UNIVERSIDADE FEDERAL FLUMINENSE

THAYLON GUEDES SANTOS

Análise e Captura de Dados de Proveniência no Apache Spark

NITERÓI

2018

UNIVERSIDADE FEDERAL FLUMINENSE

THAYLON GUEDES SANTOS

Análise e Captura de Dados de Proveniência no Apache Spark

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Engenharia de Sistemas e Informação.

Orientador:
Prof. D.Sc. DANIEL CARDOSO MORAES DE OLIVEIRA

Co-orientador:
Prof. D.Sc. MARCOS VINÍCIUS NAVES BÊDO

NITERÓI

2018

Ficha catalográfica automática - SDC/BEE

G924a Guedes Santos, Thaylon
Análise e Captura de Dados de Proveniência no Apache Spark / Thaylon Guedes Santos ; Daniel Cardoso Moraes de Oliveira, orientador ; Marcos Vinicius Naves Bêdo, coorientador. Niterói, 2018.
82 f. : il.

Dissertação (mestrado)-Universidade Federal Fluminense, Niterói, 2018.

DOI: http://dx.doi.org/10.22409/PGC.2018.m.05122782377

1. Sistemas paralelos e distribuído. 2. Fluxo de trabalho. 3. Sistema de computador. 4. Produção intelectual. I. Título II. Cardoso Moraes de Oliveira,Daniel, orientador. III. Naves Bêdo, Marcos Vinicius, coorientador. IV. Universidade Federal Fluminense. Escola de Engenharia.

CDD -

Bibliotecária responsável: Fabiana Menezes Santos da Silva - CRB7/5274

THAYLON GUEDES SANTOS

Análise e Captura de Dados de Proveniência no Apache Spark

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Engenharia de Sistemas e Informação.

Aprovada em Setembro de 2018.

BANCA EXAMINADORA

Prof. D.Sc. Daniel Cardoso Moraes de Oliveira - Orientador, IC/UFF

Prof. D.Sc. Marcos Vinícius Naves Bêdo - Co-Orientador, INFES/UFF

Prof. D.Sc. Leonardo Gresta Paulino Murta, IC/UFF

Prof. D.Sc. Maria Claudia Reis Cavalcanti, IME

Prof. D.Sc. Marta Lima de Queiros Mattoso, COPPE/UFRJ

Prof. D.Sc. Ary Henrique Morais de Oliveira, UFT

Niterói

2018

"O sucesso nasce do querer, da determinação e persistência em se chegar a um objetivo. Mesmo não atingindo o alvo, quem busca e vence obstáculos, no mínimo fará coisas admiráveis."(José de Alencar)

Agradecimentos

Aos meus pais, Merisvaldo Chaves e Iracilda Guedes, por todo apoio e carinho que recebi, por sempre prezarem pela minha educação, por serem meu exemplo de vida e por me ensinarem os valores que cultivo como pessoa. Aos meus irmãos, Ariadne Guedes e Hiago Guedes, por suas amizades e companheirismo ao decorrer dos anos.

Ao Prof. Daniel de Oliveira, meu orientador, pelos seus ensinamentos, orientações, e conselhos e plena disponibilidade (mesmo em horários não-convencionais). Também ao Prof. Marcos Bedo, meu coorientador, pelas orientações que foram muito importantes para o desenvolvimento desta dissertação.

À Profa. Marta Mattoso, por sua importante participação nos trabalhos que eu desenvolvi e por sempre realizar uma excelente revisão dos artigos. Ao Prof. Ary Henrique, meu orientador da graduação, por ser um dos meus primeiros professores a ter acreditado no meu potencial e ter me ajudado a explorar minhas habilidades.

Ao Vítor Silva, pela parceria nos trabalhos que desenvolvi durante o mestrado e por todas as contribuições que impactaram diretamente nesta dissertação. Ao Eduardo Smil, colega com quem fiz os trabalhos das diversas disciplinas que cursamos juntos no mestrado.

Aos professores Leonardo Murta e Maria Cavalcanti por terem aceitado participar da banca da defesa da minha dissertação de mestrado.

São com essas poucas palavras que quero expressar a minha grande gratidão a vocês. Por terem participado da minha vida acadêmica e terem deixado conhecimentos que eu sempre irei utilizar.

Por fim, gostaria de agradecer à CAPES pela bolsa concedida.

Resumo

Experimentos científicos de longa duração e que fazem o uso intensivo de dados têm se tornado cada vez mais comuns. Esses experimentos precisam de métodos e ferramentas para agilizar as suas execuções e também para analisar os dados produzidos da forma mais eficiente e transparente possível. Nesse contexto, um *framework* de processamento distribuído que tem ganhado notoriedade é o Apache Spark, capaz de reduzir o gargalo de E/S encontrado em outros competidores do paradigma MapReduce, como o Apache Hadoop. Contudo, o Apache Spark deixa a desejar quanto ao apoio à captura e gerenciamento de proveniência, elemento fundamental para reproduzir e avaliar a qualidade de resultados no contexto de experimentos científicos. O monitoramento padrão do Apache Spark informa somente o *status* das execuções e não permite acessar os dados intermediários produzidos pelos experimentos. Além disso, o Apache Spark também não apoia a execução de aplicações "caixa-preta" externas que se comunicam por meio de arquivos, no sentido de que o *framework* ignora o conteúdo consumido e produzido por essas aplicações. Dessa forma, o cientista precisa esperar até o final da execução para pesquisar os arquivos gerados pelos seus experimentos e concluir algo sobre os resultados. Esta dissertação apresenta uma solução para estender o Apache Spark e preencher essa lacuna da literatura. Nossa proposta, denominada `SAMbA`, é focada na captura e recuperação de dados de proveniência prospectiva, retrospectiva e de domínio. O `SAMbA` captura automaticamente dados de proveniência prospectiva e retrospectiva, além de permitir que os cientistas selecionem os atributos de interesse de dados manipulados por aplicações Apache Spark (ou nativas na linguagem Scala ou programas "caixa-preta" externos) que são persistidos na base de proveniência como dados de domínio. A solução também trata da otimização de aplicações "caixa-preta" no Apache Spark ao *(i)* disponibilizar o módulo `SAMbA-FS`, que atua como um sistema de arquivos em memória principal e *(ii)* usar a ferramenta Git, para armazenar os dados intermediários coletados do `SAMbA-FS`. Portanto, o cientista se encontra habilitado a consultar informações dos experimentos ao acessar a base de dados de proveniência, seja por meio de relatórios em tempo de execução (em uma interface *web*) ou por buscas *post-mortem* efetuadas por meio da linguagem SQL. Os testes realizados com o `SAMbA` sobre um estudo de caso real mostram que a solução é capaz de gerenciar dados de proveniência sem adicionar custos significativos ao tempo de execução de experimentos científicos. Por último, mas não menos importante, discutimos formas alternativas de se consultar dados de domínio produzidos por experimentos científicos, sempre que o usuário optar por não gerenciá-los diretamente no `SAMbA`. Comparamos duas abordagens estado da arte para buscas em arquivos sobre dados de domínio brutos, a saber, *(i)* PostgresRaw, uma ferramenta para consultas adaptativas, e *(ii)* FastBit, uma solução baseada em indexação de arquivos, considerando um estudo de caso com múltiplos arquivos de grande cardinalidade. Os resultados indicaram que consultas adaptativas são as mais adequadas para reduzir o tempo-para-consulta nesse contexto.

Palavras-chave: Proveniência, Spark, Consultas Adaptativas, *Workflows* Científicos.

Abstract

Long lasting and data-intensive scientific experiments are increasingly drawing attention. Such experiments require methods and tools to *(i)* speed up their execution, and *(ii)* analyze the produced data in the most efficiently and transparently way possible. Within this context, Apache Spark is a parallel processing framework that gained notoriety by reducing the I/O bottleneck of other competing frameworks of MapReduce paradigm, *e.g.*, Apache Hadoop. Nevertheless, Apache Spark has limitations on supporting capture and management of provenance data, which is a transparent approach for the replication, reproducibility and quality evaluation of scientific experiments results. Standard Apache Spark monitoring tool only reports execution status so that it does enable the access and visualization of intermediate data produced in the experiments. Additionally, Apache Spark struggles in providing provenance support for the execution of external black-box applications that communicate through files, *i.e.*, the framework ignores the content consumed and produced by those applications. As a result, scientists are expected to wait until the end of the experiment execution for searching the files generated by the experiments and conclude something about the results. In this study, we tackle the lack of provenance support on Apache Spark and reports on a seamless extension of Apache Spark. Our proposal, called `SAMbA`, is focused on the capture and recovery of prospective and retrospective provenance as well as domain data. `SAMbA` automatically captures prospective and retrospective provenance information and also enables scientists to select their attributes of interest from data managed by Apache Spark (both native Scala language and external black-box applications) that are be persisted on the provenance database as domain data. Our solution addresses the optimization of black-box applications in Apache Spark by *(i)* providing `SAMbA-FS` module, which acts as an in-memory filesystem, and *(ii)* using the Git tool for the storage of intermediate data collected from `SAMbA-FS`. Accordingly, scientists become capable of consulting intermediate data and information about the experiments by accessing the provenance database, which can be achieved by either a *web* interface or *post-mortem* queries in SQL statements. We experiment with the `SAMbA` on a real case study and results indicate our solution is capable of managing the provenance data without adding significant costs to the execution time of scientific experiments. Last, but not least, we discuss alternative strategies for querying domain data produced by scientific experiments whenever users choose to not manage them directly on `SAMbA`. We compared two state-of-the-art approaches for querying raw domain data files, namely, *(i)* PostgresRaw, a strategy based on adaptive querying, and *(ii)* FastBit, a solution based on bitmap indexing, on a case study with multiple and large files. Experimental evaluations indicated adaptive querying is the most suitable strategy for reducing time-to-query.

Keywords: Provenance Data, Spark, Adaptive Queries, Scientific *Workflows*.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Nas últimas décadas, a computação se consolidou como parte integrante da pesquisa e metodologia científica unindo-se aos paradigmas tradicionais de teoria e experimentação [25, 34]. Diversos experimentos dessa nova geração de experimentação científica apoiada por simulações computacionais, ou experimentos *in silico*, são conhecidos por serem computacionalmente dispendiosas, e são caracterizados por sua *(i)* complexidade de execução, *(ii)* manipulação de grandes quantidades de dados e *(iii)* utilização intensiva de recursos computacionais [17, 33, 67]. A maior parte dessa nova categoria de experimentos científicos é composta de vários procedimentos lógicos diferentes, códigos e rotinas computacionais que realizam diversas operações sobre os dados a serem analisados, tais como carga (*data loading*), processamento (*data processing*) e agregação (*data aggregation*).

Esses experimentos podem ser representados por meio de fluxos de dados (*i.e., data-flows)* que consistem em uma cadeia de transformações sobre os dados. Nesse contexto, cada transformação consome um conjunto de dados de entrada e produz um conjunto de dados como saída, que, por sua vez, pode ser consumido por outras transformações até que a cadeia seja totalmente executada [37, 62]. Ainda que seja possível modelar a execução de um fluxo de dados por uma sequência de comandos (*i.e., scripts*), o emprego desses artefatos em experimentos com uso intensivo de dados pode não ser interessante por duas razões discutidas na sequência.

Primeiro, normalmente, *scripts* não são escaláveis, uma vez que eles são desenvolvidos para serem executados em uma única máquina, salvo exceções específicas, como no caso do Swift/T [71]. Segundo, o método experimental preza pela *reprodutibilidade* e *rastreabilidade* dos resultados, os quais acabam "borrados" devido ao fato que nos *scripts* ficam à cargo dos cientistas que os desenvolvem capturar tais informações em aplicações, ambientes de programação e sistemas operacionais específicos. Apesar de existirem soluções

inovadoras que permitem a captura automática de dados de proveniência para garantir a reprodutibilidade em *scripts*, como o *noWorkflow* [55], elas ainda não são preparadas para lidar com *scripts* que são executados de forma distribuída.

Com o objetivo de auxiliar a execução de experimentos científicos computacionalmente intensivos, diversas ferramentas foram projetadas para dar apoio à sua execução de forma paralela e distribuída em ambientes de computação de alto desempenho (*High Performance Computing* – HPC), tais como *clusters* e nuvens computacionais [39]. Algumas dessas ferramentas são focadas na modelagem e automatização da execução de experimentos científicos que podem ser compostos por programas de diferentes domínios, como os Sistemas de Gerência de *Workflows* Científicos (SGWfC) que são adequados para modelar e coordenar a execução de fluxos de dados [25, 29, 47]. Exemplos de SGWfC são o Swift/T [71], o Pegasus [26], o Kepler [8] e o SciCumulus [23]. Apesar de eficazes, tais SGWfCs requerem uma grande curva de aprendizado para serem usados em larga escala, *i.e.* dependendo do tipo de processamento a ser realizado, tanto a etapa de configuração quanto o uso do SGWfC pode não ser trivial [55].

Uma opção aos SGWfC existentes são as ferramentas para Computação Escalável e Intensiva em Dados (*Data-Intensive Scalable Computing* – DISC), que fornecem modelos de programação na qual os usuários desenvolvem a lógica para o processamento dos dados [14, 17]. Os modelos construídos são compilados durante a sua execução na forma de um grafo acíclico direcionado (*Directed Acyclic Graph* – DAG) constituído por operadores de dados paralelos [38]. Portanto, é possível realizar otimizações durante o escalonamento das execuções das atividades de processamento de dados [60]. A maioria das ferramentas DISC implementa o paradigma de programação *MapReduce*, que é inspirado em duas funções do modelo de programação funcional: o *Map* e o *Reduce* [24, 61, 69, 74]. Alguns exemplos de ambientes DISC são o Apache Hadoop [61] e o Apache Spark [74].

Outro aspecto a ser considerado por ferramentas DISC são as operações de leitura e escrita em disco, que são alguns dos procedimentos experimentais computacionalmente mais onerosos. Assim, experimentos que utilizam muito esse recurso tendem a ter o seu tempo de execução penalizado [16]. O uso do Apache Spark é particularmente interessante nesse contexto, uma vez que, ao contrário de outras alternativas, esse *framework* é o primeiro que utiliza a memória principal também como unidade de armazenamento para tratar os resultados intermediários. Para modelar um experimento, a ferramenta dispõe de um modelo de programação baseado em uma abstração de dados denominada Conjunto de Dados Distribuídos Resilientes (*Resilient Distributed Datasets* – RDD).

Através dessa abstração, o Apache Spark consegue fornecer uma representação da memória principal de várias máquinas como uma única "memória compartilhada", onde a lógica desenvolvida pelos cientistas é aplicada sobre essa estrutura [73]. No Apache Spark, cada RDD representa um conjunto de dados imutável, onde é possível aplicar tanto transformações, como *map*, *reduce* e *filter*, o que, consequentemente, gera outros RDDs; quantos ações, como *count* e *collect* sobre o conjunto de dados abstraído [38,73,74]. Vale ressaltar que as transformações são operações aplicadas apenas sob demanda, o que contribui para um melhor aproveitamento de todos os recursos computacionais [74]. Além disso, o Apache Spark também tem outras vantagens sobre os SGWfC. Por exemplo, o uso da "localidade de dados" permite que os dados sejam transferidos para outros nós de processamento apenas quando necessário, o que minimiza o tráfego de rede [73, 74].

Finalmente, é de crucial importância identificar e armazenar os chamados dados de *proveniência* de experimentos científicos, o que inclui os dados que foram usados e gerados durante a execução de testes, informações de como os dados foram manipulados, as aplicações utilizadas nas transformações, as variáveis de ambiente e as características de *hardware* [22,29,55]. Com o uso destas informações é possível avaliar a qualidade, confiabilidade e reprodutibilidade dos resultados obtidos após a execução dos experimentos [53]. Além disso, dados de proveniência permitem que cientistas possam comparar os resultados obtidos com execuções anteriores e encontrar pontos de falha ou pontos que podem ser usados como novos inícios para reexecução [48, 50]. Contudo, é importante ressaltar também que o tratamento de dados de proveniência traz diversos desafios para o contexto de ambientes DISC [5]. Um dos maiores desafios é definir quais dados de proveniência serão capturados, pois armazenar todos os elementos de dados consumidos e produzidos durante a execução de um experimento pode ser proibitivamente caro em termos de armazenamento, ou até impraticável, dependendo do volume e complexidade [5, 68].

SGWfC normalmente já implementam métodos para a captura de dados de proveniência, os quais são coletados em diversos níveis de granularidade [21, 23]. Por outro lado, embora as abordagens DISC forneçam um *framework* com boa eficiência computacional para execução de experimentos científicos de alto desempenho, elas ainda não possuem um suporte à proveniência tão sólido quanto os SGWfC. Por exemplo, no caso do Apache Spark, o "tratamento" de proveniência se resume ao registro de um conjunto de informações armazenadas em *log*, além disso, ele também não permite capturar informações sobre os dados científicos produzidos ao longo do tempo [32, 38]. Desta forma, os cientistas têm que esperar até o fim do processamento para que seja possível extrair algum conhecimento sobre uma determinada execução.

Nesse trabalho, buscamos atacar essa lacuna existente na literatura e estudamos estratégias para dotar o Apache Spark com apoio à proveniência tal como o que já ocorre nos SGWfC [21, 62]. Em particular, estudamos os tipos de dados de proveniência a serem tratados, quais componentes e fases de execução do Apache Spark que eles estão ligados e a forma mais adequada de capturá-los e armazená-los.

1.1 Motivação

Considerando o crescente número de experimentos que manipulam grandes quantidades de dados, torna-se importante a utilização de ferramentas que otimizem a utilização dos recursos computacionais em ambientes de HPC. Embora o Apache Spark atenda com proficiência esse requisito, o *framework* apresenta limitações no tocante ao apoio do ciclo de vida dos experimentos científicos [51]. Dentre essas limitações destacam-se:

1. Falta de apoio à captura de dados de proveniência. Dados de proveniência são fundamentais para registrar a linhagem da geração de dados e facilitam a reprodutibilidade e análise de dados produzidos por experimentos científicos [29, 64].

2. Falta de acesso aos dados em memória. Os cientistas não podem analisar os resultados intermediários em tempo de execução e, consequentemente, orientar seus experimentos, *e.g.*, parar a execução ou modificar algum parâmetro.

3. O Spark não foi projetado para invocar aplicativos de "caixa-preta". Este cenário, comum em experimentos científicos onde as aplicações leem e gravam dados em arquivos externos, não têm apoio nativo da ferramenta em termos de uma integração otimizada com as abstrações de RDDs. Embora seja possível mover dados de, e para, a memória secundária no Apache Spark, esse fluxo de ações adiciona uma sobrecarga não desprezível ao experimento. Em um cenário com apoio completo à proveniência, o conteúdo dos arquivos deveria ser carregado em memória principal, enquanto que um subconjunto do conteúdo dos arquivos poderia ser extraído e armazenado juntamente com outros dados de proveniência.

4. O Spark não faz o controle de versão dos dados manipulados. Uma vez que os dados são mantidos em memória principal e o Apache Spark não dá suporte a aplicações "caixa-preta" que manipulam arquivos, controlar a versão dos dados manipulados não é possível. Entretanto, em um *workflow* que empregue aplicações "caixa-preta",

o controle de versão dos arquivos manipulados por essas chamadas externas enriquece a análise final da execução dos experimentos, pois permite comparar arquivos entre diversas execuções.

Além dessas limitações, também é importante levar em consideração um segundo momento no qual os usuários do Apache Spark queiram consultar os dados dos experimentos científicos após a sua execução. Nesse contexto, o tratamento de experimentos científicos no Apache Spark com apoio à proveniência também inclui desafios relacionados a consultas sobre os dados oriundos do experimento, a saber:

1. A forma mais adequada de indexar dados produzidos por experimentos científicos. Dados oriundos de experimentos científicos são usualmente salvos em arquivos brutos, sendo que índices específicos podem ser construídos para consultar essas estruturas. Portanto, faz-se necessário avaliar o desempenho das abordagens mais representativas para a indexação desse tipo de dados.

2. A forma mais eficiente para consultar dados de experimentos científicos. Junto aos métodos de indexação, consultas sobre dados oriundos de experimentos científicos em seu formato original podem ser realizadas segundo diferentes paradigmas, tais como consultas *in-situ* [42] e diretamente sobre os índices. Assim, faz-se necessário uma discussão experimental sobre a forma computacionalmente mais eficiente para consultar dados de experimentos científicos.

1.2 Contribuições

O presente trabalho apresenta um estudo sobre o tratamento adequado de dados de proveniência e de domínio no ambiente Apache Spark. Em particular, apresenta-se uma categorização de tipos de dados de proveniência a serem tratados junto a uma extensão para o Apache Spark considerando a coleta, armazenamento e consulta desses tipos de dados. Portanto, com o uso da extensão proposta torna-se possível *(i)* coletar dados de proveniência a partir de RDDs, *(ii)* fornecer recursos analíticos em tempo de execução de experimentos e *(iii)* executar aplicações "caixa-preta" que manipulam arquivos.

Outro ponto debatido nesse trabalho é com relação às duas formas de consultar dados de proveniência, a saber *(i)* buscas em tempos de execução e *(ii)* buscas *post-mortem*, que requerem que os dados de proveniência estejam armazenados em memória secundária. Nesse sentido, além do tratamento de proveniência no Apache Spark, esse trabalho

também contribui com o estado da arte ao fornecer uma análise experimental sobre a estratégia de armazenamento em memória secundária mais adequada para a execução de consultas sobre dados de domínio.

Em resumo, as contribuições deste trabalho são:

- Uma abordagem para captura de dados de proveniência, incluindo dados de domínio, no Apache Spark.
- Uma comparação quantitativa de métodos para armazenamento e recuperação de dados de proveniência e de domínio.
- Um sistema de arquivos em memória, que permite executar aplicações "caixa-preta" que manipulem arquivos que estão persistidos em RDDs.
- Uma forma interativa de explorar o fluxo de dados durante, e após, o seu registro.
- Uma avaliação quantitativa de experimentos científicos executados sobre a proposta da extensão do Apache Spark para o apoio à proveniência.
- Uma avaliação quantitativa para os métodos de indexação-e-consulta de dados de experimentação científica em seus formatos originais.

1.3 Organização do trabalho

Além da introdução, esta dissertação é composta por mais quatro capítulos. O Capítulo∼2 expõe trabalhos relacionados ao cenário do processamento DISC de dados científicos e as definições teóricas necessárias para melhor compreensão da abordagem proposta, além de aspectos das tecnologias utilizadas para apoiar a solução desenvolvida. O Capítulo∼3 descreve a abordagem proposta para a captura e armazenamento de proveniência no Apache Spark e detalha a implementação da solução proposta. O Capítulo∼4 discute sobre as formas mais adequadas para consultar dados científicos, considerando armazenamento em arquivos, relacional, indexação de arquivos *in-situ* e linguagens de consulta de alto nível. No final de cada um desses dois capítulos são apresentados os respectivos resultados experimentais. Finalmente, o Capítulo 5 conclui esta dissertação.

Capítulo 2

Referencial Teórico

Neste capítulo são abordados os principais conceitos utilizados ao longo dessa dissertação. Na Seção 2.1 definimos dados científicos e dados de domínio, visto que eles são conceitos essenciais para o entendimento das principais contribuições desse trabalho. Na Seção 2.2 é apresentado o conceito de fluxo de dados, assim como a sua definição formal. A Seção 2.3 apresenta alguns métodos e ferramentas existentes que apoiam o processamento de grandes volume de dados. Nessa seção também apresenta-se o Apache Spark, um *framework* para processamento de dados em memória que foi utilizado como base no desenvolvimento da solução proposta nessa dissertação.

As seções seguintes caracterizam os tipos de dados de proveniência e sua relação com dados de domínio, trabalhos correlatos para tratá-los na perspectiva do Apache Spark e os pontos em aberto da literatura que serão abordados no restante desse estudo.

2.1 Dados científicos

Na comunidade acadêmica não há um consenso do que é considerado um dado científico, uma vez que essa definição varia de acordo com o domínio e o tipo de pesquisa realizada [3, 57, 59]. Desta forma, nessa dissertação assumimos a definição dada pela EPSRC [3]:

> *"Dados científicos são definidos como registros factuais comumente preservados e aceitos pela comunidade científica como necessários para validar os resultados de uma pesquisa"*[1].

Em uma perspectiva geral, os experimentos científicos apoiados por computador analisam e produzem grandes volumes de dados [31]. Isso é resultado da evolução natural das

[1] `https://epsrc.ukri.org/about/standards/researchdata/scope/`

simulações computacionais e da precisão dos instrumentos científicos usados na captura desses dados [6, 43, 45]. Esse comportamento é o que caracteriza a *computação intensiva em dados* [33]. Além da quantidade, uma outra característica de dados científicos é a sua complexidade em termos de representação, armazenamento e interdependência entre os elementos de informação [6].

Dados científicos podem assumir diversos formatos, que variam de acordo com o domínio do experimento. Comumente, eles são armazenados em arquivos estruturados, os quais contêm metadados que os descrevem, tais como a sua estrutura interna, unidades de medidas, origem, além de outras informações [44, 75]. Como exemplos de formatos de arquivos para armazenamento desses dados em memória secundária, pode-se citar: arquivos FITS [70], que são utilizados para armazenar dados de astronomia; arquivos netCDF[2] e HDF5 [28], que são empregados para representar matrizes multidimensionais no domínio de dinâmica de fluidos computacionais; arquivos ROOT [13], usados para compactar dados de física de altas energias; e arquivos FASTA [49] usados para representar sequências de nucleotídeos no domínio de bioinformática.

Vale mencionar que, dentre todo o volume dos conjuntos de dados manipulados por experimentos computacionais, somente uma parte é normalmente relevante para confirmar ou refutar uma hipótese científica ou um comportamento específico do domínio [12, 27]. No contexto dessa dissertação, usaremos o termo "dados de domínio" alternativamente a essa porção de dados científicos de interesse.

2.2 Fluxos de dados

Os resultados das execuções de experimentos científicos com o uso intensivo em dados podem ser modelados como fluxos de dados (*dataflows*). Um fluxo de dados é representado por meio de um grafo direcionado, onde os vértices representam as transformações sobre os dados, enquanto as arestas representam as dependências entre elas [62]. A seguir, é apresentada a formalização de fluxos de dados adaptada de acordo com as definições dos trabalhos de Ikeda *et al.* [37] e de Silva *et al.* [62].

Definição 1 (Elemento de Dados) *Um elemento de dados* $e = \{v_1, v_2, \ldots, v_n\}, v_i \in \mathbb{V}$ *é composto de uma sequência de valores* v_i *cujo domínio é* $\mathbb{V}$.

Definição 2 (Coleção de Dados) *Uma coleção de dados c é composta por um conjunto*

[2] `http://www.unidata.ucar.edu/software/netcdf/`

de elementos de dados, i.e., uma matriz de valores.

Definição 3 (Conjunto de Dados) *Um conjunto de dados* $s = \langle A, C \rangle$ *é um par composto de uma sequência de atributos* $A = \{a \mid a \in \mathbb{A}\}$*, onde o domínio* $\mathbb{A}$ *é o par* $\langle$*nome, tipo*$\rangle$*, e de um conjunto de coleções de dados* C*, onde* $\forall\ c \in C \wedge e \in c,\ |e| = |A|$*.*

Definição 4 (Transformação de Dados) *Uma transformação de dados* $t(I),\ t : I \rightarrow O$ *é uma função que mapeia um conjunto de dados de entrada* I *em um conjunto de dados de saída* O*.*

Definição 5 (Dependência de Dados) *Uma dependência de dados* $\phi = \langle s, t, t' \rangle$ *é uma terna composta por um conjunto de dados* s *e duas transformação de dados* t *e* t'*, onde* $s \subseteq t(s) \wedge s \subseteq t'(t(s))$*.*

Definição 6 (Fluxo de Dados) *Um fluxo de dados* $D = \langle T, S, \Phi \rangle$ *é uma terna composta de um conjunto de transformações de dados* T*, de um conjunto de dados* S *e de um conjunto de dependências de dados* Φ*.*

Definição 7 (Instância da Transformação de Dados) *Dada uma transformação de dados* $t(I) = O$*, uma instância de transformação de dados* $t(I') = O'$ *consiste no resultado do mapeamento de quaisquer subconjuntos de coleções de dados nos conjuntos de entrada* $I' = \bigcup_{k=1}^{|I|} C'_k,\ C'_k \subseteq C_k \in I$ *em quaisquer subconjuntos de coleções de dados de saída* $O' = \bigcup_{k=1}^{|t(I)|} C'_k,\ C'_k \subseteq C_k \in t(I)$*.*

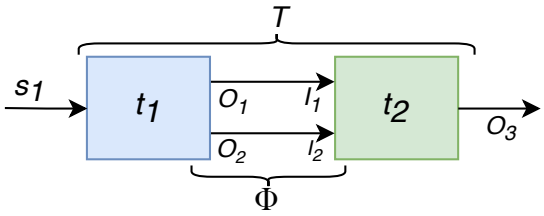


Figura 2.1: Exemplo de um fluxo de dados composto de duas transformações.

Na Figura 2.1 é apresentado um fluxo de dados com duas transformações t_1 e t_2. No exemplo em questão, a transformação t_1 consumiu um conjunto de dados s_1, que foi usado como entrada e produziu dois conjuntos de dados de saída O_1 e O_2, respectivamente. A transformação t_2 consume os conjuntos de dados produzidos por t_1 e produz, como saída, um terceiro conjunto de dados O_3.

2.3 Sistemas de processamento em larga escala

Diversos sistemas foram desenvolvidos para suprir a demanda por processamento de grande volume de dados que são produzidos por experimentos científicos. Nessa seção, discutimos métodos e ferramentas para o processamento de dados científicos em larga escala, tais como os sistemas baseados na abstração de *workflows*.

2.3.1 *Workflows* científicos

Fluxos de dados de experimentos científicos apoiados por computador são obtidos pelo encadeamento de diversas tarefas. Para gerenciar a execução dessas tarefas, uma das abstrações mais adotadas é a de *workflow* científico [25]. A definição de *workflow* é proveniente do ambiente de negócios e foi cunhado pela coalizão *Workflow Management Coalition* (WfMC) com o objetivo de desenvolver modelos de referências para os sistemas de *workflows* de negócios [65]. O WfMC define o termo *workflow* como:

> *"A automação do processo de negócio, por completo ou em partes, no qual documentos, informações ou tarefas são passados de um participante para o outro para a execução de uma ação, de acordo com um conjunto de regras procedimentais"* [35].

Tal definição é dirigida aos *workflows* de negócios, mas também pode ser expandida para o contexto científico [65]. *Workflows* científicos são empregados como uma abstração que permite modelar experimentos através de um conjunto de atividades e dos relacionamentos entre elas [25, 65]. Fazendo uma comparação com fluxos de dados, as atividades representam a execução de uma transformação de dados, que pode incluir a invocação de um programa científico, a análise de dados, dentre outras rotinas computacionais. Enquanto isso, o relacionamento entre as atividades representa a dependência entre elas, o que também significa uma troca de informações entre as atividades. Desta forma, um *workflow* pode ser descrito como um grafo direcionado acíclico (Figura 2.2).

Embora seja possível executar os *workflows* manualmente, é conveniente que se utilizem aplicações ou interfaces de alto nível para a construção, gerência e execução dos *workflows* científicos. Essas ferramentas são conhecidas como Sistemas de Gerência de *Workflows* Científicos (SGWfC), cujos objetivos incluem oferecer aos usuários uma forma simples e abstrata, geralmente com o uso de interfaces gráficas, de modelar e monitorar a execução do *workflows* [11, 56]. Desta forma, um pesquisador que não tem domínio sobre

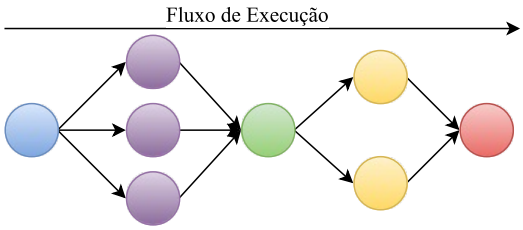


Figura 2.2: Representação gráfica de um *workfow*, os vértices representam as atividades e as arestas indicam as dependências e precedências entre as atividades.

programação pode aproveitar ao máximo o poder computacional que um determinado sistema tem a oferecer [29], pois fica a cargo do SGWfC gerenciar a complexidade dos recursos computacionais. SGWfCs mais avançados também procuram gerenciar a execução dos *workflows* em ambientes paralelos e distribuídos, como *clusters*, *grids* e nuvens computacionais. Entretanto, SGWfCs têm algumas limitações, a saber:

- Paralelismo em granularidade grossa, uma vez que esses sistemas tem por objetivo somente invocar as "unidades" de tarefas das atividades, e desta forma, não permitem a paralelização das ações internas realizadas por essas tarefas.
- *Workflows* intensivos em dados que requerem várias transferências entre atividades, visto que a granularidade grossa nas execuções das atividades, pode levar a movimentações desnecessárias de dados entre os nós de processamentos e sobrecarga do canal compartilhado (sobrecarga na rede).

Uma alternativa aos SGWfC, e que permitem o paralelismos em dois níveis das atividades (o processamento e suas passos) são os *frameworks* MapReduce, os quais são apresentados a seguir.

2.3.2 MapReduce

O MapReduce é um modelo de programação que foi desenvolvido pela Google [24] para otimizar o processamento paralelo e distribuído de grandes volumes de dados. Esse modelo permite a execução de experimentação e processamento paralelo em milhares de máquinas heterogêneas que, eventualmente, possam vir a apresentar alguma falha. De forma mais específica, o MapReduce foi inspirado em duas funções comumente utilizadas em programação funcional: o *Map* e o *Reduce*. Portanto, o paradigma MapReduce é baseado no método dividir para conquistar, que consiste em dividir (*Map*) recursivamente um problema complexo em diversos sub-problemas até que estes se tornem escaláveis e pos-

sam ser resolvidos rapidamente [17]. Os resultados intermediários são agrupados (*Reduce*) para produzir a solução global do problema em análise.

Ao desenvolver suas soluções baseadas em MapReduce, os usuários devem expressar toda a computação por meio desses dois métodos. Além disso, *frameworks* MapReduce partem do princípio que as tarefas de computação recebem como entrada um conjunto de pares ⟨chave, valor⟩ e produzem como saída pares de igual formato [24]. Neste contexto, cada função *Map* consome uma tupla e produz zero, ou mais, tuplas de saída como resultados intermediários. Após consumir todos os dados de entrada na função *Map*, o MapReduce agrupa as tuplas que têm a mesma chave, e para cada grupo, é criado uma nova tupla onde a chave é a chave do grupo e o valor é composto por uma lista de valores de todos os dados. Finalmente, esse novo conjunto de pares é passado para as funções do tipo *Reduce*. As funções *Reduce* fazem uma junção dos valores e produzem uma tupla com o menor número de valores para cada grupo. A principal diferença entre essas funções é que rotinas *Map* consomem somente uma tupla por vez, enquanto que funções *Reduce* podem consumir mais de uma tupla por turno.

A Figura 2.3 ilustra o fluxo de execução de um problema de contagem de palavras de um texto por meio do modelo de programação MapReduce. Esse problema pode ser dividido em uma função *Map* e uma função *Reduce*. A primeira função recebe como entrada o nome do documento (como chave) e uma linha desse arquivo (como valor). Para cada palavra presente na linha, a função produz como saída pares onde a palavra é a chave e o valor 1 é a saída. Por sua vez, a função *Reduce* recebe como entrada os pares produzidos pela função *Map*, onde as chaves correspondem às palavras encontradas e os valores são uma lista com os "1"s colocados para cada vez que aquela chave apareceu. Essa função produz como saída tuplas em que as chaves são as palavras encontradas, enquanto que o valor de cada chave representa o número de vezes que aquela palavra apareceu nos documentos pesquisados.

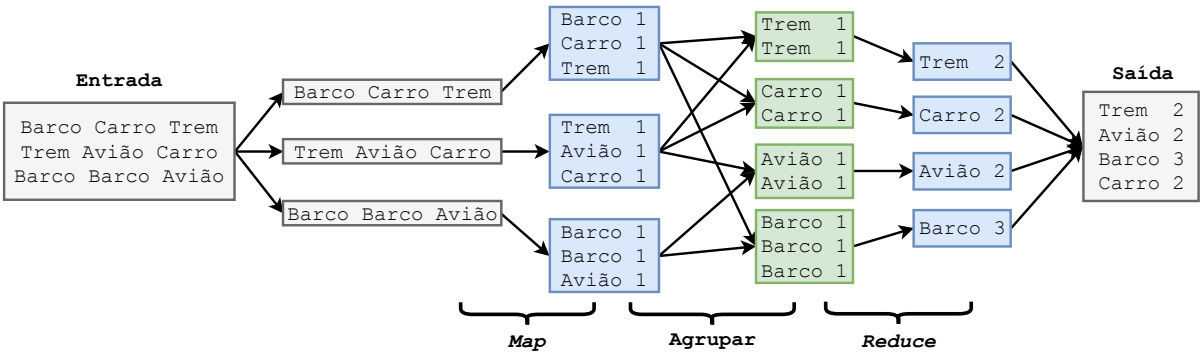


Figura 2.3: Etapas do MapReduce para o problema de contagem de palavras.

2.3.2.1 Apache Hadoop

O Apache Hadoop[3] é uma das implementações de código aberto mais conhecidas do paradigma MapReduce. Esse *framework* objetiva gerar soluções para armazenamento e processamento de grandes volumes de dados. O Apache Hadoop foi projetado para garantir a integridade dos dados, disponibilidades dos nós, escalabilidade das aplicações e recuperações de falhas sem que seja necessário equipamentos específicos. A ferramenta é composta de diversos componentes, sendo os principais para esse trabalho:

- Hadoop *Distributed File System* (HDFS): Um sistema de arquivos distribuídos que permite alta vazão no acesso aos dados;
- Hadoop MapReduce: Um sistema para processamento paralelo de grandes conjuntos de dados que estão armazenados no HDFS.

O Hadoop trabalha em uma arquitetura mestre-escravo. Nesse cenário, os nós mestres supervisionam as duas principais partes funcionais que compõem o Hadoop: o armazenando de dados (HDFS) e a execução das computações dos dados (MapReduce). Cada nó escravo no *cluster* do Apache Hadoop faz duplo papel, armazenando os dados e também processando funções *Map/Reduce*. O Hadoop parte do princípio de que é mais fácil levar a computação até os dados, do que levar os dados até os nós de processamento. Desta forma, a ferramenta sempre tenta realizar o processamento na máquina disponível mais próxima aos dados para aumentar a velocidade de acesso.

No Apache Hadoop, o resultado de cada função *Map* e *Reduce* é escrito no HDFS e, em seguida, esses dados são lidos pelo próximo processamento. Embora os resultados estejam possivelmente sendo produzidos e consumidos na mesma máquina, fazer essas operações de escrita e leitura na memória secundária resulta em execuções dispendiosas e limitadas por velocidade de E/S, que tendem a ser inferior a capacidade de processamento do nó. Para contornar essa problemática, foi desenvolvido o Apache Spark.

2.3.2.2 Apache Spark

O Apache Spark[4] é um *framework* baseado em processamento distribuído de dados em memória por meio de uma abstração chamada RDD (*Resilient Distributed Dataset*) [73].

[3] `http://hadoop.apache.org`
[4] `https://spark.apache.org/`

RDDs são coleções particionadas de elementos de dados imutáveis entre os nós que compõem o *cluster* de processamento. Através dessa abstração é possível executar diversas operações sobre os dados, incluindo as funções de *Map* e *Reduce*. No Apache Spark, a aplicação desenvolvida pelo o usuário é chamada de *driver program* e é responsável por gerenciar a distribuição das operações paralelas no *cluster* através do *SparkContext*, como exemplificado na Figura 2.4.

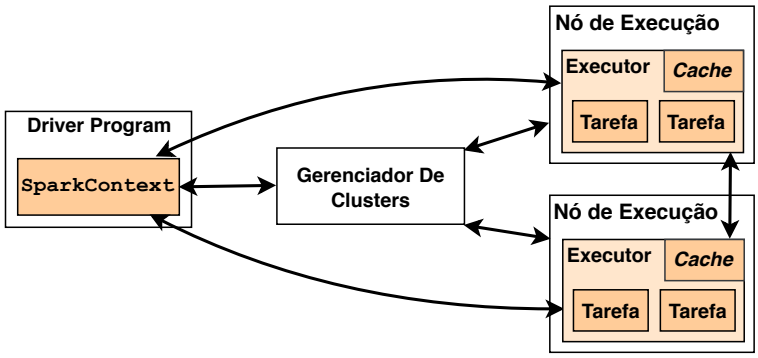


Figura 2.4: Gerenciamento e distribuição de tarefas no *cluster*. Adaptado de [2].

Usuários podem criar RDDs de duas formas: *(i)* ao carregar os dados de uma fonte externa, que inclusive pode estar armazenada no HDFS, ou *(ii)* ao distribuir uma coleção de dados como uma lista, através do *driver program* [40]. Uma vez criados, os RDDs oferecem dois tipos de operações: transformações e ações. As transformações criam um novo RDD à partir da aplicação de uma função sobre o RDD anterior. Entre as transformações mais comuns suportadas pelo Spark estão *filtragem*, *junção* e *agrupamento por chave*, além das funções *Map* e *Reduce*. Enquanto isso, as ações ou processam os resultados dos RDDs e os devolvem para o *driver program*; ou os persistem em um sistema de armazenamento em memória secundária.

Uma vez modelada a aplicação, sempre que o usuário executa uma ação, o *SparkContext* analisa o fluxo das transformações de dados e cria um grafo direcionado e acíclico (DAG) [73]. O Apache Spark também utiliza o DAG para coordenar a execução paralela de instruções nas partições de dados que compõem um RDD. Uma otimização que o Apache Spark faz nesse ponto é agrupar um conjunto de transformações que podem ser executadas de formas independentes em estágios (*stages*).

Portanto, a principal diferença de ganho entre o Apache Hadoop e o Apache Spark é que as trocas de dados entre as transformações que compõem um mesmo estágio é feito com os dados ainda em memória principal.

2.4 Dados de Proveniência

Neste trabalho adotamos a definição de proveniência dado pelo W3C (*The World Wide Web Consortium*) *Provenance Working Group's* [54], que diz que a proveniência é um registro que descreve pessoas, instituições, entidades e atividades que produzem, influenciam ou entregam pedaços de dados ou de algo. De forma geral, a proveniência de dados facilita a repetição ou a reprodução de um resultado, além de permitir a avaliação da qualidade e a confiabilidade de uma determinada informação em um experimento científico [46, 54].

Dados de proveniência de um fluxo de dados podem ser capturados em diferentes níveis de granularidade. Com o objetivo de distinguir as informações coletadas de um ambiente com captura de proveniência em diferentes níveis, separamos os tipos de dados de proveniência de acordo com seus aspectos, da seguinte maneira:

1. Dados de proveniência prospectiva. Representam a sequência de passos necessários para produzir o resultado de um experimento científico, por exemplo, os programas que são necessários para efetuar a computação [20, 21, 29, 52]. No contexto de fluxo de dados, essa sequência é composta pelas definições das transformações e o vínculo de dados entre elas.

2. Dados de proveniência retrospectiva. São dados obtidos durante a execução dos experimentos, que incluem o registro dos dados que foram usados e gerados durante as transformações, o ambientes em que eles foram executados e os parâmetros que foram utilizados nos programas [20, 21, 29, 52]. Em outra palavras, os dados de proveniência retrospectiva podem ser vistos como um *log* detalhado de uma execução de um experimento científico [29].

3. Dados de domínio. São os dados produzidos pelas aplicações científicas e cujo conteúdo os cientistas precisam trabalhar e inspecionar para corroborar ou refutar uma hipótese científica [12, 62]. Vale ressaltar que os dados de domínio podem ser viso como um subconjunto dos dados de proveniência retrospectiva.

Para exemplificar essa distribuição de dados de proveniência, a Figura 2.5 apresenta a classificação dos dados de um *workflow* composto por três atividades de acordo com a sua classe no fluxo de dados e no tipo de proveniência. As atividades de *workflow*, ou seja, as transformações do fluxo de dados, são dados de proveniência prospectiva, que são definidas durante a fase de concepção do experimento científico. As coleções de dados que

foram usados e gerados durante a execução do fluxo de dados são os dados de proveniência retrospectiva e, finalmente, os elementos de dados (utilizados durante a fase de análise de validação dos experimentos) são classificados como dados de domínio.

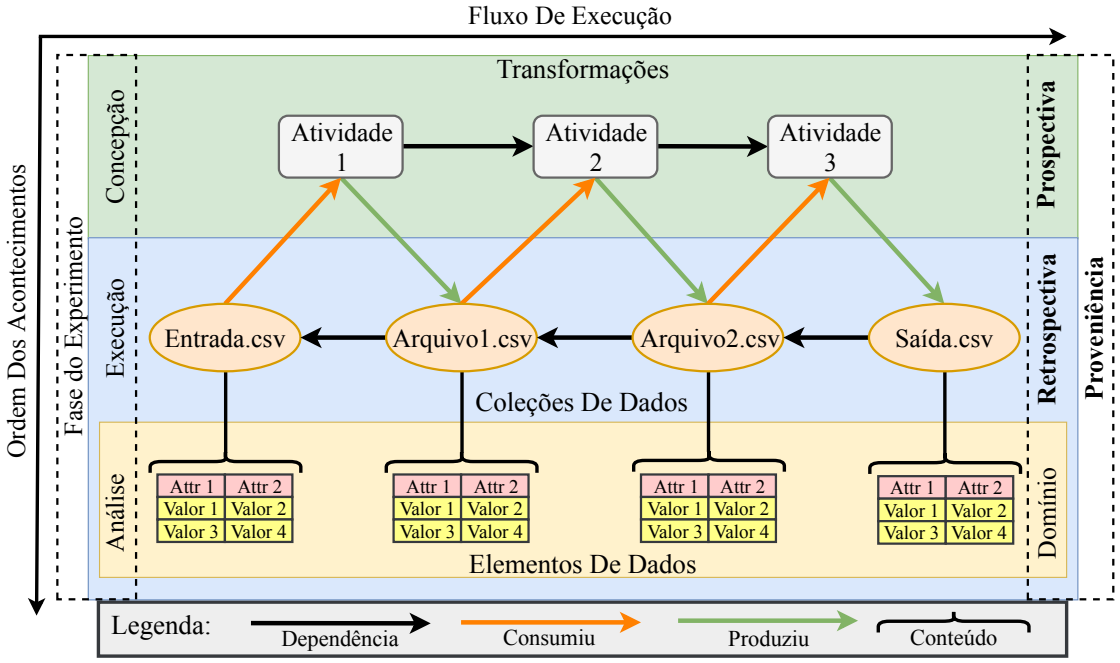


Figura 2.5: Classificação dos dados de um experimento científico de acordo com a sua classe no fluxo de dados e na proveniência.

Existem muitos modos para se representar as informações de proveniência, incluindo o padrão o PROV [54] criado pelo W3C. O W3C PROV é um modelo de dados genérico e sua definição gira em torno de três elementos-básicos, a saber: *Entidade*, *Atividade* e *Agente*, e das ligações entre eles, as quais possuem rótulos específicos. Mais detalhes sobre um modelo de dados de proveniência para representar o fluxo de dados em experimentos científicos são apresentados na subseção a seguir. Além disso, as principais soluções baseadas em proveniência de dados em ambientes DISC são discutidas na sequência.

2.4.1 Modelos de representação de proveniência

No W3C PROV itens físicos, digitais, conceituais ou de quaisquer outros tipos são chamados de entidade (*Entity*) [30]. Por exemplo: arquivos, gráficos, parâmetros, documentos e fotos são considerados entidades e podem ser descritos por diferentes atributos a partir de diferentes perspectivas. No contexto de *workflows* científicos, entidades representam os artefatos usados ou gerados durante a execução de um experimento.

As atividades (*Activity*) representam os aspectos dinâmicos do mundo, como ações e processos [30]. Exemplos de atividades são o processamento de uma matriz ou a análise

de um determinado artefato. Portanto, atividades são ações ou processos que atuam sobre entidades, através do uso ou da geração deles. No contexto de *workflows* científicos, a atividade representa a execução de uma tarefa. Já o agente (*Agent*) representa algo, ou alguém, que possui alguma responsabilidade sobre uma atividade, pela existência de uma entidade ou pelas atividades de outros agentes [30]. Um agente pode ser uma pessoa, um aplicação ou uma organização. No contexto de *workflows* científicos, um agente pode ser o pesquisador ou um SGWfC.

Diversos tipos de relacionamentos são definidos no modelo W3C PROV, sendo os principais descritos em [54]. O relacionamento *wasGeneratedBy* indica que uma entidade foi gerada por uma atividade; O *used* representa a utilização de uma entidade por uma atividade; O *wasInformedBy* representa a dependência entre atividades. Resumidamente, esse relacionamento relata a troca de entidades entre atividades. O *wasDerivedFrom*, significa que uma entidade foi derivada de outra; O *wasAttributedTo*, indica a atribuição da entidade a um agente, ou seja, o agente teve uma contribuição na geração ou uso dessa entidade; O *wasAssociatedWith* relata a responsabilidade de um agente sobre uma atividade; Finalmente, o *actedOnBehalfOf* representa a atribuição responsabilidade de um agente sobre outro agente para a realização de uma atividade específica como delegado.

A Figura 2.6 mostra uma visão geral dos elementos que compõem o W3C PROV e os relacionamentos entre eles. O W3C PROV é um modelo genérico que permite extensões para se adaptar ao domínio onde será aplicado. Desta forma, os dados de proveniência prospectiva, retrospectiva e de domínio podem ser eficientemente modelados como modelos compatíveis com o W3C PROV. Nessa dissertação consideramos o uso do modelo PROV-Df [63], visto que ele é focado na abstração de fluxo de dados.

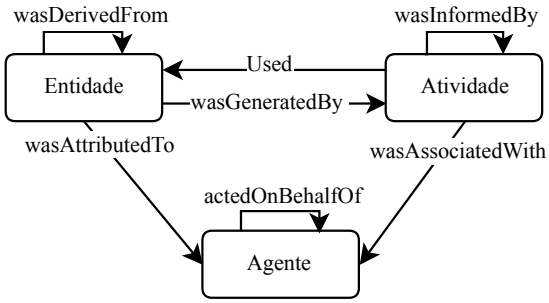


Figura 2.6: Tipos e relacionamentos base do W3C PROV. Adaptado de [54].

O PROV-Df representa como dados de proveniência prospectiva de um fluxo de dados, tanto o encadeamento de transformações, quanto as suas respectivas dependências de dados. Dados de proveniência retrospectiva correspondem aos dados de domínio de interesse do usuário, além dos metadados consumidos e produzidos pelas transformações.

O PROV-Df consiste na especialização dos componentes *Entidade*, *Agente*, *Plano* e *Atividades* definidos no padrão W3C PROV. A Figura 2.7 mostra o diagrama de classe do PROV-Df [63], sendo que esse diagrama é composto de três grupos de classes. Na figura, as classes em branco representam a estrutura do fluxo de dados referente a proveniência prospectiva. As classes em cinza representam as execuções, juntamente com os dados de domínio referentes a um fluxo de dados. Por fim, as classes em cinza claro representam as configurações do ambiente computacional. Os estereótipos no diagrama são usados para representar a associação de cada classe com os elementos do W3C PROV.

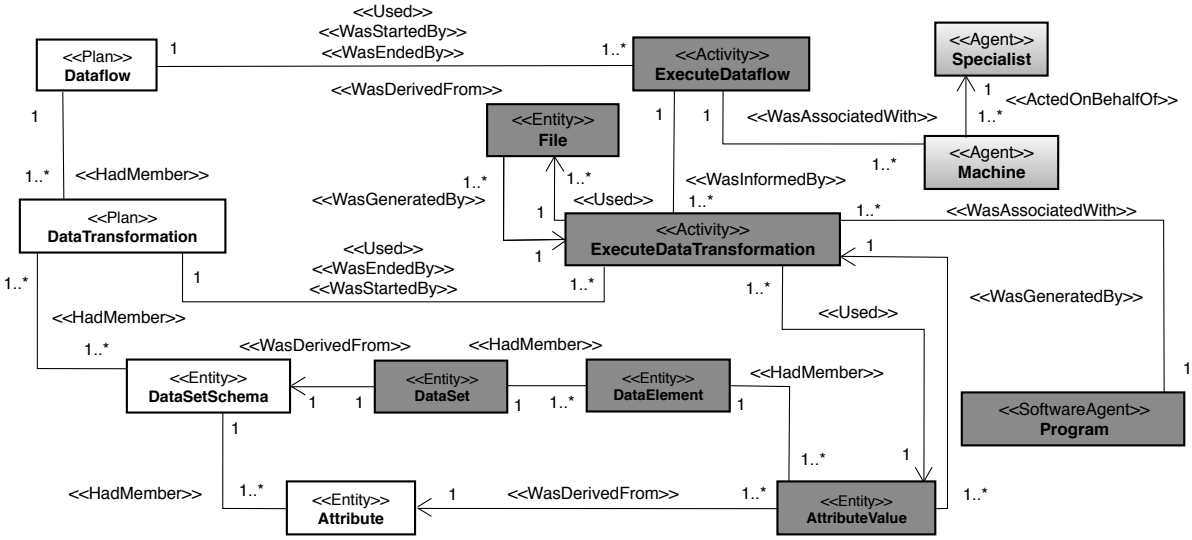


Figura 2.7: Diagrama de classe do PROV-Df. Adaptado de [63].

2.4.2 Dados de proveniência em ambientes DISC

Soluções para a captura de dados de proveniência em ambiente DISC têm sido desenvolvidas com diferentes objetivos tanto para o Apache Hadoop quanto para o Apache Spark. Algumas soluções baseiam-se na instrumentação do código, tais como RAMP [36] e Newt [48]. Outras são diretamente ligadas a um *framework*, tais como o Titian [38] e o BigDebug [32]. Nessa seção, uma descrição de cada uma dessas soluções é apresentada e a Tabela 2.1 discute uma comparação direta entre as abordagens revisadas especificamente com relação à captura e análise de dados de proveniência.

Spark *Log* [9]. O Apache Spark inclui um sistema simplificado de proveniência prospectiva e retrospectiva projetado somente para monitorar as execuções dos *workflows*. Os dados de monitoramento podem ser acessados por uma interface *web* ou por um arquivo de texto que registra o *log* da execução. Na interface *web*, os usuários podem visualizar o DAG das transformações e verificar o estágio de suas execuções. Embora os dados esta-

tísticos, como o consumo de memória e o tempo decorrido, possam ser obtidos por essa interface, análises mais sofisticadas exigem o *parsing* e o processamento do arquivo de *log*.

Newt [48]. Newt utiliza a proveniência para encontrar erros nos resultados obtidos pela execução dos *workflows*. A ferramenta captura dados de proveniência através da instrumentação de código, onde os usuários são responsáveis por definir um identificador para cada instância de dados do experimento e fornecer explicitamente o relacionamento entre eles. Os dados de proveniência são armazenados em um *cluster* do SGBD MySQL, de modo que consultas SQL em alto nível podem ser utilizadas para rastrear todos os identificadores envolvidos no experimento e que culminaram na produção do erro na saída. Com acesso a essa informação, o usuário pode realizar a reexecução da aplicação com um sub-conjunto dos dados de entrada.

RAMP [36]. O RAMP (*Reduce And Map Provenance*) foi projetado especificamente para capturar dados de proveniência em processamentos MapReduce no Hadoop. O RAMP emprega a API do Apache Hadoop para coletar os dados de proveniência e os armazena em arquivos no HDFS. Portanto, ferramentas específicas, como Hive [66] ou Pig [58], devem ser usadas para a consulta de proveniência. Essa solução, assim como o Newt, também utiliza identificadores para representar cada coleção de dados. Porém, como a API do Apache Hadoop é usada na conexão, a chave das tuplas é empregada como o seu identificador para posterior recuperação.

Titian [38]. Titian é uma ferramenta para Apache Spark que permite capturar dados de proveniência à partir da execução de transformações de dados encadeadas. Essa captura permite os usuários rastrearem, a partir de uma determinada transformação, quais foram os dados consumidos e produzidos durante o processamento computacional de interesse. Assim como nas outras soluções, ele também captura e armazena somente os relacionamentos entre os dados, ou seja, expressa as dependências entre os elementos de dados, sem compreender o seu conteúdo ou o seu valor semântico. Contudo, não é gerado um identificador para cada registro produzido por uma transformação. Pelo contrário, é gerado um identificador para cada registro usado ou gerado em um estágio da execução do Apache Spark. Desta forma, diminui-se o número de registros necessários para representar uma execução. Para analisar o histórico da execução do experimento científico, o Titian fornece um novo tipo de RDD, chamado de *LineageRDD*, que pode ser obtido à partir de qualquer RDD. Por meio dessa estrutura, o usuário pode recuperar dados de interesse na forma de um RDD, sejam eles dados anteriores ou posteriores de uma transformação. Portanto, os usuários podem aplicar novas transformações sobre os dados recuperados,

ao invés de apenas “repetir” a execução de um fluxo de dados, como em outras soluções. A tabela com os dados de proveniência é mantida em memória principal, embora seja possível armazená-la em disco por meio de implementação de código próprio.

	Armazenamento			**Recuperação**		
	Prospectiva	**Retrospectiva**	**Dados de Domínio**	**Prospectiva**	**Retrospectiva**	**Dados de Domínio**
Spark *log*	*Log* em arquivo texto	*Log* em arquivo texto	Não Suporta	Relatório *web* e *parsing* do *log*	Relatório *web* e *parsing* do *log*	Não Suporta
Newt	SGBD Relacional	SGBD Relacional		SQL padrão	SQL padrão	
RAMP	Arquivo HDFS	Arquivo HDFS		Hive/Pig	Hive/Pig	
Titian	Lineage RDD	Lineage RDD		RDD	RDD	
BigDebug	Lineage RDD	Lineage RDD		Break/watch point	Break/watch point	

Tabela 2.1: Comparação de soluções existentes para tratar dados de proveniência em ambientes DISC.

BigDebug [32]. O BigDebug segue um viés de proveniência diferente dos demais e tem por objetivo fornecer acesso em tempo de execução aos dados por meio da simulação de *breakpoints* e *watchpoints*. A solução fornece um sistema de depuração equivalente aos disponíveis para as linguagens de programação, como o GDB[5], porém focada em um ambiente paralelo e distribuído. O BigDebug foi construído sobre o Titian, aproveitando a linhagem de dados em RDDs para a depuração de aplicações em Spark. Ele permite a recuperação de dados intermediários a partir de *breakpoints* ou *watchpoints* para os dados que passaram por um determinado filtro e sem interromper a execução da aplicação. Os dados recuperados são apresentados ao usuário através de uma interface *web*, em que o usuário pode controlar as execuções das transformações.

Destacamos que nenhuma das abordagens revisadas apoia a captura e o armazenamento de dados de domínio. Além disso, as abordagens não tratam os dados científicos manipulados pela aplicação, *e.g.*, Newt, RAMP e Titian. De forma geral, os métodos persistem em uma tabela o rastreio das transformações dos dados na qual é atribuído um identificador para cada registro manipulado pela aplicação e descrevem o relacionamento entre esses identificadores. Embora essas informações sejam dados de proveniência, elas são de granularidade grossa, o que limita as análises exploratórias em dados científicos produzidos pelos experimentos que, comumente, envolvem investigação de proveniência e domínio.

2.5 Consulta a dados de domínio

A grande quantidade de dados de domínio produzidos por experimentos científicos traz desafios também no que se refere à capacidade de analisá-los em tempo hábil. Um dos métodos disponíveis para se realizar buscas genéricas sobre estes dados é por meio de um SGBD, onde utiliza-se linguagens de consultas de alto nível, como o SQL padrão. Entretanto, nesse caso, é necessário primeiro realizar a carga dos dados no SGBD, partindo do princípio que dados de experimentos científicos de ambientes DISC são persistidos em arquivos. No entanto, carregar esses arquivos para um formato estruturado se mostra ou impraticável ou inviável na maioria dos casos, uma vez que o carregamento pode demandar um tempo considerável para preparação e transformação, ou ainda, os arquivos de dados podem precisar ser mantidos em seu formato original porque novos experimentos podem ler e escrever informações sobre eles [7, 18, 41]. Métodos de indexação sobre dados de domínio brutos têm sido propostos, nesse contexto, com o objetivo de permitir consultas

[5] `https://www.gnu.org/software/gdb/`

genéricas sobre dados em seus formatos originais, *i.e.*, sem que seja necessário fazer a carga em SGBDs. Dentre eles destaca-se o mapa de *bits* (*bitmap*) e o índice posicional.

2.5.1 Bitmap

A indexação baseada em um mapa de *bits* consiste na análise de determinados atributos presentes em dados de domínio para gerar índices binários por meio da verificação de equações ou inequações algébricas [72]. Por exemplo, assumindo-se que os valores do atributo X para um arquivo de dados apresenta um domínio com apenas cinco valores possíveis, então o mapa de *bits* necessita de cinco colunas para identificar a presença ou não de um valor desse atributo por meio de uma equação algébrica, conforme exemplificado na Tabela 2.2. Existem casos também em que o uso de inequações para a indexação *bitmap* é mais vantajosa, principalmente em cenários em que o domínio de um atributo é extenso.

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

Tabela 2.2: Mapa de bits para indexação *bitmap* do atributo X em um exemplo sintético.

O FastBit [72] e o FastQuery [19] são exemplos de sistemas que realizam a indexação de dados em mapas de *bits*. Mais especificamente, o FastBit propõe um conjunto de variações nas técnica de indexação, permitindo *(i)* o ajuste do tipo de codificação utilizado, *(ii)* do algoritmo de compressão e *(iii)* do nível de precisão dos valores numéricos para se construir o mapa de *bits*. Por outro lado, o FastQuery é uma implementação paralela do Fastbit. Essas duas estratégias são usadas em consultas *post-mortem*, ou seja, após a finalização da execução de experimentos, pois os algoritmos de indexação precisam ter ciência de todos os dados de domínio presentes nos arquivos oriundos dos experimentos.

2.5.2 Índice posicional

A contra-parte de índices para consultas *post-mortem* de dados de domínio é a geração de índices posicionais, que utilizam informações para facilitar a localização (*i.e.*, posição) dos dados em arquivos. Em comparação com a indexação *bitmap*, índices posicionais podem fornecer uma sobrecarga de dados menor, pois estes não utilizam *bits* redundantes (*i.e.*,

sequência de zeros no mapa de *bits*) que crescem de acordo com o domínio dos atributos indexados. Além disso, índices posicionais permitem referenciar estruturas de dados mais complexas, como árvores e malhas.

O NoDB [42] é um exemplo de sistema que emprega índices posicionais e realiza a extração de dados de domínio presentes em arquivos, carregando-os em uma versão modificada do SGBD PostgreSQL, conhecido como PostgresRAW. Apesar de evitar mudanças nas estruturas de dados adotadas pelos arquivos, o NoDB baseia-se no processamento de *consultas adaptativas*, que utiliza estatísticas e estratégias de *caching* para ter acesso aos dados científicos e realizar apenas transformações de dados estritamente necessárias. A ideia é que apenas os dados científicos necessários pelas consultas são efetivamente indexados.

A Tabela 2.3 apresenta uma comparação entre as soluções FastBit e PostgresRAW em relação às suas técnicas de indexação, suporte a processamento de consulta em alto nível, necessidade de carga de dados de domínio e complexidade de tempo para consultar um arquivo de dados em seu formato original.

Abordagem	**Estratégia**	**Indexação**	**Linguagem Para Consulta**	**Carga dos Dados**	**Acesso ao dados no Arquivo**
FastBit	Indexação dos dados dos arquivos	Bitmap	SQL Compatível	Requerido	Constante
PostgresRAW	Consulta Adaptativa	Posicional	SQL Padrão	Não Requerido	Linear

Tabela 2.3: Comparação entre PostgresRAW e FastBit.

2.6 Considerações Finais

Neste capítulo discutimos os conceitos investigados para o desenvolvimento de uma solução que considere o armazenamento e consulta de dados de proveniência oriundos de execuções em ambientes DISC, em particular, do Apache Spark. De acordo com a revisão bibliográfica, destacamos que, embora o Apache Spark (objeto central dessa dissertação) execute com eficiência *workflows* científicos, esse *framework* ainda tem limitações no que se refere ao suporte à captura, armazenamento e consulta de dados de proveniência prospectiva, retrospectiva e, principalmente, de domínio.

Mais do que isso, os trabalhos relacionados, além de não disponibilizarem suas implementações publicamente para efeitos de comparação, são focados apenas no grafo de transformações. Assim, os aspectos de proveniência capturados não são o produto final de suas aplicações, mas apenas etapas contempladas dentro de outros objetivos, *e.g.*, *debug-*

ging de aplicações Apache Spark. Também destacamos que os trabalhos relacionados não se preocupam em tratar dados de domínio, em particular não há suporte para os casos em que esse tipo de dados é manipulado por aplicações "caixa-preta" externas. Portanto, uma solução mais abrangente para o tratamento de proveniência também deve capturar, extrair e armazenar dados de domínio de arquivos brutos.

Nas próximas seções usaremos os conceitos e soluções revisadas até aqui para completar essa lacuna de ausência de uma solução abrangente para captura e consulta de dados de proveniência de aplicações científicas que rodam sobre o Apache Spark.

Capítulo 3

Captura e gerência de dados de proveniência no Apache Spark

Nesta seção é apresentada a proposta desenvolvida neste trabalho para a coleta e armazenamento de dados de proveniência oriundos de experimentos científicos executados no Apache Spark. Em particular, discute-se quais os componentes do Apache Spark que precisam ser estendidos para que a proveniência prospectiva e retrospectiva e os dados de domínio possam ser tratados na execução de qualquer *workflow* sem que seja necessário modificar o código da aplicação existente. Além disso, com uso de instrumentação torna-se possível aos usuário selecionar os dados de interesse a serem persistidos na base de proveniência.

Dentre as modificações e os novos componentes propostos, destaca-se o suporte para dados de domínio oriundos de transformações realizadas por aplicações "caixa-preta", um ponto que nenhum dos trabalhos revisados consegue tratar. A solução proposta para esse problema é discutida em duas frentes. Na primeira, é usado um "envelopamento" do RDD com relação ao seu conteúdo e também é empregado um *template* para a seleção e extração de dados representados por um RDD. Na segunda frente, é proposto e implementado um sistema de arquivos em memória principal, com o objetivo de se otimizar a execução de *workflows* científicos que usam aplicações "caixa-preta" durante o seu processamento. Assim, evitá-se operações de leitura e escrita em memória secundária quando há invocação de aplicações externas. Nesse sentido, nossa solução apoia a captura e armazenamento da proveniência buscando evitar, também, a queda no desempenho ou prejuízo na execução do experimento científico.

Outro aspecto que foi considerado para a implementação da solução, foi a modificação do código-fonte do Apache Spark para que se tornasse possível a identificação de cada

coleção de dados manipulada em cada transformação (seja ela uma aplicação externa ou não). Ao empregar esse mecanismo de identificação, o rastro de proveniência da execução do experimento pode ser persistido em uma base de dados, assim como as transformações e as dependências de dados podem ser diretamente associadas aos dados persistidos. As diferentes formas de armazenamento (SGBDs relacionais, colunares, etc.) e consultas sobre dados de proveniência (consultas em tempos de execução e *post-mortem*) foram tratadas experimentalmente para determinar o conjunto de ferramentas mais adequado para o gerenciamento de proveniência no Apache Spark.

Para permitir o monitoramento da execução de experimentos e a visualização de relatórios sob demanda, foi implementada uma interface *web* que mostra o fluxo de dados como um DAG interativo. Além disso, para dar flexibilidade aos cientistas que querem validar suas hipóteses usando as informações de proveniência com consultas genéricas em alto nível, implementamos um conversão de tipos que permite ao usuário expressar suas buscas em linguagem Cassandra CQL ou SQL padrão, o que dá portabilidade à base de dados de proveniência coletada.

Na sequência é apresentado todo o projeto e implementação da solução proposta para a captura e gerenciamento de dados de proveniência no Apache Spark.

3.1 Projeto da solução para captura de proveniência

Nesta seção, é apresentada a solução proposta por esse trabalho para a captura dos três aspectos de proveniência em aplicações científicas executadas sobre o Apache Spark, os quais foram definidas na Seção 2.4. Nossa abordagem, que denominamos `SAMbA` (Spark *provenAnce MAnagement on* RDDs)[1], consiste em envelopar tanto a estrutura quanto o conteúdo de um RDD em tempo de execução. A partir dessa estratégia, criamos um conjunto de componentes que permitem que: *(i)* dados científicos consumidos e produzidos sejam armazenados pelo `SAMbA` de forma eficiente e eficaz e, em determinados casos, estruturada e *(ii)* dados de proveniência possam ser consultados durante e após a execução de experimentos científicos.

A proposta do `SAMbA` é estender os módulos básicos do Apache Spark para dar suporte à captura e consulta de dados de proveniência. Nesse sentido, as quatro fases de execução de uma aplicação no Apache Spark estão envolvidas na extensão proposta. Em particular, essas fases estão relacionadas com: *(i)* o Apache Spark criar um fluxo de transformação

[1]Detalhes da solução estão disponíveis em: `https://github.com/UFFeScience/SAMbA`

dos dados através da API (*Application Programming Interface*) fornecida pelo *SparkContext*. Ao identificar uma ação, *(ii)* o *SparkContext* se conecta ao gerenciador de *clusters* e solicita a criação de *Executors* em cada nó de processamento que compõe o *cluster*. No Apache Spark, os *Executors* são processos do sistema operacional que efetivamente executam as transformação sobre os dados e armazenam os dados intermediários. Uma vez iniciados os *Executors*, *(iii)* o *SparkContext* envia os binários da aplicação para cada nó de processamento. Por fim, *(iv)* o *SparkContext* submete as tarefas para os *Executors* para execução. Dada essa característica do Apache Spark (do envio de instâncias de transformações de dados para os *Executors*), o *SparkContext* consegue otimizar o balanceamento de carga e o desempenho geral do sistema.

A Figura 3.1 apresenta a arquitetura do `SAMbA` sobre os módulos básicos do Apache Spark. Nesta figura, os elementos coloridos representam os novos componentes e destacam os elementos do Apache Spark que foram estendidos. Dentre os elementos do Apache Spark utilizados na execução de uma aplicação, dois são estendidos pelo `SAMbA` para o gerenciamento de dados de proveniência: o *SparkContext* e os *Executors*.

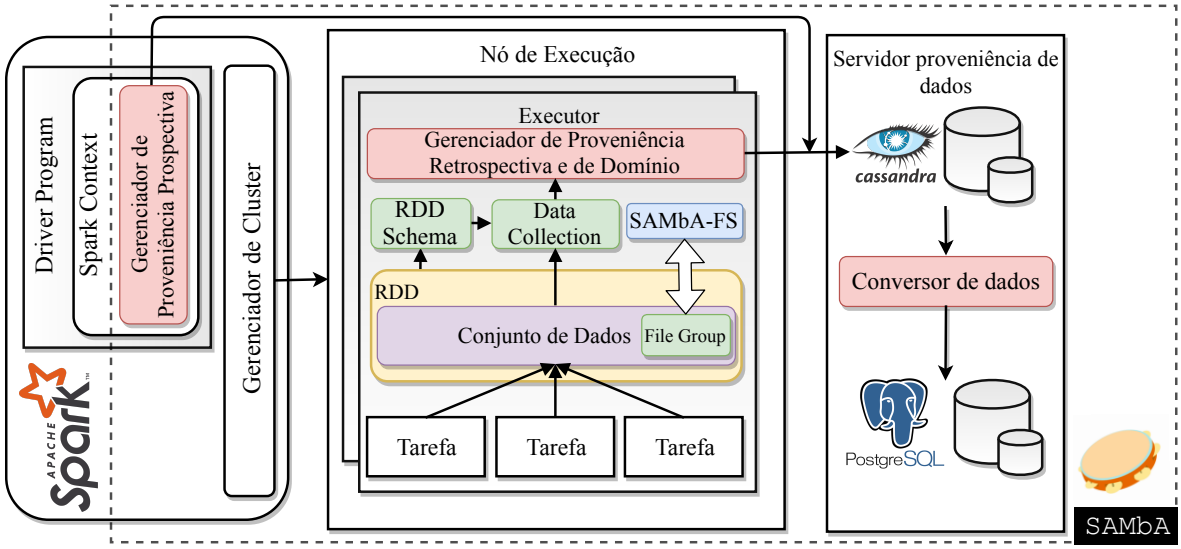


Figura 3.1: Arquitetura do `SAMbA`.

O Gerenciador de Proveniência Prospectiva (`GPP`) fica acoplado ao *SparkContext*, enquanto o Gerenciador de Proveniência Retrospectiva e Domínio (`GPRD`) fica acoplado a cada um dos *Executors* gerenciados pelo *SparkContext*. Uma vez que o processamento dos dados é executado no *Executor*, uma parte substancial do projeto do `SAMbA` é embutido em cada um dos *Executors* do Apache Spark. Os dados de proveniência coletados pelos gerenciadores são armazenados no SGBD colunar Cassandra. O `SAMbA` usa o Cassandra para o gerenciamento iterativo de dados de proveniência. A escolha pelo SGBD

Cassandra como primeira opção se deve ao fato que esse sistema supera, em termos de vazão de escrita, a maioria dos competidores, como o HBase, VoltDB e MySQL [1].

Não obstante, o `SAMbA` também fornece um `Conversor de dados` para analisar e converter os dados de proveniência do Cassandra para o SGBD relacional PostgreSQL. Dessa forma, os usuários podem realizar consultas *post-mortem* aos dados de proveniência em alto nível com base em funções Cassandra CQL ou por meio de rotinas PL-SQL, desde que o `Conversor de dados` tenha sido invocado antes.

3.1.1 Gerenciador de Proveniência Prospectiva

O módulo Gerenciador de Proveniência Prospectiva coleta dados relacionados às transformações que serão aplicadas aos dados durante a execução do *workflow*. Ele se concentra na estrutura do RDD, que passa a ser estendido pelo `SAMbA` e conter novos atributos e métodos, os quais são apresentados ao decorrer dessa seção. Um RDD passa a ser salvo no banco de dados de proveniência logo após a aplicação de uma transformação ou ação sobre os dados que ele representa.

Cada RDD tem a ele associado um identificador e também uma lista de identificadores dos outros RDDs dos quais ele depende, o que caracteriza o fluxo das transformações. Embora não seja obrigatório, os usuários podem rotular cada uma dessas transformações para aprimorar a semântica das consultas de proveniência no banco de dados resultante. A ação de dar nome às transformações é feita por meio do método `setName(nome:String)`, que é nativa da estrutura do Spark RDD.

```
val sc   = new SparkContext(...)
val texto1 = sc.readText("texto1.txt")
               .setName("Ler Arquivo 1")
val texto2 = sc.readText("texto2.txt")
               .setName("Ler Arquivo 2")
val uniaoDosArquivos = texto1.union(texto2)
               .setName("Uniao")
val algumaTransformacao = uniaoDosArquivos
                  .map(...)
                  .setName("Transformacao")
val outraTransformacao = algumaTransformacao
                           .map(...)
                           .ignoreIt()
outraTransformacao.foreach(...)
```

Listagem 3.1: Exemplo de um conjunto de transformações.

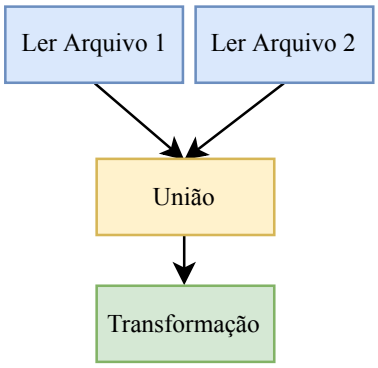


Figura 3.2: Grafo de Transformações para o exemplo da Listagem 3.1.

Por padrão, toda transformação terá seus dados persistidos no banco de dados de

proveniência. Para ignorar quaisquer delas, o `SAMbA` fornece o método `ignoreIt()`. Essa funcionalidade é especialmente útil para ignorar transformações com baixo valor semântico, tais como transformações que fazem somente conversões entre tipos de dados. Como resultado, diminui-se também o uso de armazenamento secundário para transformações pouco relevantes, além do tempo gasto em sua persistência. A Figura 3.2 apresenta um grafo que será capturado pelo gerenciador de proveniência prospectiva para o código em Scala da Listagem 3.1. O código consiste em cinco transformações: realiza a leitura de dois arquivos, faz a união de conteúdos e, por fim, aplica duas transformações genéricas.

Armazenar todo o conteúdo das instâncias de dados (coleções de dados) consumidos e produzidos pelas transformações de um *workflow* pode ser custoso, até mesmo impraticável, devido ao grande volume de dados que será transferido da memória principal para o banco de dados de proveniência. No entanto, é muito provável que o usuário desenvolvedor da aplicação conheça os dados do domínio. Nesse sentido, foi criada criamos uma abstração para o `SAMbA`, denominada `RDD Schema`, com a qual os usuários podem definir exatamente quais são os atributos de interesse a serem extraídos da coleção de dados em uma estratégia similar à de casamento de *templates*.

O `RDD Schema` consiste em uma interface com dois métodos: `getFieldsNames()` e `splitData()`. Essa interface deve ser implementada pelo usuário para se adequar a especificidade de cada transformação do seu *workflow* científico e ser associado a um RDD através do método `setSchema(schema)`. O método `getFieldsNames()` retorna a lista de nomes dos atributos marcados como de interesse pelo usuário. Todos os atributos marcados por esse método são registrado como proveniência prospectiva.

O método `splitData()` realiza a extração dos dados para os atributos definidos no método anterior, ou seja, faz a extração dos elementos de dados que correspondem à definição do fluxo de dados. Essa rotina, utilizada pelo Gerenciador de Proveniência Retrospectiva e de Domínio, recebe como parâmetro uma instância de dados de domínio (coleção de dados) e retorna uma matriz, onde cada coluna representa um atributo e cada linha uma instância de elementos de dados.

```
class ExemploDeUmSchemaRDDParaString extends DataElementSchema[String]{

  override def getFieldsNames(): Array[String] = Array("Atributo 1", "Atributo 2")

  //Exemplo de uma Instancia de dados (Colecao de Dados)
  //valor = "Linha 1 Valor1;Linha 1 Valor 2;Linha 2 Valor 1;Linha 2 Valor 2"
  override def getSplitedData(valor: String): Array[Array[String]] = {
```

```
  val data = valor.split(";")
  return Array(
      Array(data(0), data(1)),
      Array(data(2), data(3))
    )
  }
}
```

Listagem 3.2: Exemplo de um `RDD Schema`.

Na Listagem 3.2 é apresentado um exemplo de código em Scala que implementa um `RDD Schema`. O `RDD Schema` em questão tem dois atributos (`‘‘Atributo 1’’, ‘‘Atributo 2’’`) e cada coleção de dados é composta por dois elementos de dados (*`valor`*). Para a extração dos valores dos elementos de dados é realizada a quebra do conteúdo de acordo com o caráter de separação ‘;’.

O `SAMbA` fornece uma versão padrão do `RDD Schema` para os casos em que os usuários não queiram especificar ou implementar essa abstração. O `RDD Schema` padrão contém atributos que variam de acordo com o tipo do RDD. Por exemplo, se um *PairRDD* for usado, o `RDD Schema` inclui dois atributos, a saber, *key* e *value* para a criação de Elementos de Dados compostos por pares chave-valor. Caso o RDD seja do tipo *FileGroup* (Seção 3.1.3), onde cada instância de dados representa um diretório do sistema operacional, o `RDD Schema` padrão contém três atributos, a saber: *nome do arquivo*, *caminho até o arquivo* e o *tamanho do arquivo*. Por fim, para os demais casos, é fornecido um `RDD Schema` com apenas um atributo, chamado *value*. Nesse caso, o valor do elemento de dado, de uma instância de dado, é obtido através da invocação método `.toString()` disponível em todas as classes Scala.

3.1.2 Gerenciador de Proveniência Retrospectiva e de Domínio

No Apache Spark, cada partição de um RDD consiste em um conjunto de instâncias de dados, manipulados através de *Iterators* do tipo `RDD[Tipo De Dados]`). Para permitir o rastreio do fluxo de dados, e sua posterior persistência no banco de dados de proveniência, criamos uma abstração chamada `DataCollection[Tipo De Dados]`, que representa uma coleção de dados para a abstração do fluxo de dados.

Essa abstração “envelopa” as instâncias de dados de acordo com a aplicação de um `RDD Schema`. Além do dado, cada `DataCollection` tem um identificador, uma lista de identificadores de outros `DataCollections` dos quais depende, o identificador da transformação

que o produziu e uma variável de controle para indicar se ele deve ou não ser persistido na base de dados de proveniência. Portanto, essa modificação realizada pelo `SAMbA` no Apache Spark, consiste em trocar o conteúdo das partições dos RDD por suas "versões envelopadas". Na prática, o Apache Spark deixa de manipular instâncias para, de forma transparente, passar a manipular `DataCollections` de instância de dados.

`DataCollections` são persistidos logo após a sua criação através do Gerenciador de Proveniência Retrospectiva e de Domínio. Ao receber um `DataCollection`, o `GPRD` aplica o `RDD Schema`, definido para a transformação, sobre a instância de dados que está envelopada. Ao receber o resultado do método `splitData()`, o gerenciador persiste os elementos de dados e as informações do rastreio do `DataCollection` na base de dados de proveniência.

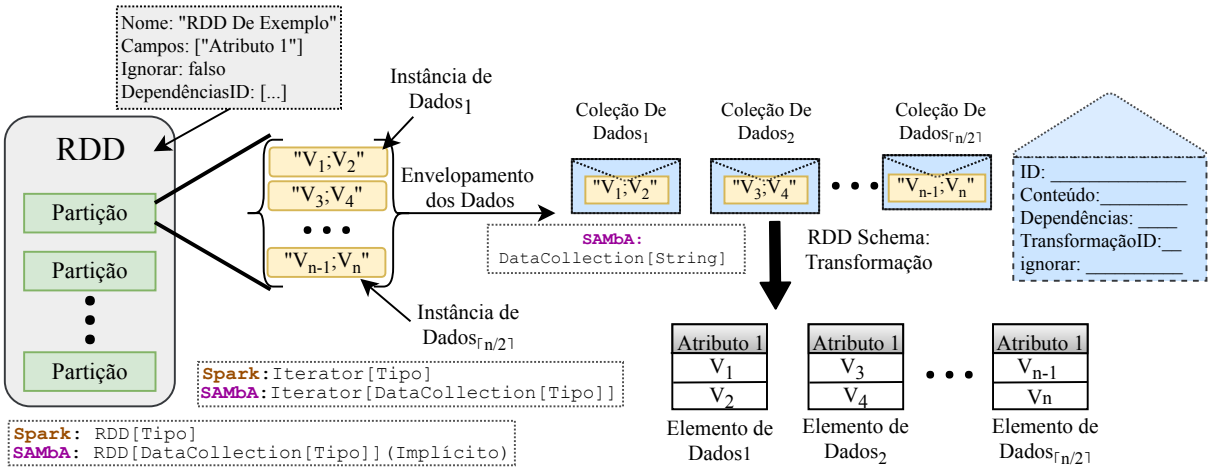


Figura 3.3: Gerenciador de proveniência do `SAMbA` para dados retrospectivos e de domínio. O `SAMbA` recupera as instâncias de dados de partições RDD e as envolve em uma entidade que inclui a instância, a identificação da instância e as suas dependências.

A Figura 3.3 apresenta todo o fluxo realizado pelo `SAMbA` para a obtenção dos elementos de dados. Nesse sentido, em um primeiro momento, é aplicado o `RDD Schema` sobre o RDD em análise, o que permite que as instâncias de cada partição possam ser obtidas individualmente como coleções de dados. Note que os dados dentro de cada partição podem ter sido gerados por transformações anteriores ou carregados do sistema de arquivos, sendo que a aplicação do `RDD Schema` garante que os dados de domínio possam ser adequadamente extraídos. As coleções envelopadas são rotuladas de acordo com informações relevantes para futuras consultas de proveniência. Finalmente, o `SAMbA` aplica a transformação para a obtenção dos elementos de dados das coleções envelopadas e os persiste na base de dados de proveniência.

3.1.3 SAMbA-FS– Mapeamento de arquivos em memória principal

Um caso de particular interesse no projeto para o tratamento de proveniência no Apache Spark está relacionado com o uso de aplicações "caixa-preta" e externas na execução de experimentos científicos. De forma geral, o uso de aplicações externas requer que os usuários invoquem, via chamadas de sistema, o processo que corresponda a aplicação "caixa-preta". Por padrão, o Apache Spark suporta somente aplicações "caixa-preta" que consomem dados pela entrada padrão (*stdin*) e produzem dados pela saída padrão (*stdout*), de forma que o produto final será um RDD de `String`. Portanto, aplicações "caixa-preta" que manipulam arquivos não são suportadas, pois os dados ficam isolados na memória e não são acessíveis a aplicações externas. A comunicação entre o processo baseado no Apache Spark e a aplicação externa é feita por meio de dados em memória secundária – arquivos.

Nesse contexto, existem duas formas principais de se executar aplicações "caixa-preta" que executam computações sobre arquivos no Apache Spark. A primeira, envolve manter os arquivos em um sistema de arquivos distribuído, e utilizar o Apache Spark somente para gerenciar a execução dos programas externos. O ponto negativo dessa abordagem é que se perde as duas principais vantagens do *framework* Apache Spark: o uso da localidade dos dados e o processamento em memória principal. A segunda abordagem envolve manter todos os arquivos em memória principal e, quando da chamada de aplicações externas, persistí-los em memória secundária para que possam ser acessados pela aplicação externa. Essa abordagem gera uma sobrecarga não-desprezível no tempo de execução de *workflows* compostos por atividades que executam essas operações.

Portanto, é relevante tratar eficientemente a manipulação e gerenciamento dos arquivos consumidos (ou produzidos) por essas aplicações externas, uma vez que essas operações de E/S tendem a ser "gargalos" da execução. Com o intuito de reduzir esse potencial obstáculo e agilizar o processamento dos experimentos científicos e a captura de proveniência, implementamos um protótipo de sistema de arquivos em memória para o Apache Spark, um componente que chamamos de `SAMbA-FS`. Esse componente foi construído com o apoio da biblioteca `libfuse`[2] do Linux e tem por objetivo permitir o acesso aos dados que estão na memória principal através de diretórios que serão criados no sistema de arquivos do sistema operacional. Através desses diretórios, as aplicações "caixa-preta" se tornam capazes de acessar o conteúdo da memória do Apache Spark, sem que seja necessário mover os dados entre a memória principal e secundária.

Essa estrutura faz com que o `SAMbA` passe a ser o responsável por gerenciar a E/S

[2] `https://github.com/libfuse/libfuse`

de arquivos, ao invés de delegar essas rotinas a um sistema de arquivos distribuído. Tal característica permite manter o comportamento *shared-nothing* do Apache Spark, uma vez que não é necessário o uso do sistema de arquivo distribuído entre os nós de processamento para gerenciar os arquivos intermediários produzidos. Finalmente, manter o conteúdo dos arquivos em memória principal possibilita que *workflows* científicos não apenas se beneficiem do processamento do Apache Spark com a diminuição de latência de E/S, mas também permite que o `SAMbA` seja capaz de tratar o aspecto de proveniência de dados de domínio com maior eficiência e transparência uma vez que o `SAMbA` se torna ciente dos resultados produzidos durante a execução de um experimento.

A principal abstração do `SAMbA-FS` é o `File Group`, um tipo de dados que é utilizado para representar um grupo de arquivos em memória. Um `File Group` é composto por três atributos:

- Um vetor de `FileElement`, que representa o conteúdo dos arquivos,
- Um mapa *hash*, para atribuir informações extras ao *File Group*, *e.g.,* nome do arquivo ou parâmetros para aplicações que irão consumir o `File Group`;
- E um nome, utilizado para realizar a sua identificação posterior (Seção 3.1.4).

Um `File Group` pode ser criado de duas formas, a partir de um conjunto de *bytes* ou através do carregamento dos dados a partir de arquivos. Além dos operadores comuns do Apache Spark, um `RDD` do tipo `File Group` tem dois novos operadores: o `runCommand()` e o `runScientificApplication()`. O primeiro é usado para a execução de comandos únicos do sistema operacional, enquanto o segundo tem por objetivo executar uma sequência de comandos que podem incluir diversas chamadas a aplicações externas. Assim, ao se executar um desses operadores para cada *File Group*, o `SAMbA-FS` associa seu conteúdo em memória com um diretório temporário e, em seguida, o `SAMbA` executa a aplicação "caixa-preta" sobre o diretório.

3.1.4 Controle de versão

Um último aspecto, considerado em nosso projeto é a possibilidade de se incluir um mecanismo de controle de versão para o conteúdo de arquivos relacionados as aplicações externas, com o objetivo de enriquecer a capacidade do `SAMbA` em tratar consultas analíticas de proveniência. A idéia é que cada execução de um experimento no `SAMbA` seja representada por um *branch* no sistema de controle de versão. Portanto, quando uma

aplicação externa "caixa-preta" é invocada através de um dos novos operadores disponíveis para um RDD de *File Group*, o *File Group* resultante de operação é armazenado em um repositório externo usando a ferramenta Git[3].

Para o gerenciamento de versões de *File Groups* produzidos por outros operadores do Apache Spark, como a operação de filtro, o usuário deve requisitar explicitamente tal desejo através do método `persistFileGroupInGit()`. No repositório versionado, os arquivos representados por um `FileGroup` são armazenados com o seguinte caminho relativo `../[RDD Name]/[FileGroup Name]`, *e.g.*, se um exemplo roda a aplicação externa `Mafft` para gerar um RDD de mesmo nome com relação ao `FileGroup ORTHOMCL1`, então o caminho do diretório dos arquivos modificados será `/Mafft/ORTHOMCL1`. Esse mecanismo de controle de versão possibilita que usuários comparem os resultados de diferentes execuções do mesmo experimento considerando diversas versões do mesmo arquivo proveniente das mesmas transformações.

3.1.5 Banco de dados de proveniência

O `SAMbA` armazena dados dos três aspectos de proveniência em um SGBD externo durante a execução dos experimentos científicos no Apache Spark. Desta forma, a solução fornece aos usuários uma visão antecipada de proveniência na medida em que os dados são processados. O SGBD padrão usado para o armazenamento de dados de proveniência é o SGBD Cassandra[4]. Se, por um lado, foi considerada como critério de escolha do SGBD a maior vazão no momento de escrita dos dados, por outro lado, também destaca-se que a linguagem de consulta do SGBD Cassandra CQL é limitada na comparação com a linguagem SQL enriquecida, *e.g.*, PL-SQL, suportada por SGBD relacionais. O `SAMbA` trata esse *trade-off* ao fornecer o componente de conversão de dados para gerar um *schema* relacional para o SGBD PostgreSQL equivalente ao *schema* colunar do SGBD Cassandra. Assim, os usuários também podem executar consultas analíticas *post-mortem* com SQL enriquecido.

Por último, mas não menos importante, o projeto da solução para o tratamento de proveniência implementado também inclui uma interface *web* com relatórios em tempo de execução pré-definidos, que são executados sobre a base de dados de proveniência armazenada no SGDB Cassandra. Por meio dessa interface, os usuários podem verificar a lista de experimentos que já foram executados e os que estão em execução. Assim, os usuá-

[3] `https://git-scm.com`

[4] O schema colunar do Cassandra para o `SAMbA` está disponível em: `https://gist.github.com/thaylongs/5b2e1cbce7eeb2c8fecca9befa664c89`

rios podem escolher qualquer uma dessas execuções para verificar maiores informações. O relatório de proveniência se inicia com o tempo de início e fim de execução (quando houver) e um grafo da sequência de todas as transformações realizadas. A partir desse grafo, os usuários podem escolher visualizar todo o fluxo de dados ou todas as coleções de dados produzidas ou consumidas por uma transformação específica. Sempre que os usuários selecionarem uma coleção de dados em particular, uma tabela com os elementos de dados é apresentada. Além disso, se a coleção de dados for oriunda ou associado a um `FileGroup`, todos os arquivos também podem ser diretamente acessados.

3.2 Avaliação Experimental

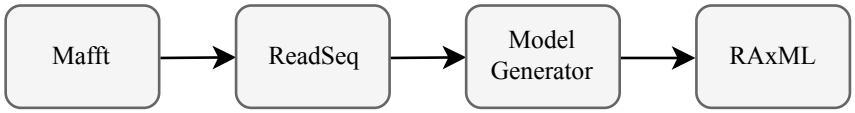


Figura 3.4: Fluxo do *workflow* `SciPhy`.

Nós avaliamos o `SAMbA` para a coleta e consulta dos três aspectos de proveniência com relação a um caso real de um *workflow* de bioinformática, denominado `SciPhy` [56]. Esse *workflow* é, essencialmente, composto por um conjunto de blocos encadeados de programas externos para a construção e avaliação de árvores filogenéticas como detalhado na Figura 3.4 e constitui um método para a descoberta de drogas baseada na análise de genomas.

Nesse contexto, o descobrimento de drogas depende de sequências de determinados organismos, enquanto os valores faltantes são frequentemente inferidos a partir de sequências homólogas. Portanto, dado um conjunto de uma nova sequência genômica, o objetivo do `SciPhy` é encontrar árvores filogenéticas que possam fornecer as melhores alternativas para inferir os valores faltantes. O *workflow* inspeciona entradas de bases de dados consolidadas de genomas de protozoários e infere relações filogenéticas de enzimas-alvo de drogas encontradas na análise dos genomas.

O `SciPhy` representa um problema limitado pela capacidade de processamento. Na avaliação experimental, foram lidos 197 arquivos de entrada e gerados 4.531 arquivos brutos de saída. O *workflow* em si está composto de quatro atividades: *(i)* alinhamento múltiplo de sequências, *(ii)* conversão de alinhamento, *(iii)* busca pelo melhor modelo evolucionário e *(iv)* construção de árvores filogenéticas. As atividades são executadas pelas seguintes aplicações "caixa-preta": *(i)* `Mafft`, *(ii)* `ReadSeq`, *(iii)* `ModelGenerator` e

(iv) RAxML. As comunicações entre essas aplicações ocorrem por arquivos.

A avaliação tem por objetivo explorar todos os módulos criados ou estendidos pelo SAMbA para ilustrar as capacidades da solução proposta com esse estudo de caso. Os experimentos com o SAMbA medem o desempenho geral do sistema com relação ao armazenamento e recuperação de dados do *workflow* por meio de dois critérios complementares:

1. Esforço computacional, que considera o tempo gasto para a coleta de dados de proveniência (incluindo dados de domínio das aplicações externas) em comparação com o tempo gasto apenas na execução do *workflow* pelo Apache Spark. Também foi considerado o impacto de fatores associados, tais como o uso de repositórios externos para o armazenamento de arquivos dados brutos.

2. Análise de Dados, que avalia a capacidade do SAMbA em fornecer relatórios e consultas sobre dados de proveniência durante e após a execução dos experimentos. Além dos relatórios disponíveis na interface *web*, foram submetidas oito consultas, definidas por especialistas do domínio do *workflow*, por meio de expressões SQL de alto nível. As consultas submetidas são diversificadas para cobrir a análise de dados de proveniência prospectiva, retrospectiva e de domínio.

3.2.1 Configurando o SAMbA

Uma vez que o SciPhy depende de aplicações "caixa-preta" para executar suas transformações chave, o RDD Schema deve conter os atributos de interesse que serão extraídos dos arquivos e que serão ser armazenados na base de proveniência. A Figura 3.5 mostra o exemplo de um RDD Schema para o gerenciamento dos arquivos relacionados com o programa ReadSeq. Nesse caso, foram selecionados três atributos de domínios FILE_NAME, NUM_ALIGNS e LENGTH, que serão representados como abstrações de elementos de dados no SAMbA. De acordo com o exemplo, o Gerenciador de Dados de Proveniência Retrospectiva e de Domínio é capaz de ler os conteúdos dos RDDs e persistir os dados de interesse, extraídos pelo usuário, no banco de dados de proveniência.

Na Figura 3.3 é apresentada a modelagem do *workflow* SciPhy através do SAMbA. O código em questão lê o arquivo inputFastaList.txt, o qual contém a lista de arquivos de entrada para o *workflow*. Para cada arquivo de entrada, é criado um File Group Template, que são utilizados para gerar o RDD de entrada. Por fim, após a definição dos RDD Schema's e dos File Group's, o SAMbA executa o *workflow* SciPhy ao invocar todos os programas externos em cascata por meio da rotina runScientificApplication().

```
//Implementacao do RDD Schema para a Atividade ReadSeq
class ReadSeqSchema extends SingleLineSchema[FileGroup] {

  override def getFieldsNames(): Array[String]
      = Array("FILE_NAME", "NUM_ALIGNS", "LENGTH")

  override def splitData(value: FileGroup): Array[String] = {
   //Entre os arquivos presente no FileGroup, identifique o primeiro que termine com ".phylip"
    val file = value.getFileElements. filter ( file => file.getFileName.endsWith(".phylip")).head
   //Pega a primeira linha do arquivo
    val line = Source.fromInputStream(file.getContents.toInputStream).getLines().next()
    val data = line.trim. split (" ")
    Array( file .getFileName, data(0), data(1))
  }
}
```

Figura 3.5: Um exemplo de um fragmento do `RDD Schema` para a coleta de proveniência de aplicações "caixa-preta" do `SciPhy`.

```
 //Metodo que le o arquivo com a lista de arquivos de entrada do workflow e cria os Files Group
  def parserInputFile( fileLocation : String): Seq[FileGroupTemplate] = {
    val scanner = new Scanner(new File(fileLocation))
    scanner.nextLine()
    val list = new ArrayBuffer[FileGroupTemplate]()
    while (scanner.hasNext) {
        //Nome do Arquivo;Caminho ate o Arquivo
        val linha = scanner.nextLine(). split (";")
        val nome = linha(0)
        val filePath = linha(1)
        list += FileGroupTemplate.ofFile(new File(filePath), false, Map("NAME" -> nome))
    }
    return list
  }
 // Experimento SciPhy usando Spark
 val fileGroup = new SparkContext(
    new SparkConf().setMaster("local[4]").setAppName("SciPhy")
      .setScriptDir("/workflows/sciphy/scripts")
 ).fileGroup(parserInputFile("inputFastaList.txt"): _*)//FileGroup
 //{{NAME}} e uma chave no HashMap presente em cada FileGroup
 fileGroup
    . runScientificApplication ("mafft.cmd {{NAME}}")
          .setName("Mafft")
    . runScientificApplication ("readseq.cmd {{NAME}} {{NAME}}.mafft")
          .setName("ReadSeq")
          .setSchema(new ReadSeqSchema)
```

```
    .runScientificApplication("modelgenerator.cmd {{NAME}}")
        .setName("Model Generator")
        .setSchema(new ModelGeneratorSchema)
    .runScientificApplication("raxml.cmd {{NAME}}.phylip {{NAME}}.mg.modelFromMG.txt")
        .setName("RAxML")
        .setSchema(new RAxMLSchema)
    .saveFilesAt(new File("/workflows/sciphy/output"))
```

Listagem 3.3: Modelagem do Workflow SciPhy através do `SAMbA`.

3.2.2 Esforço computacional

Nesse experimento foi mensurado o tempo gasto pelo `SAMbA` para a captura dos três aspectos de proveniência no `SciPhy`. O *workflow* foi executado seis vezes no *cluster* Lobo Carneiro – LoboC[5] para todos os arquivos de entradas tanto no Apache Spark quanto no `SAMbA`, de forma separada. Foi configurado o ambiente de teste para usar 48 núcleos de processamento, dos quais 44 núcleos foram empregados na execução do *workflow* e os 04 núcleos restantes foram colocados a cargo da persistência de dados no SGBD Cassandra.

Em nossos testes, o *workflow* `SciPhy` foi executado em três sistemas: no SciCumulus, que é um SGWfC no qual o `SciPhy` foi inicialmente implementado e que será utilizado como *baseline*, no Apache Spark nativo e no `SAMbA`. No total o `SciPhy` foi testado em 6 abordagens, as quais são detalhadas a seguir:

- **Abordagem 1**: foi utilizada o SciCumulus;
- **Abordagem 2**: foi utilizado o Apache Spark sem modificações;
- **Abordagem 3**: o `SAMbA` com a proveniência e o Git desabilitados;
- **Abordagem 4**: o `SAMbA` com a proveniência desabilitada e uso de repositórios externos habilitado;
- **Abordagem 5**: o `SAMbA` com a proveniência habilitada mas sem suporte ao versionamento;
- **Abordagem 6**: o `SAMbA` com a proveniência e Git habilitados.

Na Tabela 3.1 é apresentada as relações entre as abordagens que foram executadas e as funcionalidades que elas estão utilizando. O tempo médio gasto nas execuções do `SciPhy`

[5]Especificação completa em: `https://www.nacad.ufrj.br/recursos/sgiicex`

em cada uma das abordagens são apresentados na Figura 3.6, onde o desvio-padrão foi inferior a 0, 5s para todos os casos. Os resultados mostram que o SAMbA superou a execução tanto do SciCumulus quanto do Apache Spark padrão, muito devido ao seu sistema de arquivos em memória. O SAMbA usado sem realizar a coleta de dados proveniência e uso do Git superou o Apache Spark em até 1.3% e o SciCumulus em 40.8%, enquanto que adicionar ou Git ou proveniência diminuiu o ganho de desempenho por apenas uma fração centesimal.

Funcionalidade	Abordagem 1	Abordagem 2	Abordagem 3	Abordagem 4	Abordagem 5	Abordagem 6
Paralelismo	■	■	■	■	■	■
Proveniência	■				■	■
Git				■		■
In-memory filesystem (SAMbA-FS)			■	■		■

Tabela 3.1: Tabela com a relação entre as abordagens avaliadas e os recursos utilizados em cada uma delas.

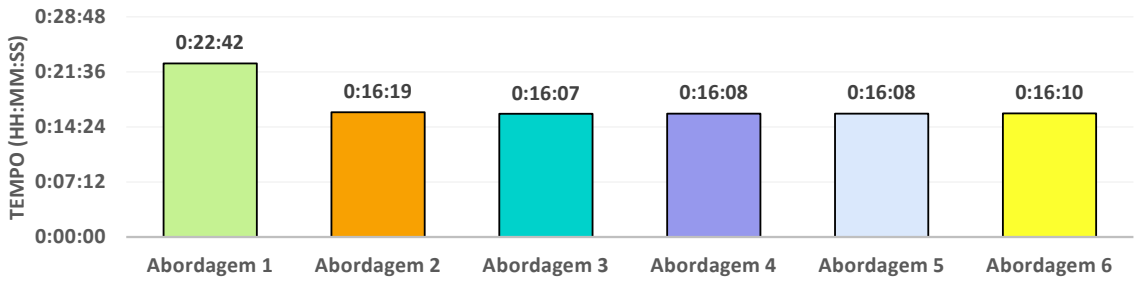
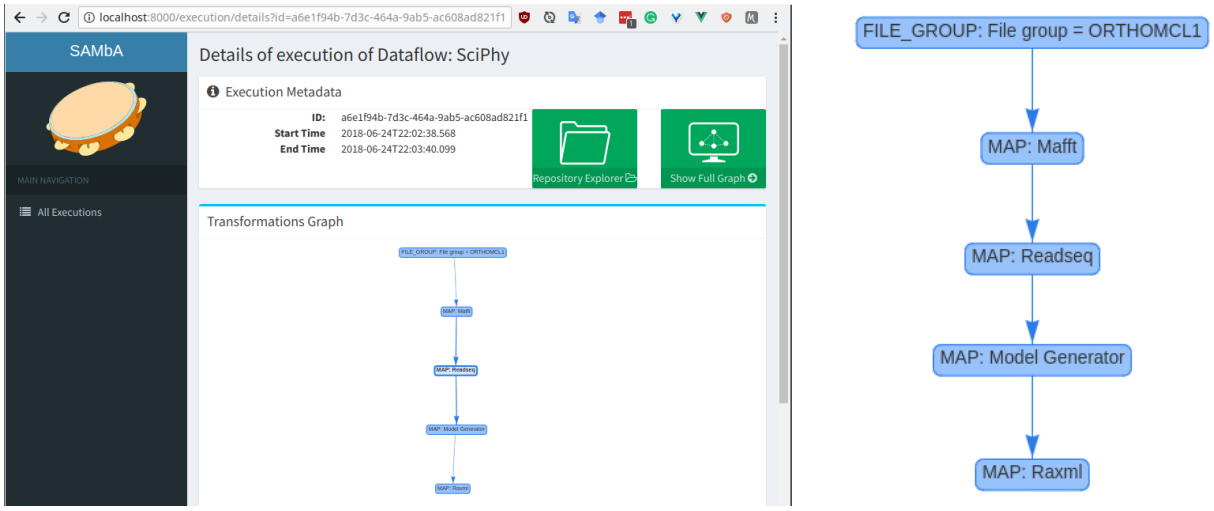


Figura 3.6: Tempo gasto na execução do *workflow* SciPhy em 6 abordagens, variando as funcionalidades utilizadas.

No cenário de interesse dessa dissertação, o SAMbA também superou o Apache Spark na execução do *workflow*. Nesse contexto, o SAMbA persistiu todos os dados de proveniência de interesse para consultas no SGBD Cassandra, armazenou os arquivos de dados brutos em repositórios externos e ainda foi ligeiramente superior à execução Apache Spark padrão. Esse resultado indica que a nossa proposta tende a não prejudicar o desempenho geral da execução de experimento científico, uma vez que aspectos de otimização (*e.g.*, sistema de arquivos em memória) foram adequadamente tratados.

De formal geral, com o custo de se manter os dados de transformações oriundas de aplicação externas em memória, o SAMbA foi capaz de: *(i)* gerenciar com eficiência e transparência o conteúdo de arquivos manipulados por aplicações "caixa-preta" e *(ii)* persistir os três aspectos de proveniência em um SGBD externo.



(a) (b)

Figura 3.7: Gráfico do fluxo de transformações para o processamento de um conjunto de entrada do `SciPhy`. (a) Interface *web* para verificar o fluxo de transformações persistidas em disco, para o qual o usuário pode expandir todo o fluxo de dados ou verificar os repositórios associados aos arquivos manipulados por aplicações "caixa-preta" externas. (b) Fluxo de transformações para uma instância de execução do `SciPhy`.

3.2.3 Relatórios em tempo de execução

Foram implementados três relatórios dinâmicos para visualização em tempo de execução. Os relatórios são acessíveis por meio de uma interface *web* que permite exibir os dados armazenados no SGBD Cassandra. Em particular, foram definidos três relatórios dinâmicos, as quais são listadas a seguir:

1. Relatório retrospectivo: Detalha o início e fim (quando a execução já estiver finalizada) do experimento.
2. Gráfico de Transformações: Detalha o gráfico de transformações realizadas para um conjunto de dados de entrada.
3. Gráfico de Fluxo de Dados: Detalha as informações de cada uma das etapas do fluxo de dados.

A Figura 3.7 apresenta tanto a interface *web* quanto o gráfico de transformações para uma instância de execução do `SciPhy`. Através dessa interface *web* (Figura 3.7(a)) os usuários podem expandir o fluxo de dados, e visualizar todas as coleções de dados manipuladas no experimento, ou acessar o repositório que contém os arquivos consumidos e produzidos por aplicações "caixa-preta" externas. A Figura 3.7(b) apresenta o fluxo de transformações detalhado, onde o usuário pode expandir cada um dos nós.

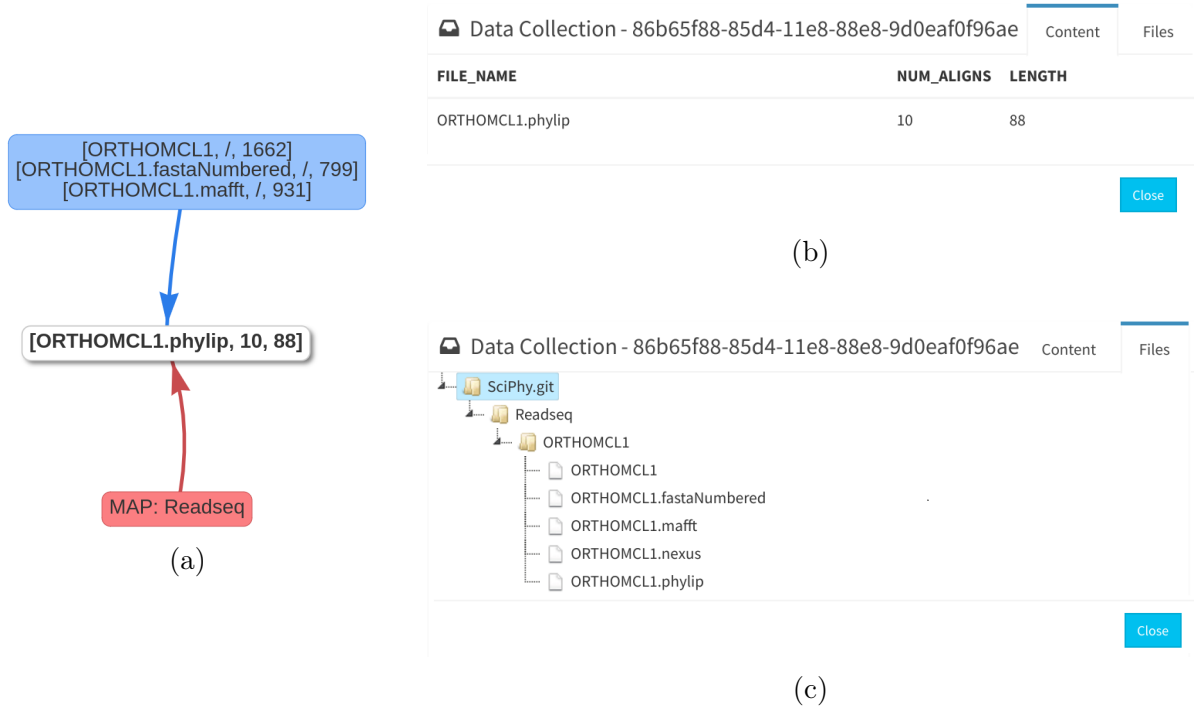


Figura 3.8: Visualização das coleções de dados envolvidos em uma transformação. (a) Sub-fluxo associado a uma transformação. (b) Valores dos elementos de dados consumidos em uma transformação nativa. (c) Acesso aos arquivos representados em um `File Group`.

Ao clicar em uma transformação, é carregada dinamicamente uma nova página com o sub-fluxo de dados do nós em questão. Antes da exibição, os usuários devem escolher se querem visualizar, ou não, as coleções de dados consumidas pela transformação. As coleções consumidas e produzidas são mostradas na interface da Figura 3.8(a) como retângulos azuis e brancos, respectivamente. Nessa interface, ao clicar em uma coleção, é apresentado uma nova janela com uma tabela com os elementos de dados da coleção selecionada, como exemplificado na Figura 3.8(b). Caso a coleção de dados selecionada seja um `File Group`, além da tabela com os elementos dados é apresentado, em uma nova aba, a árvore de diretórios com os arquivos que compõem o `File Group` – Figura 3.8(c). Nessa árvore, basta o usuário selecionar um dos arquivos para realizar o seu *download*.

Também é possível expandir todo o fluxo de dados de uma execução do `SciPhy`, a partir da interface da Figura 3.7. Para isso, o relatório Gráfico de Fluxo de Dados é invocado e o resultado é exibido em uma segunda página – Figura 3.9. Nessa interface, o usuário pode visualizar todo o fluxo de dados para um conjunto de dados de entrada e também explorar as suas transformações para investigar eventuais resultados intermediários.

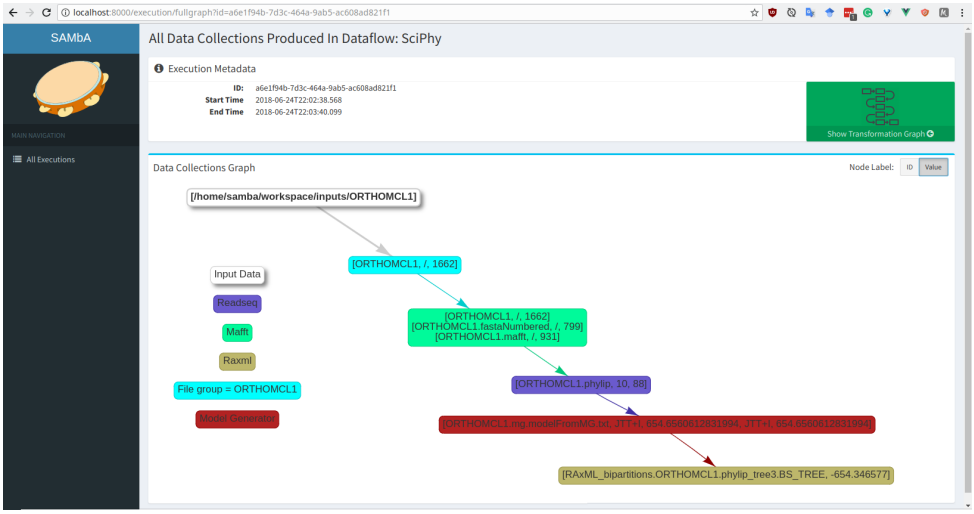


Figura 3.9: Visualização do fluxo de dados de uma execução do `SciPhy`.

3.2.4 Análise de dados

Para a análise de dados de proveniência *post-mortem* foram utilizadas oito consultas definidas por especialistas no *workflow* `SciPhy`. Nós categorizamos as oito consultas em termos de seus aspectos de proveniência. A Figura 3.10 mostra a divisão das consultas em conjuntos não-disjuntos com relação aos dados de proveniência prospectiva e retrospectiva e de domínio.

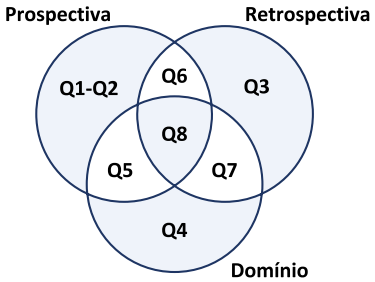


Figura 3.10: Categorias das consultas de proveniência *post-mortem* no *workflow* `SciPhy`.

As consultas de proveniência prospectiva analisam as especificações do fluxo de dados do `SciPhy`, tais como as transformações no *workflow*, os atributos consumidos e produzidos pelas transformações e a dependência de dados. As consultas de proveniência retrospectiva investigam os caminhos nos fluxos de dados, tais como o histórico das atividades de processamento do *workflow*. Por último, as consultas por dados de domínio investigam os dados de interesse dos usuários do domínio do `SciPhy`.

A Tabela 3.2 descreve as oito consultas seguindo a categorização da Figura 3.10. Cada consulta pode ser combinada com as outras sete para buscar múltiplos aspectos de proveniência. Por exemplo, as consultas `Q3` e `Q7` podem ser combinadas para encontrar os

Tabela 3.2: Descrição das oito consultas definidas por especialistas para consulta do Banco de Dados de Proveniência do `SAMbA`.

Consulta	**Descrição**
Q1	Recuperar todas as transformações de dados de uma execução específica.
Q2	Recuperar o nome e o tipo de todos os atributos com relação a um conjunto de dados de entrada.
Q3	Recuperar todos os arquivos gerados durante as duas últimas horas.
Q4	Recuperar o phylip e o número de alinhamentos para a execução do programa `ModelGenerator` cujos resultados alcançaram um valor para o primeiro modelo obtido maior que 1.500.
Q5	Recuperar todas as árvores filogenéticas, *i.e.*, elementos de dados, do conjunto `ds_raxml` depois da execução do `SciPhy`.
Q6	Recuperar o tempo inicial e final de uma execução `SciPhy`.
Q7	Recuperar o tempo gasto por execuções do `SciPhy` que geram, pelo menos, quatro árvores filogenéticas nas quais o *bootstrap* é maior que 90%.
Q8	Recuperar o algoritmo usado na transformação do `SciPhy` que gera árvores filogenéticas com *bootstrap* médio maior que 70%.

Tabela 3.3: Exemplos de expressões SQL submetidas ao banco de dados de proveniência.

Q2	`SELECT att.name, att.type`
	` FROM public.tb_attributes att`
	`  INNER JOIN tb_transformations dt`
	`  ON dt.id= att.task_id`
	` WHERE dt.description='Mafft'`

Q4	`SELECT rs.''FILE_NAME'', rs.''NUM_ALIGNS''`
	` FROM sciphy.ds_readseq rs`
	`  INNER JOIN sciphy.rel_ds_readseq_ds_model_generator rel`
	`  ON rel.prev_id = rs.id_data_element`
	`  INNER JOIN sciphy.ds_model_generator mg`
	`  ON mg.id_data_element = rel.next_id`
	` WHERE mg."PROB1" > 1500`
	`  AND mg.id_execution='id'`

Q6	`SELECT exec.''startTime'', exec.''endTime''`
	` FROM public.tb_execution exec`
	`  INNER JOIN public.tb_dataflow df`
	`  ON df.id = exec.dataflow_id`
	` WHERE df.name='Sciphy'`

arquivos nos resultados parciais que satisfaçam as condições de *bootstrap* maior que 90% e que também foram executadas nas últimas duas horas. Analogamente, a Tabela 3.3 fornece expressões em SQL padrão para a execução das consultas por proveniência. Em

particular, nós detalhamos as consultas `Q2`, `Q4` e `Q6` que ilustram aspectos de proveniência retrospectiva, prospectiva e de domínio e retrospectiva e de domínio, respectivamente.

Por último, mas não menos importante, nós medimos o tempo gasto na execução das consultas SQL após a execução de seis instâncias do `SciPhy`. Os dados de proveniência do SGBD Cassandra, com dados de 5 execuções, foram convertidos para o *schema* relacional do SGBD PostgreSQL por meio do Conversor de Dados do `SAMbA`[6], cujo tamanho médio final foi de 2.1MB. A Figura 3.11 detalha o tempo médio de execução das consultas `Q2`, `Q4`, e `Q6` após 10 execuções sem uso de *caching*. A consulta `Q6` foi mais rápida que os competidores, pois usa predicados menos seletivos.

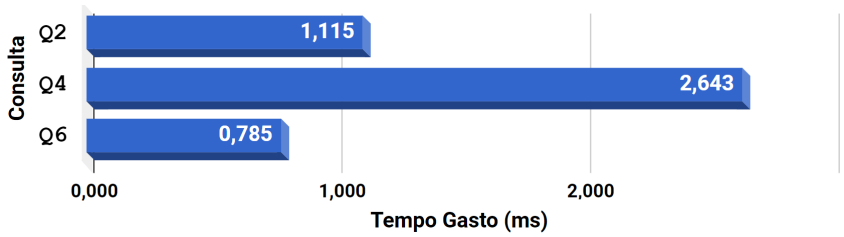


Figura 3.11: Tempo médio gasto na execução das consultas de proveniência.

3.3 Conclusões Parciais

Nessa seção, foi discutida uma solução prática para o armazenamento e recuperação de dados de proveniência de *workflows* científicos no Apache Spark. Em particular, foi apresentada a arquitetura da solução `SAMbA`, que estende os componentes do Apache Spark para a captura e gerenciamento de proveniência. As funcionalidades da solução proposta foram avaliadas em um *workflow* científico real e os resultados indicam:

- O `SAMbA` não prejudicou o desempenho do Apache Spark.
- O `SAMbA` foi capaz de fornecer uma análise de dados eficiente sobre os três aspectos de proveniência consultados. A proveniência pode ser consultada por: *(i)* relatórios, na interface *web* e *(ii)* linguagens de alto nível, após conversão de dados.

Todos os experimentos foram realizados considerando possíveis otimizações, tais como o uso do `RDD Schema` para a extração de dados de domínio de interesse, a implementação do `SAMbA-FS` para manipular dados em memória principal e o uso de ferramentas de versionamento de arquivos para armazenar o resultado de transformações conduzidas por aplicações externas.

[6]A base de dados relacional e as consultas SQL estão disponíveis em: `https://drive.google.com/file/d/101P_Hxwsek8n6M7YlSoCxEJooHICM538/view?usp=sharing`.

Capítulo 4

Consultas sobre dados de domínio

Os arquivos brutos gerados por transformações oriundas de aplicações "caixa-preta" são versionados e armazenados em um repositório externo durante a execução do *workflow* científico no `SAMbA` acoplado ao Apache Spark e com suporte ao Git. Nesse sentido, é também importante que os usuários sejam capazes de consultar com eficiência os dados de domínio nesses arquivos bruto externos. Nesse capítulo, tratamos esse segundo momento, no qual os responsáveis pela análise dos dados obtidos dos *workflows* científicos precisam consultar os dados em arquivos oriundos dos experimentos. O usuário pode escolher não tratar esse desafio diretamente no `SAMbA`, uma vez que:

- O usuário pode não saber, ou não definir, quais são os atributos de interesse que precisam ser extraídos de uma execução.
- A extração leva a uma pequena sobrecarga computacional e o usuário pode optar por evitá-la a fim de executar o experimento mais rapidamente.
- O usuário pode querer explorar um atributo que, inicialmente, era desinteressante se comparado a outros atributos do experimento.

Nesse contexto, o tratamento de experimentos científicos no Apache Spark com suporte à proveniência também inclui os desafios relacionados a consulta sobre os dados de domínio armazenados em arquivos resultantes do experimento, a saber:

1. A forma mais adequada de se indexar dados de experimentos científicos. No contexto do `SAMbA`, dados de domínio são salvos em arquivos brutos, sendo que índices específicos podem ser construídos para se consultar essas estruturas. Portanto, faz-se necessária a avaliação do desempenho de métodos de indexação para a consulta de dados de domínio.

2. A forma mais eficiente para se consultar dados de experimentos científicos. Consultas sobre dados de domínio podem ser realizadas segundo diferentes paradigmas, tais como consultas *in-situ* e consultas diretamente sobre os índices. Assim, faz-se necessária uma discussão experimental sobre a forma mais eficiente para consultar desse tipo de dados.

Aqui são discutidas e avaliadas duas soluções existentes, que podem ser divididas entre abordagens baseadas em consultas adaptativas e as baseadas na indexação de dados brutos, com relação às vantagens e desvantagens de cada uma na análise de dados de domínio armazenados em múltiplos arquivos brutos.

4.1 Estudo de caso de dados de domínio

Para ilustrar o problema de consultas sobre dados de domínio, suponha a execução de uma simulação de dinâmica de fluidos computacionais, especificamente no problema de análise de deposição de sedimentos em bacias sedimentares. Esse estudo de caso é particularmente interessante, pois envolve um grande volume de dados, descritos por uma grande quantidade de atributos. Portanto, se todos os atributos forem de interesse do usuário, aplicar o `RDD Schema` no `SAMbA` apenas levaria a uma cópia integral dos dados de domínio, ao custo de uma sobrecarga no processamento e armazenamento. Nesse sentido, o usuário pode optar por não definir dados de domínio de interesse e, ao invés disso, consultar os dados brutos gerados pelas transformações.

Assim, aqui se propõe tratar esse cenário tanto com processamento de consultas adaptativas, quanto com indexação de dados brutos. Foram executados os experimentos para esse estudo de caso usando o resolvedor libMesh-sedimentation [15]. Essa aplicação foi criada a partir da biblioteca libMesh[1] para simular correntes turbidíticas, tipicamente encontradas em processos geológicos. Dentro deste contexto, os sedimentos são transportados devido à movimentação do fluido e seguem as regras das equações de Navier-Stokes para o fluido incompressível combinado com a equação de transporte denominada por advecção (concentração de sedimentos).

Todos os dados gerados são armazenados em múltiplos arquivos em formatos de arquivos específicos do domínio: `XDMF` e `HDF5`. Nessa simulação, os resultados parciais e finais das malhas são armazenados em arquivos de dados brutos, enquanto outros dados de interesse são mantidos na memória principal, *e.g.*, variáveis do código do *solver*. Assim,

[1] `http://libmesh.github.io`

o libMesh-sedimentation foi integrado com o ParaView Catalyst [10] para que os dados que estão em memória possam ser extraídos da malha simulada. Os dados de interesse que estão em memória são armazenados em outros aquivos brutos (arquivos `.CSV`).

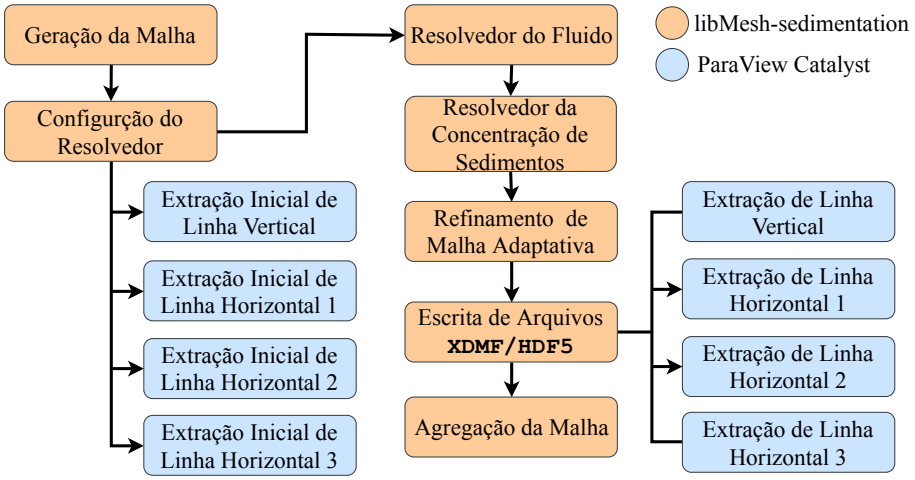


Figura 4.1: Sequência de execução para o experimento de deposição de sedimentos.

A Figura 4.1 apresenta as transformações de dados envolvidas no libMesh-sedimentation (cor laranja), assim como as transformações de dados usando o ParaView Catalyst (cor azul) para a extração de dados brutos. Essa sequência está relacionada à geração de dados científicos, que são armazenados diretamente em arquivos e as informações que estão em memória, as quais são extraídas das estrutura de dados em memória para arquivos `.CSV`. Inicialmente, uma malha é gerada para a simulação e é utilizada para alimentar a entrada do resolvedor libMesh. Nesse ponto, o ParaView Catalyst extrai os dados em memória da simulação das estruturas de dados do resolvedor (linhas horizontais) e os persiste em arquivos. Para as partes iterativas restantes do resolvedor, o libMesh gera arquivos `XDMF` e `HDF5`, que contém os resultados das simulações, tais como, a malha de saída, enquanto o ParaView Catalyst mantém o controle da malha alocada em memória e a armazena em arquivos `HDF5`. Por último, a malha agregada resultante é mantida em um único arquivo `XDMF` com referências aos arquivos `HDF5`.

4.2 Consultas Adaptativas *vs.* Indexação de Arquivos

Conforme discutido nas Seções 2.5 e 4.1, foram elencadas duas abordagens candidatas para acessar, extrair, indexar e consultar dados científicos em arquivos brutos: *consultas adaptativas* e *indexação dos dados em arquivos brutos*, sendo que o PostgresRAW e o FastBit/FastQuery foram identificadas como soluções estado-da-arte para essas duas categorias. Portanto, foram avaliadas tanto o PostgresRAW quanto o FastBit para o es-

tudo de caso dos dados de domínio de deposição de sedimentos, com relação aos seguintes critérios:

1. O tempo gasto no processo de consulta em cada uma das soluções, *i.e.*, o tempo necessário para executar uma consulta, independentemente da preparação dos dados.

2. O tempo-para-consulta (*time-to-query*) exigido para cada solução, *i.e.*, o tempo gasto da preparação de dados até a geração dos conjuntos-resposta.

Foi executado o experimento da simulação de deposição sedimentar resolvida pelo libMesh-sedimentation no *cluster* Lobo Carneiro usando 960 núcleos. Toda a simulação computacional gerou cerca de 47GB de dados, distribuídos em 44.160 arquivos de interesse oriundos das transformações da aplicação. Os dados extraídos incluem 62 intervalos de tempo, *i.e,* iterações do *solver*, divididos, na média, em 480 arquivos por intervalos impares e 960 arquivos por intervalos pares. Essa variação na quantidade de arquivos extraídos é resultado do algoritmo *Adaptive Mesh Refinement*, que é aplicado pelo *solver* em tempo de execução. A Figura 4.2 detalha o tamanho acumulado dos conjunto de dados (em MB) para cada intervalo de tempo.

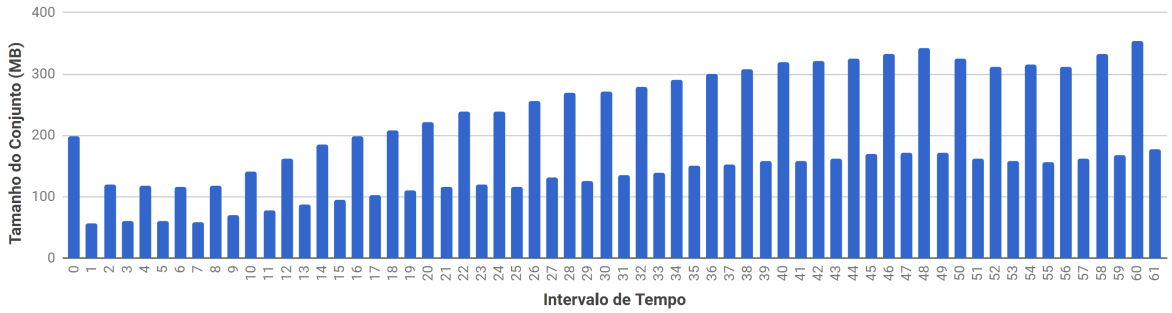


Figura 4.2: Tamanho dos conjuntos de dados para cada intervalo de tempo da simulação computacional de deposição de sedimentos.

Todas as medidas foram tomadas após uma série de dez repetições e todas as consultas foram executadas na mesma infraestrutura de *hardware* em uma Máquina Virtual(VM) no Google Cloud Platform[2]. A VM em questão, tinha oito núcleos de processamento e 30GB de memória RAM e os dados foram armazenados em unidades de SSD. No que se refere a *software*, foi utilizado o Sistema Operacional Ubuntu 16.04.4 LTS, o Fastbit na versão 2.0.3[3] e o PostgresRAW no último *commit* disponível no repositório[4] público, datado do dia 25 de abril de 2018.

[2] `https://cloud.google.com/`
[3] `https://code.lbl.gov/projects/fastbit/`
[4] `https://github.com/HBPMedical/PostgresRAW`

4.2.1 Configurando o PostgresRAW

O PostgresRAW gerencia os arquivos a serem acessados através de um vetor no qual cada posição corresponde a um ponteiro para o arquivo. Esse vetor é inicializado toda vez que uma nova conexão com o banco é aberta. Por padrão, o PostgresRAW é configurado para manipular um vetor com 50 posições, o que é insuficiente para avaliar a simulação computacional do estudo de caso de deposição sedimentar. Entretanto, quando o tamanho dessa matriz é aumentado, seja de forma estática ou dinâmica, isso causa um atraso substancial no tempo de abertura das conexões com o banco de dados. A causa desse atraso está relacionada à estratégia utilizada para preencher o vetor e não a estrutura em si.

Depois de analisar o código-fonte PostgresRAW, onde se faz a leitura do arquivo de configuração que é usado para preencher o vetor, foi observado que se percorre todo o vetor para cada linha do arquivo de configuração com o objetivo de encontrar uma posição vazia para associar à um novo arquivo, ou uma posição cujo identificador é igual ao identificador de arquivo a ser atualizado. Cada posição do vetor tem quarto atributos (a localização do arquivo, o nome da tabela, caracter delimitador e se o arquivo possui cabeçalho ou não). Assim, cada entrada no arquivo de configuração, *i.e.*, um arquivo passível de consulta, tem a seguinte estrutura:

```
filename-ID  = 'Localização do arquivo'
relation-ID  = 'Nome da tabela'
delimiter-ID = ','
header-ID    = 'TRUE | FALSE'
```

Foi modificado o algoritmo de carga do vetor do PostgresRAW para usar o identificador como o índice do vetor. Desta forma, desde que haja memória suficiente, o vetor terá tamanho fixo e o tempo para procurar uma posição no vetor terá complexidade $\mathcal{O}(1)$. Também foi modificado o código fonte do PostgresRAW para lidar com a limitação de não manipular mais do que cinco tabelas ao mesmo tempo. Mais uma vez, o código fonte do PostgresRAW original era limitado por um vetor de tamanho fixo que mantinha metadados sobre a consulta e as tabelas na memória principal, o que exigiu a conversão do vetor de tamanho fixo em uma lista com tamanho dinâmico.

4.2.2 Configurando o FastBit

Ao contrário do PostgresRAW, que depende de consultas adaptativas para gerar seus índices posicionais, o Fastbit permite que os usuários definam parâmetros para indexação e ajuste fino a *priori*. Em particular, o FastBit pode ser configurado com parâmetros para o *binning* ou *encoding*. O parâmetro *binning* determina como o *bitmap* é produzido e, para os testes, foi utilizado o seu valor padrão, isto é, *precision* = 2. Por outro lado, a parametrização do *encoding* requer encontrar um *trade-off* de duas configurações (espaço e desempenho), o que geralmente é obtido após várias avaliações experimentais. Nos experimentos de deposição, foi usada a parametrização *interval-equality*, que é a recomendada [4] para o tipo de dados de domínio avaliada nas consultas.

4.2.3 Construção dos índices

O FastBit gera índices *bitmap* para arquivos de dados brutos, enquanto o PostgresRAW gera referências para esses mesmos arquivos. Em ambos os casos, os índices devem ser gerados *antes* das consultas aos dados. Nesse experimento, foi medido o tempo gasto no processo de indexação, além do espaço em disco necessário para as duas soluções competidoras. A Tabela 4.1 apresenta a comparação do tempo gasto na geração dos índices, com base nos dados extraídos dos arquivos `XDMF` e `HDF5`. O PostgresRAW exige um tempo muito menor (em até 1 ordem de magnitude, na média) para a construção dos índices, quando comparado ao FastBit. Esse resultado é devido ao fato que o PostgresRAW usa uma função incremental em memória para adicionar entradas em seu índice posicional, enquanto o FastBit que se baseia em uma rotina de "passada única" sobre os dados, *i.e.*, essa solução indexa todos os valores de um arquivo de dados bruto. Nesse sentido, espera-se que as primeiras consultas submetidas ao PostgresRAW levem mais tempo do que no FastBit, uma vez elas irão requer a atualização do índice posicional.

		Soluções de indexação	
Medidas		**FastBit**	**PostgresRAW**
Tempo decorrido para Construção dos índices (s)	Média	20.396,07	264,49
	Desvio padrão	1.406,65	0,73
Uso do disco (GB)		18,00	0,53

Tabela 4.1: Comparação do tempo e espaço em disco gasto na geração de índices no FastBit e PostgresRAW para o estudo de caso avaliado.

A Tabela 4.1 também apresenta o espaço em disco necessário para cada uma das abordagens comparadas. Os resultados gerais indicam que o uso do disco segue um

comportamento semelhante ao da geração do índice: o PostgresRAW exige até 30 vezes menos espaço em disco para o armazenamento do índice na comparação com o FastBit. Embora o tamanho dos índices no PostgresRAW deva aumentar sempre que as consultas sobre novos atributos sejam realizadas, o resultado sobre o tamanho dos índices reforça que o FastBit depende muito da estratégia redundante para representar os valores nos arquivos de dados brutos. Portanto, a estratégia do FastBit parece escalar linearmente com o número de entradas × arquivos.

4.2.4 Consulta aos dados de domínio em arquivos brutos

Nessa seção, foram usados os índices construídos para executar um grupo de consultas representativas sobre os arquivos de dados obtidos na simulação de deposição de sedimentos. As consultas escolhidas se baseiam em buscas típicas que os usuários fazem no domínio da simulação. A idéia geral é avaliar seis consultas em SQL padrão aplicadas sobre cada intervalo de tempo da solução, *i.e.*, foram submetidas $6 \times 62 = 372$ consultas. Foi medida a média do tempo necessário para que cada consulta retorne com o conjunto resposta, considerando uma sequência de dez repetições. A Tabela 4.2 descreve as consultas do domínio em termos de seletividade e objetivo.

Rótulo	**Seletividade**	**Descrição**
`Q1`	Baixo	Recupera dados de várias colunas e arquivos ao mesmo tempo.
`Q2 / Q3`	Alto	Recupera valores extremos em várias colunas e arquivos. A `Q2` usa a função de agregação `MAX` do SQL, enquanto a `Q3` aplica a função de agregação `MIN` do SQL.
`Q4`	Baixo	Recupera dados de uma única coluna de vários arquivos.
`Q5`	Médio	Filtra dados de uma única coluna de vários arquivos.
`Q6`	Médio	Filtra dados de várias colunas e arquivos.

Tabela 4.2: Seis consultas de domínio submetidas para cada intervalo de tempo da simulação de deposição de sedimentos.

A consulta `Q1` seleciona dados de todos os arquivos que representam um determinado intervalo de tempo, ou seja, une o conteúdo de todos os arquivos de um determinado intervalo em uma única saída. As consultas `Q2` e `Q3` aplicam funções de agregação do SQL para obter os valores máximo e mínimo de atributos específicos de vários arquivos. A consulta `Q4` seleciona os dados de um único atributo sobre vários arquivos, enquanto que a consulta `Q5` filtra esses valores usando um predicado com seletividade média. Por

fim, a consulta `Q6` filtra todos os atributos após um produto cartesiano, filtrado por um predicado de seletividade média.

A Figura 4.3 apresenta o tempo gasto na execução de cada consulta, acumulado o tempo gasto por intervalos entre 0 e 61 de 10 em 10. A primeira parte de cada gráfico empilhado corresponde aos intervalos de tempo do *solver*, de 0 até 9 e assim por diante. Como exceção, a última parte das barras empilhadas inclui o intervalo de tempo de 50 até 61. O primeiro par de barras na Figura 4.3 apresenta os resultados para a `Q1`, onde o PostgresRAW foi $37,92\%$ mais rápido que o FastBit. Esse comportamento é porque o FastBit precisou ler todos os arquivos de colunas e fazer a junção de cada linha para produzir os resultados para esta consulta.

Para todas as outras consultas, no entanto, o FastBit teve um desempenho melhor que o PostgresRAW. Em detalhes, o FastBit foi $92,79\%$, $98,23\%$, $76,53\%$, $71,61\%$ e $69,57\%$ superior ao PostgresRAW para as consultas `Q2`, `Q3`, `Q4`, `Q5`, e `Q6`, respectivamente. Além disso, a execução das consultas no PostgresRAW apresentaram quase sempre o mesmo comportamento, independentemente da seletividade da consulta.

O FastBit se mostrou sensível aos intervalos de tempo e também à seletividade das consultas. Por exemplo, a soma do tempo gasto nas consultas `Q2`, `Q3` e `Q4` foi menor que a soma do tempo gasto nas `Q5` e `Q6`. Além de seletividade diferente, as consultas `Q5` e `Q6` também empregaram produtos cartesianos. A execução do FastBit para as consultas `Q5` e `Q6` indicam que quanto maior a porção (e a quantidade) de índices que o FastBit precisa varrer, maior é o custo acumulado da consulta, uma vez que passa a envolver a recuperação dos blocos do índice que estão armazenados no sistema de arquivos em disco.

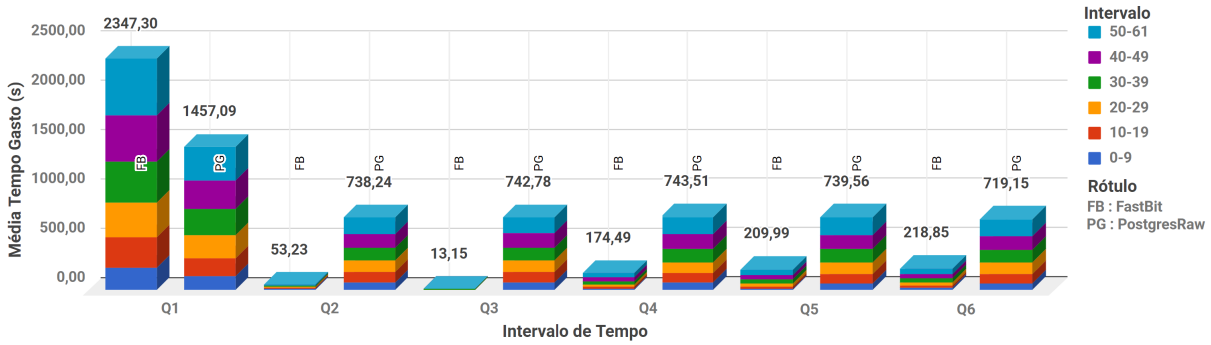


Figura 4.3: Tempo gasto no processamento das consultas após a indexação dos dados.

Para detalhar o comportamento do desempenho dos índices com relação aos intervalos de tempo é apresentada uma segunda medida nos gráficos das Figuras 4.4 e 4.5. Esses gráficos mostram a comparação para cada intervalo de tempo referente à consulta `Q3`. Essa consulta é representativa porque varre uma coluna inteira em comum entre os

vários arquivos de dados de domínio. A parte inferior das Figuras 4.4 e 4.5 apresentam o desempenho dos índices para consultar os arquivos de dados que armazenam as informações sobre intervalos de tempo ímpares, enquanto o topo das Figuras 4.4 e 4.5 mostram o comportamento dos índices com em relação aos intervalos de tempo pares.

Os resultados indicam que o FastBit foi capaz de recuperar dados em tempo constante para consulta `Q3`, uma vez que utiliza uma estrutura de indexação de tamanho fixo e que permite uma única varredura de todo o conjunto de valores das colunas. Por outro lado, o tempo gasto pelo PostgresRAW aumenta de acordo com o tamanho do conjunto de dados, o que destaca que as suas estruturas de indexação são bastante sensíveis à cardinalidade dos arquivos contendo os dados de domínio.

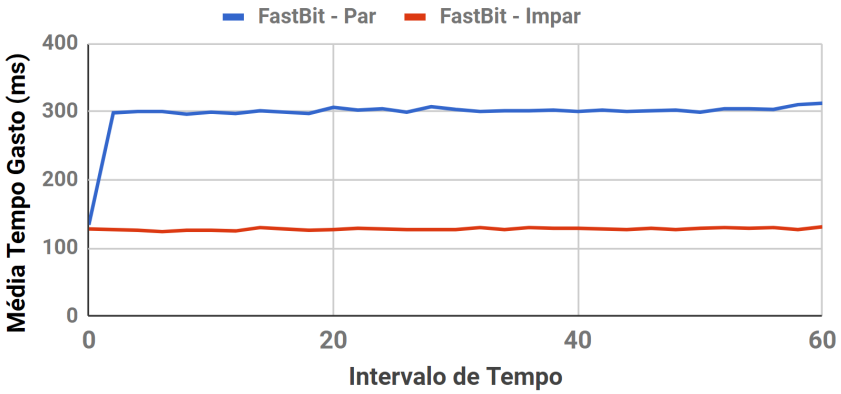


Figura 4.4: Consulta `Q3` no FastBit com relação aos intervalos de tempo.

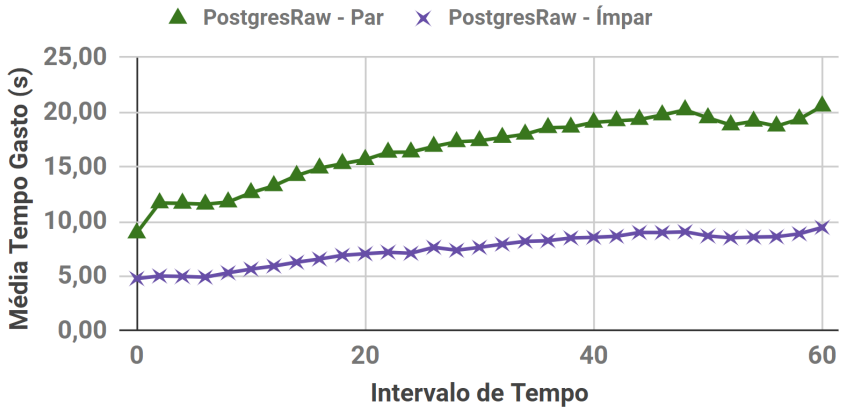


Figura 4.5: Consulta `Q3` no PostgresRAW com relação aos intervalos de tempo.

4.2.5 Comparação do tempo-para-consulta

Para esse último experimento, mediu-se a média do tempo-para-consulta de cada solução na busca de dados de domínio da simulação de deposição de sedimentos. Nesse contexto, supõe-se que os usuários desejam executar uma só consulta sobre todos os intervalos

de tempo, em vez de seis consultas sequenciais, tal como apresentado na Seção 4.2.4. Portanto, o tempo gasto na indexação dos arquivo de dados brutos é *adicionado* ao tempo de consulta aos dados de domínio.

A Figura 4.6 apresenta o tempo-para-consulta gasto pelas seis consultas aos arquivos oriundos da simulação computacional em todos os 62 intervalos de tempo. Os resultados indicam que o PostgresRAW foi mais adequado do que o FastBit nesse contexto, uma vez que precisou de menos tempo para alcançar os resultados. Em particular, o PostgresRAW foi até $92,43\%$, $95,10\%$, $95,06\%$, $95,10\%$, $95,13\%$ e $95,23\%$ melhor que FastBit com relação às consultas `Q1`, `Q2`, `Q3`, `Q4`, `Q5` e `Q6`, respectivamente. O menor ganho do PostgresRAW sobre o FastBit foi na consulta `Q1`, que une dados de domínio de vários arquivos. Além da consulta em si, esse ganho está relacionado com os procedimentos de construção do índice. Nesse caso, enquanto valores inteiros foram discretizados para a construção dos índices *bitmap*, o PostgresRAW apenas precisou mapear as colunas envolvidas na consulta.

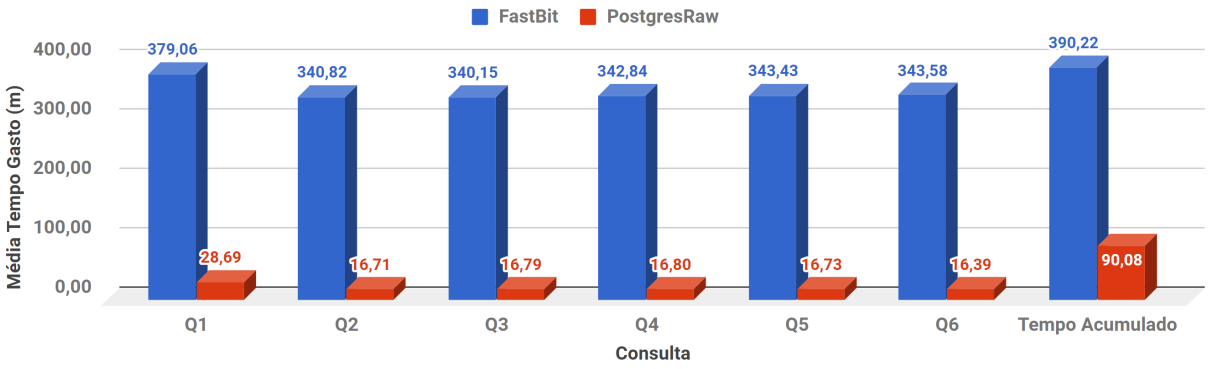


Figura 4.6: Comparação do tempo-para-consulta no FastBit e PostgresRaw.

Embora a execução das consultas no FastBit, depois de construídos os índices seja melhor do que a execução no PostgresRAW, o tempo gasto na construção das estruturas de indexação demora muito mais para se pagar. Em cenários de consultas a dados de domínio, como as do estudo de caso de deposição de sedimentos, o tempo-para-consulta do PostgresRaw foi muito menor do que o do FastBit para todas as consultas avaliadas, tal como detalhado nos resultados da Figura 4.6.

4.3 Conclusão

Extrair e armazenar dados de domínio diretamente na solução de captura de proveniência, *e.g.*, `SAMbA`, pode ser uma opção que o cientista prefira não usar. A alternativa para consultar dados de domínio, nesses casos, é buscar diretamente sobre os dados brutos oriundos

dos experimentos. Nessa seção, foi discutida e avaliada as estratégias *(i)* PostgresRaw, para processamento de consultas adaptativas e *(ii)* FastBit, para indexação e consultas sobre dados em arquivos. Foi examinada o comportamento de ambas as estratégias com relação à um estudo de caso de simulação no domínio de deposição de sedimentos.

Nas avaliações experimentais, foi medido o tempo gasto para a construção dos índices e do processamento de consultas para seis buscas diferentes sobre 62 intervalos de tempo, somando 372 consultas em 44.160 arquivos com um total de $12,2$GB de dados armazenados. A avaliação experimental contribuiu com os seguintes resultados principais:

1. O FastBit foi mais rápido que o PostgresRAW na execução das consultas para todos os cenários, exceto o de baixa seletividade. Essa avaliação considerou que os índices foram previamente construídos.

2. O PostgresRAW foi mais responsivo que o FastBit na execução completa das consultas, pois exigiu um tempo-para-consulta (o tempo gasto entre a geração dos índices e a execução das consultas) muito menor do que a solução FastBit.

Capítulo 5

Conclusão

Experimentos científicos que fazem uso intensivo de dados têm se tornado cada vez mais comuns e requerem ferramentas e métodos para execução e análise da forma mais eficiente e transparente possível. Nesse contexto, um *framework* que tem ganhado notoriedade é o Apache Spark, capaz de reduzir o gargalo de E/S encontrado em outros competidores do modelo MapReduce, como o Apache Hadoop. Não obstante, uma vez que a execução de *workflows* de experimentos científicos pode demorar muito tempo para finalizar, é de crucial importância que seja possível ao cientista acompanhar, monitorar, interpretar e rastrear resultados intermediários para o auxílio na tomada de decisões sobre experimentos enquanto eles ainda estão em execução. O método de monitoramento padrão do Apache Spark é o registro de *log* da execução, que fornece apenas informações sobre o *status* das execuções das tarefas do *workflow*. Além disso, os dados intermediários ficam somente na memória principal, o que restringe o seu acesso por aplicações externas e o cientista precisa esperar a finalização da execução para poder concluir algo sobre os resultados.

Uma forma eficiente de acompanhar em detalhes a execução de um experimento científico é por meio da captura e análise de dados de proveniência. A avaliação desses dados permite que cientistas tomem decisões sobre os dados gerados por suas aplicação em tempo real, o que possibilita, por exemplo, que a execução de um experimento seja interrompida quando há a identificação de um erro de execução. Infelizmente, nenhuma das abordagens existentes na literatura para análise de dados de experimentos científicos possibilita capturar e tratar dados de proveniência no Apache Spark. Em particular, ao se considerar os tipos de dados de proveniência entre prospectivos, retrospectivos e de domínio tem-se que as soluções existentes para análise de dados experimentais no Apache Spark apenas cobrem um pequeno aspecto desses dados proveniência como forma de atingir outros objetivos, *e.g.*, no caso de *debugging* de *workflows* científicos. Mais do que

isso, no melhor do nosso conhecimento, todas as ferramentas revisadas não tratam de outra limitação-chave do Apache Spark para o gerenciamento de dados de proveniência: a incapacidade de se executar aplicações "caixa-preta" externas que lidam com arquivos e, ao mesmo tempo, capturar os dados de domínio em memória de forma transparente.

Nesse trabalho foram abordados esses pontos em aberto na literatura ao-se propor uma solução para a coleta e recuperação de dados de proveniência no Apache Spark, denominada `SAMbA`. Em particular, a solução proposta nessa dissertação se mostrou, empiricamente, capaz de:

- Estender o Apache Spark de forma a apoiar a coleta e gerenciamento de três tipos de dados de proveniência em *workflows* científicos, com o cuidado de atacar possíveis gargalos de execução que podem vir a prejudicar o desempenho da execução de experimentos científicos com proveniência.
- Permitir que os usuários realizem a extração de quaisquer dados de domínio de interesse que são persistidos na base de dados de proveniência. Os dados de domínio podem ser facilmente especificados por meio da abstração `RDD Schema` do `SAMbA`.
- Fornecer formas eficientes de se realizar a análise dos dados de proveniência coletados. Em particular, duas formas de análise se destacam:
  - Relatórios de fluxos de transformações e fluxos de dados, visualizados e explorados em tempo de execução por meio de uma interface *web* interativa.
  - Consultas *post-mortem* com SQL feitas sobre a base de dados de proveniência.
- Fornecer uma estratégia de otimização para a execução de aplicações "caixa-preta" externas que se comunicam por meio de arquivos em memória secundária. A primeira parte da estratégia proposta é permitir que a comunicação entre essas aplicações seja feita em memória principal através do sistema de arquivos `SAMbA-FS`. A segunda parte consiste em manter os arquivos consumidos e produzidos por transformações oriundas de aplicações "caixa-preta" em repositórios externos versionados.

Em um segundo momento também foi discutida a forma mais adequada para se recuperar informações sobre grandes quantidades de dados de domínio produzidas em experimentos científicos. A premissa para essa discussão é que os cientistas podem optar por não tratar os dados de domínio diretamente na solução `SAMbA`, seja por não querer identificar *a priori* os atributos de interesse, no caso de análises exploratórias, ou por ter

interesse em todos os atributos do domínio, no caso de validações completas. Em ambos os casos, capturar todos os dados de domínio diretamente no `SAMbA` apenas levaria à uma cópia dos dados transformados para a base de dados de proveniência do `SAMbA`. Portanto, a questão de otimizar a consulta sobre esses dados de domínio se torna a questão de identificar a forma mais adequada de se recuperar dados de domínio de arquivos brutos. Nesse trabalho foi realizada uma análise comparativa de métodos que permitem consultas diretamente sobre os arquivos de dados de domínio em seus formatos originais. Especificamente, foi feita uma comparação detalhada entre o PostgresRaw, como uma ferramenta estado-da-arte para a abordagem de consultas adaptativas, e o FastBit, como uma solução estado-da-arte para a abordagem baseada em indexação de arquivos. Os resultados dessa avaliação experimental sobre um estudo de caso indicam que:

- O FastBit é mais adequado que o PostgresRAW na execução de consultas para grande parte de cenários, considerando que os índices já se encontram construídos *antes* da execução das consultas.
- O PostgresRAW é mais responsivo que o FastBit na execução completa de uma consulta, considerando o tempo gasto entre a geração dos índices e a execução das consultas em si.

5.1 Limitações

Muito embora a solução proposta seja abrangente o suficiente para dar suporte ao gerenciamento de proveniência em quase todas os experimentos científicos que usam o Apache Spark, o `SAMbA` possui algumas limitações e aspectos que podem ser melhorados. A saber:

- O `SAMbA` ainda não trata falhas de execução de tarefas. Quando falhas ocorrem, o Apache Spark aplica seu algoritmo de recuperação nativo, o que pode levar a duplicação de alguns dados de proveniência na base de dados do `SAMbA`.
- O `SAMbA` é limitado pela vazão de escrita do SGBD acoplado que armazena a base de dados de proveniência. Aplicações rodando sobre o `SAMbA` que gerem um volume de dados maior do que o SGBD possa persistir no mesmo intervalo de tempo entram em pausa automática, pois os dados a serem persistidos formam uma fila de inserção.

5.2 Trabalhos futuros

A seguir, apresentamos proposta para trabalhos futuros sobre os tópicos desse trabalho:

- Expandir a implementação do `SAMbA` para todos ou outros componentes da arquitetura e linguagens do Apache Spark, além da linguagem Scala.
- Implementar um relatório na interface *web* para permitir realizar a comparação entre duas execuções e identificar as diferenças entre elas além da comparação via Git já existente.
- Implementar técnicas de compressão de arquivos em memória, para otimizar o uso do `SAMbA-FS` e avaliar o impacto desse fator de acordo com a quantidade de dados manipulados por aplicações "caixa-preta" externas. Exemplos de algoritmos que poderiam ser aplicados são o LZ4[1] e o Zstandard[2].
- O `SAMbA-FS` permite identificar quais os arquivos utilizados em uma determinada execução e o `SAMbA` permite recuperar quais foram os programas executados sobre os dados. Um trabalho futuro seria combinar essas funcionalidades para gerar pacotes "autocontidos", *i.e.*, pacotes com todos os arquivos e programas utilizados em um experimento. O resultado dessa combinação proporcionaria a reprodução de experimentos, de forma similar ao raciocínio empregado pela ferramenta ReproZip[3].

5.3 Publicações

Nessa seção são reportados os artigos publicados ao decorrer da duração do mestrado. A Seção 5.3.1 apresenta as publicações cujo conteúdo é produto da pesquisa descrita nessa dissertação e a Seção 5.3.2 indica outras publicações em colaboração.

5.3.1 Artigos oriundos dessa dissertação

Revista:

1. (*Minor Review*). GUEDES, Thaylon; SILVA, Vítor; CAMATA, José; BEDO, Marcos V. N.; MATTOSO, Marta; OLIVEIRA, Daniel. "*Towards an Empirical Evalu-*

[1] `https://github.com/lz4/lz4/`
[2] `https://github.com/facebook/zstd`
[3] `https://www.reprozip.org/`

ation of Scientific Data Indexing and Querying". **Journal of Information and Data Management (JIDM)**. 10 páginas.

Publicações em Conferências:

1. GUEDES, Thaylon; SILVA, Vítor; CAMATA, José; MATTOSO, Marta; DE OLIVEIRA, Daniel. "Análise de Dados Científicos: uma Análise Comparativa de Dados de Simulações Computacionais"[4]. **XXXII Simpósio Brasileiro de Banco de Dados (SBBD) - Qualis B2**, 2017. 06 páginas.

2. GUEDES, Thaylon; SILVA, Vítor; BEDO, Marcos V. N.; MATTOSO, Marta; OLIVEIRA, Daniel. "Análise *Online* de Dados de Proveniência e de Domínio de Aplicações Spark com SAMbA". Demonstração, **XXXIII Simpósio Brasileiro de Banco de Dados (SBBD)**, 2018. 06 páginas.

3. GUEDES, Thaylon; SILVA, Vítor; BEDO, Marcos V. N.; MATTOSO, Marta; OLIVEIRA, Daniel. "*A practical roadmap for provenance storage and retrieval in scientific Spark applications*". **Workshop Workflows in Support of Large-Scale Science (WORKS)**. 10 páginas.

5.3.2 Trabalhos em colaboração

Publicação em Revista:

1. (Submetido). GUEDES, Thaylon; JESUS, Leonardo; OCAÑA, Kary; DRUMMOND Lúcia; OLIVEIRA, Daniel. "*Provenance-based Fault Tolerance Technique Recommendation for Cloud-based Scientific Workflows: a Practical Approach*". **Cluster Computing**. 18 páginas.

Publicações em Conferências:

1. GUEDES, Thaylon; OCAÑA, Kary; OLIVEIRA, Daniel. "SciPhyloMiner: um *Workflow* para Mineração de Dados Filogenômicos de Protozários". **XI Brazilian e-Science Workshop (BreSci)**, 2017. 08 páginas.

2. JASBICK, Daniel; GUEDES, Thaylon; OLIVEIRA, Rodolfo; SANTOS, Lucio F. D.; OLIVEIRA, Daniel; BEDO, Marcos V. N. "*VP-Viewer: Keeping track of your*

[4]Esse artigo obteve menção honrosa e convite para extensão em revista.

query from a vantage point". Demonstração, **XXXIII Simpósio Brasileiro de Banco de Dados (SBBD)**, 2018. 06 páginas.

Referências

[1] Apache cassandra nosql performance benchmarks. `https://academy.datastax.com/planet-cassandra/nosql-performance-benchmarks`. Último Acesso: 09/07/2018.

[2] Cluster mode overview. `https://spark.apache.org/docs/latest/cluster-overview.html`. Último Acesso: 07/07/2018.

[3] Epsrc policy framework on research data. `https://epsrc.ukri.org/about/standards/researchdata/scope/`. Último Acesso: 07/07/2018.

[4] Options for building bitmap indexe. `https://sdm.lbl.gov/fastbit/doc/indexSpec.html`. Último Acesso: 27/07/2018.

[5] Ailamaki, A. Fast, just-in-time queries on heterogeneous raw data. Disponível em: `https://cds.cern.ch/record/2294705`.

[6] Ailamaki, A.; Kantere, V.; Dash, D. Managing scientific data. *Communications of the ACM 53*, 6 (2010), 68–78.

[7] Alagiannis, I.; Borovica, R.; Branco, M.; Idreos, S.; Ailamaki, A. Nodb: efficient query execution on raw data files. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (2012), ACM, pp. 241–252.

[8] Altintas, I.; Ludaescher, B.; Klasky, S.; Vouk, M. A. Introduction to scientific workflow management and the kepler system. In *SC'06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing* (2006), p. 205.

[9] Armbrust, M.; Das, T.; Davidson, A.; Ghodsi, A.; Or, A.; Rosen, J.; Stoica, I.; Wendell, P.; Xin, R.; Zaharia, M. Scaling spark in the real world: performance and usability. *Proceedings of the VLDB Endowment 8*, 12 (2015), 1840–1843.

[10] Ayachit, U.; Bauer, A.; Geveci, B.; O'Leary, P.; Moreland, K.; Fabian, N.; Mauldin, J. Paraview catalyst: Enabling in situ data analysis and visualization. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization* (New York, NY, USA, 2015), ISAV2015, ACM, pp. 25–29.

[11] Berriman, G. B.; Deelman, E.; Good, J.; Jacob, J. C.; Katz, D. S.; Laity, A. C.; Prince, T. A.; Singh, G.; Su, M.-H. Generating complex astronomy workflows. In *Workflows for e-Science*. Springer, 2007, pp. 19–38.

[12] Blanas, S.; Wu, K.; Byna, S.; Dong, B.; Shoshani, A. Parallel data analysis directly on scientific file formats. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data* (2014), ACM, pp. 385–396.

[13] Brun, R.; Rademakers, F. Root—an object oriented data analysis framework. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 389*, 1-2 (1997), 81–86.

[14] Bryant, R. E. Data-intensive scalable computing for scientific applications. *Computing in Science & Engineering 13*, 6 (2011), 25–33.

[15] Camata, J. J.; Silva, V.; Valduriez, P.; Mattoso, M.; Coutinho, A. L. In situ visualization and data analysis for turbidity currents simulation. *Computers and Geosciences 110* (2018), 23 – 31.

[16] Caulfield, A. M.; Grupp, L. M.; Swanson, S. Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications. *ACM Sigplan Notices 44*, 3 (2009), 217–228.

[17] Chen, C. P.; Zhang, C.-Y. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences 275* (2014), 314–347.

[18] Cheng, Y.; Rusu, F. Scanraw: A database meta-operator for parallel in-situ processing and loading. *ACM Transactions on Database Systems (TODS) 40*, 3 (2015), 19.

[19] Chou, J.; Wu, K., et al. Fastquery: A parallel indexing system for scientific data. In *2011 IEEE International Conference on Cluster Computing* (2011), IEEE, pp. 455–464.

[20] Clifford, B.; Foster, I.; Voeckler, J.-S.; Wilde, M.; Zhao, Y. Tracking provenance in a virtual data grid. *Concurrency and Computation: Practice and Experience 20*, 5 (2008), 565–575.

[21] da Cruz, S. M. S.; Campos, M. L. M.; Mattoso, M. Towards a taxonomy of provenance in scientific workflow management systems. In *Services-I, 2009 World Conference on* (2009), IEEE, pp. 259–266.

[22] Davidson, S. B.; Freire, J. Provenance and scientific workflows: challenges and opportunities. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (2008), ACM, pp. 1345–1350.

[23] de Oliveira, D.; Ogasawara, E.; Baião, F.; Mattoso, M. Scicumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows. In *International Conference on Cloud Computing* (2010), IEEE, pp. 378–385.

[24] Dean, J.; Ghemawat, S. Mapreduce: simplified data processing on large clusters. *Communications of the ACM 51*, 1 (2008), 107–113.

[25] Deelman, E.; Gannon, D.; Shields, M.; Taylor, I. Workflows and e-science: An overview of workflow system features and capabilities. *Future generation computer systems 25*, 5 (2009), 528–540.

[26] Deelman, E.; Singh, G.; Su, M.-H.; Blythe, J.; Gil, Y.; Kesselman, C.; Mehta, G.; Vahi, K.; Berriman, G. B.; Good, J., et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming 13*, 3 (2005), 219–237.

[27] Dong, B.; Byna, S.; Wu, K. Parallel query evaluation as a scientific data service. In *Cluster Computing (CLUSTER), 2014 IEEE International Conference on* (2014), IEEE, pp. 194–202.

[28] Folk, M.; Heber, G.; Koziol, Q.; Pourmal, E.; Robinson, D. An overview of the hdf5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases* (2011), ACM, pp. 36–47.

[29] Freire, J.; Koop, D.; Santos, E.; Silva, C. T. Provenance for computational tasks: A survey. *Computing in Science & Engineering 10*, 3 (2008).

[30] Gil, Y.; Miles, S.; Belhajjame, K.; Deus, H.; Garijo, D.; Klyne, G.; Missier, P.; Soiland-Reyes, S.; Zednik, S. Prov model primer. *W3C Working Draft, 11th December* (2012).

[31] Gray, J.; Liu, D. T.; Nieto-Santisteban, M.; Szalay, A.; DeWitt, D. J.; Heber, G. Scientific data management in the coming decade. *Acm Sigmod Record 34*, 4 (2005), 34–41.

[32] Gulzar, M. A.; Interlandi, M.; Yoo, S.; Tetali, S. D.; Condie, T.; Millstein, T.; Kim, M. Bigdebug: Debugging primitives for interactive big data processing in spark. In *Proceedings of the 38th International Conference on Software Engineering* (2016), ACM, pp. 784–795.

[33] Hey, T.; Tansley, S.; Tolle, K. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, October 2009.

[34] Hey, T.; Tansley, S.; Tolle, K. M. Jim gray on escience: a transformed scientific method., 2009.

[35] Hollingsworth, D.; Hampshire, U. Workflow management coalition: The workflow reference model. *Document Number TC00-1003 19* (1995).

[36] Ikeda, R.; Park, H.; Widom, J. Provenance for generalized map and reduce workflows.

[37] Ikeda, R.; Sarma, A. D.; Widom, J. Logical provenance in data-oriented workflows? In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on* (2013), IEEE, pp. 877–888.

[38] Interlandi, M.; Shah, K.; Tetali, S. D.; Gulzar, M. A.; Yoo, S.; Kim, M.; Millstein, T.; Condie, T. Titian: Data provenance support in spark. *Proceedings of the VLDB Endowment 9*, 3 (2015), 216–227.

[39] Jha, S.; Qiu, J.; Luckow, A.; Mantha, P.; Fox, G. C. A tale of two data-intensive paradigms: Applications, abstractions, and architectures. In *Big Data (BigData Congress), 2014 IEEE International Congress on* (2014), IEEE, pp. 645–652.

[40] Karau, H.; Konwinski, A.; Wendell, P.; Zaharia, M. *Learning spark: lightning-fast big data analysis*. "O'Reilly Media, Inc.", 2015.

[41] Karpathiotakis, M.; Alagiannis, I.; Heinis, T.; Branco, M.; Ailamaki, A. Just-in-time data virtualization: Lightweight data management with vida. In *Proceedings of the 7th Biennial Conference on Innovative Data Systems Research (CIDR)* (2015), no. EPFL-CONF-203677.

[42] Karpathiotakis, M.; Branco, M.; Alagiannis, I.; Ailamaki, A. Adaptive query processing on raw data. *Proceedings of the VLDB Endowment 7*, 12 (2014), 1119–1130.

[43] Karpatne, A.; Liess, S. A guide to earth science data: Summary and research challenges. *Computing in Science Engineering 17*, 6 (Nov 2015), 14–18.

[44] Kersten, M.; Zhang, Y.; Ivanova, M.; Nes, N. Sciql, a query language for science applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases* (2011), ACM, pp. 1–12.

[45] Kwon, Y.; Nunley, D.; Gardner, J. P.; Balazinska, M.; Howe, B.; Loebman, S. Scalable clustering algorithm for n-body simulations in a shared-nothing cluster. In *International Conference on Scientific and Statistical Database Management* (2010), Springer, pp. 132–150.

[46] Lim, C.; Lu, S.; Chebotko, A.; Fotouhi, F. Prospective and retrospective provenance collection in scientific workflow environments. In *2010 IEEE International Conference on Services Computing* (2010), IEEE, pp. 449–456.

[47] Liu, J.; Pacitti, E.; Valduriez, P.; Mattoso, M. A survey of data-intensive scientific workflow management. *Journal of Grid Computing 13*, 4 (2015), 457–493.

[48] Logothetis, D.; De, S.; Yocum, K. Scalable lineage capture for debugging disc analytics. In *Proceedings of the 4th annual Symposium on Cloud Computing* (2013), ACM, p. 17.

[49] Markel, S.; León, D. *Sequence Analysis in a Nutshell: A Guide to Tools: A Guide to Common Tools and Databases*. "O'Reilly Media, Inc.", 2003.

[50] Mattoso, M.; Werner, C.; Travassos, G.; Braganholo, V.; Murta, L. Gerenciando experimentos científicos em larga escala. *SBC-SEMISH 8* (2008), 121–135.

[51] Mattoso, M.; Werner, C.; Travassos, G. H.; Braganholo, V.; Ogasawara, E.; Oliveira, D.; Cruz, S.; Martinho, W.; Murta, L. Towards supporting the life cycle of large scale scientific experiments. *International Journal of Business Process Integration and Management 5*, 1 (2010), 79–92.

[52] McPhillips, T.; Bowers, S.; Belhajjame, K.; Ludäscher, B. Retrospective provenance without a runtime provenance recorder. In *USENIX Workshop on Theory and Practice of Provenance* (2015).

[53] Moreau, L.; Groth, P. Provenance: An introduction to prov. *Synthesis Lectures on the Semantic Web: Theory and Technology 3*, 4 (2013), 1–129.

[54] Moreau, L.; Missier, P.; Belhajjame, K.; B'Far, R.; Cheney, J.; Coppens, S.; Cresswell, S.; Gil, Y.; Groth, P.; Klyne, G., et al. Prov-dm: The prov data model. *Retrieved July 30* (2013), 2013.

[55] Murta, L.; Braganholo, V.; Chirigati, F.; Koop, D.; Freire, J. noworkflow: capturing and analyzing provenance of scripts. In *International Provenance and Annotation Workshop* (2014), Springer, pp. 71–83.

[56] Ocaña, K. A.; de Oliveira, D.; Ogasawara, E.; Dávila, A. M.; Lima, A. A.; Mattoso, M. Sciphy: a cloud-based workflow for phylogenetic analysis of drug targets in protozoan genomes. In *Brazilian Symposium on Bioinformatics* (2011), Springer, pp. 66–70.

[57] of Edinburgh, U. *Edinburgh University Data Library Research Data Management Handbook*. University of Edinburgh, 2011.

[58] Olston, C.; Reed, B.; Srivastava, U.; Kumar, R.; Tomkins, A. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (2008), ACM, pp. 1099–1110.

[59] Pilat, D.; Fukasaku, Y. Oecd principles and guidelines for access to research data from public funding. *Data Science Journal 6* (2007), OD4–OD11.

[60] Sakellariou, R.; Zhao, H. A hybrid heuristic for dag scheduling on heterogeneous systems. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International* (2004), IEEE, p. 111.

[61] Shvachko, K.; Kuang, H.; Radia, S.; Chansler, R. The hadoop distributed file system. In *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on* (2010), Ieee, pp. 1–10.

[62] Silva, V.; Leite, J.; Camata, J. J.; De Oliveira, D.; Coutinho, A. L.; Valduriez, P.; Mattoso, M. Raw data queries during data-intensive parallel workflow execution. *Future Generation Computer Systems 75* (2017), 402–422.

[63] Silva, V.; Oliveira, D.; Valduriez, P.; Mattoso, M. Analyzing related raw data files through dataflows. *Concurrency and Computation: Practice and Experience 28*, 8 (2016), 2528–2545.

[64] Simmhan, Y. L.; Plale, B.; Gannon, D. A survey of data provenance in e-science. *ACM Sigmod Record 34*, 3 (2005), 31–36.

[65] Taylor, I. J.; Deelman, E.; Gannon, D. B.; Shields, M. *Workflows for e-Science: scientific workflows for grids*. Springer Publishing Company, Incorporated, 2014.

[66] Thusoo, A.; Sarma, J. S.; Jain, N.; Shao, Z.; Chakka, P.; Zhang, N.; Antony, S.; Liu, H.; Murthy, R. Hive-a petabyte scale data warehouse using hadoop. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on* (2010), IEEE, pp. 996–1005.

[67] Venugopal, S.; Buyya, R.; Ramamohanarao, K. A taxonomy of data grids for distributed data sharing, management, and processing. *ACM Computing Surveys (CSUR) 38*, 1 (2006), 3.

[68] Wang, J.; Crawl, D.; Purawat, S.; Nguyen, M.; Altintas, I. Big data provenance: Challenges, state of the art and opportunities. In *Big Data (Big Data), 2015 IEEE International Conference on* (2015), IEEE, pp. 2509–2516.

[69] Wang, L.; Tao, J.; Ranjan, R.; Marten, H.; Streit, A.; Chen, J.; Chen, D. G-hadoop: Mapreduce across distributed data centers for data-intensive computing. *Future Generation Computer Systems 29*, 3 (2013), 739–750.

[70] Wells, D. C.; Greisen, E. W. Fits-a flexible image transport system. In *Image Processing in Astronomy* (1979), p. 445.

[71] Wozniak, J. M.; Armstrong, T. G.; Wilde, M.; Katz, D. S.; Lusk, E.; Foster, I. T. Swift/t: Scalable data flow programming for many-task applications. In *ACM SIGPLAN Notices* (2013), vol. 48, ACM, pp. 309–310.

[72] Wu, K.; Ahern, S.; Bethel, E. W.; Chen, J.; Childs, H.; Cormier-Michel, E.; Geddes, C.; Gu, J.; Hagen, H.; Hamann, B., et al. Fastbit: interactively searching massive data. In *Journal of Physics: Conference Series* (2009), vol. 180, IOP Publishing, p. 012053.

[73] Zaharia, M.; Chowdhury, M.; Das, T.; Dave, A.; Ma, J.; McCauley, M.; Franklin, M. J.; Shenker, S.; Stoica, I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. 2–2.

[74] Zaharia, M.; Chowdhury, M.; Franklin, M. J.; Shenker, S.; Stoica, I. Spark: Cluster computing with working sets. *HotCloud 10*, 10-10 (2010), 95.

[75] Zhang, T.; Sun, X.; Xue, W.; Qiao, N.; Huang, H.; Shu, J.; Zheng, W. Parsa: High-throughput scientific data analysis framework with distributed file system. *Future Generation Computer Systems 51* (2015), 111–119.