oferecem apoio à proveniência dos arquivos manipulados dentre as suas funcionalidades. Os SGWfC geralmente armazenam o fluxo de dados e metadados associados às etapas de execução do experimento em uma base de dados. A partir desta base de dados de proveniência, o usuário pode realizar consultas em dados estruturados. Por sua natureza generalista, os SGWfC, em geral, não proveem recursos para a análise

exploratória de dados científicos, tornando-os pouco úteis para as consultas que precisam considerar dados científicos armazenados em arquivos.

Somando-se a isso, outra limitação dos SGWfCs é o custo de configuração já que é necessário que o usuário especifique detalhadamente as etapas do experimento. O uso de SGWfC geralmente exige que a simulação seja modelada como um *workflow*, especificando um conjunto de atividades e as suas dependências de dados. Modelar uma simulação pode ser uma tarefa árdua caso a mesma seja complexa ou o usuário não tenha familiaridade com a linguagem de especificação usada pelo SGWfC. O esforço empregado em tarefas de configuração das ferramentas de proveniência são um fator relevante para o desenvolvimento de uma simulação já que é comum que o fluxo de dados do experimento seja ajustado de acordo com as hipóteses já avaliadas, o que traz a necessidade de reconfigurar a ferramenta de proveniência. Assim, o usuário possivelmente tem mais trabalho remodelando o *workflow* com frequência.

Tendo em vista a rigidez dos SGWfC, diversas propostas surgiram para oferecer análise de dados por meio de consultas a bases de dados. Ferramentas como o NoDB (ALAGIANNIS *et al.*, 2015) oferecem recursos para consulta diretamente em arquivos de dados sem a necessidade de criação de uma base de dados. Ou ainda o DataTracker (EICHINSKI & ROE, 2016) e o NoWorkflow (PIMENTEL *et al.*, 2017) que são capazes de capturar proveniência em tempo de execução e construir uma base de dados de proveniência sem que o usuário precise especificar ou configurar quais dados precisam ser coletados. Estas ferramentas são capazes de atender as características dinâmicas do desenvolvimento de experimentos científicos e buscam oferecer as vantagens de uma base de dados estruturados, mas com reduzido esforço de configuração e de especificação do fluxo de dados.

O NoDB é uma ferramenta que suporta a execução de consulta SQL em dados tabulares fornecidos pelo usuário, em geral no formato de arquivos *csv*. O NoDB usa uma adaptação do PostgreSQL que suporta realização de consultas e indexação de dados no arquivo *csv* sem a necessidade de definição de um esquema. Contudo a ferramenta é voltada apenas para a análise de dados já que não auxilia coleta de dados nem a suporta a integração com arquivos de domínio deixando para o usuário essa árdua tarefa.

Consequentemente, esta ferramenta possibilita realizar apenas análises *post mortem* já que não permite a coleta de dados durante a execução da simulação.

Soluções como DataTracker e NoWorkflow são ferramentas que fornecem recursos de proveniência de dados e de arquivos para scripts desenvolvidos nas linguagens R e Python respectivamente. Para realizar a captura de proveniência ambas atuam sobrescrevendo chamadas básicas de acesso a arquivos ou chamadas de função disponibilizadas pela linguagem de programação. O objetivo de alterar o comportamento padrão da linguagem é coletar, em momentos específicos, informações de produção e consumo de dados. Como os dados são coletados sempre que as funções sobrecarregadas são chamadas não é necessário que o usuário especifique o experimento como um fluxo de dados.

Por outro lado, a coleta indiscriminada de dados de proveniência no cenário de grandes simulações computacionais pode acarretar num grande volume de dados coletados. Num cenário de volume excessivo de dados, a formulação de consultas e a análise de resultados podem ser tarefas custosas já que exigiriam uma filtragem pelos dados de interesse dentro de um conjunto de dados muito maior.

Essas e outras ferramentas possuem recursos para capturar dados que retratem a aplicação, o conjunto de dados e entrada e as condições do ambiente de maneira que permitam sua reprodução, análise e monitoramento. Contudo, estas ferramentas buscam atender outras categorias de aplicação com necessidades que podem ser diferentes das encontradas em simulações computacionais. Por isso, estas ferramentas podem não atender as necessidades das simulações computacionais em larga escala.

A análise de resultados de simulações computacionais requer a integração de dados de proveniência, dados em arquivos de domínio e logs de execução de maneira que seja possível realizar consultas para análise exploratória. Além disso, a rotina de execução deste tipo de experimento exige monitoramento da produção de resultados em ambiente de PAD durante a execução do experimento.

A solução que mais se aproxima desses objetivos é a ferramenta DfAnalyzer (SILVA *et al.*, 2017), que por sua vez é uma instância da arquitetura ARMFUL (SILVA, 2018). DfAnalyzer não possui as limitações das demais soluções, pois é capaz de prover análise de dados de simulações computacionais em tempo de execução, contemplando

dataflows, dados de proveniência e execução não intrusiva em ambientes PAD. No entanto, observou-se em DfAnalyzer a limitação de realizar a captura de dados científicos e de proveniência e seu armazenamento, de forma síncrona, o que pode interferir no desempenho da simulação computacional na presença de um grande volume de dados. Além disso, DfAnalyzer em suas versões iniciais (SILVA *et al.*, 2017), exigia um grande esforço do usuário para especificação da simulação como um fluxo de dados.

Desta forma, observou-se que as soluções existentes, tanto em análise de dados científicos, como em SGWfC, quanto nos geradores de dados de proveniência e mesmo a DfAnalyzer, caracterizam um cenário que evidenciou o problema de sobrecarga no tempo de execução para a coleta e análise de dados durante a execução da simulação computacional, além da necessidade em reduzir o esforço de adoção e configuração para que não seja uma tarefa custosa para o usuário da solução.

1.3 - Abordagem proposta

Esta dissertação propõe estratégias para facilitar análises exploratórias em dados de simulações computacionais em larga escala, durante sua execução, sem apresentar as limitações das soluções existentes. Mais especificamente, a abordagem visa a diminuir o tempo de execução da coleta e análise de dados e o esforço necessário para adoção do recurso de apoio. Como o tempo de execução é uma questão importante para este tipo de aplicação, a solução proposta para coleta e análise de dados deve buscar não impactar o tempo de execução da simulação. Soluções de apoio podem exigir grande esforço de configuração e adaptação da simulação. Neste trabalho, a solução proposta busca ter um esforço de adoção reduzido.

Como a ferramenta DfAnalyzer é a que mais se aproxima dos objetivos pretendidos, esta dissertação propõe manter as vantagens da DfAnalyzer em prover dados de domínio e de proveniência para consulta em tempo de execução, ao mesmo tempo em que diminui a sobrecarga de sua adoção aliada ao baixo esforço de configuração. Para alcançar estes objetivos, foram traçadas novas estratégias de captura de dados a partir de troca de mensagens entre a simulação e a ferramenta de análise exploratória de dados. A coleta de dados proposta é assíncrona em relação à simulação o que permite a coleta de dados da simulação durante sua execução com reduzida sobrecarga no processamento da

simulação. Foram traçadas ainda estratégias para reduzir o esforço de configuração relacionado à especificação das etapas do experimento por meio da flexibilização da especificação do fluxo de dados.

Para validar as estratégias propostas, foi construída a ferramenta *Asynchronous Dataflow Analyzer* (ADfA), que permite a coleta e armazenamento de dados de proveniência de modo assíncrono, caracterizando-se como uma alternativa às estratégias implementadas na DfAnalyzer. Para isso, se baseia na arquitetura ARMFUL (SILVA *et al.*, 2017). A ARMFUL propõe uma estrutura modular para instrumentar a captura, armazenamento e análise de dados de arquivos científicos gerados no contexto de simulações computacionais.

A ADfA estende e mantém todas as vantagens da DfAnalyzer, porém realiza a captura dados de proveniência por meio de troca de mensagens com os programas que compõem a simulação computacional. A estratégia de troca de mensagens tem a vantagem de ser não bloqueante em relação à execução da simulação computacional. Além disso, apresenta baixa sobrecarga do uso de recursos computacionais da simulação computacional. Para simplificar a configuração, a ADfA usa uma base de dados com esquema de dados flexível, cuja estrutura mutável é adaptada para se adequar aos dados recebidos. Isso permite que as etapas de especificação de fluxo de dados e instrumentação da simulação sejam simplificadas em uma única tarefa. A ADfA foi construída em paralelo à evolução de DfAnalyzer e contribuições geradas nesta dissertação foram incorporadas em versões mais recentes da DfAnalyzer (SILVA *et al.*, 2018).

1.4 - Organização da Dissertação

Além desta introdução, esta dissertação está organizada em outros cinco capítulos. O Capítulo 2 - apresenta a fundamentação teórica e os conceitos relevantes para o entendimento das particularidades da ferramenta desenvolvida O *Capítulo 4* - retrata as abordagens existentes para o problema proposto, ressaltando-se as vantagens e as limitações de cada uma. *O Capítulo 4* - descreve as estratégias e as decisões de projeto adotadas para resolver o problema, assim como os detalhes de implementação da ADfA. O Capítulo 5 - discute os experimentos realizados e os resultados obtidos. O *Capítulo 6* - conclui esta dissertação.

Capítulo 2 - Fluxos de dados de Experimentos Computacionais e suas tecnologias

Neste capítulo, são apresentados conceitos e tecnologias fundamentais para o desenvolvimento deste trabalho: a Seção 2.1 disserta sobre experimentos científicos com simulações computacionais em larga escala; a Seção 2.2 apresenta o ciclo de vida do experimento computacional; a Seção 2.3 define o conceito de proveniência de dados no contexto de experimentos computacionais e discute sua relevância no âmbito da exploração de resultados de simulações computacionais; a 2.4 discute recursos computacionais para gerência de proveniência de experimentos. A seção 2.5 discute a estratégia de captura de proveniência por meio de troca de arquivos. Esta estratégia é adotada pela ferramenta DfAnalyzer, conforme SILVA *et al.* (2017).

2.1 - Experimentos Científicos baseados em Simulações Computacionais

Experimentos científicos podem ser definidos como "*ensaios realizados em ambiente controlado com o objetivo de demonstrar a veracidade de um fato, validar uma hipótese ou determinar a eficácia de algo ainda não tentado*" (SOANES & STEVENSON, 2003) ou ainda "*um cenário criado em laboratório, que avalia, sob condições controladas o relacionamento entre fenômenos de interesse*" (JARRARD, 2001).

A partir destas definições, pode-se assumir que um experimento científico pode ser associado a um conjunto de ações e ambiente controlados. Estas ações controladas incluem testes e comparação de resultados para confirmar a validade de uma hipótese. Um experimento só pode ser considerado científico as informações sobre as ações tomadas e o ambiente controlado forem suficientes para reproduzir com precisão os resultados obtidos (MATTOSO *et al.*, 2010).

Conforme a evolução da tecnologia, o uso de recursos computacionais tem se tornado cada vez mais comum no ambiente científico para realização de experimentos científicos nas mais variadas áreas (DEELMAN *et al.*, 2009). Conforme a classificação estabelecida em Travassos e Barros (2003), as simulações computacionais, alvos desta

dissertação, podem ser classificados como experimentos *in silico* visto que o ambiente e os objetos de teste são tratados como modelos computacionais.

Simulações computacionais em larga escala são simulações computacionais que usam modelos ou processos complexos, os quais são computacionalmente custosos. Em geral, simulações em larga escala exigem ambientes de PAD e manipulam grande quantidade de dados ao longo de sua execução. Nas últimas décadas, as simulações computacionais têm se tornado mais complexas e exigindo mais recursos de PAD.

Consequentemente, o volume de dados também tem aumentado fazendo com que as dificuldades das tarefas de análise de resultados, executadas pelo usuário, se tornem mais críticas para a conclusão do experimento. Neste cenário, técnicas e ferramentas que apoiem o desenvolvimento, a reprodutibilidade, a exploração de resultados e compartilhamento de simulações assumem papel importante no desenvolvimento científico.

2.2 - Ciclo de Vida do Experimento Científico

O ciclo de vida de um experimento científico é uma abstração das atividades de configuração, execução do experimento e a análise de resultados. O modelo adotado neste trabalho, ilustrado pela Figura 2.1, compreende três fases: composição, execução e análise (MATTOSO *et al.*, 2010). O uso de uma abstração que divide o desenvolvimento de um experimento científico em fases ajuda a entender as necessidades e dificuldades encontradas em cada tarefa da realização de um experimento.

A fase de composição do experimento consiste em atividades de alto nível. Nela, realiza-se a montagem de um fluxo de ações abstratas interligadas e quais os substratos e resultados que se deseja consumir e produzir em ação. No caso de experimentos computacionais, a etapa de composição do experimento define quais programas serão executados, quais parâmetros e arquivos devem ser fornecidos para sua execução, quais dados e arquivos devem produzidos e como os dados produzidos devem ser usados para a execução de outros programas, ou outras ações do experimento científico.

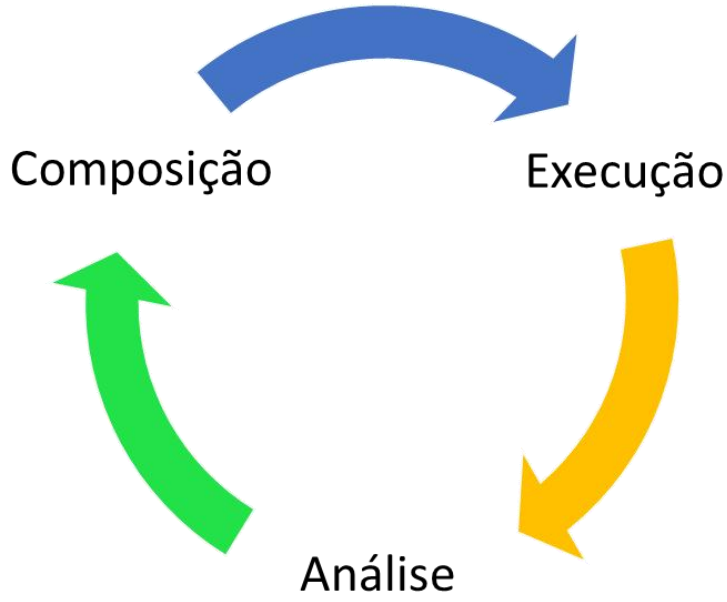

Figura 2.1 - Ciclo de vida do experimento científico

Na fase de execução de um experimento, o fluxo de ações desenvolvido na etapa de composição torna-se um experimento concreto a partir da associação a um ambiente computacional, a definição de valores de parâmetros e especificação de arquivos de entrada. Durante a execução, realiza-se e o monitoramento do comportamento do experimento em ambiente computacional. O monitoramento é realizado para garantir que o experimento está sendo executado corretamente e os resultados têm características esperadas. Para experimentos em larga escala, o monitoramento é de extrema importância, visto que os a execução dos experimentos pode perdurar por meses e que os recursos de PAD são escassos. Assim, se o experimento apresentar alguma inconformidade, o usuário precisa estar apto identificar o problema e reagir rapidamente.

Na fase de análise, dados gerados nas etapas de composição e execução são explorados por meio de consultas e estratégias de visualização. Nesta fase os dados de proveniência coletados a partir da composição e execução do experimento complementam a análise exploratória dos resultados na qual os usuários confrontam suas hipóteses com os resultados observados. De acordo com os resultados observados e as conclusões obtidas, o experimento pode passar por um novo ciclo contemplando

composição, execução e análise. As três fases da realização de um experimento são repetidas conforme a necessidade de refinar os resultados ou reproduzi-los em condições ou ambientes diferentes. Muitas vezes as informações de proveniência são relevantes ainda para auxiliar a comparação entre resultados produzidos em diferentes versões do experimento.

2.3 - Proveniência e Fluxos de dados

Neste trabalho, considerou-se que a proveniência é informação chave para o funcionamento das ferramentas de apoio à análise exploratória de resultados de simulações computacionais. Entende-se por proveniência o histórico (FREIRE *et al.*, 2008) de um dado: valores de parâmetros, arquivos, processos e configurações de ambiente que fizeram parte de sua construção. Podemos definir como proveniência *a origem de um dado em um experimento científico*.

Dados de proveniência podem ser classificados em dois tipos (FREIRE *et al.*, 2008) de acordo com o seu significado. Dá-se o nome de proveniência prospectiva a informação sobre o encadeamento de programas e como dependem dos dados produzidos entre si. Chama-se proveniência retrospectiva qualquer informação de proveniência relacionada aos dados produzidos ou à execução de uma transformação.

De forma geral, podemos assumir que a proveniência prospectiva é uma representação da composição do experimento enquanto a proveniência retrospectiva caracteriza o fluxo de execução do experimento. A partir da análise de resultados associados a proveniência prospectiva e retrospectiva podemos: (i) garantir a reprodutibilidade do experimento, visto que os parâmetros, arquivos e configurações de ambiente podem ser capturados como informações de proveniência; (ii) identificar inconsistências nos resultados e rastrear suas causas através da análise das transformações de dados que geraram o resultado inconsistente; (iii) facilitar as tarefas de interpretação de dados por suportar a execução de consultas complexas relacionando resultados e informações de proveniência.

Fluxo de dados pode ser entendido como todo o encadeamento de programas e os dados associados à sua execução, ou seja, todo o conjunto de proveniência de uma simulação. Uma definição mais precisa, apresentada por SILVA (2018) define um fluxo

de dados (*dataflow*) como uma composição de transformações de dados (*data transformations*) e seus relacionamentos. Uma transformação de dados é a abstração para um procedimento computacional que executa uma tarefa específica consumindo elementos (*data elements*) de um ou mais conjuntos de dados (*datasets*) e produzindo elementos de dados como saída.

SILVA (2018) descreve ainda um fluxo de dados como um grafo direcionado em que os vértices representam as transformações de dados e as arestas representam os conjuntos de dados que relacionam entradas e saídas das transformações. Na prática, uma transformação de dados é um programa ou um trecho de um programa que produz dados de interesse no contexto do domínio da simulação.

2.4 - Recursos para Gerência de Proveniência

Existem diversos recursos para gerência e análise de proveniência em experimentos computacionais. Em geral, essas ferramentas constroem uma base de dados para abrigar os dados de proveniência do experimento. As ferramentas encontradas na literatura adotam diferentes estratégias para construção, população e acesso às suas bases de dados de proveniência. Devido a essas diferenças, as ferramentas podem ser mais ou menos adequadas de acordo com as características do experimento computacional.

No caso das simulações computacionais, deseja-se disponibilizar a proveniência, com rapidez, durante a execução do experimento. Assim, uma característica importante das ferramentas de proveniência é a estratégia de captura de proveniência retrospectiva. De maneira geral, as ferramentas de proveniência podem ser divididas em três grandes grupos em relação ao processamento e disponibilidade da proveniência retrospectiva: *post mortem, in situ e in transit*. Essas categorias foram são usadas no contexto de manipulação e acesso a dados de programas computacionais (MORELAND *et al.*, 2011, BAUER *et al.*, 2016) e foram adotadas por SILVA (2018) para classificação de ferramentas de proveniência.

Na abordagem *post mortem,* os arquivos de resultados do experimento são inspecionados para reconstruir a proveniência dos resultados. Simulações computacionais são experimentos que levam muito tempo para serem executadas e os usuários precisam avaliar resultados parciais ao longo da execução. Essa categoria de ferramenta não atende

aos requisitos das simulações computacionais já que é necessário finalizar o experimento para realizar a coleta de proveniência.

As abordagens *in situ* propõem realizar a coleta de proveniência durante a execução do experimento, muitas vezes, sem a necessidade de produzir resultados em arquivos. Nessa categoria de ferramenta, se faz necessário configurar a ferramenta de proveniência e o experimento para que a produção de resultados parciais e de dados de proveniência possa ser interpretada pela ferramenta.

As abordagens *in transit*, por sua vez, propõem que a coleta de proveniência aconteça durante a execução do experimento, porém, sem consumir os recursos computacionais destinados ao experimento. Para que isso seja possível, ferramentas nesta categoria são capazes de realizar o processamento de proveniência em recursos computacionais distintos.

Para as simulações computacionais, a abordagem *in transit* é mais adequada que a abordagem *in situ* já que o volume de dados produzido pode ser muito grande e compartilha a capacidade de processamento entre a simulação e a ferramenta de proveniência pode prejudicar o tempo de execução.

As estratégias para definição de proveniência variam de acordo com o refinamento desejado para a granularidade da proveniência e com o tipo desejado. Para as simulações computacionais, é importante que a proveniência permita avaliar os resultados de cada etapa para cada dado de entrada, isto é, é necessário ter granularidade fina em cada transformação de dados. Devido a isso, a maioria das ferramentas *in situ* ou *in transit* que coletam proveniência com granularidade fina podem exigir que os programas que compõem o experimento passem por instrumentação que é o procedimento em que o usuário da ferramenta deve ajustar seu experimento para definir quais dados de proveniência devem ser produzidos e para adaptar os dados de proveniência de maneira que se tornem acessíveis pela ferramenta.

Outro aspecto relevante para a gerência de proveniência de experimentos de simulação é a especificação da proveniência prospectiva. Em geral, ferramentas exigem que o experimento seja modelado como um *script* ou declarados em linguagem própria da ferramenta. No caso das simulações computacionais a especificação da proveniência

pode ser trabalhosa e propensa a erros. Por isso é importante que as ferramentas ofereçam recursos para facilitar esta etapa da configuração das ferramentas.

A captura de dados de proveniência é uma funcionalidade central nas ferramentas de apoio à análise de proveniência. Ela permite que a base de proveniência seja populada e explorada na etapa de análise. O processo de captura de proveniência pode exigir especificação do fluxo de dados e alguma forma de comunicação entre a simulação e a ferramenta. De acordo com cada estratégia adotada, pode ser necessário realizar adaptações nos programas de simulação, ou fornecer uma especificação do fluxo de dados ou ainda depurar arquivos de saída dos programas da simulação. Cada estratégia pode se mostrar mais conveniente de acordo com o tipo de aplicação científica que se deseja atender.

No contexto de simulações computacionais, entretanto, o uso intenso de recursos de PAD e o grande volume de dados configuram barreiras importantes que limitam a eficiência de algumas dessas estratégias presentes nas ferramentas de apoio à análise de proveniência. As simulações computacionais exigem que que a coleta de proveniência seja realizada durante a execução do experimento sem afetar seu tempo de execução.

Capítulo 3 - Trabalhos Relacionados

Existem diversas ferramentas de apoio a análise de dados baseadas na captura de dados de proveniência no contexto mais geral de experimentos científicos. No caso de simulações em larga escala, é necessário estar apto a capturar dados de proveniência em ambiente de PAD e relacioná-los aos dados de domínio ainda durante a execução da simulação. Este capítulo apresenta o estado da arte das ferramentas de captura e análise de proveniência em experimentos computacionais. NoDB e SqlShare são ferramentas *post mortem* capazes de indexar e suportar consultas SQL em arquivos sem a necessidade de construção de uma base de dados. Flashview é uma ferramenta *in situ* que permite consultas em arquivos durante a execução do experimento. noWorkflow e DataTrack são ferramentas in situ capazes de realizar a coleta de proveniência diretamente a partir do experimento sem a necessidade de especificar o fluxo de dados do experimento e sem a necessidade de instrumentação. A biblioteca Tigres oferece recursos de monitoramento e consulta aos resultados de experimentos especificados conforme a linguagem de especificação adotada pela ferramenta. YesWorkflow é uma ferramenta de gerência de proveniência cuja especificação da proveniência a ser capturada se dá pela instrumentação dos programas da simulação. DfAnalyzer é uma é uma ferramenta para captura e gerência de dados de proveniência que permite consultas com suporte a arquivos de dados científicos em tempo de execução a partir de uma base de dados de proveniência que contém informações sobre todo o fluxo de dados. A DfAnalyzer é uma implementação da arquitetura ARMFUL voltada para a análise de arquivos de dados científicos de maneira intensiva em conjuntos com dados de proveniência referente ao fluxo de dados.

No entanto, conforme observado nas análises de cada uma dessas oito abordagens, apresentadas a seguir, DfAnalyzer é a única que provê análise de dados de proveniência em tempo de execução, considerando ambientes PAD. No entanto, nenhuma delas, incluindo aí a DfAnalyzer conforme em SILVA *et al.* (2017), segue uma abordagem assíncrona para o mecanismo de coleta e armazenamento de dados de proveniência. Ao se adotar estratégias síncronas de captura e armazenamento, compromete-se o tempo de execução da simulação computacional devido ao compartilhamento de recursos PAD

entre a simulação computacional e o mecanismo de captura e armazenamento de dados de proveniência.

3.1 - NoDB

O NoDB (ALAGIANNIS et al., 2015) representa um novo paradigma em exploração de dados científicos visto que não requer carregamento de dados para oferecer o conjunto de funcionalidades presentes nos SGBDs. O NoDB propõe realização de consultas estruturadas construída linguagem SQL em arquivos de dados sem a necessidade de construir uma base de dados com o conteúdo dos arquivos reorganizados. Para atender seu propósito, o NoDB desenvolve um mecanismo de indexação adaptativa através da geração de índices posicionais e uma estrutura de cache flexível.

Os dados científicos são extraídos e carregados sem mudanças significativas em suas estruturas no PostgresRaw que é uma adaptação do SGBD PostgreSQL para dados científicos. Embora não exija reestruturação dos dados, o NoDB é baseado no processamento de consultas adaptativas que formam uma base estatística que permite o uso eficiente de estratégias de cache em consultas posteriores. Como apenas os dados necessários para uma consulta são indexados, o uso do NoDB para realização de consultas implica em um menor número de transformações e consequentemente em menos tempo de execução da consulta.

Por outro lado, o NoDB exige a inserção dos dados científicos no PostgresRaw o que implica na sobrecarga da criação dos dados no SGBD e a criação de estruturas de dados auxiliares necessárias para a manipulação de dados e realização de consultas. O NoDB não oferece recursos para promover coleta de proveniência e a análise de dados em tempo de execução, tendo em vista seu objetivo principal é facilitar o acesso a conteúdo de arquivos de dados de domínio.

3.2 - SQLShare

No contexto de experimentos científicos computacionais, os conjuntos de dados produzidos costumam ser analisados por meio de ferramentas *ad hoc* que realizam consultas complexas. Nesse sentido, o uso de sistemas de bancos de dados pode ser

vantajoso já que possuem diversos recursos para facilitar construção e para melhorar o desempenho de consultas complexas. Por outro lado, os dados em questão são usados mais intensamente por curtos períodos de tempo. Esta característica é um limitante para a adoção de SGBDs visto que construir uma base de dados pode ser um processo custoso em termos de modelagem de dados e de carregamento.

O SQLShare (JAIN *et al.*, 2016) é construído sobre a premissa de que pequenas modificações em sistemas de bancos de dados podem tornar seu uso mais vantajoso para atividades de análise de dados de experimentos científicos. O SQLShare é uma ferramenta web que foca em oferecer o potencial analítico das ferramentas oferecidas pelos SGBD para realização de consultas sem as desvantagens da modelagem de base e da inserção de dados.

O SQLShare é um sistema web para gerência de dados científicos hospedado na nuvem de computadores. O sistema oferece as funcionalidades de realizar análise exploratória de dados via consultas SQL sem que seja necessário o esforço de projetar e popular uma base de dados como acontece nos SGBD convencionais.

O SQLShare é baseado no conceito de esquema relaxado (*relaxed schemas*) onde o esquema é inferido automaticamente a partir dos dados contidos na base. Neste aspecto, existe uma grande vantagem em relação aos SGBD tradicionais que precisam de uma definição prévia de esquema. O SQLShare foi projetado para análise de dados, produção colaborativa de resultados derivados. Assim o sistema oferece funcionalidades para criação e compartilhamento de conjuntos de dados derivados de consultas analíticas. Estes conjuntos derivados são criados como conjuntos virtuais baseados em recursos de visões (*views*) relacionais presentes nos SGBD tradicionais.

Assim como o recurso de derivação de resultados, outras transformações de dados que fogem ao escopo de consultas estruturadas são disponibilizadas pelo encadeamento de *views* em vez de operações de atualização e escrita de granularidade da tupla.

Uma limitação do SQLShare é a forma como os dados científicos são adicionados ao sistema. O usuário deve submeter um arquivo através de uma interface REST em algum dos formatos suportados pelo sistema. Os formatos de arquivo suportados compreendem formatos que armazenam dados estruturados em linhas e colunas. Contudo, no contexto da realização de experimentos científicos, reunir todas as informações

relevantes sobre o experimento em tuplas distribuídas em alguns arquivos pode ser uma tarefa custosa e propensa a erros conforme a complexidade do experimento aumenta.

3.3 - FlashView

O FlashView (PANG *et al.*, 2017) uma ferramenta descrita como um sistema de exploração visual de dados. O FlashView busca atender as dificuldades de se trabalhar com grandes quantidades de dados produzidos rapidamente visto que o uso de ferramentas dominantes, como os SGBD, pode introduzir alto custo para carregamento de dados e geração de índices. O FlashView aplica um processamento de consultas aproximado para produzir resultados em tempo real e assim torna a etapa de análise mais ágil e ajuda a selecionar os dados a serem inseridos numa base de dados em SGBD onde consultas mais sofisticadas podem ser realizadas.

As principais contribuições do FlashView para a análise de dados em tempo real seriam funcionalidades que permitem: (1) manipular arquivos de dados diretamente com *parsing* adaptativo que é feito gradualmente conforme o usuário explora os dados. (2) avaliar respostas as consultas de agregação com resultados aproximados de maneira que o usuário receba respostas rapidamente e conforme novos dados são acessados o resultado é atualizado continuamente. (3) acessar interface gráfica para visualização e exploração de resultados. Nela, os dados estão hierarquicamente organizados em subconjuntos. É possível navegar pelos resultados das consultas de agregação em cada subconjunto e gerar novos particionamentos.

Com estas funcionalidades o FlashView proporciona uma forma de realizar uma análise preliminar dos dados para melhor entendimento das características dos dados. Com os as conclusões desta análise preliminar, é possível segmentar os dados e posteriormente carregar porções de interesse em ferramentas com mais funcionalidades como SGBDs. Com esta abordagem, o SGBD seria usado para carregar e indexar apenas dados que serão efetivamente analisados evitando assim atividades de inserção e indexação custosas.

O FlashView oferece um conjunto de possibilidades de consulta bastante restritos, de maneira que atende apenas uma etapa intermediária entre a coleta de proveniência e

sua efetiva inserção em uma base de dados gerenciada por um SGBD que permita consultas mais elaboradas.

3.4 - noWorkflow

A ferramenta noWorkflow (PIMENTEL *et al.*, 2017) é uma ferramenta de coleta de proveniência em scripts programados em Python. A ferramenta noWorkflow propõe a coleta de proveniência com granularidade no nível das chamadas de função. Para atender a essa classe de aplicação, o noWorkflow propõe coletar três categorias de proveniência: definição, implantação e execução.

Proveniência de definição (*definition provenance*) retrata características estruturais da aplicação. Isto é, características que não dependem dos dados entrada nem de características do ambiente de execução da aplicação. Por exemplo, a definição de funções e seus argumentos. A proveniência de implantação (*deployment provenance*) retrata características dos recursos de execução como informações sobre variáveis de ambiente, versão de bibliotecas e sistema operacional. Scripts não são executados em ambientes controlados. Por isso, fatores ambientais podem interferir nos resultados obtidos A proveniência de execução (*execution provenance*) captura as entradas e saídas produzidas pelo script.

Em termos arquiteturais, o noWorkflow pode ser dividido em três módulos: *Provenance Collection*, *Provenance Storage* e *Provenance Analysis*. O *Provenance Collection* extrai proveniência do script e o módulo *Provenance Storage* armazena estes dados que podem ser lidos pelo módulo *Provenance Analysis* e exibidos de diferentes maneiras.

Para usar o noWorkflow, o script deve ser invocado usando um comando específico em vez da invocação trivial de scripts Python. Por exemplo, deve-se usar *now run script.py* em vez do tradicional *python script.py*. A partir da invocação do noWorkflow, o script é inspecionado para coleta de proveniência. Para fazer esta avaliação estática do script, o noWorkflow inspeciona a sintaxe do script e a partir de uma estrutura *abstract syntax tree* (AST) identifica definição e chamada de funções argumentos e referências a variáveis globais.

A proveniência de implantação é capturada antes da execução do script e é associada com uma invocação específica (*trial*). O noWorkflow usa recursos fornecidos pela própria linguagem de programação para capturar informações sobre o hardware do ambiente de execução, sistema operacional e bibliotecas utilizadas.

A proveniência de execução é coletada ao longo da execução do script e consiste em armazenar ativação de funções, argumentos, valores de retorno, tempo de execução e outras informações que sejam específicos de uma execução. A captura destas informações é feita através de especializações da API de perfilagem do Python para que estas invoquem funções do noWorkflow apropriadamente. Para não gerar informações de proveniência em excesso, o usuário pode indicar no código quais variáveis e funções não precisam ter dados coletados com granularidade fina.

Para armazenamento de dados estruturados o noWorkflow mantém uma base de dados relacional SQLite e mantém um diretório para armazenamento de arquivos. Estas duas instâncias de armazenamento são relacionadas por meio de códigos *hash*. Para contornar limitações de sistema operacional o noWorkflow armazena arquivos em diretórios correspondentes a parte do código *hash*. E para reduzir espaço ocupado em disco, arquivos referenciados mais de uma vez na base relacional pelo mesmo código *hash* são salvos apenas uma vez.

A captura e armazenamento de proveniência contribui para a reprodutibilidade do experimento e facilita a análise de resultados e a comparação de técnicas. O noWorkflow suporta análises de grafos de execução, análise comparativa e a realização de consultas.

As análises de grafos produzem visualizações de grafos que facilitam a avaliação do script como fluxo de dados. Nessa forma de visualização, as funções representadas como vértices e as chamadas de função, sequências de chamadas num mesmo contexto e retornos são representadas como arestas. A análise comparativa permite comparar duas execuções (*trials*) distintos. Este tipo de análise é útil para entender diferenças nos resultados já que permite comparar cada etapa do script em diferentes aspectos: modificações no código, mudanças no ambiente de execução ou em versões de bibliotecas. Embora o recurso de consultas SQL seja uma escolha fácil já que os dados estruturados estão em uma base relacional, elas podem ser difíceis de construir e custosas para processar nas bases de dados construídas pelo noWorkflow. Para abordar esta

questão, o noWorkflow conta com uma interface de consulta por meio de regras de inferência baseada em Prolog. Além das regras de inferência que podem ser criadas pelo usuário, o noWorkflow disponibiliza um conjunto de regras para facilitar a realização de consultas.

Como age bem próximo ao interpretador Python, o noWorkflow consegue extrair muitos detalhes sem que o script precise ser alterado. Porém, esta vantagem pode ser um inconveniente do ponto de vista analítico, visto que, quando o usuário não especifica as transformações de dados de interesse, o noWorkflow captura os dados de proveniência de todas as transformações e com granularidade muito fina. No caso de simulações computacionais em larga escala, é possível que grande parte dos dados coletados desta forma podem não ser conveniente do ponto de vista analítico e dificultar a extração das informações que são efetivamente relevantes. Além disso, no contexto de aplicações científicas, é comum um experimento envolver o uso de programas em várias linguagens diferentes e o uso de recursos de PAD.

3.5 - DataTrack

O DataTrack (EICHINSKI & ROE, 2016) é uma biblioteca desenvolvida na linguagem R que suporta coleta de dados de proveniência que representam o fluxo de dados da execução de scripts desenvolvidos em R. Sabendo que várias ferramentas que gerenciam proveniência exigem considerável esforço de aprendizado e desenvolvimento para serem usadas, o DataTrack busca atender as necessidades de cientistas em coleta e análise de proveniência minimizando a necessidade de esforço por parte do usuário. Lerner e Boose (2014) postulam que a alternativa que envolve menor esforço seria coletar toda informação possível de ser extraída da execução do script. Contudo, os autores reconhecem que esta abordagem leva a um conjunto de dados excessivamente volumoso. Visando resolver esta questão, o DataTrack disponibiliza funções para instrumentação dos scripts que o usuário deve usar para definir trechos de código nos quais a coleta de proveniência deve ser mais detalhada.

3.6 - Tigres Workflow Library

Tigres Workflow Library (HENDRIX *et al.*, 2016) é uma biblioteca desenvolvida para apoiar a composição, gerenciar a execução de experimentos computacionais em diferentes plataformas desde desktops até supercomputadores e suportar análise colaborativa através do uso de artefatos de código reutilizáveis. Tigres usa o conceito de *Templates* para facilitar a composição, execução e análise de experimentos computacionais. As etapas dos experimentos computacionais, programas já existentes, não precisam ser alterados e são encadeados através de um fluxo de produção e consumo de dados. O Tigres realiza este encadeamento entre estes programas por meio de scripts na linguagem Python onde as funcionalidades da biblioteca estão disponíveis.

O Tigres conta com componentes para definição do fluxo do experimento (*Template* API), gerência da execução (*Execution Management*), Monitoramento da execução (*Monitoring*) e garantia da consistência do encadeamento de tarefas (*State Management*). No âmbito deste trabalho, vamos nos dedicar a avaliar as características relacionadas à coleta e análise de fluxo de dados, deixando em segundo plano os aspectos relacionados à gerência de execução do experimento.

O usuário realiza a composição do experimento por meio do componente *Templates* API. A API provê *templates* básicos, *sequence*, *parallel*, *merge* e *split*. As semânticas dos *templates* básicos estão relacionadas à cardinalidade dos dados produzidos de maneira semelhante ao modelo de programação MapReduce. A partir dos *templates* o usuário define tarefas (*tasks*) que representam unidades de trabalho que precisam ser executadas ao longo do experimento.

Como um experimento computacional é um encadeamento de programas ligados por dependências de dados, a *Templates* API provê maneiras de compor o experimento definindo dependências de dados implícitas e explícitas que influenciam na ordem de execução das tarefas que compõem o experimento.

O componente Monitoring tem a função de registrar a ocorrência de eventos provenientes do sistema ou definidos pelo usuário em pares chave-valor. Os eventos de sistema proveem informações sobre o estado de um programa como o início da execução

de uma tarefa ou seu estado de execução atual. O Tigres disponibiliza uma API com funções básicas para que o usuário defina eventos e para tratamento de erros de execução.

O componente *State Management* é o responsável por garantir a consistência do experimento como um fluxo de dados antes e depois da execução. Ao longo da execução do experimento, este componente define quais tarefas devem ser invocadas pelo módulo de execução. O componente State Management inda faz a interface entre os módulos de execução e o módulo de monitoramento ao atualizar eventos relacionados ao estado de execução da aplicação.

HENDRIX et al. (2016) traz diversos testes de performance com experimentos sintéticos com características computacionais variadas. As métricas de desempenho adotadas estão voltadas ao tempo de execução do experimento por completo, o tempo de execução de uma tarefa individualmente e o tempo da sobrecarga provocada pelo uso da ferramenta.

A ferramenta Tigres requer que o experimento seja especificado em linguagem Python como um workflow antes da execução do experimento. Essa exigência representa um esforço não trivial a ser realizado pelo usuário a cada modificação no experimento. Como não usa base de dados integrada, a ferramenta não permite consultas mais complexas que são essenciais para análise integrada de resultados.

3.7 - YesWorkflow

O YesWorkflow (MCPHILLIPS *et al.*, 2015) é um sistema de captura e análise de dados de proveniência de scripts em linguagem Python. Para habilitar o uso do YesWorkflow, o usuário deve inserir comentários com marcações próprias e cujo conteúdo sinalize ao YesWorkflow como aquele trecho de código se integra ao fluxo de dados do experimento computacional. É necessário que o usuário identifique trechos de código como transformações de dados e especifique os conjuntos consumidos e produzidos por cada transformação. Com isso, o YesWorkflow é capaz de inferir a relação de dependência de dados entre as transformações.

Para realizar análise exploratória dos dados o YesWorkflow conta com uma interface de consultas e formas de visualização que privilegiam as transformações de dados e nos conjuntos de dados transformados.

3.8 - DfAnalyzer

A DfAnalyzer, conforme visto em SILVA *et al.* (2016b), é uma ferramenta de apoio a realização de experimentos computacionais que provê análise de fluxo de dados de aplicações científicas. Ao acoplar componentes da DfAnalyzer aos códigos-fonte de aplicações científicas, os dados científicos e de proveniência são extraídos e relacionados ao longo de um fluxo de dados disponível para consultas em tempo de execução. A DfAnalyzer possui recursos para monitoramento, depuração e análise interativa de dados em tempo de execução.

As contribuições desta dissertação foram desenvolvidas a partir da versão inicial do DfAnalyzer (SILVA *et al.*, 2017) na qual a coleta de proveniência acontece baseada em uma estratégia bloqueante que faz uso de leitura e escrita de arquivos. As subseções a seguir caracterizam a ferramenta DfAnalyzer e evidenciam os objetivos desta dissertação com contribuição no contexto do desempenho de experimentos de simulações computacionais em larga escala. Este trabalho usa as contribuições alcançadas pela DfAnalyzer como ponto de partida para propor melhorias no processo de coleta e armazenamento de proveniência de simulações computacionais. Assim se faz necessário compreender as contribuições gerais e as estratégias adotadas pela DfAnalyzer especialmente na captura de proveniência. Esta seção aborda uma visão geral da ferramenta DfAnalyzer e suas estratégias para captura de proveniência.

A arquitetura ARMFUL (Análise de Dados Científicos a partir de Múltiplos Arquivos) (SILVA *et al.*, 2017) é o modelo de gerência de fluxos de dados que norteia o desenvolvimento da DfAnalyzer e consequentemente da ADfA. A ARMFUL define componentes para a coleta e análise de dados de domínio e de proveniência de execução da simulação (Figura 3.1). A ARMFUL propõe armazenar dados de domínio e proveniência garantindo atomicidade, consistência, isolamento, durabilidade, controle de concorrência e recuperação fazendo uso de um sistema de gerência de banco de dados.

Para analisar e coletar dados de domínio, a ARMFUL propõe os seguintes serviços durante a execução de aplicações científicas: acesso de arquivos de dados científicos enquanto eles ainda estão sendo gerados, busca por informação relevante dentro destes arquivos, extração de subconjuntos de dados dentro de arquivos, indexação de regiões de interesse, além da execução de consultas que envolvem múltiplos arquivos de dado científico, dados de proveniência e dados de desempenho (ou de recursos computacionais consumidos).

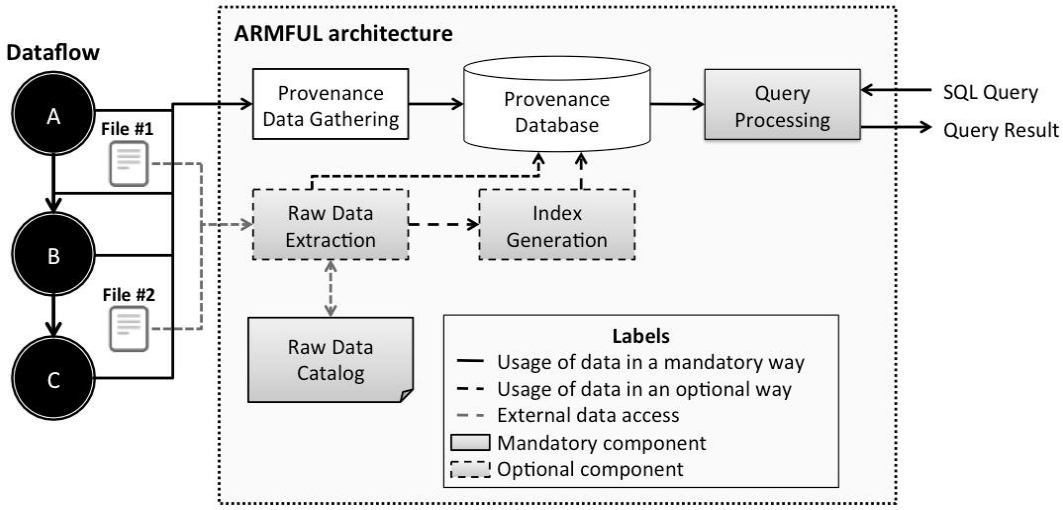

Figura 3.1 - Componentes da Arquitetura ARMFUL. Extraído de SILVA *et al.* (2017a).

A arquitetura ARMFUL apresenta os seguintes componentes:

- **Provenance Data Gatherer (PDG):** é o componente responsável por coletar dados de proveniência, assim como dados de domínio e suas dependências das atividades que compõem a simulação computacional. Este componente precisa coletar e processar as saídas -- impressões no terminal e arquivos gerados -- de cada programa invocado no fluxo de dados para encontrar informações relevantes para a gerência do fluxo de dados e gerar estruturas de dados compatíveis com o modelo de dados adotado pela ferramenta.

- **Raw Data Extractor (RDE):** é o componente responsável por extrair dados de domínio de arquivos de dados científicos que serão consumidos pelo

componente PDG. Para que os dados sejam corretamente extraídos, o usuário deve informar previamente qual cartucho (*cartridge*) deve ser utilizado. Os cartuchos são algoritmos específicos para tratar a extração de dados de domínio de arquivos de um determinado formato de arquivos. Os dados extraídos dos arquivos de dados científicos são armazenados em um arquivo de saída que segue o formato CSV (sigla do termo em inglês *Comma-Separated Values*).

- **Raw Data Indexer (RDI):** Responsável por indexar arquivos de dados científicos, visto que em certos casos essa indexação diminui a quantidade de dados lidos em eventuais consultas. Assim como o componente RDE, o RDI usa cartuchos para lidar com formatos de arquivos específicos.

- **Data Ingestor (DI):** é o componente encarregado de armazenar os dados de domínio, os índices para os dados presentes em arquivos e os dados de proveniência na base de dados relacional. Ademais, o componente DI comunica-se com o componente PDG para adquirir os dados estruturados contendo informações sobre a estrutura do fluxo de dados, sua produção e consumo de dados. Uma vez que este componente adquire novos dados, eles são armazenados e relacionados aos registros já existentes na base de dados.

- **Query Processor (QP):** É o componente que permite a realização de consultas na base de dados de proveniência. Uma vez que os dados de proveniência e os dados de domínio foram extraídos pelo componente PDG e persistidos na base de dados pelo componente DI, o QP pode ser usado pelo cientista para consultar a base de dados por meio de consultas desenvolvidas na linguagem declarativa SQL.

- **Provenance Database (ProvDB)**: A base de dados de proveniência segue um modelo de dados PROV-Df (SILVA et al., 2016) que permite representar qualquer fluxo de dados com relação aos seus dados de proveniência e dados de domínio da aplicação científica (). Este modelo é uma especialização do modelo W3C PROV (MOREAU & GROTH, 2013). O modelo de dados utilizado no ARMFUL contempla dados de proveniência prospectiva, e proveniência retrospectiva além de dados (e metadados) sobre a configuração e o desempenho do ambiente de execução.

A DfAnalyzer consiste em um conjunto de componentes especializados para cada função da ferramenta. Ao seguir a ARMFUL, a captura de dados de proveniência é realizada pelo componente *Provenance Data Gatherer* (PG). A inserção dos dados na base de proveniência é realizada pelo componente *Data Ingestor* (DI). Na DfAnalyzer, a comunicação entre os componentes PG é feita com a leitura e escrita de arquivos no formato JSON em um diretório acessível aos dois componentes.

O componente PG, cuja execução ocorre baseada em invocações realizadas no contexto da simulação, coleta dados de proveniência e de domínio, as formata em objetos serializados no formato JSON. Os objetos são escritos em arquivos em um diretório compartilhado. O componente DI executa como um processo de plano de fundo que consome arquivos adicionados ao referido diretório e persiste novos dados na base de proveniência. Antes de estar apto a inserir dados de proveniência relativos à execução das transformações de dados, o componente DI precisa processar dados de proveniência prospectiva que caracterize toda a simulação. Ou seja, é necessário fornecer uma especificação do fluxo de dados da simulação.

A necessidade de realizar manipulação de arquivos pode se tornar um problema em casos onde a quantidade de elementos de dados capturada seja muito grande. Neste cenário, o tempo entre a produção e a disponibilização de proveniência para consulta pode aumentar de maneira drástica, ou ainda, o uso intenso de recursos de disco para manipulação de proveniência pode afetar o desempenho da simulação.

Há também a necessidade de um diretório compartilhado entre os componentes impede sua execução em alguns ambientes distribuídos como em arquiteturas *shared-nothing* onde não há dispositivo de armazenamento comum entre os nós computacionais.

A obrigatoriedade de especificar o fluxo de dados antes da execução da simulação através de uma linguagem de especificação própria pode ser uma barreira para a adoção da ferramenta. Em casos de simulações complexas com muitas transformações ou em casos em que o usuário não esteja familiarizado com modelagens de fluxos de dados a descrição de proveniência prospectiva pode ser uma tarefa difícil e propensa a erros.

Nesta dissertação são propostas estratégias para especificação do fluxo de dados e captura de proveniência de simulações computacionais diante das limitações encontradas no estado da arte das ferramentas de apoio a análise de proveniência, incluindo a DfAnalyzer, conforme em SILVA *et al.* (2017). A captura de dados assíncrona é uma estratégia que objetiva alcançar melhor desempenho na interação entre a simulação e a ferramenta de proveniência. E a especificação de fluxo de dados flexível busca reduzir o esforço do usuário na configuração da ferramenta.

A ferramenta ADfA construída para validar as estratégias propostas, consiste em uma extensão da DfAnalyzer focada em melhorar o desempenho em termos de tempo e de operações em disco ao monitorar o fluxo de dados em simulações computacionais além de simplificar a inserção de proveniência prospectiva.

Na construção da ADfA, os componentes PG e DI, responsáveis pelas atividades de captura e inserção de proveniência, são redefinidos. Os componentes PG e DI passam a se comunicar por troca de mensagens de maneira que o gargalo provocado pelo acesso a disco deixa de existir. Além disso a ADfA é mais versátil já que pode ser usada em ambientes sem disco compartilhado.

Na estratégia da especificação adaptativa do fluxo de dados, as características do fluxo de dados são descobertas pela ferramenta ADfA a partir da produção de novos dados de proveniência. A partir dos metadados associados aos dados de proveniência, a ADfA remonta o fluxo de dados durante a execução da simulação. Desta maneira, a etapa de especificação do fluxo de dados é simplificada. Esta simplificação torna a ADfA bem mais acessível a usuários de simulações computacionais não especializados em especificação de fluxos de dados.

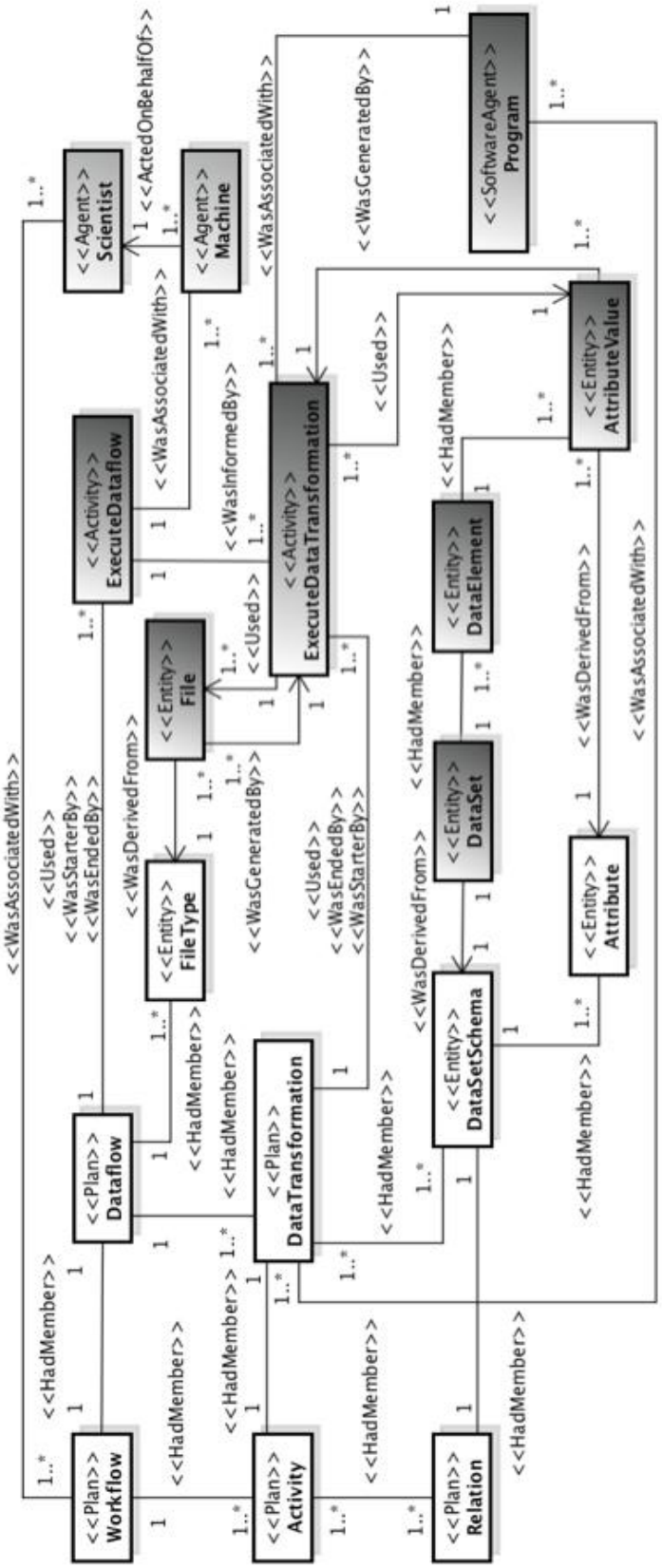

Figura 3.2 - Modelo de dados PROV-Df (SILVA *et al.*, 2016b).

Capítulo 4 - Captura de proveniência assíncrona

Este capítulo descreve a abordagem proposta por esta dissertação, destacando-se os seus objetivos e aspectos de implementação da ferramenta *Asynchronous Dataflow Analyzer*.

Devido ao contínuo aumento da produção de dados científicos por meio de simulações computacionais em larga escala, o uso de informações de proveniência tem se mostrado indispensável na rotina do desenvolvimento científico. Contudo, a construção de soluções específicas para gerenciar proveniência de cada experimento pode ser uma abordagem custosa e lenta.

Como as simulações computacionais são altamente dependentes de grande quantidade de recursos de PAD, e sua execução pode durar meses e uma grande quantidade de dados é manipulada, o usuário precisa ser capaz de identificar inconsistências mitigar problemas rapidamente. Assim, as ferramentas de apoio à proveniência devem trabalhar durante a execução da simulação e disponibilizar informações rapidamente sem concorrer por recursos computacionais com a simulação.

Diversas ferramentas trabalham com consumo e produção de arquivos para realizar a captura de proveniência. Para atender a experimentos com necessidades computacionais elevadas, a manipulação de arquivos pode ser lenta ou representar um custo computacional adicional no tempo de execução da simulação. Por isso, é necessário capturar dados de proveniência sem afetar o fluxo de execução e ainda assim ser capaz de disponibilizar dados para consulta em tempo de execução.

Neste contexto, a ADfA é proposta como uma ferramenta que almeja agregar as estratégias de coleta de proveniência e especificação de fluxo propostas à DfAnalyzer sem abrir mão do conjunto de funcionalidades já disponível na ferramenta. A funcionalidade central da ferramenta é a capacidade de capturar e organizar sistematicamente os dados das simulações computacionais, visto que todas as funcionalidades de análise e indexação dependem do sucesso do processo de aquisição de dados. Assim, este trabalho se debruça especialmente nesta etapa da gerência de dados de proveniência. O objetivo principal é produzir uma ferramenta que faça a aquisição de proveniência em tempo de execução com eficiência.

Para realizar a coleta e processamento de proveniência de maneira eficiente, a ADfA adota um padrão de comunicação de troca de mensagens. Assim, a coleta de proveniência acontece por meio de mensagens transmitidas pela simulação que são recebidas e processadas pelos recursos da ferramenta.

Para que a simulação seja capaz de produzir mensagens no formato definido pelo ADfA, ela precisa ser instrumentada com trechos de código que façam a captura de dados de proveniência, a configuração das mensagens e o envio para a ADfA. Para simplificar o procedimento de instrumentação da simulação, a ADfA contém uma biblioteca de funções que auxilia na configuração e envio das mensagens de maneira que o usuário possa focar seus esforços em definir corretamente quais dados devem ser capturados.

O esquema da base de proveniência é construído com base em uma especificação do fluxo de dados produzida pelo usuário. Como esta pode ser uma tarefa difícil e trabalhosa, a ADfA oferece recursos para reduzir o esforço de configuração por meio da realização a especificação do fluxo de dados no momento da instrumentação da simulação. Assim a especificação da proveniência prospectiva e a instrumentação para definir a coleta da proveniência retrospectiva são reduzidos a uma única atividade.

A ADfA é uma implementação da arquitetura ARMFUL considerando novas estratégias para capturar dados de proveniência. A ADfA objetiva prover as funcionalidades especificadas na arquitetura ARMFUL em tempo de execução, com esforço reduzido de configuração, de implantação e de agregação de novas funcionalidades.

Nesse sentido, as decisões de projeto e estratégias de desenvolvimento adotadas na ADfA objetivam prover recursos de gerência de proveniência acessíveis ao cientista sem, contudo, abrir mão de desempenho computacional que é uma questão crítica para simulações computacionais. As seções a seguir descrevem as contribuições técnicas presentes na ADfA para implementar estes princípios: implantação dos recursos de proveniência como serviço, a estratégia de coleta de proveniência, a comunicação por troca de mensagens e a flexibilidade na definição do modelo de dados.

4.1 - Gerência de Proveniência como Serviços

A ADfA visa oferecer recursos de acesso a proveniência com baixo esforço de adoção e de implantação em ambientes de PAD em geral. Além disso, é importante que a ferramenta estabeleça uma interface atrativa para integração de novas ferramentas de consulta, análise e captura. Para isso, a ferramenta foi construída como um conjunto de micro serviços (JARMAN, 2015) e com o padrão de comunicação REST.

A arquitetura de micro serviços consiste em destacar as funcionalidades da aplicação em componentes de *software* que executem de forma independente e cuja comunicação seja feita por troca de mensagens. Uma das vantagens da comunicação via troca de mensagens obriga que os componentes mantenham uma interface para troca de dados coerente com o sistema. Com isso, o sistema pode ser desenvolvido de forma incremental por diferentes equipes sem a necessidade de que os desenvolvedores conheçam profundamente todos os componentes da aplicação.

Outro conceito proveniente do desenvolvimento de *software web* que pode ser útil para o desenvolvimento deste sistema é o padrão de comunicação REST (RODRIGUEZ, 2008). Aplicações desenvolvidas neste padrão são caracterizadas por: terem seus componentes representados como recursos acessíveis por URIs (*unique resource identifier*) com semântica representativa para o usuário; por interagir com as aplicações clientes usando o protocolo HTTP; e por disponibilizar apenas recursos *stateless*, isto é, recursos que não precisam armazenar dados de contexto já que requisições de acesso são feitas por meio de mensagens que contém todos os dados necessários para o seu processamento.

Neste contexto, os componentes de uma arquitetura de serviços podem ser classificados em três tipos de componentes básicos: consumidores, que fornecem serviços por meio as URIs, um *provedor* para a publicação de serviços e produtores que acessam serviços enviando mensagens para o *provedor*. Os consumidores são os componentes da ADfA que realizam alguma função dentro do contexto da arquitetura ARMFUL, como o componente de coleta de proveniência ou o componente de processamento de consulta. O *provedor* é o componente que recebe as requisições e, de acordo com a URI, as direciona para ser consumidas por recursos específicos. Os produtores são processos

externos que interagem com a ADfA por meio de requisições HTTP que serão processadas pela ferramenta. As principais aplicações produtoras seriam as simulações científicas e as ferramentas de análise que consultam a base de dados.

Seguindo a estratégia de serviços REST, a ADfA disponibiliza serviços de coleta e análise de proveniência como um conjunto de recursos independentes acessíveis por requisições remotas (Figura 4.1). No ADfA, os recursos são disponibilizados como serviços *web* que são capazes de receber e interpretar mensagens de acordo com o protocolo HTTP e processar estas requisições de acordo com seu conteúdo. O *provedor* do ADfA pode ser concebido como uma camada de interface entre a aplicação cliente e os recursos com acesso à base de dados de proveniência. Por exemplo, na coleta de novos dados, o *provedor* é a interfaces entre a simulação e o serviço de ingestão de dados. No caso da exploração de resultados, o *provedor* é a interface entre a aplicação analítica e a base de proveniência.

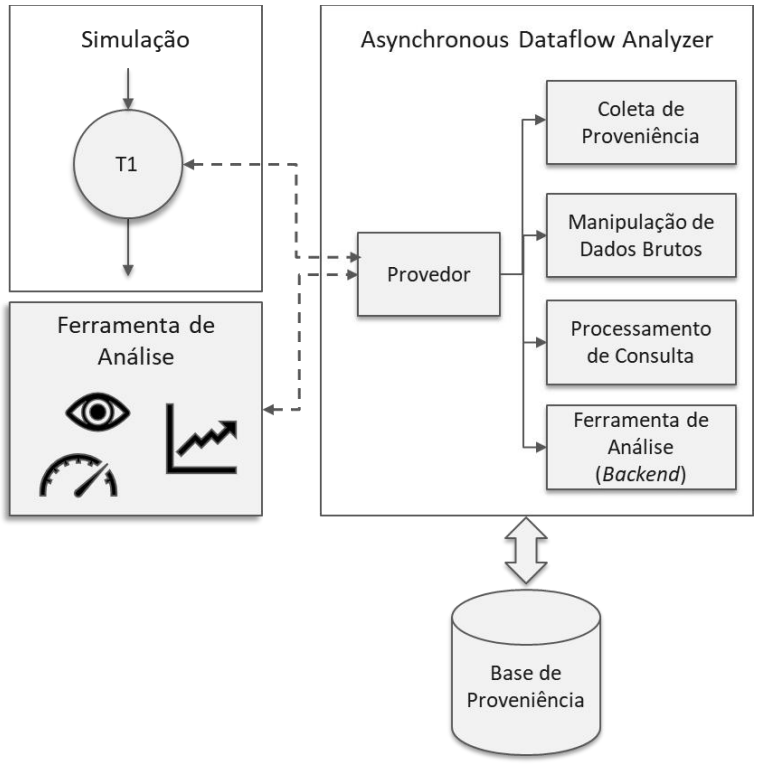

Figura 4.1 - Esquema dos componentes da ADfA e sua interação com diferentes tipos de aplicação cliente.

O *provedor* disponibiliza URIs únicas que definem cada recurso oferecido a aplicações externas. As URIs são acessíveis por meio de requisições HTTP cujo conteúdo especifica dados gerados ou consultas a serem realizadas. No caso do recurso de aquisição de proveniência, as requisições são enviadas pela simulação e seu conteúdo deve estar formatado seguindo os padrões da ADfA para mensagens com dados de proveniência. Para facilitar a composição e o envio de mensagens para o recurso de processamento de proveniência da ADfA foi criada uma biblioteca de funções para facilitar a construção e transmissão de mensagens que faz a interface entre a ferramenta e a aplicação científica.

Escolhas arquiteturais e decisões de projeto voltadas para não acoplamento entre componentes do sistema favorecem a agregação de novos componentes à infraestrutura desenvolvida. Os requisitos de integração de software podem ser resumidos em compatibilidade do padrão de mensagens e conhecimento sobre o esquema da base de dados.

O componente *provedor* foi construído a partir do arcabouço para construção de aplicações web Spring Boot (PIVOTAL SOFTWARE, 2015) que oferece recursos para publicação de serviços e comunicação com o padrão REST. A aplicação é hospedada por um servidor web Tomcat (APACHE SOFTWARE FOUNDATION, 2017) que faz a manipulação das mensagens HTTP.

4.2 - Coleta de Proveniência

Dentre os recursos fornecidos no ADfA, a coleta de dados fornecida pelo componente PDG pode ser considerado o mais central visto que é o recurso responsável por produzir os dados básicos nos quais os outros componentes realizarão seus recursos. Esta seção se destina a detalhar como é feita a coleta de proveniência no ADfA.

A principal função de uma ferramenta como a ADfA é a análise de proveniência durante a execução da simulação. Esta análise se dá a partir do processamento e exploração de dados produzidos em etapas estão em execução ou estão finalizadas bem como dados sobre a produção e consumo de arquivos nestas etapas da simulação. A coleta de dados de proveniência é feita a partir da comunicação entre a simulação e a ferramenta ADfA, nesse contexto, produtor e consumidor respectivamente. A comunicação é uma troca de mensagens que se dá através de requisições HTTP. Para que a comunicação seja

possível, é necessário instrumentar a simulação, isto é, adaptá-la para produzir e transmitir informações de proveniência no padrão adotado pela ferramenta.

O *provedor* disponibiliza URIs com semântica relacionada à sua função, que são acessadas por requisições enviadas a partir da simulação. As mensagens contêm dados que identificam a simulação, a etapa em execução, informações de controle, informações de performance e informações sobre os dados consumidos e gerados na transformação em questão. O padrão de mensagem adotado formato JSON (do inglês *JavaScript Object Notation*) que é um formato de troca de dados entre sistemas muito usado em aplicações web.

A adaptação dos programas da simulação para que estejam aptas a se comunicar com o ADfA pode ser uma tarefa custosa. Para favorecer o uso da ferramenta em aplicações reais, foi construída uma API na linguagem C que simplifica esta tarefa. A API contém estruturas de dados e métodos para auxiliar a construção e o envio de mensagens HTTP com conteúdo coerente em relação ao padrão esperado pelo ADfA.

Para cada mensagem recebida, o *provedor* encaminha o procedimento de acordo com URL fornecida na requisição. O recurso converte o conteúdo da mensagem em objeto um objeto Java que o represente. Por exemplo, os dados da execução de uma tarefa são representados pela classe *Task*. Os objetos contidos na mensagem são encapsulados em um objeto da classe *Transaction*, que representa uma interação em potencial com a base de dados e inseridos em uma fila de proveniência (*ProvenanceQueue*). Um componente de inserção de proveniência consome dados da fila na ordem em que foram adicionados, inspeciona informações sobre o fluxo de dados e as insere na base de dados.

4.3 - Especificação do Fluxo de Dados e a Base de Proveniência

Esta seção disserta sobre como é construído o esquema da base de dados que armazena a proveniência do fluxo de dados da simulação. Na ADfA, para que a proveniência seja armazenada em uma base de dados, é necessário fornecer uma definição do fluxo de dados. Com essa especificação, a ferramenta é capaz de construir as estruturas de dados adequadas para armazenar os dados que serão produzidos pela simulação.

A necessidade especificação da proveniência prospectiva, isto é a modelagem do fluxo de dados, em linguagem própria pode ser uma grande barreira para a adoção de ferramentas de proveniência visto o esforço que pode ser exigido do usuário ao longo do ciclo de vida da simulação computacional. Muitas ferramentas exigem a especificação do fluxo de dados antes do início da execução do experimento, momento em que a ferramenta cria uma base de dados de proveniência.

A especificação prévia é utilizada, nestas ferramentas, para composição da estrutura da base de dados de proveniência (Tabelas, Colunas, Relacionamentos, Índices) que abrigarão os dados do experimento. Especificar o fluxo de dados pode ser uma tarefa árdua para o usuário do domínio científico, visto que a especificação de fluxo precisa ser detalhada e correta para que a base de dados seja construída corretamente.

Em cenários dinâmicos como o desenvolvimento de simulações computacionais, ocorrem várias iterações do ciclo de vida do experimento científico, e assim, é comum que partes do experimento sejam melhoradas a partir da obtenção de resultados. Assim, é razoável esperar que o encadeamento de programas e os atributos extraídos evolua entre rodadas de execução de uma simulação com alterações no encadeamento de programas e surgimento de novos atributos. Com isso, a especificação do fluxo de dados pode mudar em pouco tempo levando sucessivas especificações a cada iteração do ciclo de vida.

Esta característica também obriga a reconfiguração em diversas ferramentas que oferecem apoio a exploração de dados de proveniência, visto que nestas tecnologias é necessário especificar detalhes sobre o fluxo de dados antes do início do experimento. Além de ser necessário atualizar a instrumentação da simulação.

Há, também, ferramentas que capturam dados de proveniência sem exigir especificação do fluxo. Contudo, visto que o usuário não tem a possibilidade de especificar quais as transformações de dados que precisam ser monitoradas, a ferramenta realiza uma coleta automática de todas as transformações de dados. A coleta indiscriminada de dados de proveniência, provocada pela não de especificação das transformações de interesse pode ser uma desvantagem sobretudo em simulações devido ao grande volume de dados tratado por este tipo de aplicação.

A ADfA adota o modelo em que o usuário precisa realizar especificação do fluxo de dados, contudo oferece recursos para reduzir esforço e retrabalho nesta tarefa. Na

ADfA, a especificação do fluxo de dados é definida junto com a instrumentação da simulação. Com isso, as informações de proveniência prospectiva são recebidas pela ADfA junto com a proveniência retrospectiva, ou seja, durante a execução da simulação.

```
{
   "sets":[
         { "elements":["1;8;1"],
            "tag":"sparsegridinput",
            "attributes":[{"name":"vmid","type":"NUMERIC"},
                           {"name":"dimension","type":"NUMERIC"},
                           {"name":"level","type":"NUMERIC"}],
            "type":"INPUT"
         },
         { "elements":['$elements'],
            "tag":"sparsegridoutput",
            "attributes":[{"name":"vmid","type":"NUMERIC"},
                           {"name":"dimension","type":"NUMERIC"},
                           {"name":"level","type":"NUMERIC"},
                           {"name":"region","type":"FILE"},
                           {"name":"weights","type":"FILE"},
                           {"name":"points","type":"FILE"}],
            "type":"OUTPUT"
         }],
   "subid":"'${i}'",
   "workspace":"\/home\/luciano\/Desktop\/pg",
   "dataflow":"uq_rtm_complex",
   "dependency":{},
   "resource":"'$RESOURCE_NAME'",
   "id":"'${i}'",
   "transformation":"sparse_grid_construction",
   "status":"FINISHED",
   "program": "programa_executado",
   "output": "output"
   "error": "error"
}
```

Figura 4.2 - Exemplo de Mensagem Completa no padrão ADFA

Para que seja possível fundir as etapas de instrumentação da simulação e especificação de fluxo em uma tarefa, a ADfA adota um padrão de mensagens e um algoritmo de processamento de mensagens que permite adaptar a base de dados para novos dados de proveniência em tempo de execução. Assim, ao longo da execução do experimento, mensagens contendo proveniência são processadas novas características do

fluxo de dados são identificadas e com isso a base de dados é incrementada com novas tabelas e colunas de acordo com as informações de proveniência capturadas.

```
Mensagem 1:
{"sets":
   [ {
         "tag":"output_set",
         "type":"OUTPUT",
         "attributes":[{"name":"A","type":"NUMERIC"}],
         "elements":["x"]
   }]
         …
}

Mensagem 2:
{"sets":
   [ {
         "tag":"output_set",
         "type":"OUTPUT",
         "attributes":[{"name":"B","type":"NUMERIC"}],
         "elements":["y1", "y2"]
   }…]
         …
}

Mensagem 3:

{"sets":
   [ {
         "tag":"output_set",
         "type":"OUTPUT",
         "attributes":[{"name":"C","type":"NUMERIC"}],
         "elements":["z"]
   }…]
```

Figura 4.3 - Três mensagens válidas para uma mesma transformação de dados.

Para que isto seja possível, é necessário que o componente de processamento de proveniência apto a atualizar o esquema da base de dados conforme os programas da simulação produzam mensagens com especificações sobre o fluxo de dados que permitam a inserção correta dos dados de proveniência. Estes metadados devem permitir a identificação de conjuntos de entrada e saída, seus atributos e descrever o processo de

extração de dados nos casos de operação de arquivos científicos. Nos metadados deve haver ainda informações que definam as dependências da transformação.

Após processamento da Mensagem 1:

Id	A
1	0,17

Após processamento da mensagem 2:

Id	A	B
1	0,17	NULL
2	NULL	1,0
3	NULL	0,2

Após processamento da mensagem 3:

Id	A	B	C
1	0,17	NULL	NULL
2	NULL	1,0	NULL
3	NULL	0,2	NULL
4	0,6	NULL	8

Figura 4.4 - Materialização das mensagens na base de dados

A Figura 4.3 mostra um exemplo de três mensagens válidas para uma mesma transformação de dados. Cada mensagem contém informação sobre uma tarefa que é uma execução de uma mesma transformação. Contudo, a primeira mensagem tem o atributo A que não está presente na segunda mensagem. A segunda mensagem por sua vez tem o

atributo B, que não está presente na primeira mensagem. A materialização destes elementos de dados na base de proveniência é ilustrada na Figura 4.4.

No momento de inserção de novos registros na base de dados, cada tarefa inserida terá seus metadados avaliados e eventualmente a base será adaptada caso um novo atributo seja descoberto. Cada tarefa tem os metadados da transformação extraídos e comparados com a estrutura conhecida do fluxo de dados. Caso haja alguma diferença, a representação do fluxo de dados é acrescida, passando a suportar os dados anteriores e os dados da nova tarefa.

Visto que tempo com a preparação do ambiente para análise de dados está sendo poupado pode se dizer que o ADfA privilegia o desenvolvimento do experimento e a realização de análises em relação à manutenção de infraestrutura de software e de armazenamento de dados.

A estratégia de adaptação do esquema, por outro lado, aumenta a chance de que os dados estejam mais esparsos na base de dados. Por exemplo, numa mesma transformação, existe um grande número de variações nos atributos das tarefas, boa parte delas não terá dados em todas as colunas tornando a base de dados esparsa. O que por sua vez poderia impactar o desempenho de consultas realizadas durante a etapa de análise.

Contudo, esta proposta foi construída e avaliada a partir de uma base de proveniência hospedada em um SGBD colunar, o MonetDB (ABADI, 2007). Algumas características do MonetDB são convenientes para a estratégia de esquema adaptativo. Algumas características do MonetDB são convenientes para o armazenamento e a recuperação de dados esparsos que podem ser mais frequentes quando adotada a estratégia de esquema adaptativo.

4.4 - Base de Dados Colunar

Os SGBDs mais comuns como Oracle, SQLServer, MySQL, PostgreSQL organizam os valores armazenados com base em linhas das tabelas de maneira que os valores das colunas de uma tabela são armazenados em sequência. Por outro lado, o MonetDB é uma base de dados colunar, o que significa que cada coluna de uma tabela é

armazenada separadamente de maneira que valores sucessivos são armazenados em sequência.

Bases de dados colunares, em geral, apresentam bons resultados com tabelas com muitos atributos e tabelas esparsas. Visto que em base de dados colunar apenas as colunas que serão acessadas por uma consulta precisam ser lidas pelo SGBD, estratégias de otimização de uso de banda podem ser aplicadas com maior sucesso.

O tamanho ocupado por um atributo de interesse costuma ser menor que a menor granularidade de leitura disponibilizada em computadores modernos. Uma vez que, em bases de dados orientadas ao armazenamento de linha, os atributos de uma tupla são armazenados lado a lado, a leitura de um atributo de interesse implica na leitura de outros valores que não serão relevantes para a consulta em execução. No caso das bases colunares, os valores lidos em uma mesma operação pertencem a uma mesma coluna. Assim é possível que os dados necessários para realização da consulta sejam recuperados com menos operações de leitura. Ou ainda, que a taxa de acerto da cache seja maior.

Em dados de simulações computacionais, os elementos dados produzidos por uma transformação de dados podem ser compostos de muitos atributos. A quantidade de colunas de uma tabela tem menor impacto sobre o desempenho da execução de uma consulta em um SGBD colunar em relação aos SGBD orientados ao armazenamento de linhas. O MonetDB conta com estratégias para lidar com dados esparsos e com a facilidade de inserir novas colunas sem afetar os dados já armazenados. Estas características são de alta relevância para o ADfA, visto que sua estratégia adaptativa implica que novas colunas podem ser criadas em tabelas existentes. O uso do MonetDB torna esse processo muito menos custoso uma vez que não haverá reescrita dos elementos de dados existentes.

Capítulo 5 - Estudos de caso e experimentos

Este capítulo descreve a avaliação de desempenho da ADfA em relação à sua capacidade de prover serviços de coleta e sua capacidade de disponibilização de proveniência para realização de consultas analíticas em experimentos científicos centrados em simulações computacionais de larga escala e executados em ambiente de PAD.

As principais contribuições da ADfA para o estado da arte das ferramentas de apoio a análise exploratória de dados de arquivos científicos são contribuições que melhoram o desempenho da captura de proveniência e facilitam o trabalho do usuário com o processo de captura de dados de proveniência. Para avaliar o desempenho da ferramenta com as contribuições deste trabalho, foram utilizados dois casos para estudo: um fluxo de dados sintético gerado por um programa construído para testar a ferramenta e uma simulação computacional real de sedimentação.

O desempenho da ADfA foi avaliado segundo um conjunto de métricas que nos permite avaliar o comportamento de componentes internos do serviço de captura de proveniência da solução proposta. Os experimentos foram realizados no cluster Lobo Carneiro da COPPE/UFRJ devido à necessidade de recursos de processamento de alto desempenho.

A adoção da estratégia adaptativa, proposta nessa dissertação, para captura de proveniência prospectiva reduz o esforço do usuário na etapa de especificação do fluxo de dados. Por outro lado, é necessário avaliar o impacto no tempo de processamento da simulação. É necessário quantificar como a ADfA afeta o tempo de execução da simulação. Além disso, é preciso avaliar o tempo necessário para que os dados capturados estejam disponíveis para consulta. A estratégia foi avaliada de maneira comparativa com os mesmos dados sendo capturados via leitura de arquivos e com definição do fluxo de dados estática. Para validar a eficiência da proposta de captura adaptativa, realizamos um experimento comparativo entre a ferramenta desenvolvida neste trabalho e a versão em produção da ferramenta DfAnalyzer que é a tecnologia que melhor atende este nicho de aplicações.

O restante deste capítulo expõe sobre os casos de estudo (Seção 5.1 -), o ambiente de testes e a configuração dos experimentos (Seção 5.2 -), as métricas de desempenho (Seção 5.3 -), os resultados e as conclusões obtidas com base nos dados coletados para fluxos de dados sintéticos (Seção 5.4 -) e um experimento que compara a tecnologia de mensagens do ADfA com a tecnologia de comunicação por arquivos da DfAnalyzer (Seção 5.5 -).

5.1 - Casos de Estudo

Testes preliminares mostraram que o uso de uma estratégia de troca de mensagens para captura de dados reduz o esforço computacional necessário para capturar dados de proveniência. Contudo, as simulações computacionais podem ser muito distintas no que diz respeito ao fluxo de dados e, consequentemente, aos dados de proveniência produzidos. Portanto, é importante avaliar o comportamento da solução em situações variadas.

A fim de validar as contribuições da ferramenta ADfA em situações diversas, foram utilizados dois casos de teste: um fluxo de dados sintético representando uma simulação sintética e um experimento científico que simula correntes turbidíticas para o estudo de processos de sedimentação de partículas baseado em uma aplicação chamada *libmesh-Sedimentation*. Os dois casos de estudo foram avaliados segundo um conjunto de métricas que nos permite avaliar o comportamento de componentes internos do serviço de aquisição de proveniência da solução proposta.

Para produzir o fluxo de dados sintético, foi construído um pequeno programa que permite variar livremente as características dos dados gerados como a quantidade de dados enviados, a quantidade de elementos de dados em cada mensagem, o tamanho em bytes de cada elemento de dado ou a frequência que as mensagens devem ser emitidas. Com este recurso pode-se observar como as características da mensagem afetam o tempo de disponibilidade da proveniência. O fluxo de dados sintético é usado ainda para realizar uma avaliação comparativa entre a abordagem de troca de mensagens da ADfA é uma solução baseada em inspeção de arquivos. No segundo estudo de caso, utilizamos um experimento científico centrado na simulação de sedimentação de partículas para avaliar a ferramenta em uma situação realística.

5.1.1 - Fluxo de dados sintético

O fluxo de dados sintético é produzido por uma aplicação capaz de produzir um conjunto de dados de proveniência com características que permitam analisar aspectos internos da solução proposta. Esta aplicação conta com parâmetros de entrada para produzir cenários convenientes em relação às características dos dados e a taxa de envio de mensagens. Projetada para cobrir variados casos, a aplicação permite configurar o conteúdo das mensagens em relação à quantidade de dados em cada elemento de dados produzido, à quantidade de elementos por mensagem e em relação à quantidade de mensagens enviadas num intervalo de tempo (vazão). Para facilitar a análise, os experimentos deste estudo de caso representam o consumo de dados de apenas uma transformação.

Simulações computacionais, em geral, são aplicações paralelas e escaláveis que executam transformações de dados em diversos processos computacionais distintos e em ambiente distribuído. Para que o experimento se assemelhe com uma simulação real, a aplicação suporta execução paralela em ambiente distribuído usando o arcabouço de computação paralela MPI (*Message Passing Interface*).

Analisar o comportamento da ferramenta para diferentes cargas de dados com diferentes características de mensagens é importante para compreender como cada característica da carga de dados influencia no tempo de disponibilidade de dados. Por exemplo, para enviar 100 elementos de dados produzidos em um nó computacional, podemos enviar todos em uma única mensagem ou em 100 mensagens diferentes. Embora a carga de dados total seja a mesma, a distribuição dos dados nas mensagens é diferente e isso pode afetar algumas etapas do fluxo de processamento de mensagens.

A aplicação sintética é uma aplicação executável por linha de comando que recebe como parâmetros: a quantidade de processo que o programa deve invocar, np, a quantidade de elementos de dados que deve ser enviada, el, o intervalo de tempo entre cada envio de mensagem, dt e o número de mensagens que deve ser enviado por cada processo m. Assim, a quantidade total de elementos de dados enviada para a ferramenta ADfA é dada por np x m x el. E o número de mensagens enviadas por segundo é dada por (np x m x el) / dt.

Este último valor será referenciado como a vazão total de geração que será avaliada em conjunto com a vazão de consumo, que é a quantidade de elementos de dados adicionados na base de dados por unidade de tempo. Os experimentos que seguem mostram como variações nas características dos dados gerados afeta o comportamento de cada componente do serviço e, consequentemente, como afeta o tempo de disponibilidade da proveniência.

5.1.2 - libMesh-sedimentation

A aplicação libMesh-sedimentation (CAMATA *et al.*, 2018), usada no estudo de caso de avaliação desta ferramenta, é uma simulação computacional que modela correntes de turbidez tipicamente encontradas em processos geológicos. Correntes de turbidez são fluxos de partículas cujo principal condutor é a turbulência. Estas partículas podem ser carregadas por longas distâncias e se acumular formando depósitos de sedimentos que constroem formações geológicas. Sedimentação e erosão promovidas por estes fluxos de partículas podem modelar o solo oceânico, produzindo diferentes estruturas geológicas, como cânions, dunas e ondulações. O processo de formação geológica é de interesse da indústria de óleo e gás.

A aplicação libMesh-sedimentation utiliza uma abordagem de elementos finitos baseada em um método de variação residual multi escala para simulação de grandes redemoinhos descrito em Guerra et al. (2013). Para produzir resultados mais detalhados, a simulação conta com uma estratégia de refinamento de malha construído com recursos da libMesh (BAUMAN & STOGNER, 2015), 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 (AMR/C, do termo, em inglês, *Adaptive Mesh Refinement and Coarsing*).

As macro etapas de um processo de simulação de correntes de turbidez podem ser resumidas em: (i) pré-processamento dos dados de entrada e geração da malha em memória; (ii) simular o processo de fluxo de partículas e sedimentação ao longo do tempo e salvar dados de interesse, como pressão, velocidade e concentração de sedimentos em vários momentos da simulação; (iii) realizar pós-processamento dos dados para gerar informações que permitam análise dos resultados.

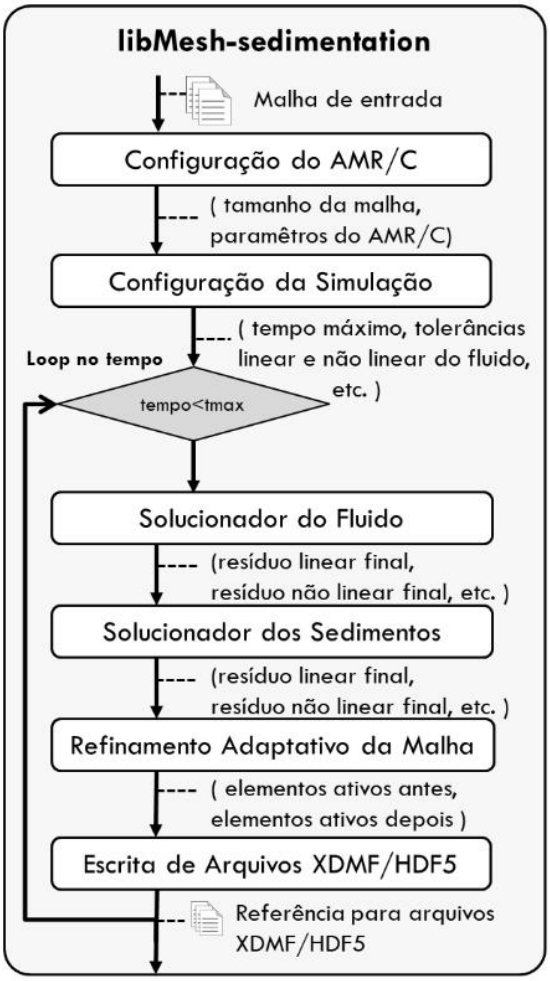

Figura 5.1- Representação dos macro passos do programa libMesh-sedimentation. Extraído de SILVA *et al.* (2018).

Do ponto de vista de coleta e análise de fluxo de dados é importante caracterizar as transformações de dados, suas dependências e seus resultados. A partir da identificação das transformações de dados de interesse do usuário, a simulação pode ser instrumentada através de chamadas de função da biblioteca ADfA Provenance para produzir os dados de proveniência empacotadas em mensagens HTTP.

Na Figura 5.1, a simulação implementada na libMesh-sedimentation é descrita como um fluxo de transformações de dados interligadas por suas dependências (SILVA, 2018). As duas primeiras transformações de dados compreendem a leitura do *dataset* de entrada e a computação de parâmetros que irão determinar o comportamento do fluxo de

simulação. A transformação de dados Configuração do AMR/C consome a malha que representa o domínio onde a simulação ocorre e produz parâmetros a serem usados no refinamento adaptativo da malha. A transformação Configuração da Simulação é onde são definidos parâmetros de controle da simulação por meio da leitura e pré-processamento de arquivos de configuração.

O conjunto de transformações de dados dentro do *loop* de passos de tempo da simulação são repetidos uma vez a cada passo de tempo até que o contador de passos chegue ao valor máximo *tmax*. A transformação Solucionador de Fluido representa o cálculo do comportamento do fluido de acordo com o modelo matemático adotado e o estado mais recente da simulação; a etapa Solucionador de Sedimentos age de maneira análoga à etapa anterior para calcular o comportamento dos sedimentos.

Em seguida, a transformação de dados Refinamento Adaptativo da Malha avalia e executa ajustes de aumentar ou diminuir o refinamento da malha. A avaliação é feita com base em parâmetros de entrada associados às estimativas de erros calculadas no estado mais recente da simulação. A última etapa do *loop* é a etapa de Escrita de Arquivos XDMF/HDF5 que é o momento onde o algoritmo salva em disco os dados científicos que caracterizam o estado mais recente da simulação. Por questões de disponibilidade de recursos, esta última etapa não é executada em todos os passos de tempo. A escrita de arquivos acontece apenas em grandes intervalos de tempo para reduzir custos com armazenamento e reduzir a sobrecarga da escrita de arquivos no tempo de execução da simulação.

O experimento de sedimentação é baseado no experimento de sedimentação em um tanque proposto por DE ROOIJ & DALZIEL (2001). O tanque é modelado como uma caixa retangular com dimensões 20 x 2 x 2. A simulação inicia com os sedimentos separados fisicamente do fluido em uma câmara de dimensões 0,75 x 2 x 2 em um dos extremos do tanque conforme mostra a Figura 5.2. A configuração do experimento é caracterizado detalhadamente em CAMATA *et al.* (2018).

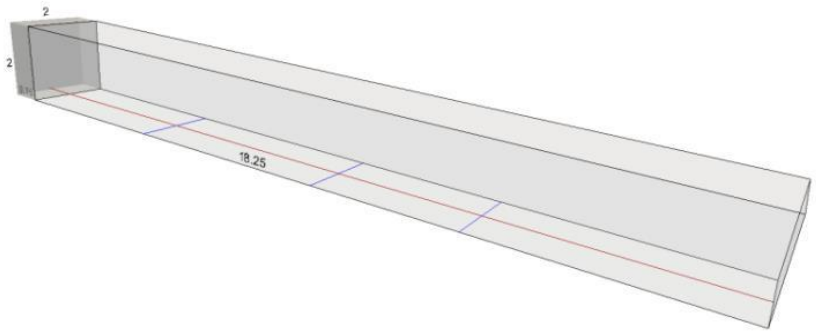

Figura 5.2 - Representação do tanque de sedimentação proposto por de Rooij e Dalziel. Adaptado de CAMATA *et al.* (2018)

Ao habilitar o uso das funcionalidades da ADfA, o usuário pode executar consultas exploratórias que associam os dados de domínio produzidos pela simulação com informações de proveniência de dados. Por exemplo pode ser a inspeção por resultados incorretos e a busca pelas causas do problema. A libMesh-Sedimentation permite ajustar parâmetros em tempo de execução e retroceder no loop de passos de tempo para uma versão consistente da simulação. As funcionalidades da ADfA permitem realizar consultas na base de proveniência ainda durante a execução da simulação permitindo a tomada de decisão antes da finalização do experimento.

5.2 - Ambiente de Testes

Os experimentos descritos neste capítulo foram executados no cluster de computadores Lobo Carneiro (LoboC), do Núcleo Avançado de Computação de Alto Desempenho da COPPE/UFRJ. O LoboC é um cluster SGI ICE-X Linux que conta com 504 processadores Intel Xeon E5-2670v3 (Haswell) distribuídos em 252 nós de processamento. Cada processador contém 12 núcleos de processamento, totalizando 6048 núcleos. Os processadores contam com a tecnologia Hyper-Threading (HT), assim oferecendo 48 threads de processamento por nó. Cada nó conta com 64 Gb de memória RAM e são interligados por rede Infiniband FDR - 56 Gbs (Hypercube). O cluster funciona sob a arquitetura de disco compartilhado, com um sistema de arquivos paralelo Intel Lustre com capacidade de armazenamento de 500 Tb. O LoboC pertencente ao

Núcleo Avançado de Computação de Alto Desempenho (NACAD) e está localizado na COPPE / UFRJ.

Nos experimentos descritos neste capítulo, um nó computacional foi reservado para execução da infraestrutura da ADfA e de uma instância do SGBD MonetDB que contém a base de proveniência. O *provedor* possibilita que as mensagens HTTP contendo dados científicos e de proveniência sejam consumidas pelos recursos REST adequados. Os recursos REST realizam operações na base de dados de proveniência, instanciada no SGBD MonetDB, por meio do driver JDBC para carregar novos dados produzido pelo processamento de mensagens.

Nos experimentos comparativos com a DfAnalyzer, os arquivos com os dados de proveniência fornecidos pelos arquivos são alocados em um diretório definido pelo usuário, no sistema de arquivos paralelo. A DfAnalyzer monitora este diretório e inspeciona arquivos em busca de novas informações de proveniência prospectiva ou retrospectiva. A DfAnalyzer e sua base de dados associada também foi instanciado em um nó computacional da mesma forma que a ADfA.

5.3 - Avaliação de Desempenho

Os principais fatores de cujo desempenho afeta a rotina do usuário são o tempo de disponibilidade da proveniência e o tempo de geração de proveniência. Tempo de disponibilidade da proveniência é o tempo necessário para que os dados estejam disponíveis para consulta a partir do momento em que são produzidos pela simulação. Este tempo é importante para que o usuário possa monitorar e interagir com a simulação em tempo real.

Nos testes realizados, o processamento de mensagens é avaliado para cargas de dados com diferentes características como o tamanho em bytes dos elementos de dados, quantidades de elementos de dados por mensagem e taxa de envio de mensagens. O objetivo é identificar quais fatores influenciam mais no tempo de disponibilidade da proveniência.

O tempo de geração de proveniência é a sobrecarga no tempo de processamento causadas pelo código que produz informações de proveniência. Considerar este fator é

necessário já que tempo de processamento de simulações é um fator crítico para a produção de conhecimento científico.

Para realizar a avaliação de desempenho do serviço é necessário coletar os tempos de início e fim em cada etapa do processamento de uma mensagem. Incrementamos a ferramenta com estruturas de dados que armazenam dados de desempenho e métodos que produzem logs que caracterizam o conjunto de mensagens ao longo e o estado do serviço ao longo de seu tempo de atividade.

Um recurso de monitoramento também foi desenvolvido para acompanhar o desempenho da ferramenta durante sua execução. Desta forma, o usuário pode usar ferramentas de análise podem acompanhar o estado do servidor em tempo real. Assim como os outros recursos disponibilizados pela ferramenta, os recursos de monitoramento de performance configuram um recurso REST acessível via URIs.

5.4 - Tempo de disponibilidade - Carga única e Streaming

Neste experimento fixamos uma carga de dados (quantidade de tuplas) e variamos a forma como essa carga é enviada para ver como essa variação afeta o serviço de aquisição e a disponibilidade de dados. Configuramos a aplicação para produzir um conjunto de mensagens características distintas, compatíveis com o volume de dados produzidos em experimentos reais. A Tabela 5.1 mostra as características dos dados enviados para cada um dos experimentos realizados.

Tabela 5.1 - Características dos conjuntos de dados

Processos	Quantidade de Mensagens
12	12
24	24
48	48
96	96
192	192
384	384

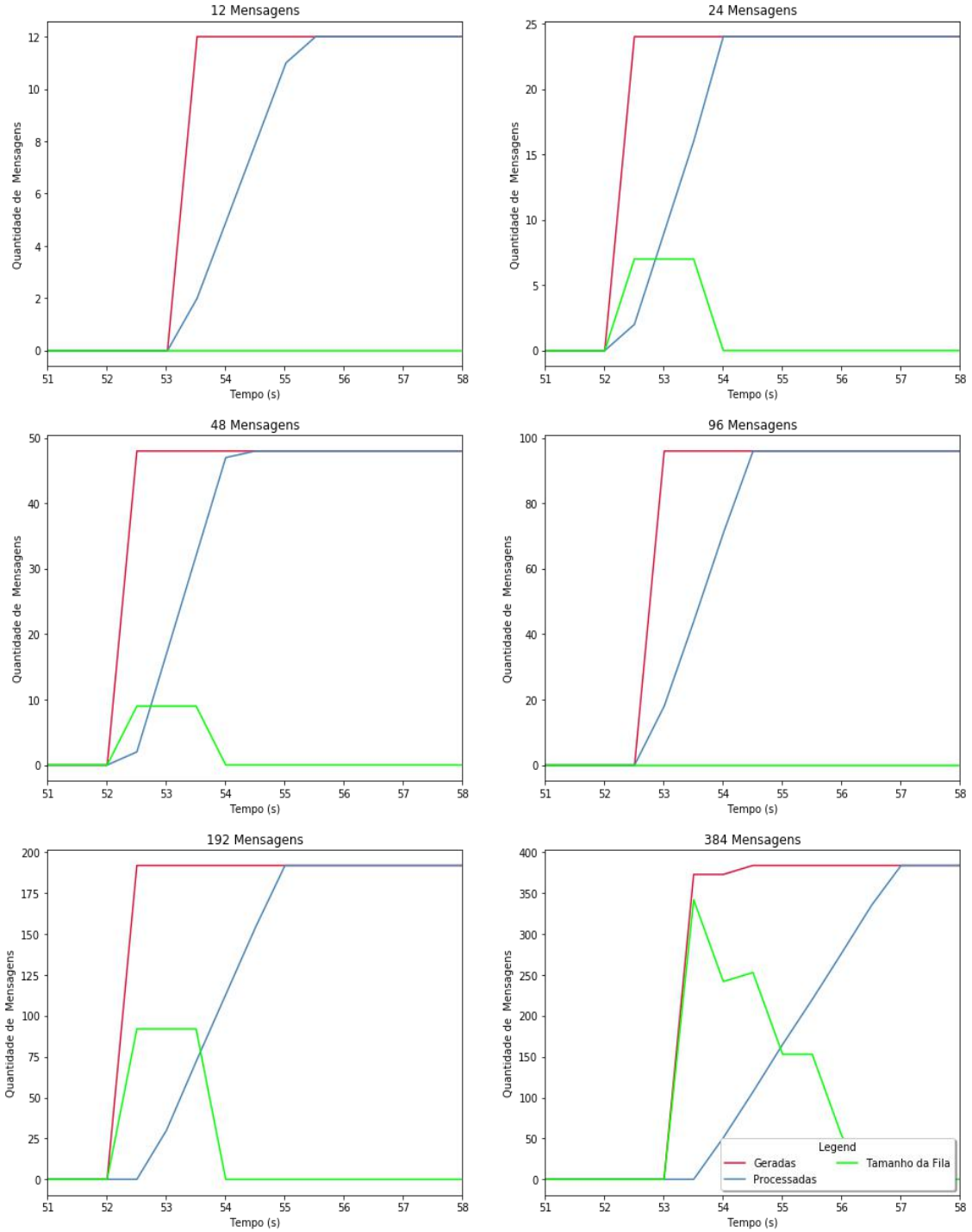

Figura 5.3 - Gráficos do estado do serviço durante a captura de dados

Em todos os experimentos, o programa sintético é invocado de maneira que cada processo MPI seja alocado em uma *thread* do sistema operacional. Assim, cada nó computacional abrigue 48 processos geradores de mensagem. Cada processo envia

apenas uma mensagem contendo 100 elementos de dados. Como a carga de dados é "instantânea" não calculamos a vazão de entrada e o valor do intervalo entre mensagens não é relevante neste experimento.

Conforme pode ser visto os gráficos da Figura 5.3, as mensagens são produzidas num curto intervalo de tempo, de maneira que a fila de proveniência cresce rapidamente em todos os casos. Pode ser visto que a fila diminui conforme os elementos de dados são inseridos na base de proveniência e o gráfico se estabiliza quando toda a carga de dados está inserida na base. Nota-se que o aumento no número de mensagens prolonga o tempo de disponibilidade de proveniência.

A Tabela 5.2 mostra o atraso de disponibilidade da carga total que é a diferença entre o término da recepção de mensagens e a inserção de todos os elementos na base de proveniência. No caso A, não se nota crescimento da fila, isso pode ser explicado considerando que a capacidade de inserção dos dados na base é maior que a velocidade de recepção das mensagens. As mensagens recebidas geram transações que são adicionadas e consumidas da fila de proveniência num tempo muito reduzido.

Tabela 5.2 - Atraso de disponibilidade nos testes

Processos	Quantidade de Mensagens	Atraso de disponibilidade (s)
12	12	1,5
24	24	1,5
48	48	1,8
96	96	2,0
192	192	2,8
384	384	4,5

Um experimento complementar ao caso anterior é o experimento de *streaming (fluxo contínuo*). Este experimento consiste em manter uma vazão de entrada constante ao enviar uma mensagem por processo a cada intervalo de tempo. Nesse experimento, a ADfA recebe mensagens ao longo de um intervalo de tempo o que faz com que a fila de proveniência seja acessada por *threads* para inserção e para consumo de tarefas ao longo

do experimento. No caso anterior, de carga única, a fila era acessada para inserção num curto período de tempo no início do experimento.

Na Figura 5.4, temos o gráfico que mostra o comportamento do serviço para um conjunto de dados com a inserção de 384 elementos de dados por segundo e um total de 38400 elementos de dados. Pode-se observar que a fila cresce lentamente ao longo do tempo de inserção. Isto é, a fila de tarefas pendentes não cresce significativamente ao longo do tempo para cargas de dados com esta natureza. A Figura 5.5 mostra um detalhe do experimento, destacando que a fila de transações pendentes aumenta e diminui em pequenas quantidades conforme novas mensagens são processadas ou transações são consumidas da fila.

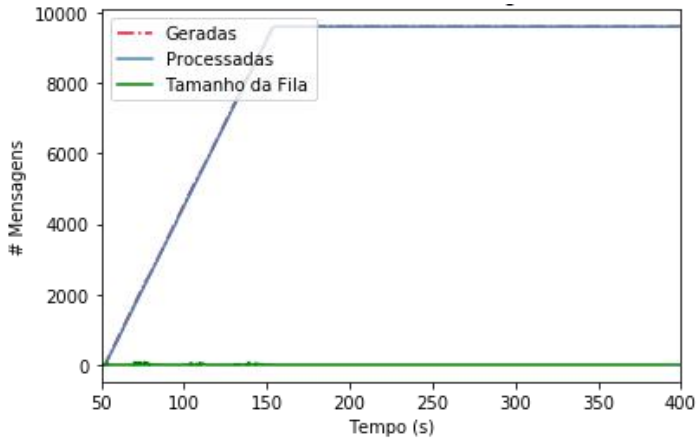

Figura 5.4 - Gráficos do estado do serviço durante a captura de dados em streaming

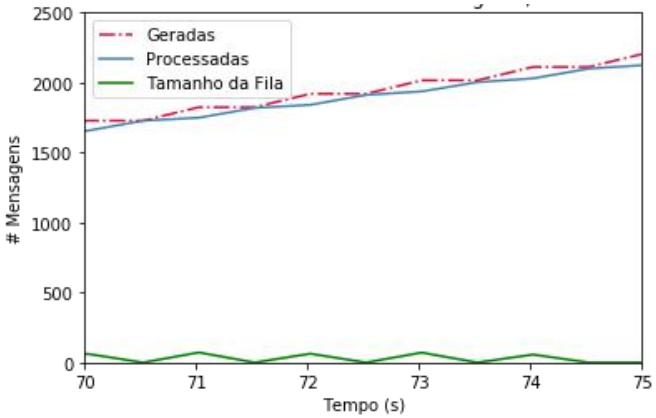

Figura 5.5 – Detalhe do estado do serviço durante a captura de dados em streaming

Por outro lado, em casos em que a quantidade de dados e a vazão são suficientemente altos, o tempo de disponibilidade dos dados pode ser afetado. Por exemplo, se repetirmos o experimento da figura anterior com uma quantidade de dados total maior, a fila de proveniência cresceria o suficiente para afetar o tempo de disponibilidade dos dados. A Figura 5.6 ilustra a inserção de 384000 elementos com uma vazão de 1920 elementos por segundo. Nota-se que a alta vazão de geração de dados provoca o crescimento acelerado da fila de proveniência e o alto tempo para que o número de tarefas processadas alcance seu valor máximo.

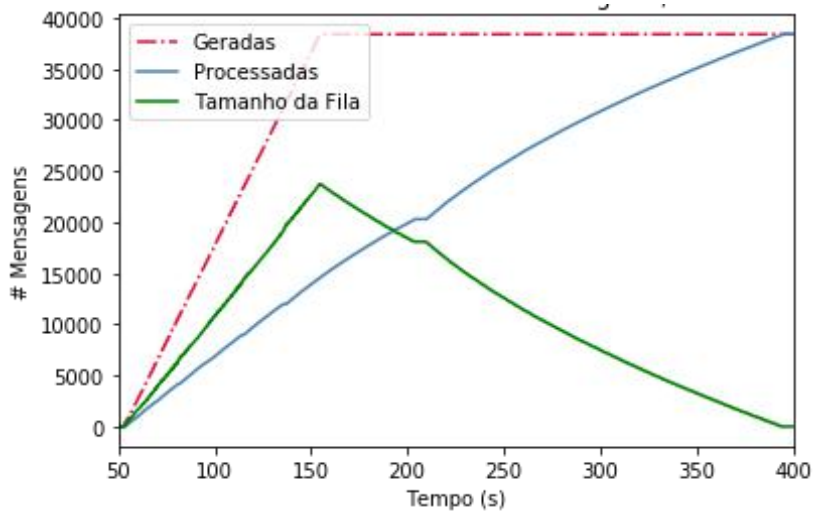

Figura 5.6 - Estado do serviço durante a captura de dados

Neste experimento, vemos que a capacidade de consumo de tarefas é menor que a quantidade de tarefas adquirida num mesmo intervalo de tempo, isso faz com que ao longo do tempo a fila cresça e o tempo de processamento dos dados termine muito após o fim do experimento computacional. Contudo, consideramos este exemplo não realístico, visto que é improvável que uma única simulação gere dados em volume e velocidade tão altos. Por outro lado, se considerarmos o caso de um serviço compartilhado para muitos experimentos ou usuários, é possível que este tipo de situação ocorra e disponibilidade da proveniência fique muito atrasada em relação à execução da simulação.

5.5 - Comparação com a DfAnalyzer

Para termos uma base de comparação, usamos a mesma configuração de experimento para a DfAnalyzer, ferramenta na qual a ADfA é baseada. A DfAnalyzer inspeciona arquivos produzidos por uma API integrada à simulação computacional. A este experimento comparativo é importante pois avalia, em termos de desempenho, a validade das estratégias escolhidas e a validade da implementação realizada. Vale lembrar que as principais contribuições propostas na ferramenta ADfA são: (i) o padrão de comunicação assíncrona e (ii) a flexibilização da especificação do fluxo de dados. O experimento comparativo permite avaliar as duas contribuições em relação ao desempenho computacional.

A adoção da arquitetura REST neste projeto tem, entre seus benefícios, a possibilidade de adotar um padrão de troca de mensagens entre a simulação e a ferramenta de gerência de proveniência. As mensagens seguem o protocolo HTTP e são processados por serviços computacionais específicos de acordo com o endereço fornecido na mensagem.

O experimento comparativo entre as duas tecnologias nos permite avaliar de forma quantitativa o impacto da mudança do padrão de comunicação no fluxo de trabalho do usuário. É esperado que a ferramenta ADfA precise de menos tempo para coleta e disponibilização de proveniência confirmando a vantagem do padrão de comunicação proposto. Espera-se também que a sobrecarga causada na simulação devido aos processos de coleta de proveniência seja menor na ADfA também pelo padrão de comunicação adotado.

A estratégia adotada para flexibilização da especificação de fluxo de dados que tem como principal benefício reduzir o esforço do usuário para projetar a base de dados de proveniência também é avaliada neste experimento comparativo. O algoritmo utilizado na ADfA para processamento de dados de proveniência permite que a base de dados de proveniência seja incrementada dinamicamente conforme o fluxo de dados seja identificado por meio das mensagens produzidas ao longo da execução da simulação. Para que isto seja possível, é necessário inspecionar todos os dados de proveniência e avaliar se existem metadados cuja especificação não está contemplada na base de dados. Para

metadados não contemplados, é necessário incrementar o esquema da base de dados com estruturas que permitam o armazenamento consistente dos dados recém descobertos. Assim, é importante avaliar o impacto da adoção deste novo algoritmo, mais complexo, no tempo de disponibilidade.

Foram feitos estudos comparativos com o fluxo de dados sintético e com a simulação baseada na aplicação libMesh-Sedimentation (CAMATA *et al.*, 2018). Ambos os casos foram definidos na Seção 5.1 - . Para no experimento usando o fluxo de dados sintético, comparamos o tempo de disponibilidade de proveniência para um total dados de proveniência equivalentes a 3840 tuplas nas duas ferramentas.

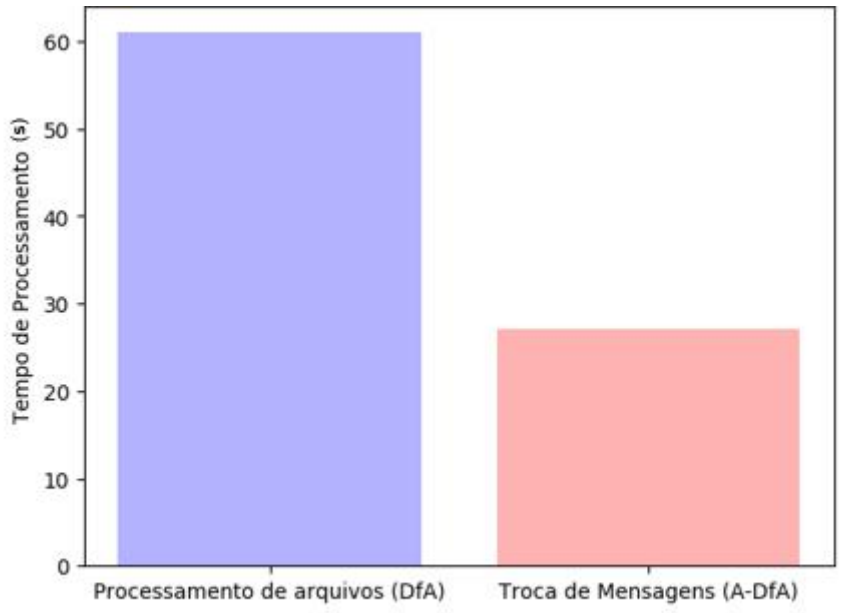

Figura 5.7 – Comparação entre soluções de troca de mensagens e compartilhamento de arquivos

A análise dos resultados mostra que a ADfA precisa de 27 segundos em média para popular a base de dados com informações de proveniência sobre todo o fluxo de dados. A DfAnalyzer, por sua vez, consumiu 61 segundos em média para realizar a mesma tarefa. Foram realizadas 10 execuções do experimento. A Figura 5.7 mostra o tempo de processamento médio de todos os dados gerados pelo fluxo, ou o tempo total de disponibilidade de proveniência para as duas estratégias de comunicação. A ADfA se

mostrou mais eficiente para este caso, consumindo apenas 44% do tempo usado pela DfAnalyzer.

No caso da simulação baseada na aplicação libMesh-Sedimentation, foram avaliados o tempo de processamento da proveniência de toda a simulação de maneira análoga ao experimento com o fluxo de dados sintético. Neste caso, também se observa que a estratégia de troca de mensagens da ADfA tem resultados melhores em termos do tempo de processamento dos dados de proveniência. Para inserir na base de dados as tuplas produzidas a partir da proveniência coletada em todas as transformações da simulação, a ADfA consumiu 19567 segundos, enquanto a DfAnalyzer precisou de 139341 segundos para realizar a mesma tarefa. A estratégia de troca de mensagens se mostrou mais eficiente ao realizar esta carga de trabalho em 0,14% do tempo da estratégia envolvendo arquivos que não é rápida o suficiente para disponibilizar dados de proveniência durante a execução do experimento.

A coleta de dados de proveniência feita por meio da instrumentação da simulação computacional, embora abra oportunidades para diversos benefícios, tem o custo de adicionar uma sobrecarga no tempo de processamento da simulação. O tempo de execução é um fator crítico em simulações computacionais devido ao alto custo e a escassez de recursos de PAD. Por isso, é desejável que a sobrecarga da geração de dados de proveniência seja reduzida.

Para avaliar o impacto da produção de proveniência, executamos a simulação de sedimentação de três formas distintas: sem geração de proveniência; com proveniência gerenciada pela ferramenta DfAnalyzer; com a proveniência gerada e transmitida pela biblioteca de componentes ADfA, na forma de mensagens HTTP. Desta forma, podemos comparar as duas ferramentas com o caso original onde não há sobrecarga da proveniência. A análise de resultados mostrou que a sobrecarga de produzir proveniência seguindo o padrão da ADfA aumenta o tempo de processamento em cerca de 9,8% do tempo original da simulação enquanto a DfAnalyzer adiciona uma sobrecarga de 14,24%.

Capítulo 6 - Conclusão

Essa dissertação foi construída com o objetivo de atender as necessidades de gerência de dados de proveniência no contexto de simulações computacionais em ambiente de PAD. Baseado no estudo das ferramentas existentes, o desempenho da captura de proveniência e a complexidade de configuração das ferramentas foram identificadas como principais oportunidades de contribuição.

Foram propostas estratégias para construção de ferramentas que atendam estas necessidades: a coleta de proveniência assíncrona e a flexibilização da especificação do fluxo de dados. A estratégia de coleta de proveniência assíncrona é uma estratégia baseada em troca de mensagens que objetiva reduzir o tempo necessário para processamento e disponibilização de proveniência. Para implementar esta estratégia, usou-se a arquitetura cliente-servidor e usar mensagens seguindo o protocolo HTTP. A estratégia de flexibilização busca resolver a dificuldade de definição de fluxo de dados sem impactar na qualidade da coleta de proveniência. Sabendo que o ciclo de vida de experimentos científicos computacionais impõe a necessidade de redefinição do fluxo em diversas ferramentas, foi proposta uma definição de fluxo flexível onde a base de dados de proveniência é construída conforme os dados são consumidos pela ferramenta. Desta maneira, o usuário não precisa reespecificar o fluxo de dados a cada iteração do ciclo de vida do experimento de simulação.

As estratégias desenvolvidas foram incorporadas à ferramenta ADfA, que é uma biblioteca de componentes, que estende a DfAnalyzer, para a coleta de dados de proveniência construídos a partir das estratégias propostas neste trabalho. Foram realizados experimentos de validação da ADfA usando a aplicação sintética e uma simulação de sedimentação em um tanque. Esta simulação de sedimentação (CAMATA *et al.*, 2018) é produto de linhas de pesquisa desenvolvidas no NACAD/COPPE/UFRJ.

Os resultados obtidos foram satisfatórios em termos de performance, tanto na aplicação sintética quanto na simulação, validando as estratégias de definição adaptativa de fluxo e coleta de proveniência assíncrona se mostraram mais eficientes que DfAnalyzer em uma análise comparativa. As contribuições em termos de resultados de

desempenho e pelas contribuições qualitativas que diferenciam esta proposta de outras alternativas presentes na literatura.

As decisões de projeto na construção desta ferramenta consideraram a facilidade de integrar melhorias propostas futuras. A adoção da arquitetura web e do padrão de desenvolvimento REST, promove a independência entre os serviços oferecidos por cada recurso. Estas características facilitaram o desenvolvimento de novas aplicações, principalmente de consulta e visualização de dados.

Os experimentos realizados mostraram que a ADfA mantém a disponibilização dados de proveniência para o usuário em uma base de dados relacional durante a execução da simulação provida pela DfAnalyzer, no entanto provê a coleta e o armazenamento de modo bem mais eficiente. Esta funcionalidade abre a oportunidade de desenvolvimento de ferramentas que façam o monitoramento e facilitem a interferência do usuário no fluxo de simulação. Esta oportunidade é também mais facilitada pela arquitetura de serviços inspirada em serviços web da ADfA, que facilita a alocação do serviço como um recurso acessível por um navegador. De fato, estratégias propostas nesta dissertação foram incorporadas à versão final da DfAnalyzer (SILVA *et al.*, 2018).

Outra demonstração da viabilidade deste uso da ADfA é apresentada por PINA (2018) que propõe uma interface web para execução de consultas em fluxos de dados de simulações computacionais. Embora a interface em PINA (2018) atenda necessidades gerais de consulta, monitoramentos e visualizações específicas para cada simulação, sua substituição por estratégias da ADfA podem melhorar ainda mais esses recursos. Interfaces web também podem ser um mecanismo para facilitar a intervenção no fluxo do experimento quando necessário. Contudo, para esta última possibilidade, seria necessário propor componentes arquiteturais capazes de ativamente modificar o comportamento da simulação (SOUZA *et al.*, 2017). Os componentes propostos neste trabalho se restringem a recursos de coleta de dados e monitoramento e por isto não abordaram intervenções na solução.

Um aspecto que pode ser explorado é o protocolo de troca de mensagens. A ADfA foi desenvolvida a partir de um protocolo de troca de mensagens definido neste trabalho, a partir de tecnologias de desenvolvimento de software bem estabelecidas. Para troca de mensagens, foram especialmente relevantes o protocolo HTTP e o formato de transmissão

de dados JSON. Contudo, o esquema de troca de mensagens proposto não define comportamentos para garantia de qualidade de serviço. A qualidade do serviço pode ser garantida com estratégias para comportamentos inesperados como por exemplo, um protocolo de tratamento de falhas.

Uma arquitetura de componentes independentes como a que é proposta neste trabalho é capaz de continuar funcionando mesmo que alguns de seus componentes não estejam acessíveis durante um intervalo de tempo. Assim, a ADfA pode ser incrementado com estratégias que aumentem a resiliência do serviço evitando a perda de dados de execução da simulação.

A arquitetura de troca de mensagens e recursos REST abre a oportunidade de explorar o potencial da ADfA como um serviço para múltiplos usuários. Para isto, seria necessário lidar com aspectos ainda mais complexos de qualidade de serviço, visto que com uma maior carga de usuários, a possibilidade de sobrecarga de tarefas em algum componente da ferramenta é mais provável. Uma solução natural para esta questão seria a implementação distribuída da ADfA para que a carga de trabalho possa ser dividida entre vários nós computacionais. As tecnologias MonetDB e SpringBoot contam com recursos de distribuição e paralelismo visando atender cargas de trabalho de grande porte. Por outro lado, nesta implementação paralela, seria necessário propor algoritmos que adaptem o comportamento da ferramenta para ambiente distribuído. Por exemplo, a implementação dos acessos de leitura e escrita na fila de tarefas de proveniência ou a distribuição de novas mensagens entre diversos recursos computacionais em ambiente distribuído.

Referências Bibliográficas

ABADI, D.J., 2007, "Column Stores for Wide and Sparse Data.". In: *Cidr*. pp. 292–297.

ALAGIANNIS, I., BOROVICA-GAJIC, R., BRANCO, M., et al., 2015, "NoDB". In: *Communications of the ACM*. v. 58, pp. 112–121.

APACHE SOFTWARE FOUNDATION, 2017. APACHE TOMCAT. Disponível em: <http://tomcat.apache.org/>. Acessado em: 2 Setembro 2018.

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

BAUMAN, P.T., STOGNER, R.H., 2015, "GRINS: A Multiphysics Framework Based on the libMesh Finite Element Library". In: *SIAM Journal on Scientific Computing*. v. 4400, pp. 1–21.

CAMATA, J.J., SILVA, V., VALDURIEZ, P., et al., 2018, "In situ visualization and data analysis for turbidity currents simulation". In: *Computers and Geosciences*. v. 110, pp. 23–31.

DA CRUZ, S.M.S., CAMPOS, M.L.M., MATTOSO, M., 2009. "Towards a taxonomy of provenance in Scientific Workflow Management Systems". In: *SERVICES 2009 - 5th 2009 World Congress on Services*. S.l.: IEEE. Julho 2009. pp. 259–266.

DEELMAN, E., GANNON, D., SHIELDS, M., et al., 2009, "Workflows and e-Science: An overview of workflow system features and capabilities". In: *Future Generation Computer Systems*. v. 25, pp. 528–540.

DEELMAN, E., MEHTA, G., SINGH, G., et al., 2007. "Pegasus: Mapping large-scale workflows to distributed resources". In: TAYLOR, Ian J, DEELMAN, Ewa, GANNON, Dennis B & SHIELDS, Matthew (orgs.), *Workflows for e-Science: Scientific Workflows for Grids*. London: Springer London. pp. 376–394.

EICHINSKI, P., ROE, P., 2016. "Datatrack: An R package for managing data in a multi-stage experimental workflow data versioning and provenance considerations in interactive scripting". In: *2016 IEEE 12th International Conference on e-Science (e-Science)*. S.l.: IEEE. Outubro 2016. pp. 147–154.

FREIRE, J., KOOP, D., SANTOS, E., et al., 2008, "Provenance for computational tasks: A survey". In: *Computing in Science and Engineering*. v. 10, pp. 11–21.

GREISEN, E.W., CALABRETTA, M.R., 2002, "Representations of world coordinates in FITS". In: *Astronomy & Astrophysics*. v. 395, pp. 1061–1075.

GUERRA, G.M., ZIO, S., CAMATA, J.J., et al., 2013, "Numerical simulation of particle-laden flows by the residual-based variational multiscale method". In: *International Journal for Numerical Methods in Fluids*. v. 73, pp. 729–749.

HENDRIX, V., FOX, J., GHOSHAL, D., et al., 2016, "Tigres Workflow Library: Supporting Scientific Pipelines on HPC Systems". In: *Proceedings - 2016 16th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing,*

CCGrid 2016. pp. 146–155.

JAIN, S., MORITZ, D., HALPERIN, D., et al., 2016, "SQLShare". In: *Proceedings of the 2016 International Conference on Management of Data - SIGMOD '16*. pp. 281–293.

JARMAN, P., 2015, "Microservices – A New Application Paradigm". In: *Infosys*. pp. 8.

JARRARD, R.D., 2001. "Scientific methods: an online book". Acessado em: 12 Agosto 2018.

KARPATHIOTAKIS, M., BRANCO, M., ALAGIANNIS, I., et al., 2014, "Adaptive query processing on RAW data". In: *Proceedings of the VLDB Endowment*. v. 7, pp. 1119–1130.

LERNER, B.S., BOOSE, E.R., 2014, "RDataTracker: Collecting Provenance in an Interactive Scripting Environment". In: *Proceedings of TAPP 2014*. pp. 1–4.

MA, B., SHOSHANI, A., SIM, A., et al., 2012. "Efficient attribute-based data access in astronomy analysis". In: *Proceedings - 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, SCC 2012*. S.l.: s.n. 2012. pp. 562–571.

MATTOSO, M., WERNER, C., TRAVASSOS, G.H., et al., 2010, "Towards supporting the life cycle of large scale scientific experiments". In: *International Journal of Business Process Integration and Management*. v. 5, pp. 79.

MCPHILLIPS, T., SONG, T., KOLISNIK, T., et al., 2015, "YesWorkflow: A User-Oriented, Language-Independent Tool for Recovering Workflow Information from Scripts". In: . v. 10, pp. 298–313.

MOREAU, L., GROTH, P., 2013, "Provenance: An Introduction to PROV". In: *Synthesis Lectures on the Semantic Web: Theory and Technology*. v. 3, pp. 1–129.

MORELAND, K., HERELD, M., PAPKA, M.E., et al., 2011. "Examples of in transit visualization". In: *Proceedings of the 2nd international workshop on Petascal data analytics: challenges and opportunities - PDAC '11*. New York, New York, USA: ACM Press. 2011. pp. 1.

OLIVEIRA, D., OGASAWARA, E., BAIÃO, F., et al., 2010, "SciCumulus: A lightweigh cloud middleware to explore many task computing paradigm in scientific workflows". In: *Proceedings - 2010 IEEE 3rd International Conference on Cloud Computing, CLOUD 2010*. pp. 378–385.

PANG, Z., WU, S., CHEN, G., et al., 2017, "FlashView: An Interactive Visual Explorer for Raw Data". In: . v. 10, pp. 1869–1872.

PIMENTEL, J.F., MURTA, L., BRAGANHOLO, V., et al., 2017, "noWorkflow". In: *Proceedings of the VLDB Endowment*. v. 10, pp. 1841–1844.

PINA, D.B., 2018. *Uma Interface Para a Análise De Fluxo De Dados Em Simulações Computacionais Intensivas Em Dados*. Projeto de Graduação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Março.

PIVOTAL SOFTWARE, Inc., 2015. "Spring Boot". Disponível em:

<http://projects.spring.io/spring-boot/>. Acessado em: 2 Setembro 2018.

RODRIGUEZ, A., 2008, "Restful web services: The basics". In: *Online article in IBM DeveloperWorks Technical Library*. pp. 1–11.

DE ROOIJ, F., DALZIEL, S.B., 2001. "Time- and space-resolved measurements of deposition under turbidity currents". In: *Particulate Gravity Currents*. Oxford, UK: Blackwell Publishing Ltd. pp. 207–215.

RÜDE, U., WILLCOX, K., MCINNES, L.C., et al., 2016. *Research and Education in Computational Science and Engineering*. S.l.

SILVA, V., 2018. *Análise De Dados Científicos Sobre Múltiplas Fontes De Dados Ao Longo Da Execução De Simulações Computacionais*. Tese de Doutorado, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Junho.

SILVA, V., LEITE, J., CAMATA, J.J., et al., 2017, "Raw data queries during data-intensive parallel workflow execution". In: *Future Generation Computer Systems*. v. 75, pp. 402–422.

SILVA, V., OLIVEIRA, D. DE, VALDURIEZ, P., et al., 2016a. "DfAnalyzer Tool" Disponível em: <https://hpcdb.github.io/armful/dfanalyzer.html>.

SILVA, V., OLIVEIRA, D., VALDURIEZ, P., et al., 2016b, "Analyzing related raw data files through dataflows". In: *Concurrency Computation*. v. 28, pp. 2528–2545.

SILVA, V., OLIVEIRA, D., VALDURIEZ, P., et al., 2018, "DfAnalyzer: Runtime Dataflow Analysis of Scientific Applications using Provenance". In: . v. 11, pp. 2082–2085.

SOANES, C., STEVENSON, A., 2003, *Oxford dictionary of English.* . S.l., Oxford University Press.

SOUZA, R., SILVA, V., LGA COUTINHO, A., et al., 2017, "Data reduction in scientific workflows using provenance monitoring and user steering". In: *Future Generation Computer Systems*.

THE HDF GROUP, 1997. "Hierarchical Data Format, version 5".

TRAVASSOS, G.H., BARROS, M., 2003. "Contributions of In Virtuo and In Silico Experimentes for the Future of Empirical Studies in Software Engineering". In: *2nd Workshop in Workshop Series on Empirical Software Engineering The Future of Empirical Studies in Software Engineering*. S.l.: s.n. 2003. pp. 1–14.

UNIDATA, 2016. UCAR COMMUNITY PROGRAMS. Disponível em: <http://www.unidata.ucar.edu/software/netcdf/>.