Uma Estratégia para Versionamento dos Dados de *Workflows* Científicos Executados em Nuvem

Fabrício Nogueira[1], Kary Ocaña[2], Vítor Silva[3], Vanessa Braganholo[1], Daniel de Oliveira[1]

[1] Instituto de Informática – Universidade Federal Fluminense (UFF)
[2] Laboratório Nacional de Computação Científica (LNCC)
[3] Programa de Engenharia de Sistemas e Computação – COPPE / UFRJ

{fnogueira, vanessa, danielcmo}@ic.uff.br, karyann@lncc.br, silva@cos.ufrj.br

Abstract. *Scientific experiments usually run hundreds or thousands of times, generating a huge amount of data that requires to be managed. Analyzing and comparing the results of such experiments is an extremely complex task. This becomes even more complex for workflows running in the cloud because the data is scattered across multiple virtual machines. In order to alleviate this problem, previous work proposed the use of a version control system to manage the data consumed and generated by scientific experiments. However, they add considerable overhead to the experiment, increasing the processing time and the use of disk space. In this article, we propose an alternative strategy to reduce time and space. Our initial experiments show that the time overhead of our approach is still high, but disk overhead was 5 times smaller than the approaches in the literature.*

1. Introdução

W*orkflows* científicos têm se mostrado como uma das formas mais utilizadas para representação de experimentos *in silico* (Mattoso et al., 2010). A gerência desses *workflows* requer sistemas apropriados, que são denominados Sistemas de Gerência de *Workflows* Científicos (SGWfC) (Deelman et al., 2009). Muitos experimentos existentes possuem atividades de larga escala que são executadas diversas vezes e exigem alto poder computacional. De fato, cada experimento pode envolver diversas execuções ou re-execuções dos diferentes *workflows* variando os parâmetros e/ou dados de entrada. Verifica-se, com isso, a execução de *workflows* em ambientes de processamento de alto desempenho (PAD), como por exemplo as nuvens de computadores. Para apoiar execução de *workflows* nesses ambientes, alguns SGWfCs, como o SciCumulus (Oliveira et al., 2012) e o Pegasus (Deelman et al., 2015) têm sido desenvolvidos e evoluídos.

Os SGWfCs existentes não apoiam bem a análise dos resultados de várias execuções de um mesmo *workflow,* nem fazem a gerência da evolução dos dados de execuções de *workflows* científicos. Tal problema torna-se ainda mais complexo em certos ambientes de PAD, como as nuvens, onde a execução das atividades do *workflow* pode ser feita de maneira distribuída, em máquinas virtuais (VMs) e a localização e rastreamento dos dados podem se tornar ainda mais complexos.[§]

[§] Os autores gostariam de agradecer ao CNPq e a FAPERJ por financiarem parcialmente este trabalho.

Para exemplificar o problema, tomemos como exemplo o *workflow* SciPhy (Ocaña et al., 2011) (consistentemente usado em todo o artigo). O *workflow* SciPhy realiza análises filogenéticas de sequências de DNA, RNA e aminoácidos e é constituído por cinco atividades (Figura 1.a). A primeira atividade realiza um pré-processamento dos dados de entrada (arquivo multi-fasta contendo as sequências), a segunda alinha as sequências dadas como entrada utilizando programas tradicionais como o *mafft*, a terceira atividade converte o alinhamento gerado para o padrão *phylip* com o programa *readseq*, a quarta atividade escolhe o melhor modelo evolutivo a ser usado com o programa *modelgenerator* e finalmente a quinta atividade cria as árvores filogenéticas utilizando o programa *RaxML*. Tradicionalmente o SciPhy é executado centenas ou milhares de vezes em um mesmo experimento (varredura de parâmetros), recebendo como entrada um *dataset* de arquivos multi-fasta. Em uma mesma execução do SciPhy, um arquivo multi-fasta é modificado em diversas atividades (nas atividades 1, 2 e 3) e conhecer todas as transformações e versões desse arquivo pode ser importante para analisar o comportamento do experimento como um todo. Dessa forma, caso as versões dos dados não sejam gerenciadas, pode se tornar complexo compreender *a posteriori* quais foram as transformações de dados que ocorreram. Além disso, como o SciPhy é executado *n* vezes, o cientista precisa ser capaz de comparar as *n* execuções do *workflow.* Sem um ferramental de apoio apropriado, essa comparação é inviável.

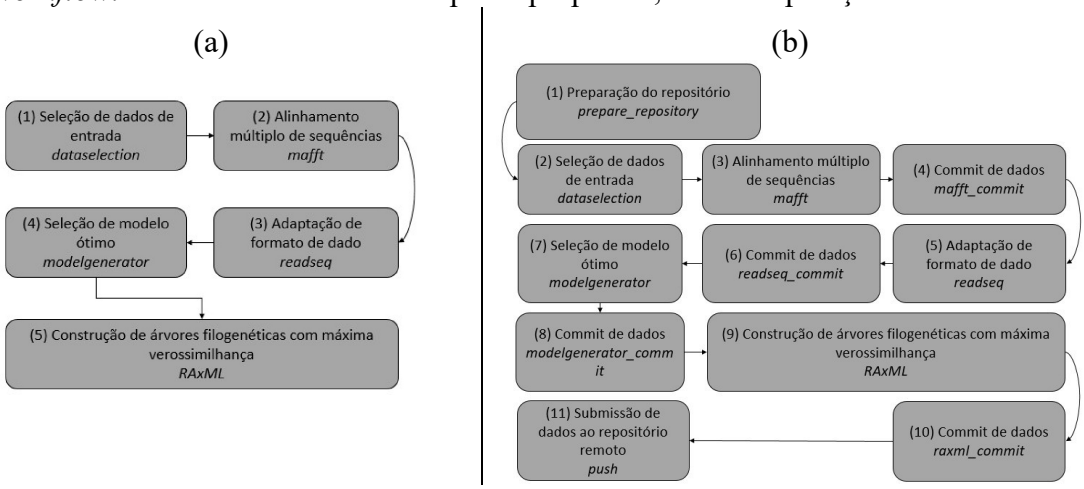

Figura 1. (a) *Workflow* SciPhy original. (b) *Workflow* SciPhy com atividades de versionamento de dados.

Uma maneira de manter a rastreabilidade das modificações sofridas pelos dados durante execuções de *workflows* científicos é o uso de técnicas de versionamento já aplicadas largamente na área de Gerência de Configuração de *Software* (GCS). Alguns trabalhos (Callahan et al., 2006; Costa et al., 2009; Ogasawara et al., 2009) propõem abordagens de versionamento da estrutura de *workflows* científicos, porém, não tratam de seus dados. Mesmo os trabalhos que consideram os dados (Neves et al., 2013; Oliveira et al., 2014) podem ser considerados incipientes uma vez que ainda podem ser otimizados, já que geram uma considerável sobrecarga de tempo de execução do *workflow* e de uso de espaço em disco. Assim, abordagens mais eficientes em termos de tempo e espaço se fazem necessárias para solucionar o problema.

A hipótese principal desse artigo é que a manutenção e o versionamento dos dados são fundamentais para o processo de análise e validação dos resultados de um experimento. Além disso, o versionamento ainda pode auxiliar no compartilhamento e

na reprodutibilidade dos experimentos, uma vez que repositórios versionados podem ser facilmente compartilhados entre diferentes usuários. Dessa forma, esse artigo apresenta uma estratégia de versionamento de dados de *workflows* científicos em um ambiente de nuvem computacional. Considerando-se que o processo de composição de workflows científico compartilha muitas características com o desenvolvimento de software, utilizou-se as técnicas de versionamento propostas pela disciplina de gerência de configuração da engenharia de software. A abordagem proposta objetiva promover rastreabilidade aos dados gerados por diversas execuções paralelas e concorrentes de um mesmo *workflow* em um ambiente de nuvem de computadores, como a Amazon EC2, diminuindo a sobrecarga de tempo e espaço das abordagens atuais (Neves et al., 2013; Oliveira et al., 2014). Como prova de conceito, a estratégia proposta foi implementada no SGWfC baseado em nuvem SciCumulus.

Este artigo está organizado em 5 seções, além desta introdução. A Seção 2 apresenta conceitos de gerência de configuração, necessários ao entendimento da abordagem proposta. A Seção 3 detalha a estratégia de versionamento proposta. A Seção 4 apresenta uma avaliação experimental da abordagem proposta. A Seção 5 discute trabalhos relacionados, e, finalmente, a Seção 6 conclui o artigo.

2. Referencial Téorico

A necessidade de versionamento surgiu na indústria de *software*, principalmente em grandes projetos, onde se fazia necessário registrar a inclusão, modificação ou exclusão de código e funcionalidades. Dessa forma, foram propostos diversos sistemas de controle de versões (SCV) para solucionar tal problema. Existem diversos sistemas como o CVS, o SVN e o Git. Nesse trabalho optamos por utilizar o Git como SCV uma vez que o mesmo é um SCV distribuído que permite que vários usuários trabalhem de forma independente e sincronizem seus trabalhos *a posteriori*, o que se assemelha bastante ao comportamento de um *workflow* executado na nuvem.

No Git existem alguns conceitos importantes que devem ser analisados. O primeiro é o conceito de repositório. Um repositório é um local de armazenamento de dados que podem ser armazenados e recuperados. Um repositório contém arquivos diversos cujas versões são gerenciadas pelo Git. O repositório reflete a história do projeto, contendo todas as versões até a mais recente (HEAD). Quando um usuário executa uma operação de *commit,* o repositório é atualizado com as modificações daquele *commit*, *i.e.* novas versões de arquivos podem ser criadas, caso modificações tenham sido realizadas. O Git possui o conceito de repositório remoto e local. No Git, o repositório local é acessado somente por um usuário e se localiza na própria máquina do usuário (no nosso caso, na(s) máquina(s) em que o *workflow* estiver executando). Já os repositórios remotos são versões do projeto (no nosso caso, da execução do *workflow*) que estão hospedados em nós de uma rede (local ou na Internet). Outro conceito importante é o de ramo no repositório (*branch*). Um ramo funciona como um *snapshot* do projeto, *i.e.* ele é uma cópia do projeto em que o usuário pode trabalhar de forma isolada. Ao fim do trabalho o usuário pode realizar um *merge* de ramos. Uma vez terminado o trabalho, o usuário pode realizar um *push* do seu repositório local para o repositório remoto. É importante observar que antes de submeter as modificações do repositório local ao repositório remoto (*push*) é necessário que haja a sincronização de ambos os repositórios, ou seja, as possíveis novas modificações existentes no repositório remoto sejam trazidas ao repositório local. Tal sincronização é realizada

através do comando *pull* do Git. A ideia desse trabalho é acoplar o SCV Git a um SGWfC de forma a prover o versionamento de dados em um *workflow*.

3. A Estratégia de Versionamento de Dados Proposta

Inspirado em trabalhos anteriores (Neves et al., 2013; Oliveira et al., 2014), a estratégia de versionamento proposta nesse artigo considera que, apesar dos dados serem gerados em VMs diferentes (uma vez que o *workflow* possui atividades executando em paralelo em múltiplas VMs) no ambiente de nuvem computacional, os mesmos devem ser armazenados em um repositório de versionamento comum a todo o *workflow*. Para isso, considerou-se que, além das atividades relacionadas à invocação de programas científicos (as atividades originais do *workflow*), a modelagem do *workflow* deve incluir as atividades relacionadas ao processo de versionamento, *i.e.* o *workflow* deve ser instrumentado. Assim, novas atividades devem ser dispostas após cada atividade original do *workflow*, de forma que os dados gerados pelos programas científicos sejam, logo em sequência, adicionados ao repositório de versionamento.

Como mencionado anteriormente, utilizamos o *workflow* SciPhy (Ocaña et al., 2011) como estudo de caso nesse artigo. O *workflow* foi modificado para incluir as atividades relacionadas ao processo de versionamento dos dados consumidos e produzidos. A Figura 1.a apresenta o *workflow* SciPhy original e sua nova representação após adição das atividades de versionamento. Na Figura 1.b, o mesmo *workflow* SciPhy é apresentado com as atividades adicionais relacionadas ao versionamento de dados (atividades 1, 4, 6, 8 e 11). A atividade 1 (Figura 1.b) cria um repositório local a partir de um repositório remoto já criado anteriormente (*clone*). As atividades de *commit* (4, 6, 8 e 10) adicionam os arquivos gerados na VM por cada uma das atividades anteriores no repositório local. É importante ressaltar que cada uma das VMs envolvidas na execução do *workflow* possui um repositório local independente. Por fim, a atividade 11 submete todos os arquivos do repositório local ao repositório remoto. É importante atentar para o fato de que todas as atividades do *workflow* SciPhy adaptado (Figura 1.b) executam em cada uma das VMs alocadas à execução do *workflow* na nuvem. Porém, a atividade de submissão de dados ao repositório remoto deve ser realizada de maneira sincronizada uma vez que todas as VMs que executam o *workflow* fazem referência a um mesmo repositório remoto, *i.e.* deve ser garantido que cada repositório local se encontre sincronizado com o repositório remoto antes da submissão dos arquivos (operação de *push*). Para lidar com essa questão, a estratégia proposta utilizou duas abordagens de versionamento, descritas a seguir.

A1. **Um ramo por execução**: Cada execução do *workflow* na nuvem cria um ramo (*branch*) no repositório. Os dados produzidos pelas atividades do *workflow* executando em diferentes VMs são armazenados em um repositório local na VM. Ao término da execução do *workflow* em todas as instâncias, cada VM submete (*push*) os dados do repositório local para o repositório remoto. O *push* deve ser realizado sequencialmente em cada uma das VMs alocadas ao *workflow* na nuvem. Isso é necessário para que cada repositório local possa ser sincronizado com o repositório remoto a fim de evitar concorrências de submissão pelas VMs que executam atividades em paralelo.

A2. **Um repositório por execução**: possibilita o *push* dos dados de maneira paralela pelas diferentes VMs. Para isso, cada VM faz uso de um ramo próprio para os dados no repositório. Ao final da execução do *workflow*, os dados gerados em cada VM se

encontram nos seus respectivos ramos e o repositório remoto faz referência àquela execução do *workflow*. Um novo repositório é criado para cada nova execução.

Ambas as abordagens apresentam vantagens e desvantagens. Em A1, a vantagem é a existência de um único repositório por *workflow*. Cada execução do *workflow* gera um ramo no repositório. A desvantagem é a necessidade de um passo sequencial (não paralelizável) em todas as VMs. O passo sequencial está associado à execução do *push* dos repositórios locais ao repositório remoto. Em A2, a vantagem é que cada VM realiza a submissão para o repositório remoto sem necessitar aguardar o *status* das demais VMs. Isso é possível porque as VMs possuem ramos exclusivos no repositório remoto. Em contrapartida, o repositório se torna exclusivo para uma execução do *workflow*. Havendo a necessidade de novas execuções, outros repositórios deverão ser criados, o que pode gerar sobrecarga tanto de espaço quanto na gerência dos repositórios, além de dificultar a etapa de análise quando se deseja comparar resultados de execuções de vários *workflows*.

As duas abordagens impõem certa sobrecarga à execução do *workflow* uma vez que novas atividades para acoplamento com o SCV são executadas. A estratégia aqui apresentada é uma evolução do trabalho de Neves *et al.* (2017) que apresenta o versionamento de dados também utilizando repositórios locais e remotos. Para conferir um versionamento de dados em maior ou menor grão, Neves *et al.* (2017) propõe estratégias que consideram: (i) a criação de um único repositório local para o *workflow* gerando ramos a cada execução do mesmo e (ii) a criação de repositórios locais para cada atividade do *workflow* gerando ramos por execução do *workflow* ou da atividade. As duas abordagens propostas neste trabalho diferenciam-se das abordagens propostas em Neves *et al.* (2017) por utilizarem repositórios locais por VMs alocadas ao *workflow* executando na nuvem e não por atividade do *workflow*, o que faz com que sejam necessários menos repositórios. Além disso, as atividades do *workflow* que são executadas numa mesma VM são capazes de compartilhar um mesmo repositório local.

4. Análise de Desempenho da Proposta

O acoplamento do SGWfC com o SCV impõe que o *workflow* seja instrumentado com novas atividades como apresentado na Seção 3. Para cada abordagem proposta, analisamos a sobrecarga de tempo de execução que as novas atividades geram na execução do *workflow* bem como o espaço em disco adicional necessário. Para melhor entendimento dos testes realizados para avaliação de desempenho, consideramos: (i) T_{A1} e T_{A2}, como sendo os tempos de execução das abordagens A1 e A2 respectivamente; (ii) T_x, como sendo o tempo de execução do *workflow* na VM *x* da nuvem sem a atividade *push*; (iii) T_{Px}, como sendo o tempo de execução da atividade *push* na VM *x* da nuvem; e (iv) T_{Cx}, como sendo o tempo de execução de todas as atividades (incluindo *push*) do *workflow* na VM *x* da nuvem. A partir de tais definições, estabelecemos que o tempo de execução da abordagem de um ramo por execução é:

$$T_{A1} = \max\{T_1, T_2, \dots, T_n\} + \sum_{i=1}^{n} T_{Pi}$$

Já a abordagem de um repositório por execução que não necessita de execução sequencial tem seu tempo de execução calculado da seguinte forma:

$$T_{A2} = \max\{T_{C1}, T_{C2}, \dots, T_{Cn}\}$$

Para mensurar o tempo de execução das duas abordagens planejou-se um experimento que consistiu em executar o *workflow* SciPhy com o SciCumulus acoplado ao Git na nuvem da Amazon EC2. Para o experimento, foram alocadas 3 VMs EC2 de baixo custo (t1.micro) de 64 bits com uma CPU de 3,3 GHz, 1 GB de RAM e aproximadamente 700 MB de armazenamento em disco. Essas 3 VMs são responsáveis pela execução das atividades do *workflow*. Além delas, foi alocada um *bucket* no Amazon S3 para o armazenamento compartilhado dos dados de saída das atividades do *workflow*. O uso desses dois tipos de recursos (EC2 e S3) é requisito padrão dos SGWfCs SciCumulus e Pegasus, que são capazes de executar na nuvem da Amazon.

Após o levantamento dos recursos necessários na nuvem, foi realizada a medição de tempo de execução do *workflow* SciPhy original, sem as atividades de versionamento dos dados. Isso foi necessário para se ter uma medida de controle (*baseline*) e comparar com os tempos da proposta com versionamento. A medição de tempo foi realizada executando-se o SciPhy por 5 vezes e calculando-se a média aritmética de tais execuções. A mesma estratégia de medição de tempo foi feita para a execução do *workflow* SciPhy adaptado com versionamento, ou seja, cada abordagem (***A1*** e ***A2***) foi executada 5 vezes obtendo-se em seguida a média aritmética dessas execuções. A Figura 2 apresenta a comparação de tempo das execuções do SciPhy.

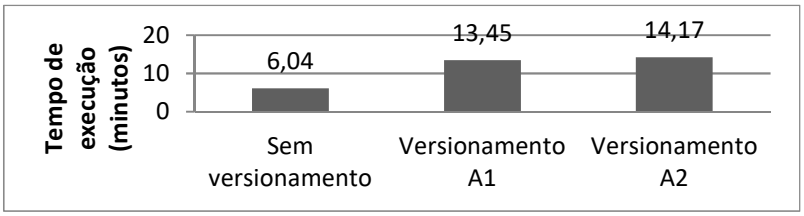

Figura 2. Comparação dos tempos de execução do *workflow* SciPhy original com as duas abordagens de versionamento propostas

As duas abordagens que usam versionamento resultaram em um tempo aproximadamente duas vezes maior que o tempo de execução do *workflow* SciPhy original. Isso pode ser explicado pelo fato da operação de versionamento (*commit)* ser executada logo após cada atividade do *workflow* SciPhy e a operação de versionamento (*push*) ser executada ao término de todas as atividades. Como o *workflow* original apresenta poucas atividades, o impacto dessas atividades de versionamento no *workflow* é considerável. Considerando-se um *workflow* com maior número de atividades, a sobrecarga das atividades de versionamento seria diluída e consequentemente o tempo adicional imposto sobre o tempo total de execução do *workflow* será menor. Pode-se observar pela Figura 2 que as abordagens A1 e A2 apresentaram um tempo de execução muito próximos. Como nas duas abordagens, as atividades de versionamento (*commit*) são executadas da mesma forma e a atividade *push* é executada de forma sequencial no versionamento A1 e de forma paralela no versionamento A2, a proximidade dos tempos de execução pode ser explicada pelo fato de que em A1 a fila de realização da atividade *push* vai sendo formada à medida que a execução do workflow é finalizada em cada instância. Após o término da execução do workflow em todas as instâncias, a fila de *push é* executada. Além disso, percebe-se que a atividade *push* impõe pouca influência à execução do *workflow*. Comparando-se os resultados de sobrecarga de tempo de execução imposta ao *workflow* com os resultados de Neves *et al.* (2017), verifica-se que as abordagens aqui propostas possuem um comportamento similar às verificadas no

referido trabalho. No trabalho de Neves *et al.* (2017) algumas atividades do mesmo *workflow* SciPhy tiveram o tempo de execução aumentado em até 7 vezes devido às atividades extras de versionamento. Nas medições feitas para o tempo do *workflow* como um todo, Neves *et al.* (2017) relata que a sobrecarga de tempo fica em média 26% superior em relação à versão sem versionamento.

Quanto à sobrecarga de uso de disco, cada VM cria um repositório local e nele adiciona os arquivos gerados. Logo, o uso de disco é dobrado uma vez que os arquivos estarão armazenados no *bucket* S3 e também nas VMs EC2, sendo que os dados da VM EC2 são somente os gerados pela própria, enquanto que na S3 são os gerados por todas as VMs envolvidas na execução. Para verificar esse comportamento, analisou-se o tamanho do conjunto de dados gerado pelas execuções do *workflow* e que foram armazenados no *bucket* S3. Esse conjunto de dados, para cada execução do *workflow*, ocupava aproximadamente 870 KB de disco. Em seguida, para cada execução, analisou-se também o conjunto de dados que foi armazenado no repositório local Git das instâncias VM EC2. O tamanho desse conjunto de dados ficou em aproximadamente 290 KB. Isso corrobora a característica de que o uso de disco é dobrado devido à geração dos repositórios de versionamento locais. No trabalho de Neves *et al.* (2017), observa-se que durante a execução do *workflow* o uso de disco chega a ser mais de 10 vezes superior em relação à execução sem versionamento. Esse comportamento ocorre principalmente por causa da replicação de dados de entrada de todas as atividades do *workflow* na nuvem. Tal comportamento já não é observado nas duas abordagens propostas neste artigo uma vez que o número de repositórios locais é reduzido, já que são definidos para as instâncias do *workflow* e não para as atividades, o que reduz consideravelmente as replicações. Pode-se concluir que, apesar do tempo de execução para o *workflow* como um todo ter sido superior à abordagem proposta por Neves (2017), a sobrecarga de disco foi consideravelmente inferior. Isso abre oportunidades para que otimizações sejam feitas para redução da sobrecarga de tempo de execução.

5. Trabalhos Relacionados

Existem trabalhos que propõem controle de versão no contexto de *workflows* científicos (Callahan et al., 2006; Costa et al., 2009; Koop et al., 2010; Neves et al., 2013; Ogasawara et al., 2009). Alguns desses trabalhos propõem o uso de versionamento para controlar alterações e adaptações que a estrutura de um *workflow* pode sofrer dentro de um mesmo experimento (Callahan et al., 2006; Costa et al., 2009; Ogasawara et al., 2009). Poucos trabalhos têm considerado o uso dessa técnica para os dados gerados nas execuções de *workflows* científicos (Koop et al., 2010; Neves et al., 2017). No ambiente de computação em nuvem, os dados são gerados em VMs independentes na nuvem. Nesse caso, é importante que, apesar de estarem em locais de armazenamento diferentes, os dados possam ser armazenados em um repositório de versionamento comum à execução do *workflow*. O trabalho de Neves *et al.* (2017) e Oliveira *et al.* (2014) utiliza SCV para armazenar os dados gerados pela execução das atividades do *workflow* científico na nuvem de maneira similar às abordagens propostas neste trabalho. A diferença entre os dois trabalhos se dá pela forma com que os repositórios locais são criados. Este trabalho considera a criação de um repositório local por VM da nuvem. Tal repositório é compartilhado por todas as atividades executadas na mesma VM. Neves *et al.* (2017) considera a criação de um repositório local por atividade do *workflow*, que possibilita o versionamento de dados em um menor grão. O trabalho de

Koop *et al.* (2010) propõe uma infraestrutura que alia informações de proveniência com a gerência de versões nos dados de maneira similar a um SCV, porém não considera o versionamento dos dados num contexto de um ambiente de nuvem computacional.

6. Conclusão e Trabalhos Futuros

A estratégia de versionamento proposta neste artigo facilita a análise e por consequência a reprodutibilidade de um experimento *in silico*. Isso é possível pelo fato dos dados gerados durante a execução do *workflow* que modela o experimento estarem disponíveis em um repositório de versionamento central e associados à execução do *workflow*. Além disso, essa proposta também facilita análise do relacionamento de dados de diferentes execuções, algo que pode ser bastante complexo sem o apoio de um ferramental adequado.

A instrumentação do *workflow* e a adição de atividades extras para controlar versão impôs uma sobrecarga na execução do *workflow* tanto relacionada ao tempo de execução quanto ao uso de disco. Porém, tais sobrecargas podem ser consideradas aceitáveis, visto que trazem um benefício de análise e reprodutibilidade ao *workflow.* Além disso, a inclusão de atividades de versionamento foi feita de maneira manual e impõe trabalho extra ao processo de definição do workflow. No futuro, pretendemos criar adaptadores automáticos para realizar essa tarefa de adaptação. O sistema de controle de versão utilizado, o Git, possui algumas otimizações como o uso de *hardlinks* que ainda podem ser explorados a fim de evitar a cópia de arquivos para o repositório local reduzindo a sobrecarga de uso de disco que a estratégia proposta apresentou.

7. Referências

Callahan, S.P., Freire, J., Santos, E., Scheidegger, C.E., Silva, C.T., Vo, H.T., 2006. VisTrails: visualization meets data management, in: SIGMOD, pp. 745–747.

Costa, B., Ogasawara, E., Murta, L., Mattoso, M., 2009. Uma Estratégia de Versionamento de Workflows Científicos em Granularidade Fina. In: e-Science Workshop, pp. 49–56.

Deelman, E., Gannon, D., Shields, M., Taylor, I., 2009. Workflows and e-Science: An overview of workflow system features and capabilities. FGCS 25, 528–540.

Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P.J., Mayani, R., Chen, W., Ferreira da Silva, R., Livny, M., Wenger, K., 2015. Pegasus, a workflow management system for science automation. FGCS 46, 17–35.

Koop, D., Santos, E., Bauer, B., Troyer, M., Freire, J., Silva, C.T., 2010. Bridging Workflow and Data Provenance Using Strong Links, in: SSDBM, pp. 397–415.

Mattoso, M., Werner, C., Travassos, G.H., Braganholo, V., Ogasawara, E., Oliveira, D., Cruz, S., Martinho, W., Murta, L., 2010. Towards supporting the life cycle of large scale scientific experiments. Int. J. Bus. Process Integr. Manag. 5, 79–92.

Neves, V., de Oliveira, D., Ocaña, K., Braganholo, V., Murta, L., 2017. Managing Provenance of Implicit Data Flows in Scientific Experiments. ACM Trans. Internet Technol. to appear.

Neves, V.C., Braganholo, V., Murta, L., 2013. Implicit Provenance Gathering through Configuration Management, in: SE-CSE, pp. 92–95.

Ocaña, K., Oliveira, D. de, Ogasawara, E., Dávila, A., Lima, A., Mattoso, M., 2011. SciPhy: A Cloud-Based Workflow for Phylogenetic Analysis of Drug Targets in Protozoan Genomes, in: Advances in Bioinformatics and Computational Biology, Lecture Notes in Computer Science. Springer, pp. 66–70.

Ogasawara, E., Rangel, P., Murta, L., Werner, C., Mattoso, M., 2009. Comparison and versioning of scientific workflows, in: CVSM, pp. 25–30.

Oliveira, D. de, Ocaña, K.A.C.S., Baião, F., Mattoso, M., 2012. A Provenance-based Adaptive Scheduling Heuristic for Parallel Scientific Workflows in Clouds. J. Grid Comput. 10, 521–552.

Oliveira, W., Neves, V.C., Ocaña, K., Murta, L., Oliveira, D., Braganholo, V., 2014. Captura e Consulta a Dados de Proveniência Retrospectiva Implícita Intra-Atividade, in: SBBD, pp. 37–46.