Análise Integrada de Grafos de Proveniência Heterogêneos por meio de uma Abordagem *PolyStore*

Yan Mendes[1], **Victor Ströele**[1], **Daniel de Oliveira**[2], **Kary Ocaña**[3]

[1]Universidade Federal de Juiz de Fora (UFJF) - Juiz de Fora - MG – Brasil

{yanmendes,victor.stroele}@ice.ufjf

[2]Universidade Federal Fluminense (UFF) - Niterói - RJ – Brasil

danielcmo@ic.uff.br

[3]Laboratório Nacional de Computação Científica (LNCC) - Petrópolis - RJ – Brasil

karyann@lncc.br

Os autores gostariam de agradecer a FAPEMIG, FAPERJ, CAPES, CNPq e PTI-Lasse por financiarem parcialmente a pesquisa apresentada nesse artigo.

Abstract. *Workflows' provenance data are captured by several existing Wofkflow Management Systems (WfMSs). Distinct WfMSs use different storing formats to represent data and, usually, captures and store data in different granularities using a graph-like shape. This allows researchers to analyze and validate their workflows' results. Yet, in more complex scenarios where scientists need to compare provenance data originated from different WfMSs and workflows, a challenge emerges. To solve this problem, we propose an approach named* `PolyFlow`*, based on Polystore systems, being able to integrate multiple heterogeneous provenance databases adopting an on-demand global schema (ProvONE), i.e., it transforms the data in execution time, allowing researchers to query multiple provenance graphs via , exploring and linking provenance of different workflows. To assess* `PolyFlow`*'s viability, we developed conceptual to two WfMSs (Swift/T and Kepler) using a real experiment to analyze phylogenetic data.*

Resumo. *Dados de proveniência de um workflow são capturados por quase todos os Sistemas de Gerência de Workflows (SGWfs) existentes. Cada SGWf utiliza um formato próprio para representar tais dados, e, comumente, captura e armazena os dados em diferentes granularidades na forma de um grafo. Isso permite cientistas analisarem e validarem resultados de um workflow específico. Entretanto, em cenários mais complexos em que o cientista necessita analisar grafos de proveniência oriundos de múltiplos SGWfs e workflows, um desafio surge. Para resolver esse problema, propomos uma abordagem chamada* `PolyFlow`*, que se baseia no conceito de Sistemas PolyStore, sendo capaz de integrar diversos bancos de dados de proveniência heterogêneos adotando um esquema ProvONE global sob demanda, i.e., sem necessidade de conversão prévia dos dados (que pode ser bastante custosa). Os cientistas podem, então, consultar múltiplos grafos de proveniência nesse banco de dados integrado via* `PolyFlow`*, explorando e vinculando a proveniência de workflows diferentes. De forma a analisar a viabilidade da abordagem* `PolyFlow`*, desenvolvemos mapeamentos para dois SGWf (Swift/T e Kepler) utilizando um experimento real de análise de dados filogenéticos.*

1. Introdução

Nos últimos anos, os *Workflows* Científicos (Wfs) têm se tornado um padrão *de fato* para representar experimentos científicos baseados em simulações computacionais

[de Oliveira et al. 2019]. Um Wf é uma abstração capaz de representar uma sequência lógica de invocações de programas e/ou serviços (*i.e.,* atividades) e suas dependências de dados [Mattoso et al. 2010]. Os Wfs podem ser implementados de diversas formas (*e.g.,* manualmente, via *scripts*), mas são comumente executados por meio de Sistemas de Gerência de *Workflows* (SGWfs).

Os SGWfs são mecanismos complexos desenvolvidos para facilitar a modelagem e a execução dos Wfs. Existem diversos SGWfs como Kepler [Altintas et al. 2004] e o Swift/T [Wozniak et al. 2013], cada um com nichos específicos (*e.g.,* bioinformática, botânica, *etc.*). Além da execução dos Wfs, os SGWfs são responsáveis por oferecerem outras capacidades fundamentais, *e.g.,* a captura de dados de proveniência [Freire et al. 2008], que descrevem todos os artefatos de dados que uma execução de um Wf usou e produziu, bem como as transformações que os dados sofreram. Dados de proveniência em Wfs são baseados em objetos (dados e programas) e seus relacionamentos (dependências) [Moreau et al. 2008], sendo tipicamente representados na forma de um grafo acíclico dirigido (DAG).

Independentemente da estrutura do Wf, o grafo de proveniência resultante é um DAG. A presença de estruturas de repetição, controle de fluxo e paralelismo, por exemplo, afeta somente a complexidade do Wf, mas não a complexidade do grafo de proveniência. Um banco de dados de proveniência é, portanto, composto de vários DAGs cujos nós possuem estruturas variáveis (*i.e.,* os nós podem desempenhar diferentes papéis). Cada nó do grafo pode possuir muitas dependências. A capacidade de análise e interoperabilidade de tais grafos tem sido o objeto de várias iniciativas da comunidade, resultando na recomendação PROV do W3C [Moreau and Groth 2013]. Além disso, extensões do PROV específicas para dados de proveniência de Wfs como o ProvONE [Prabhune et al. 2018] também foram propostas. A vantagem do ProvONE é que o mesmo é capaz de representar a proveniência prospectiva do Wf (*p-prov* - sua definição) e não somente a retrospectiva (*r-prov* - histórico de execução).

Entretanto, apesar dos SGWfs representarem um avanço, os mesmos ainda carecem de mecanismos que auxiliem na análise dos dados de proveniência se considerarmos o experimento científico como um todo, e não somente Wfs de forma isolada. No cenário atual de ciência colaborativa, um Wf pode ser considerado apenas parte de um experimento científico complexo. Tomemos como exemplo a Rede Avançada em Biologia Computacional (Rabicó)[1], que tem como objetivo desenvolver um aparato computacional para apoiar a análise dos dados aplicados a modelos biológicos. A rede possui membros de diversos institutos e universidades como o LNCC e a COPPE/UFRJ. Nesse projeto, duas ou mais equipes geograficamente distribuídas comumente trabalham independentemente com temas em comum, como por exemplo análise filogenética [Ocaña et al. 2011]. Cada equipe eventualmente adota abordagens ligeiramente diferentes, modelando Wfs diferentes em termos de representação (já que cada SGWf possui seu formato próprio) e na escolha dos programas, mas semelhantes em seus objetivos, gerando resultados passíveis de comparação.

Assumamos que dois grupos independentes na mesma rede de pesquisa projetam e implementam dois Wfs, o SciPhy [Ocaña et al. 2011] e o SwiftPhylo [Mondelli et al. 2018], usando diferentes SGWfs, o Kepler e o Swift/T, respectivamente. Cada um dos SGWfs tem suas especificidades, mas ambos são capazes de coletar *r-prov* e *p-prov*. Uma vez que ambos Wfs tem o mesmo objetivo, consomem dados de entrada iguais e geram resultados equivalentes, parece natural tentar usar os grafos de proveniência de suas execuções para comparar e discutir os resultados. No entanto, o Kepler e o Swift/T possuem modelos diferentes de seus bancos de dados de proveniência. Enquanto que o Kepler armazena os dados de proveniência em um Sistema de

[1] `https://www.labinfo.lncc.br/rabico/`

Gerência de Banco de Dados (SGBD) relacional (HSQL), o Swift/T utiliza *logs* que podem ser exportados para um banco de dados MySQL. Além da diferença na representação, cada SGWf grava os dados de proveniência em diferentes granularidades. Assim, embora seja possível que os pesquisadores consultem os dois grafos de proveniência de maneira isolada, a heterogeneidade na representação e a diferença na granularidade dos dados podem dificultar a análise conjunta dos mesmos.

Neste artigo, nos baseamos nesses grafos de proveniência heterogêneos e no ProvONE, como um modelo conceitual canônico, para mostrar como a interoperabilidade de dados de proveniência pode ser alcançada em uma abordagem prática onde podemos assumir um grau de semelhança entre os Wfs e seus grafos de proveniência. Para resolver esta falta de interoperabilidade propomos uma abordagem chamada `PolyFlow` baseada no conceito de Sistemas *PolyStore* [Dziedzic et al. 2016]. Sistemas *PolyStore* são aqueles construídos sobre múltiplos SGBDs heterogêneos e integrados. Além disso, um sistema *PolyStore* se distingue dos SGBDs federados tradicionais uma vez que necessita apenas de um mapeamento das diversas *ilhas* (*i.e.,* formatos dos dados), que são acessados em tempo de execução. Avaliamos o `PolyFlow` com *traces* de Wfs reais (SciPhy e SwiftPhylo) e os resultados apresentaram um *overhead* aceitável, além de reforçar a importância de uma análise integrada dos grafos de proveniência.

Esse artigo se encontra organizado em 4 seções além desta introdução. Na Seção 2 é apresentado o referencial teórico, bem como os trabalhos relacionados. A Seção 3 apresenta o `PolyFlow` e na Seção 4 é conduzida sua avaliação. Por fim, a Seção 5 traz as considerações finais e trabalhos futuros.

2. Referencial Teórico e Trabalhos Relacionados

Nesta seção são apresentados os conceitos necessários para o entendimento deste artigo, como o formalismo de Wfs, proveniência e sistemas *PolyStore*. Além disso, são discutidos os principais trabalhos relacionados.

2.1. *Workflows* e Proveniência

Um Wf pode ser modelado como um grafo $W(A, \phi)$, onde A é o conjunto das atividades de W e ϕ o conjunto das dependências de dados. Dessa forma, temos que $A = \{a_1, a_2, \ldots, a_n\}$ e cada atividade a_i pode ser representada como $a_i(I, P),\ a_i : \{I, P\} \rightarrow O$, onde I é o conjunto de dados de entrada, P os parâmetros e O os dados de saída da atividade a_i. Dessa forma, temos que $I = \{i_1, i_2, \ldots, i_d\}$, onde cada i_d é um arquivo de entrada de a_i, $O = \{o_1, o_2, \ldots, o_k\}$, onde cada o_k é um arquivo de saída de a_i e $P = \{p_1, p_2, \ldots, p_m\}$, onde cada p_m é um parâmetro de a_i.

Cada execução de a_i está associada a uma tupla de m parâmetros $< p_1, p_2, \ldots, p_m >$, onde o valor v_m de cada parâmetro p_m é definido por uma função $\zeta_m(p_m) = v_m$. Consequentemente, temos que o conjunto de dependências de dados $\phi = \{\varphi_{1,2}, \ldots, \varphi_{i,j}\}$, onde cada $\varphi_{i,j} = \langle i_d, a_i, a_j \rangle$, $input(a_i) \in I$, $i_d \neq \varnothing$ e $output(a_i) \in O$. Logo, $\varphi_{i,j} \leftrightarrow \exists o_k \in input(a_j) | O_k \in output(a_i)$.

Um grafo de proveniência G_p gerado a partir da execução de W é definido como $G_p = (V_p, E_p, A_p, T_p)$, onde os nós de V_p representam programas ou artefatos de dados e as arestas de E_p representam a linhagem. Um atributo *type* $\in A_p$ deve existir para todos os nós e arestas. Se um nó v_{pi} representa um programa associado a uma atividade a_i, $Value(v_{pi}, type) = program$. Se esse nó representar um artefato de dados, $Value(v_{pi}, type) = data$. O conjunto A_p também contém outros atributos encontrados nos grafos de proveniência, *e.g.,* nomes de parâmetros de entrada de programas. Os nós que representam os programas também possuem um atributo *name* $\in A_p$. O conjunto T_p representa tipos de arestas que podem ser dos tipos *WasGenera-*

tedBy, Used,WasInformedBy, WasDerivedFrom, WasAttributedTo, WasAssociatedWith ou *ActedOnBehalfOf*, de acordo com a recomendação PROV e o modelo ProvOne.

Especificamente neste artigo, usamos o ProvONE como um modelo canônico capaz de integrar grafos de proveniência (*traces*) de múltiplos Wfs e produzidos pelos diferentes SGWfs. O ProvONE estende a recomendação PROV do W3C com uma representação explícita de *p-prov*, capturando assim as informações mais relevantes sobre atividades do Wf. O ProvONE é composto por diferentes classes e os relacionamentos entre as mesmas. Por questões de restrição de espaço, somente as classes principais são detalhadas nesta seção. Mais informações podem ser obtidas em [Prabhune et al. 2016].

A classe *Program* representa uma tarefa computacional (atividade) que consome e produz dados por meio de suas portas. As instâncias da classe *Program* podem ser atômicas ou compostas. Uma *Port* habilita um *Program* a enviar ou receber dados e/ou parâmetros. Dependências de dados são explicitadas por meio da classe *Channels* que conecta dois ou mais *Programs* por meio de suas *Ports*. A classe *Workflow* representa um tipo especial de *Program*, *i.e.,* uma composição recursiva de programas. A classe *Execution* representa a execução de um *Program*. A classe *User* representa o usuário responsável pela execução. E, finalmente, a classe *Entity* representa as unidades básicas de informação consumidas ou produzidas por um *Program*.

2.2. Sistemas *PolyStore*

PolyStore é um conceito emergente que pode ser visto como um novo tipo de federação de dados, *i.e.,* um sistema de gerência de meta-banco de dados que fornece uma interface unificada e transparente para várias soluções de armazenamento autônomo [Gadepally et al. 2016]. *PolyStores* se diferenciam de sistemas federados tradicionais (que apoiam apenas um modelo de dados), pois é capaz de oferecer suporte para múltiplos modelos de dados. O uso do conceito *PolyStore* visa mitigar os problemas de usabilidade, concedendo aos usuários uma ampla variedade de soluções de armazenamento e linguagens de consulta. Este novo paradigma visa fornecer uma alternativa ao paradigma arquitetural *'one size fits all'*, armazenando e processando diferentes fragmentos de um conjunto de dados geral nos mecanismos que melhor ofereçam apoio a ingestão, consulta e análise de alto desempenho [Gadepally et al. 2016].

Em abordagens *PolyStore*, uma arquitetura conceitual necessita de um *middleware*, que é responsável por manter e orquestrar a submissão e resposta a múltiplas consultas, conhecendo os SGBDs que estão sendo usados para de fato armazenar os dados, direcionando as consultas para as *ilhas* apropriadas. Uma *ilha* é a definição de um modelo de dados e uma linguagem de consulta que representa um tipo de dados. No entanto, os próprios mecanismos de armazenamento podem não oferecer apoio aos formatos de dados e às linguagens de consulta de escolha. É definido também o operador *shim*, responsável por traduzir o modelo de dados e construtos de consulta definidas por uma *ilha* para o modelo e os construtos apoiados pela solução de armazenamento. Na solução proposta por [Gadepally et al. 2016] é possível navegar em diferentes *ilhas*, *i.e.,* podem ser utilizadas linguagens de consulta de diferentes *ilhas* para recuperar dados armazenados em um banco de dados que não pertence à mesma *ilha* que a consulta emitida. Por fim, o operador *cast* é definido para tratar a migração de dados entre soluções de armazenamento diferentes.

2.3. Trabalhos Relacionados

Essa seção apresenta trabalhos encontrados na literatura que abordam o problema de interoperabilidade em grafos de proveniência heterogêneos. Esses trabalhos foram selecionados a partir de um Mapeamento Sistemático da Literatura (disponível em [2] e analisados sob três perspecti-

[2] `https://goo.gl/2qZxUJ`

vas: (i) completude, (ii) usabilidade e (iii) extensibilidade. Sob a perspectiva da completude, os trabalhos foram avaliados considerando a capacidade de capturar *p-prov*, *r-prov* e proveniência evolutiva. Embora proveniência evolutiva não seja definida formalmente na literatura, ela está presente em diversas abordagens [Prabhune et al. 2018]. Quanto à usabilidade, foi avaliada a interface de consulta da solução, mais especificamente, se ela apoia as linguagens de consulta SQL, SPARQL, Cypher e QLP. Essas linguagens foram destacadas, pois, segundo os resultados do mapeamento, são as que os cientistas estão mais familiarizados. Finalmente, para avaliar a extensibilidade, foi considerado o modelo utilizado e o esforço necessário para apoiar os dados descritos por um novo formato para a solução.

[Ellqvist et al. 2009] propõem uma arquitetura baseada em mediadores capaz de interoperar dados de proveniência derivados de diferentes fontes de dados. A arquitetura tem um esquema global que é capaz de representar informações de proveniência expressas por outros modelos, e uma API de consulta capaz de recuperar essas informações de fontes de dados distintas. Os autores implementaram um novo modelo, o *Scientific Workflow Provenance Data Model* - SWPDM), que captura *p-prov* e *r-prov*. [Missier et al. 2010], juntamente com a integração de dados de proveniência heterogêneos, também propõem uma solução para manter o *trace* de proveniência entre as execuções dos Wfs. Como solução de integração, os autores propõem uma extensão do OPM [Moreau et al. 2008]. [Oliveira et al. 2016] propõem uma arquitetura de integração que possui uma camada responsável por transformar os *traces* dos Wfs em fatos Prolog descritos pelo modelo ProvONE, podendo capturar *p-prov* e *r-prov* e evolutiva. Em uma segunda camada é feito o compartilhamento do conhecimento, onde todas as saídas geradas pela primeira camada são armazenadas e podem ser acessadas através de consultas Prolog. [Prabhune et al. 2018] propõem uma estrutura que auxilia os pesquisadores na análise de dados heterogêneos de proveniência. Para isso, eles usam uma solução de armazenamento RDF (Apache Jena TDB), na qual os dados são representados pelo modelo ProvONE. Os usuários podem consultar os dados usando construções SPARQL. Eles implementam três algoritmos de mapeamento que transformam os dados descritos por outros formatos em dados compatíveis com ProvONE.

Em síntese, [Ellqvist et al. 2009] propõem um modelo de dados de proveniência que não captura a proveniência evolutiva. Além disso, toda consulta é feita por meio de uma API, sendo necessário implementar novos *endpoints* para novas consultas. O modelo proposto por [Missier et al. 2010] captura *p-prov* e *r-prov*, mas falta apoio à proveniência evolutiva. É adotado um modelo relacional para armazenar os dados, possibilitando consultas através de construções SQL. [Oliveira et al. 2016] utilizam o modelo ProvONE sendo um indicativo da extensibilidade da solução. Por outro lado, para acessar a base de conhecimento, os usuários devem escrever consultas Prolog o que prejudica a usabilidade da solução. [Prabhune et al. 2018] adotam SPARQL para consultas e, no que diz respeito à extensibilidade, para suportar dados descritos por outros formatos, novos algoritmos de mapeamento devem ser desenvolvidos. Além disso, todas as soluções demandam que uma conversão de todos os dados para o modelo ProvONE seja realizada, o que pode ser bastante custoso.

Considerando o exposto anteriormente, a abordagem proposta neste trabalho visa solucionar as lacunas identificadas. Nesse sentido, as principais contribuições do `PolyFlow` são: (i) ser uma solução extensível, pois utiliza um modelo de dados já definido na literatura e suporta múltiplas soluções de armazenamento, reforçando, consequentemente, seu (ii) apelo de usabilidade, fornecendo suporte a qualquer solução de armazenamento e linguagem de consulta que os usuários em potencial desejarem. Além disso, o `PolyFlow` possui um catálogo interno de mediadores, facilitando a incorporação de novos modelos e eximindo a necessidade da construção de algoritmos de mapeamento. Além disso, por não necessitar de uma conversão completa

para um modelo de dados canônico, o PolyFlow permite que as consultas sejam realizadas sob demanda nos bancos de dados originais.

3. PolyFlow: Integrando Grafos de Proveniência Heterogêneos

A abordagem PolyFlow foi desenvolvida para integrar bancos de dados heterogêneos que representam grafos de proveniência de Wfs. Nesta seção, apresentamos uma extensão do formalismo inicialmente apresentado na Seção 2, o mapeamento de dados necessário e a arquitetura do PolyFlow.

3.1. Formalismo

O PolyFlow se baseia na consulta integrada de múltiplos *traces* de Wfs representados por meio de grafos de proveniência. Dessa forma, devemos definir o conceito de banco de dados de proveniência. Neste artigo, um banco de dados de proveniência $\Im$ pode ser definido como um conjunto de θ grafos G_p, sendo $\Im = \{G_{p1}, G_{p2}, \ldots, G_{p\theta}\}$. Para cada $G_{p\theta} \in \Im$, devemos ser capazes de realizar consultas sobre esses grafos. Assim, a *consulta por valor de parâmetros de traces diferentes* pode ser definida por $Q_M(S, \Im) \Leftrightarrow \{G_{p\theta} \in \Im | \exists p_m \in G_{p\theta} \wedge \lambda(\star, \zeta_m(G_{p\theta}.p_m), v_m) \wedge (p_m, v_m) \in S\}$, onde: (i) Q_M é representado por um conjunto de pares $S = \{(p_1, v_1), (p_2, v_2), \ldots, (p_m, v_m)\}$, onde p_m é o parâmetro a ser consultado e v_m é o valor de referência; e (ii) λ é uma função que compara o valor v_m de um parâmetro p_m com o valor do mesmo parâmetro de um grafo $G_{p\theta}$ por meio $\zeta_m(G_{p\theta.p_m})$ utilizando um operador denominado $\star$. É importante ressaltar que o operador $\star$ pode ser qualquer operador utilizado na álgebra relacional.

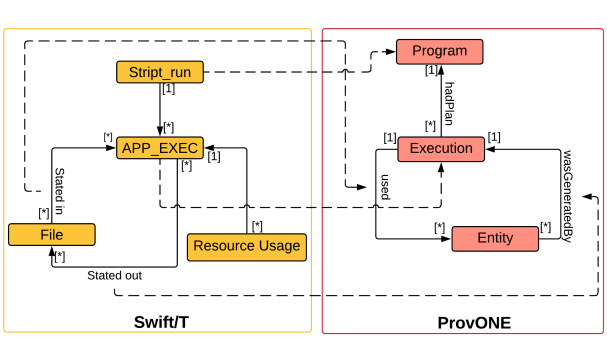
Assumindo que dois grafos $G_{p1}, G_{p2} \in \Im$ possuam parâmetros p_m e p'_m equivalentes ($p_m \equiv p'_m$), mas com denominações diferentes. Logo, dado um conjunto $P_\alpha = \{p_1, p_2, \ldots, p_\mu\} \in G_{p1}$ e $P_\beta = \{p_1, p_2, \ldots, p_\nu\} \in G_{p2}$ e uma linguagem de transformação Υ devemos ser capazes de achar um mapeamento $\upsilon \in \Upsilon$ de forma que cada $p_\mu \in P_\alpha$ e $p_\nu \in P_\beta$, $\tau(p_\mu) = p_\nu$. Logo, em bases de proveniência heterogêneas, a complexidade está em implementar consultas por valores de parâmetros $Q_M(\Gamma, \Im)$ (onde $\Gamma = S_\alpha \cup S_\beta$, $S_\alpha = \{(p_1, v_1), (p_2, v_2), \ldots, (p_\mu, v_\mu)\}$ e $S_\beta = \{(p_1, v_1), (p_2, v_2), \ldots, (p_\nu, v_\nu)\}$), considerando o mapeamento τ entre os parâmetros de *traces* representados em grafos diferentes.

3.2. Mapeamento para o Modelo ProvOne

Conforme mencionado na Seção 3.1, um desafio em uma abordagem *PolyStore* para análise de proveniência é definir o mapeamento τ entre os parâmetros de *traces* representados em grafos heterogêneos. Neste artigo focamos em prover uma solução para análise integrada (com a semântica do domínio de proveniência) a partir de bancos de dados heterogêneos. A estratégia adotada é baseada em mediação [Özsu and Valduriez 2011], que é a conciliação feita para possibilitar a tradução dos dados descritos por esquemas locais a partir de um esquema global. O PolyFlow adota o modelo ProvONE como modelo global (canônico). Assim, a estratégia adotada neste trabalho é mapear modelos de bancos de dados de proveniência diferentes para o modelo ProvONE.

Tais mapeamentos são utilizados pelos componentes do PolyFlow (*Query Resolvers* e *Entity Mappers, explicados na Seção 3.3*) para executar as consultas, sejam elas em um único ou múltiplos grafos de proveniência. No PolyFlow existem dois tipos de mapeamentos possíveis: (i) 1 - 1: onde duas entidades são equivalentes entre os *schemas*; (ii) 1 - N: quando a entidade do *Schema* Global se encontra associada a composição de várias entidades do *Schema* Local, ou seja, é representada de maneira mais granular no *Schema* Local. Note que mapeamentos N - 1 e N - N podem ser expressos por N mapeamentos 1 - 1 e 1 - N, respectivamente. A Figura 1(a) apresenta o *Schema* Local do banco de dados de proveniência do Swift/T e seu mapeamento com

um fragmento do ProvONE. As setas tracejadas indicam os mapeamentos entre os dois *schemas*. Uma vez que a granularidade entre os modelos é diferente, apenas as entidades do ProvONE envolvidas no mapeamento foram representadas. Esse mapeamento é representado por meio de objetos JSON (*i.e., Entity Mappers*, explicado a seguir). A Figura 1(b) apresenta o *Mapper* da entidade APP_EXEC (Swift/T) para a *Execution* do ProvONE.




```
{
  "name": "app_exec",
  "columns": [
    {
      "alias": "execution_id",
      "projection": "app_exec_id"
    },
    {
      "alias": "prov_startedAtTime",
      "projection": "start_time"
    },
    {
      "alias": "provone_hadPlan",
      "projection": "script_run_id"
    },
    {
      "projection": "duration"
    }
  ]
}
```

Figura 1. (a) Mapeamento do Swift/T para o ProvONE; (b) *Mapper* da entidade *APP_EXEC* para *Execution*.

3.3. Arquitetura do `PolyFlow`

A abordagem `PolyFlow` foi projetada para apoiar a interoperabilidade semântica de dados, se baseando no conceito de sistemas *PolyStore* que proveem a interoperabilidade lógica dos dados. Dessa forma, a arquitetura do `PolyFlow` segue os padrões de arquiteturas de sistemas *PolyStore*. A chave para uma abordagem *PolyStore* é que os vários mecanismos de armazenamento são distintos e acessados de forma isolada por meio de seus próprios mecanismos de consulta e de uma interface comum. Uma visão geral da arquitetura é apresentada na Figura 2. A arquitetura do `PolyFlow` é composta de três camadas: (i) Fonte de Dados, (ii) Camada de Mediação e (iii) Camada de Consulta. O código-fonte do `PolyFlow` e todos os mapeamentos realizados nos experimentos apresentados nesse artigo estão disponíveis em `https://github.com/yanmendes/polyflow.api`.

A camada que representa as fontes de dados contém os múltiplos bancos de dados de proveniência $\Im_1, \Im_2, \ldots, \Im_f$, cada uma delas representada em diferentes formatos. Cada banco de dados está relacionado com uma *ilha* do modelo *PolyStore* (*e.g.,* relacional, JSON, XML). No exemplo da Figura 2, podemos visualizar 3 bancos de dados referentes a 3 SGWfs distintos. Na etapa atual da implementação, a arquitetura não conta com implementações do operador *cast* discutido na Seção 2.2.

A camada de mediação é o núcleo do `PolyFlow`. Nessa camada é realizado todo o mapeamento entre as múltiplas ilhas que compõem a abordagem. A camada de Mediação do `PolyFlow` possui três componentes: (i) *Schema* Global, (ii) *Schema Local* e (iii) *Entity Mappers*. Os *Schemas* Locais são os modelos de dados associados a cada banco de dados de proveniência da camada de fonte de dados. O *Schema* Global é o modelo canônico a ser adotado para realizar as consultas no `PolyFlow`. Apesar da arquitetura possibilitar o uso de múltiplos *Schemas* Globais, no momento consideramos apenas o ProvONE. Finalmente, os *Entity Mappers*, confeccionados por usuários que compreendem os modelos de dados dos dois esquemas, realizam mapeamentos de equivalência entre os *Schemas* Locais e o *Schema* Global. Eles são consumidos pelos *Query Resolvers* no momento do processamento das consultas. É importante ressaltar que é nessa camada onde os usuários tendem a dispensar os maiores esforços, pois o mapeamento pode ser uma tarefa árdua. Somente os *Entity Mappers* são persistidos em formato

JSON no catálogo interno do PolyFlow. O formato JSON foi escolhido por sua popularidade e simplicidade, mitigando a curva de aprendizado que outros formatos mais semânticos (e.g. RDF) impõe por sua complexidade.

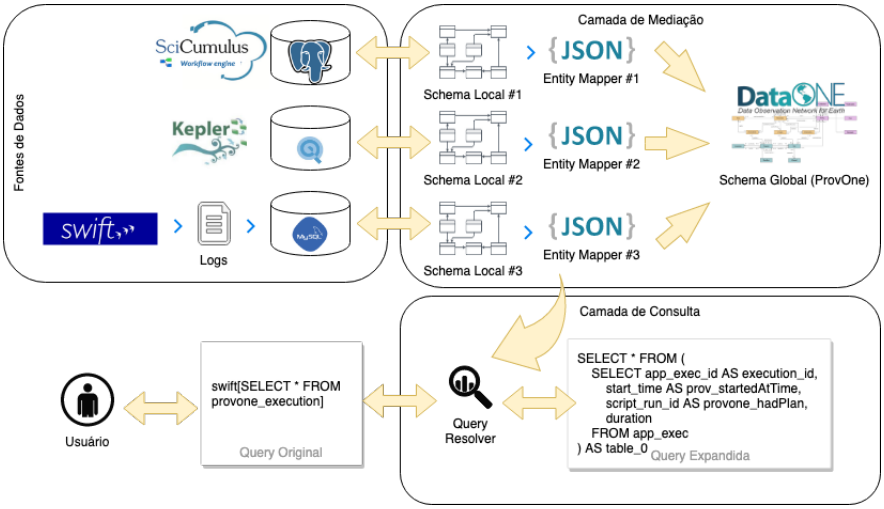


Figura 2. Arquitetura do PolyFlow

Finalmente, a camada de consulta disponibiliza para o usuário o *Resolver* que é o componente responsável por converter consultas baseadas em um *Schema* Global (ProvONE) em consultas válidas nos *Schemas* Locais. Estes funcionam de maneira similar ao operador *shim* da arquitetura *PolyStore*, resolvendo as consultas submetidas às ilhas. Dessa forma, pode submetê-las aos mecanismos de consulta da camada de fonte de dados, recuperando os dados de proveniência e representando-os utilizando o modelo ProvONE. O PolyFlow também é capaz de resolver agregações entre pares de entidades do *Schema* Local recursivamente, i.e. um *Entity Mapper* pode ser composto por pares de *Entity Mappers*. Devido à restrições de espaço, impossibilita-se o destrinchamento mais detalhado de componentes da arquitetura e exemplos. Para um mergulho mais aprofundado, veja a pasta *examples* do repositório do PolyFlow que ilustram diferentes casos de uso.

As consultas submetidas ao *Query Resolver* são SQL-*like* e devem seguir a sintaxe *nome-do-mediador[entidade-a-ser-mediada]*. Todas construções SQL válidas são suportadas por PolyFlow. As consultas são expandidas em tempo de execução por meio da substituição dos mediadores encontrados na consulta por uma *subquery* utilizando os elementos presentes no *Entity Mapper* associado. Por exemplo, assumamos que um determinado usuário necessita consultar todas as execuções de Wfs que foram realizadas no SGWf Swift/T. Ao buscar por todas as *Executions* (*provone_execution*), o PolyFlow realiza a expansão de consulta:

```
SELECT * FROM swift[provone_execution];
<=>
SELECT * FROM (
    SELECT app_exec_id AS execution_id,
    start_time AS prov_startedAtTime,
    script_run_id AS provone_hadPlan, duration
    FROM app_exec
) AS table_0;
```

4. Avaliação Experimental

Nesta seção avaliamos a abordagem PolyFlow sob duas perspectivas: **(i)** o desempenho de con-

sultas com a solução proposta em um banco de dados específico (Kepler e Swift/T consultados de forma isolada) e **(ii)** o desempenho do `PolyFlow` em consultas integradas.

4.1. Estudo de Caso

Conforme mencionado na Seção 1, foi utilizado como estudo de caso o experimento de análise filogenética no contexto do projeto Rabicó. Conceitualmente, esse experimento recebe como entrada um *dataset* de sequências de DNA, RNA e proteínas de forma a gerar uma árvore filogenética que indique a relação evolutiva entre os organismos de entrada. O experimento envolve 7 etapas bem definidas, conforme apresentado na Figura 3: (i) *Importação dos Dados*: os dados são coletados de diversos repositórios biológicos, como o RefSeq; (ii) *Numeração das Sequências*: as sequências obtidas são identificadas e numeradas (etapa opcional); (iii) *Alinhamento Múltiplo de Sequências*: as sequências são alinhadas (por programas como MAFFT, Kalign, ClustalW, Muscle, ou ProbCons), *i.e.,* são identificadas regiões similares que possam ser consequência de relações funcionais, estruturais ou evolutivas; (iv) *Conversão de Sequências*: as sequências alinhadas são convertidas para o formato PHYLIP; (v) *Escolha do Modelo Evolutivo*: o melhor modelo evolutivo para as sequências alinhadas é selecionado (etapa mais custosa do experimento); (vi) *Filtragem de Sequência*: as sequências podem ser filtradas ou trimadas para diminuir a complexidade da geração da árvore filogenética (atividade opcional); e (vii) *Geração da Árvore Filogenética*: a árvore filogenética é finalmente gerada, apontando as relações evolutivas entre os organismos.

Nesse artigo consideramos os Wfs SciPhy e SwiftPhylo como variações do mesmo experimento e cujo resultado deve ser analisado de forma integrada. O SwiftPhylo implementa todas as atividades com a exceção da importação dos dados, pois assume que os dados já se encontram disponíveis para processamento. Por outro lado, o SciPhy não implementa as atividades opcionais do experimento, ou seja, é composto por 5 atividades. Além disso, o SwiftPhylo utiliza o MAFFT como programa de alinhamento, enquanto que o SciPhy executa todos os programas de alinhamento e escolhe o alinhamento com melhor qualidade.

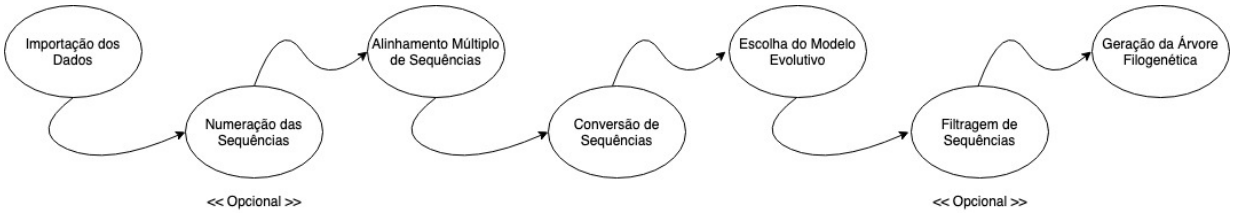


Figura 3. Definição do Experimento de Análise Filogenética

4.2. Resultados

Nesta subseção avaliamos o desempenho do `PolyFlow` em dois experimentos: (i) consultas aos bancos de dados de proveniência de forma isolada e (ii) consulta integrada a dois bancos de dados de proveniência. Para o primeiro experimento consultamos de forma isolada os bancos de dados de proveniência dos SGWf Kepler e Swift/T.

Conforme mencionado na Subseção 3.3, ao submeter uma consulta encapsulada por um mediador, o `PolyFlow` recupera os *Entity Mappers* e expande a consulta utilizando relações do *Schema* Local. Estas operações (*e.g.,* busca dos *Entity Mappers*, expansão da consulta, etc.) introduzem um *overhead* no processamento. Além disso, o tempo de execução de cada consulta no SGBD associado pode aumentar, uma vez que a consulta construída pelo `PolyFlow` não necessariamente é a mais otimizada. Como estratégia de avaliação do *overhead* de definição e expansão das consultas, calculamos o tempo total decorrido desde a requisição ao *endpoint*

do PolyFlow até o recebimento da resposta da consulta e o tempo de execução da consulta propriamente dita no SGBD (hospedado na nuvem *Digital Ocean* no caso do Kepler e localmente no caso do Swift/T).

Utilizamos a entidade *Execution* do modelo ProvONE, a entidade *Entity* e o relacionamento *Used* do PROV (mas que também existem no ProvOne) para avaliar a eficiência das transformações das consultas. Neste contexto, *Used* possui um mapeamento simples (1-1), *Execution* um mapeamento composto (1-2) e *Entity* um mapeamento composto recursivo (1-4). Ao utilizar estas três entidades, conseguimos cobrir os casos de mapeamento descritos na Subseção 3.2. Todos esses *Entity Mappers* estão disponíveis no repositório do Github mencionado anteriormente.

Para essa avaliação inicial, utilizamos consultas simples, projetando todos os atributos dessas entidades e sem considerar agregações. Cada consulta foi executada 10 vezes e os resultados estão dispostos na Tabela 1 (Linhas Kepler e Swift/T). O valor máximo corresponde à primeira requisição, sendo amortizado nas requisições seguintes por estratégias de *caching* utilizadas pelos SGBDS. As requisições foram feitas sequencialmente. O PolyFlow e o SGBD mySQL foram implantados em uma máquina com processador Intel Core i5 @ 2.3GHz, 16GB RAM e SO macOs Mojave 10.14.4. Conforme podemos perceber, uma parcela considerável do tempo total de atendimento da requisição é consumido no processamento da consulta no SGBD associado, *i.e., ilha*. Assim, podemos perceber que o *overhead* adicionado pelo PolyFlow é negligenciável no caso do Kepler (1,6% no pior caso) se comparado ao tempo necessário para processamento da consulta. No caso do Swift/T, como o SGBD estava executando localmente, as consultas foram executadas em um tempo curto se comparado ao tempo que o PolyFlow necessita para expandir a consulta (que é um tempo fixo). Logo, o *overhead* introduzido não é negligenciável nesse caso (90.4% no pior caso). Porém, se consultas mais complexas ou que envolvam a manipulação de um maior volume de dados forem submetidas, a tendência é que o *overhead* reduza em valores percentuais no caso do Swift/T.

O segundo experimento executado avalia consultas integradas aos bancos de dados de proveniência dos SGWfs Kepler e Swift/T. O Wf SciPhy foi modelado no Kepler e o Wf SwiftPhylo no Swift/T. Para essa avaliação integrada comparamos o desempenho do PolyFlow para as consultas executadas de forma integrada, *i.e.,* considerando ambos bancos de dados de proveniência. Assim como no primeiro experimento, utilizamos a entidade *Execution* do modelo ProvONE, a entidade *Entity* e o relacionamento *Used* do PROV. Nesse cenário da avaliação integrada, ambas consultas são submetidas em conjunto ao PolyFlow que se encarrega de distribuir as mesmas para o SGBD associado (*i.e., ilha*) na camada de fonte de dados, conforme exemplificado a seguir:

```
{
    query(query: "SELECT␣*␣FROM␣swift[provone_execution];"),
    query(query: "SELECT␣*␣FROM␣kepler[provone_execution];")
}
```

Assim como no experimento anterior utilizamos consultas simples sem junções ou agregações, projetando todos os atributos dessas entidades. Cada consulta foi executada 10 vezes e os resultados estão dispostos na Tabela 1 (Linha "Integrado"). O valor máximo corresponde à primeira requisição, sendo amortizado nas requisições seguintes por estratégias de *caching* utilizadas pelos SGBDS. As requisições foram feitas sequencialmente.

Como podemos perceber, similarmente ao primeiro experimento, uma parcela considerável do tempo total de atendimento da requisição é consumido no processamento da consulta no SGBD associado (*i.e., ilha*). Porém, é importante ressaltar que no caso da consulta integrada, as requisições aos SGBDs são realizadas em paralelo pelo PolyFlow, *i.e.,* o tempo de consulta

Tabela 1. *Overhead* para diferentes entidades do ProvONE com os dados do Kepler, Swift/T e de forma Integrada.

		Requisição (ms)				Execução no SGBD (ms)				Overhead (ms)				
		min	max	avg	stdev	min	max	avg	stdev	min	max	avg	stdev	avg%
Kepler	Used	842,0	1479,0	1035,3	692,0	834,0	1442,0	1054,5	676,9	6,0	37,0	10,7	29.0	1.6
	Execution	891,0	1132,0	997,1	283,5	885,0	1124,0	985,9	276,0	5,0	49,0	11,2	40,1	1.5
	Entity	903,0	1353,0	1043,8	421,8	895,0	1303,0	1032,7	392,7	5,0	50,0	11,1	41,1	1.5
Swift/T	Used	17.0	78.0	29.9	53.9	9.0	21.0	15.7	15.0	58.0	8.0	14.2	46.7	90.4
	Execution	15.0	42.0	24.6	26.6	9.0	29.0	14.8	17.3	28.0	5.0	9.8	19.8	82.1
	Entity	17.0	108.0	32.6	81.8	11.0	36.0	17.9	24.9	72.0	6.0	14.7	60.6	66.2
Integrado	Used	895.0	1317.0	1076.3	138.4	883.0	1290.0	1056.1	136.7	73.0	7.0	14.6	8.3	1.9
	Execution	894.0	1601.0	1099.8	205.7	887.0	1590.0	1081.5	203.2	72.0	7.0	9.8	1.8	1.7
	Entity	920.0	1785.0	1124.4	258.5	912.0	1775.0	1110.3	258.0	60.0	6.0	8.9	1.8	1.3

computado na Tabela 1 é o máximo entre os tempos de consulta de cada SGBD. Neste caso, sempre as consultas ao banco de dados do Kepler irão dominar o tempo total de execução da consulta, pois eles são consideravelmente superiores aos tempos de execução das consultas no banco de dados do Swift/T. Dessa forma, percebe-se que o *overhead* adicionado pelo `PolyFlow` é negligenciável (1,9% no pior caso) se comparada ao tempo necessário para processamento da consulta. É importante ressaltar que esses resultados foram obtidos a partir de consultas tradicionais da área de proveniência de dados, conforme definido por [de Oliveira et al. 2016]. Entretanto, consultas mais complexas que envolvam dados específicos do domínio, e não somente dados de proveniência, podem ser consideradas, o que aumentaria a complexidade do `PolyFlow`.

5. Considerações Finais e Trabalhos Futuros

Este artigo propõe o `PolyFlow` que possibilita a análise integrada de grafos de proveniência de múltiplos SGWfs, utilizando uma abordagem *PolyStore*. Para isso, foi utilizado o modelo ProvONE como um modelo global para o qual os diferentes bancos de dados de proveniência são mapeados, sob demanda. Bancos de dados de proveniência de experimentos reais da bioinformática executados com os SGWfs Swift/T e Kepler foram integrados utilizando `PolyFlow` para avaliação da abordagem proposta. O `PolyFlow` integrou esses bancos de dados de proveniência heterogêneos de maneira não intrusiva e permitiu ao cientista ter acesso aos dados de proveniência submetendo uma única consulta, permitindo assim uma análise abrangente em um sistema unificado. Resultados demonstram que o *overhead* do `PolyFlow` está relacionado à interação das requisições com o banco de dados, e que é negligenciável na maioria dos casos se comparado ao ganho na obtenção das informações científicas. Até o presente momento o `PolyFlow` não permite que sejam aplicados filtros ou projeções específicas aos resultados integrados de diferentes mediadores. Neste sentido, como trabalho futuro, planejamos implementar uma ilha *PolyStore* e acoplar SGBDs *PolyStore* como o BigDAWG [Gadepally et al. 2016] ao `PolyFlow`, incorporando também novos operadores *shims* e *casts* à arquitetura.

Referências

Altintas, I., Berkley, C., Jaeger, E., Jones, M. B., Ludäscher, B., and Mock, S. (2004). Kepler: An extensible system for design and execution of scientific workflows. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM 2004), 21-23 June 2004, Santorini Island, Greece*, pages 423–424.

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

de Oliveira, W. M., Missier, P., Ocaña, K. A. C. S., de Oliveira, D., and Braganholo, V. (2016). Analyzing provenance across heterogeneous provenance graphs. In *Provenance and Annota-*

tion of Data and Processes - 6th International Provenance and Annotation Workshop, IPAW 2016, McLean, VA, USA, June 7-8, 2016, Proceedings, pages 57–70.

Dziedzic, A., Elmore, A. J., and Stonebraker, M. (2016). Data transformation and migration in polystores. In *2016 IEEE High Performance Extreme Computing Conference, HPEC 2016, Waltham, MA, USA, September 13-15, 2016*, pages 1–6.

Ellqvist, T., Koop, D., Freire, J., Silva, C., and Strömbäck, L. (2009). Using mediation to achieve provenance interoperability. In *Services-I, 2009 World Conference on*, pages 291–298. IEEE.

Freire, J., Koop, D., Santos, E., and Silva, C. T. (2008). Provenance for Computational Tasks: A Survey. *Computing in Science & Engineering*, pages 20–30.

Gadepally, V., Chen, P., Duggan, J., Elmore, A., Haynes, B., Kepner, J., Madden, S., Mattson, T., and Stonebraker, M. (2016). The bigdawg polystore system and architecture. In *High Performance Extreme Computing Conference (HPEC), 2016 IEEE*, pages 1–6. IEEE.

Mattoso, M., Werner, C., Travassos, G. H., Braganholo, V., Ogasawara, E. S., de Oliveira, D., da Cruz, S. M. S., Martinho, W., and Murta, L. (2010). Towards supporting the life cycle of large scale scientific experiments. *IJBPIM*, 5(1):79–92.

Missier, P., Ludäscher, B., Bowers, S., Dey, S., Sarkar, A., Shrestha, B., Altintas, I., Anand, M. K., and Goble, C. (2010). Linking multiple workflow provenance traces for interoperable collaborative science. In *WORKS 2010*, pages 1–8. IEEE.

Mondelli, M. L., Magalhães, T., Loss, G., Wilde, M., Foster, I. T., Mattoso, M., Katz, D. S., Barbosa, H. J. C., de Vasconcelos, A. T. R., Ocaña, K. A. C. S., and Jr., L. M. R. G. (2018). Bioworkbench: A high-performance framework for managing and analyzing bioinformatics experiments. *CoRR*, abs/1801.03915.

Moreau, L., Freire, J., Futrelle, J., McGrath, R. E., Myers, J., and Paulson, P. (2008). The open provenance model: An overview. In *International Provenance and Annotation Workshop*, pages 323–326. Springer.

Moreau, L. and Groth, P. T. (2013). *Provenance: An Introduction to PROV*. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool Publishers.

Ocaña, K. A., de Oliveira, D., Ogasawara, E., Dávila, A. M., Lima, A. A., and Mattoso, M. (2011). Sciphy: a cloud-based workflow for phylogenetic analysis of drug targets in protozoan genomes. In *BSB11*, pages 66–70. Springer.

Oliveira, W., Missier, P., Ocaña, K., de Oliveira, D., and Braganholo, V. (2016). Analyzing provenance across heterogeneous provenance graphs. In *IPAW*, pages 57–70. Springer.

Özsu, M. T. and Valduriez, P. (2011). *Principles of distributed database systems*. Springer Science & Business Media.

Prabhune, A., Zweig, A., Stotzka, R., Gertz, M., and Hesser, J. (2016). Prov2one: an algorithm for automatically constructing provone provenance graphs. In *IPAW*, pages 204–208. Springer.

Prabhune, A., Zweig, A., Stotzka, R., Hesser, J., and Gertz, M. (2018). P-PIF: a provone provenance interoperability framework for analyzing heterogeneous workflow specifications and provenance traces. *Distributed and Parallel Databases*, 36(1):219–264.

Wozniak, J. M., Armstrong, T. G., Wilde, M., Katz, D. S., Lusk, E. L., and Foster, I. T. (2013). Swift/t: Large-scale application composition via distributed-memory dataflow processing. In *13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2013, Delft, Netherlands, May 13-16, 2013*, pages 95–102.