EXECUÇÃO INTERATIVA DE EXPERIMENTOS CIENTÍFICOS COMPUTACIONAIS EM LARGA ESCALA

Jonas Furtado Dias

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientadores: Marta Lima de Queirós Mattoso
Patrick Valduriez

Rio de Janeiro
Dezembro de 2013

EXECUÇÃO INTERATIVA DE EXPERIMENTOS CIENTÍFICOS COMPUTACIONAIS EM LARGA ESCALA

Jonas Furtado Dias

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

_____

Prof[a]. Marta Lima de Queirós Mattoso, D.Sc.

_____

Prof. Patrick Valduriez, Dr.

_____

Prof[a]. Lúcia Maria de Assumpção Drummond, D.Sc.

_____

Prof. Fabio Andre Machado Porto, D.Sc.

_____

Prof. Fernando Alves Rochinha, D.Sc.

_____

Prof. Claudio Luis de Amorim, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
DEZEMBRO DE 2013

Dias, Jonas Furtado

Execução Interativa de Experimentos Científicos Computacionais em Larga Escala / Jonas Furtado Dias. – Rio de Janeiro: UFRJ/COPPE, 2013.

XIII, 118 p.: il.; 29,7 cm.

Orientadores: Marta Lima de Queirós Mattoso

Patrick Valduriez

Tese (doutorado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2013.

Referências Bibliográficas: p. 108-118.

1. *Workflows* Científicos Dinâmicos. 2. Interatividade. 3. Iteração. I. Mattoso, Marta Lima de Queirós II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Agradecimentos

Há decisões que tomamos sozinhos, mas as consequências, involuntariamente, compartilhamos. Se foi o meu esforço e dedicação que trouxeram esse resultado, foi o apoio, carinho, compreensão e orientação de muitos que fazem deste resultado uma vitória. É por estarem ao meu lado que agradeço e espero poder recompensá-los com alegria além de um singelo obrigado.

Por dar ao meu coração a certeza e a coragem, agradeço a Deus pela caminhada. Aos meus pais, Eliane e Wilson, agradeço por minha vida, por minha educação e pelo exemplo de amor. Obrigado pelo apoio incondicional, pela sabedoria, paciência e alegria ao estar junto de vocês. À minha irmã, Juliana, agradeço o incentivo e o carinho. À minha noiva, Carol, um agradecimento especial pela compreensão e por todos os bons momentos que vivemos juntos nestes últimos anos. A minha madrinha, Ana Carla, agradeço o apoio e todo o carinho. Agradeço a todos da minha família por me proporcionarem tantos momentos alegres e me motivarem a seguir em frente.

Agradeço os professores que me orientaram nesta jornada de aprendizado. À Professora Marta Mattoso, obrigado pelo cuidado, dedicação e confiança. Agradeço a orientação responsável, atenciosa e por conduzir o meu trabalho através de tantas sugestões, ideias e discussões de grande valor. Ao Professor Patrick Valduriez, obrigado por compartilhar de sua experiência profissional e dar sugestões chave que estimularam o desenvolvimento de tantos conceitos do meu trabalho.

Agradeço também aos Professores Lúcia Drummond, Fabio Porto, Fernando Rochinha e Claudio Amorim por terem aceitado fazer parte da banca desta tese. Também agradeço aos demais que participaram da minha formação: Albino Aveleda, Alexandre Assis, Álvaro Coutinho, Beatriz Lima, Geraldo Xexéo, Geraldo Zimbrão, Guilherme Travassos, Jano de Souza, Leonardo Murta, Myrian Costa, Renato Elias e Valmir Barbosa.

Um agradecimento especial aos amigos Eduardo Ogasawara e Daniel de Oliveira por todas as ideias, sugestões e discussões que me ajudaram a construir e fortalecer a minha pesquisa. Agradeço também à Kary Ocaña pela amizade e pela dedicação ao nos ajudar com os experimentos de bioinformática e nas revisões. Obrigado também ao Gabriel Guerra por todas as explicações e ajuda com os experimentos de quantificação de incertezas. Agradeço também aos amigos Vítor Silva e Felipe Horta pela parceria e dedicação. Aos alunos Igor dos Santos, Fernando Pinheiro, Matheus Perrut, Kaique Rodrigues e Lu-

cas Carneiro, agradeço pela parceria no desenvolvimento da Proteus. Também agradeço ao Anderson Marinho, Fernando Chirigati e Flavio Costa pelos trabalhos em conjunto. Aos meus demais amigos, agradeço por ter vocês ao meu lado.

Agradeço também a Mara Prata, por toda ajuda e dicas com as questões administrativas da COPPE. Agradeço também à equipe administrativa do PESC, Cláudia Prata, Solange Santos, Maria Mercedes, Sônia Galliano, Natália Prata, e Gutierrez da Costa.

Ao CNPq, agradeço pela concessão da bolsa de doutorado entre março de 2011 e dezembro de 2013 e pela concessão da bolsa de doutorado sanduíche no programa Ciência sem Fronteiras entre os meses de fevereiro e agosto de 2013.

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

EXECUÇÃO INTERATIVA DE EXPERIMENTOS CIENTÍFICOS COMPUTACIONAIS EM LARGA ESCALA

Jonas Furtado Dias

Dezembro/2013

Orientadores: Marta Lima de Queirós Mattoso

Patrick Valduriez

Programa: Engenharia de Sistemas e Computação

Para lidar com a natureza exploratória da ciência e o processo dinâmico envolvido nas análises científicas, os sistemas de gerência de *workflow*s dinâmicos são essenciais. Entretanto, *workflow*s dinâmicos são considerados como um desafio em aberto, devido à complexidade em gerenciar o *workflow* em contínua adaptação, em tempo de execução, por eventos externos como a intervenção humana. Apoiar iterações dinâmicas é um passo importante na direção dos *workflow*s dinâmicos uma vez que a interação entre o usuário e o *workflow* é iterativa. Porém, o apoio existente para iterações em *workflow*s científicos é estático e não permite mudanças, em tempo de execução, nos dados do *workflow*, como critérios de filtros e margens de erro. Nesta tese, propomos uma abordagem algébrica para dar apoio a iterações centradas em dados em *workflow*s dinâmicos. Propomos o conceito de linhagem da iteração de forma que a gerência dos dados de proveniência seja consistente com as interações com o *workflow*. A linhagem também possibilita que os cientistas interajam com os dados do *workflow* por meio de dois algoritmos implementados no sistema de *workflow*s Chiron. Avaliamos a nossa abordagem utilizando *workflow*s reais em ambientes de execução em larga escala. Os resultados mostram melhorias no tempo de execução de até 24 dias quando comparado com uma abordagem tradicional não iterativa. Realizamos consultas complexas aos resultados parciais ao longo das iterações do *workflow*. A nossa abordagem introduz uma sobrecarga de no máximo 3,63% do tempo de execução. O tempo para executar os algoritmos de interação também é menor que 1 milissegundo no pior cenário avaliado.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

INTERACTIVE EXECUTION FOR LARGE SCALE COMPUTATIONAL SCIENTIFIC EXPERIMENTS

Jonas Furtado Dias

December/2013

Advisors: Marta Lima de Queirós Mattoso

Patrick Valduriez

Department: Systems and Computer Engineering

To tackle the exploratory nature of science and the dynamic process involved in scientific analysis, dynamic *workflow*s have been identified as an open challenge as they are subject to continuous adaptation and improvement. In particular, they require the ability of adapting a scientific *workflow*, at runtime, based on external events such as human interaction. Supporting dynamic iteration is an important step towards dynamic *workflow*s since user interaction with a *workflow* is iterative. However, current support for iteration in scientific *workflow*s is static and does not allow for runtime changes in data such as filter criteria or error thresholds. In this thesis, we propose an algebraic approach to support data-centric iteration in dynamic *workflow*s and a dynamic execution model for these operators. We introduce the concept of iteration lineage so that provenance data management is consistent with dynamic changes in the *workflow*. Lineage also enables scientists to interact with *workflow* data and configuration at runtime through two steering algorithms implemented in Chiron. We evaluate our approach using real large-scale *workflow*s on a large-scale environment. The results show execution time savings up to 24 days when compared to a traditional non-iterative *workflow* execution. We also perform complex queries for partial result analysis along the iterations and we assess the max overhead introduced by our iterative model as 3.63% of execution time. The performance of our proposed steering algorithms run in less than 1 millisecond in the worst-case scenario we measured.

Índice

Índice de Figuras

Índice de Tabelas

Capítulo 1 Introdução

Avanços recentes na ciência da computação permitem o desenvolvimento de experimentos científicos computacionais com análises cada vez mais complexas e refinadas envolvendo grandes volumes de dados. Experimentos em larga-escala são tipicamente compostos de diversos modelos computacionais que são explorados para revelar o comportamento do objeto de estudo. Estas explorações podem incluir tratamento de dados, algoritmos específicos para análise, simulações que progridem ao longo de passos de tempo, programas para filtrar dados, dentre outros. Muitos experimentos requerem o processamento paralelo e distribuído em ambientes de processamento de alto desempenho (PAD), pois os programas envolvidos nas análises fazem uso intensivo de dados e processamento numérico. Gerenciar tais experimentos é uma tarefa muito importante, embora complicada, pois é necessário garantir a sua consistência, reprodutibilidade e armazenamento dos dados de proveniência (Simmhan *et al.* 2005). Para possibilitar uma abordagem sistemática para modelar e executar experimentos científicos computacionais em larga-escala, muitos domínios na ciência utilizam sistemas de *workflows* científicos (Deelman *et al.* 2009).

Um *workflow* científico é composto por um conjunto de atividades e um fluxo de dados entre elas. Cada atividade pode ser um programa de computador que processa um conjunto de dados de entrada e produz um outro conjunto de dados de saída. O *workflow* científico se caracteriza por estabelecer uma série de dependências de dados onde a saída de uma atividade pode estar conectada à entrada de outra através de um enlace de dados. Pode-se associar, ao *workflow* científico, a proveniência de dados, que pode ser caracterizada como: prospectiva ou retrospectiva (Freire *et al.* 2008). A proveniência prospectiva diz respeito à composição do *workflow*: as atividades, suas características e os enlaces de dados entre elas. A proveniência retrospectiva diz respeito ao histórico da execução do *workflow*, como, por exemplo, o tempo de execução de cada atividade e o recurso computacional responsável por sua execução. A proveniência é essencial para a confiabilidade e reprodutibilidade de um experimento científico. Ela também traz oportunidades para aprimorar a gerência da execução do *workflow* (Mattoso *et al.* 2013), como por exemplo, o monitoramento da execução (Pintas *et al.* 2012), para a verificação de falhas (Costa *et al.* 2012), para a análise de

resultados (Horta *et al.* 2013), e para melhorias no escalonamento de tarefas do *workflow* (Oliveira *et al.* 2012).

Sistemas de Gerência de *Workflows* Científicos (SGWfC) (Altintas *et al.* 2004; Callahan *et al.* 2006) podem ser utilizados por cientistas para modelar e orquestrar a execução de seus experimentos em larga escala. SGWfC permitem que cientistas especifiquem *workflows*, suas atividades e os enlaces de dados. Estas informações, as quais estão associadas à proveniência prospetiva, correspondem à especificação do *workflow*. Baseado nesta especificação, o motor do SGWfC gera um plano de execução baseado em um determinado modelo (chamado de modelo de execução ou modelo de computação). Esse modelo varia de uma máquina de *workflow* para outra. Alguns SGWfC (Altintas *et al.* 2004; Pradal *et al.* 2008) permitem a escolha de diferentes modelos de execução (por ex.: encadeado, fluxo de dados síncrono, modelo contínuo no tempo) para execução do *workflow*. O modelo de execução dita como o motor do SGWfC irá orquestrar a execução, produzindo tarefas e escalonando-as para execução. *Workflows* podem ser executados utilizando recursos computacionais locais ou recursos remotos, paralelos e distribuídos (por ex. clusters, grades ou nuvens). SGWfC também fazem o registro de toda a proveniência do *workflow*, embora a forma de armazenamento da proveniência e a forma de acesso para consultá-la possam variar. Por exemplo, alguns SGWfC registram a proveniência na forma de um *log* e, ao final da execução, exportam a proveniência para um banco de dados relacional (Gadelha *et al.* 2011).

SGWfC mostram seu potencial em diferentes áreas da Ciência (Berriman *et al.* 2007; Dávila *et al.* 2008; Guerra *et al.* 2012; Ocaña *et al.* 2012). Um mesmo *workflow* científico pode ser explorado de diferentes formas dentro de um experimento. Normalmente, o cientista explora uma série de combinações possíveis dentre os fatores do experimento. O objetivo do cientista pode ser varrer um espaço de soluções possíveis em busca de um modelo ótimo ou, ao menos, satisfatório para sustentar a sua hipótese investigada. Nessas explorações, o mesmo *workflow* é executado diversas vezes utilizando diferentes combinações de parâmetros e métodos computacionais. Problemas de otimização (Kelley 1999), *workflows* de Dinâmica de Fluidos Computacional (CFD) (Ogasawara *et al.* 2009a), *workflows* de Genômica Comparativa (Ocaña *et al.* 2011a) e *workflows* de Quantificação de Incertezas (Guerra *et al.* 2012) são exemplos práticos de explorações do espaço de entrada. Tais experimentos envolvem tanto a utilização de

varredura de parâmetros (Mattoso *et al.* 2010a; Abramson *et al.* 2011) quanto a utilização de métodos iterativos (Heath 2002). Em problemas de otimização e quantificação de incertezas, uma nova iteração do método pode ser uma nova varredura de parâmetros. O espaço de parâmetros – *i.e.* o conjunto de combinações de parâmetros que se deseja testar – pode, inclusive, ter seu tamanho alterado.

Decisões a respeito do ajuste do número de iterações de um método e possíveis alterações nos parâmetros ou dados de entrada de um problema requerem um dinamismo no fluxo de dados do experimento. Este dinamismo é associado à interatividade entre o cientista e o *workflow*. Pode-se optar, inclusive, por reduzir, aumentar ou mesmo concluir antecipadamente o ciclo de um método durante a sua execução de acordo com os resultados parciais já obtidos. Quando falamos em concluir antecipadamente o ciclo de um método, significa uma interrupção controlada pela máquina de *workflow*, a qual realiza uma conclusão consistente do experimento e seus dados de proveniência. Muitas vezes, em outras abordagens, a interrupção de um ciclo requer abortar a execução do *workflow*, gerando possíveis inconsistências nos resultados e na base de proveniência. O cientista também pode optar por modificar o espaço de parâmetros para adicionar ou remover combinações de parâmetros. Essas alterações podem evitar que o algoritmo convirja prematuramente, em função da adição de novas combinações que afetem a busca movendo-a para fora da região do máximo ou mínimo local. Todas estas decisões podem ser tomadas baseadas em resultados parciais da execução do *workflow*, o que dá à execução do *workflow* um caráter dinâmico.

Workflows dinâmicos são *workflows* científicos, possivelmente distribuídos e colaborativos, que estão sujeitos à rápida reutilização e exploração acompanhados de contínua adaptação e melhorias (Gil *et al.* 2007). Um dos recursos desejados em *workflows* dinâmicos é a capacidade de adaptar a definição do *workflow*, seus parâmetros e dados de entrada baseado em eventos externos como a interação humana, a qual pode ocorrer em todas as etapas do ciclo de vida do experimento científico (Mattoso *et al.* 2010b). O ciclo de vida de um experimento é um processo iterativo que envolve as etapas de composição, de execução e de análise do experimento. Para interagir com um experimento em larga-escala, o cientista precisa monitorar sua execução e, se algum comportamento ou evento interessante ocorrer durante a execução, ele precisa fazer uma análise dos resultados parciais obtidos até o momento.

De acordo com essa análise parcial, ele poderá interagir com a especificação do *workflow* alterando a composição do experimento (Santos *et al.* 2013).

1.1 Caracterização do Problema: Natureza Dinâmica de Experimentos Científicos Computacionais

Diversos SGWfC oferecem mecanismos para executar experimentos científicos em larga-escala em ambientes distribuídos (Deelman *et al.* 2007; Abramson *et al.* 2008; Wang *et al.* 2009; Goecks *et al.* 2010; Wozniak *et al.* 2012). Porém, todos os sistemas de gerência ainda realizam a execução do *workflow* de forma '*offline'*, de acordo com Ailamaki *et al.* (2010; 2011). Ou seja, as abordagens existentes só disponibilizam os resultados e os dados de proveniência para consulta e análise depois de terminar a execução do *workflow*. Entretanto, conforme o volume de dados e a necessidade de poder computacional aumentam, cientistas precisam de mecanismos para: monitorar, analisar resultados parciais, tomar decisões e intervir dinamicamente na execução do *workflow*, ao invés de aguardar o término da execução para efetuar as mudanças. A execução das atividades do *workflow* pode precisar ser monitorada por dias, semanas ou mesmo meses, para que o cientista realize o ajuste fino do *workflow* (Gil *et al.* 2007).

Muitas das ações dinâmicas estão relacionadas à mudanças no conjunto de dados a ser processado, por exemplo mudanças nos parâmetros, critérios de filtro, margens de erro e critérios de parada. Interromper a execução de atividades ou mesmo interromper a execução de um *workflow* por completo é muitas vezes necessário. Se o SGWfC permite aos cientistas analisar os resultados parciais de um experimento, eles podem concluir que precisam mudar parâmetros para aumentar o refinamento e melhorar os resultados. Ou ainda, eles podem concluir que os resultados atuais já são satisfatórios, de forma que a execução possa ser concluída antecipadamente. Se os SGWfC permitem que o usuário interfira na execução, as mudanças mencionadas podem ser feitas em tempo de execução. Como resultado, o SGWfC pode tirar vantagem do estado atual da execução, poupando tempo e economizando re-execuções desnecessárias.

Na nossa recente experiência dando apoio a cientistas para executar seus *workflows* em paralelo, identificamos cenários importantes que podem fazer bom uso de interações dinâmicas, tais como: (i) abordagens não determinísticas de otimização (Lima *et al.* 2005), pro exemplo algoritmos genéticos que precisam refinar parâmetros como taxas de cruzamento e mutação para obter resultados melhores; (ii) *workflows* de

genômica comparativa na bioinformática (Ocaña *et al.* 2013) que precisam refinar o valor do grau de similaridade (por ex. *e-value*) de um conjunto de dados de entrada (sequências), executando o mesmo *workflow* diversas vezes ou trocar uma atividade do *workflow* dinamicamente (Ocaña *et al.* 2011a; Santos *et al.* 2013); e (iii) *workflows* de quantificação de incertezas (Guerra *et al.* 2012), que precisam ajustar o nível de interpolação em função da confiabilidade desejada. Todos estes exemplos requerem análises de dados e ajustes para refinar as configurações iniciais do experimento. A cada modificação realizada no *workflow*, o cientista quer que o motor de execução se adeque ao novo cenário e produza novos resultados para ele voltar a analisar, comparando inclusive com os resultados anteriores. Esse processo pode se repetir dezenas de vezes até que o experimento como um todo convirja. Ou seja, o processo no qual o cientista interage com o *workflow* de forma dinâmica é intrinsicamente iterativo.

Iteração é um conceito básico na Ciência da computação (Strachey 1967) e está presente na maioria das linguagens de programação. Por ser intuitivo, foi extensamente utilizado no desenvolvimento de algoritmos e modelos computacionais. Cientistas normalmente programam seus métodos iterativos diretamente em programas ou *scripts*. Entretanto, conforme os cientistas refinam seu modelo numérico, experimentam algoritmos diferentes ou exploram diferentes fatias do espaço de parâmetros para encontrar resultados melhores, o *workflow* científico precisa mudar ao longo do seu ciclo de vida. Esta natureza exploratória dos experimentos científicos computacionais em larga-escala compelem os *workflows* a serem iterativos, embora eles sejam muito dinâmicos para serem programados em uma linguagem estática.

Acreditamos que utilizar uma abordagem iterativa com dinamismo é um passo importante na direção de *workflows* dinâmicos uma vez que a i*n*teração entre o usuário e o *workflow* ao longo do experimento é iterativa. Porém, o apoio a iterações em *workflows* científicos (Laszewski *et al.* 2007; Wozniak *et al.* 2012) é baseado em estruturas de controle que estipulam quando e como um conjunto de atividades deve executar. Estas estruturas oferecem, em geral, até três tipos estáticos de iteração (Elmroth *et al.* 2010): ciclos com contagem e sem dependência, ciclos com contagem e com dependência e ciclos condicionais. Nesta tese, caracterizamos um quarto tipo – ciclos dinâmicos – como uma extensão dos ciclos condicionais mas com apoio a interações por meio da condução dinâmica de *workflows*. Os ciclos dinâmicos possuem uma especificação iterativa e um modelo de execução dinâmico.

O apoio a ciclos dinâmicos em *workflows* ajudará na composição de experimentos que inerentemente precisem de adaptação e ajustes finos ao longo do seu ciclo de vida. Da forma como é feito atualmente – (i) programar a iteração com ferramentas de gerenciamento de grandes volumes de dados (Srirama *et al.* 2012), (ii) repetir a execução do *workflow* manualmente ou (iii) executar o experimento em etapas – é muito trabalhoso e propenso a erros. Sem o acesso aos dados de proveniência de forma consistente com as iterações, os cientistas também podem perder o rastro do que já foi explorado e de como o *workflow* evoluiu. Para melhorar esse processo experimental iterativo, o cientista deve ser capaz de analisar resultados parciais durante a execução para poder intervir dinamicamente nos próximos passos do *workflow*. Ao submeter consultas de proveniência em tempo real, o cientista pode fazer um análise online detalhada da evolução do *workflow*. Portanto, existem grandes vantagens se todo o processo iterativo é modelado na forma de um *workflow* dinâmico. Embora alguns SGWfC apoiem tipos clássicos de iteração (Ekanayake *et al.* 2010; Wozniak *et al.* 2012), nenhum deles dá apoio a ciclos dinâmicos. Abordagens iterativas atuais não conseguem adaptar a execução de acordo com mudanças feitas no *workflow*. Eles tipicamente possuem uma especificação iterativa que é submetida para execução seguindo a definição de um fluxo de dados estático. Mesmo que o plano de execução seja dinâmico (por ex. mudar de execução síncrona para pipeline durante a execução), os atuais SGWfC não permitem a mudança dos dados de controle ou da configuração do *workflow*. Os cientistas precisam esperar o final da execução para analisar resultados e dados de proveniência para então decidir se é necessário ressubmeter o *workflow* com uma nova configuração.

A abordagem dinâmica facilita a gerência do experimento e poupa recursos, o que pode ser muito mais efetivo que uma otimização centrada apenas na alocação e utilização dos recursos computacionais. Se o ambiente de execução for uma nuvem computacional (Buyya *et al.* 2011), a diminuição de custos computacionais é ainda mais relevante. Neste caso, a utilização dos recursos é tarifada sob demanda. Quanto maior o número de recursos e o tempo de utilização destes recursos, maior será o valor cobrado do cientista pelo provedor do serviço de computação em nuvem. Se a abordagem utiliza os recursos computacionais de forma mais eficiente, o experimento sai mais barato. O mesmo se aplica aos *clusters*. Mesmo não sendo tarifados sob demanda, os *clusters* são máquinas caras e com alto consumo energético. Utilizá-los de maneira mais eficiente é

uma necessidade da maioria dos centros provedores de computação paralela. A utilização mais eficiente dos recursos corrobora com os esforços de uma computação mais sustentável, ou computação verde (Pedram 2009).

O uso mais eficiente de recursos é um dos objetivos de outros processos como a otimização de consultas em bancos de dados (Özsu e Valduriez 2011) e a otimização da execução de *workflows* científicos (Ogasawara 2011). Nestes dois casos, a máquina de otimização analisa um plano algébrico da execução e busca otimizá-lo realizando transformações algébricas. O objetivo desta otimização é reduzir o tempo de resposta ou o tempo de execução da consulta ou *workflow*, respectivamente. Fixado o recurso computacional, quanto menor o tempo de resposta, mais eficiente é a consulta otimizada. Analogamente, quanto menor o tempo de execução, mais eficiente é o *workflow* otimizado. Este tipo de otimização está associada à estrutura da consulta ou do *workflow* modelado. Entretanto, em um experimento científico, além da especificação do *workflow*, existem outras variáveis do domínio do problema que podem ser exploradas para melhorar a eficiência da execução. Margens de erro, critérios de parada, critérios para filtragem de dados, ajustes em fatores de um processo iterativo e escolhas de programas/algoritmos mais eficientes, são todos parâmetros do domínio do problema que podem ser otimizados para melhorar a eficiência da execução do experimento.

Otimizar variáveis do domínio do problema depende da ação do cientista, uma vez que é ele o especialista. O cientista deve analisar resultados anteriores ou parciais e então efetuar modificações no experimento, conduzindo e controlando a execução do *workflow*. Ele pode, inclusive, pré-programar algumas ações no *workflow* que disparem mudanças em função de algum resultado ou comportamento dos dados. Ou então pode conduzir o experimento, ou parte dele, em tempo real, fazendo modificações à medida que analisa os dados de proveniência. Para que este tipo de otimização possa ser realizado pelo cientista, o comportamento da máquina de execução do *workflow* precisa ser dinâmico.

A abordagem de gerência de execução dinâmica do workflow pode reduzir os custos da execução ao evitar o desperdício de recursos computacionais e de energia. Além disso, traz melhorias no tempo da execução, obtendo os resultados finais do experimento mais rapidamente e com maior qualidade, uma vez que o cientista pode

conduzir o experimento e otimizar variáveis do domínio do problema. Desta forma, a questão de pesquisa a ser investigada nesta tese é:

"*No contexto de experimentos científicos computacionais em larga escala, é possível dar apoio a execuções interativas de workflows científicos por meio de iterações e um modelo de execução dinâmico considerando diferentes tipos de interação entre o usuário e o workflow, garantindo a consistência da execução e dos dados de proveniência, permitindo consultas aos resultados parciais e à proveniência e melhorando a eficiência do experimento como um todo?*"

1.2 Abordagem Interativa para a Execução de Experimentos Científicos Computacionais

A hipótese geral desta tese é que o apoio a interações em *workflows* traz melhorias na obtenção de resultados e no tempo de execução de experimentos científicos computacionais que precisam, inerentemente, ser adaptados e sofrer ajustes finos ao longo do seu ciclo de vida. A possibilidade de interagir ao longo da execução evita que os cientistas interrompam e reiniciem o *workflow* inúmeras vezes. Com isso, reduz-se o tempo para executar o experimento como um todo. A possibilidade de analisar e correlacionar resultados de diferentes explorações pode agilizar a análise global dos resultados. Para formular esta hipótese, utilizamos como base resultados preliminares de experimentos que mostram o potencial das interações na melhoria do tempo de execução do experimento e o potencial de consultas de proveniência para otimizar a especificação do *workflow* (Dias *et al.* 2011, 2013; Santos *et al.* 2013).

Para avaliar a hipótese desta tese, nós utilizamos uma abordagem algébrica centrada em dados para dar apoio a iterações em *workflows* dinâmicos. Ciclos dinâmicos precisam de uma abordagem centrada em dados pois esta é capaz de responder a eventos estimulados por operações com dados tais como inserções e atualizações de dados em tempo de execução. Portanto, nós criamos estruturas implícitas para ciclos em fluxos de dados sem definir estruturas de controle estáticas. Nós utilizamos a abordagem algébrica para *workflows* científicos centrados em dados (Ogasawara *et al.* 2011) como o nosso modelo de especificação. Em particular, uma álgebra de *workflows* é um bom fundamento para otimização e execução paralela. Ademais, nosso modelo de execução dinâmico é fortemente acoplado a um modelo de proveniência relacional, facilitando consultas à base de proveniência em tempo real

(Costa *et al.* 2013). Este recurso mostrou-se essencial para apoiar a tomada decisão durante a execução de *workflows* (Guerra *et al.* 2012; Ocaña *et al.* 2013; Oliveira *et al.* 2013). Em Guerra *et al.* (2012), um *workflow* de quantificação de incertezas é aperfeiçoado utilizando consultas de proveniência em tempo real, mesmo sem o apoio de ciclos dinâmicos. Ao permitir análises de resultados parciais, cientistas puderam monitorar diversos parâmetros de execução como erro convergido, média e a variância da energia cinética. Ao consultar essas informações, eles podem concluir se a simulação convergiu ou foi interrompida por outro motivo.

Esta tese está inserida no contexto de "desenvolvimento de Ciência apoiada pela computação" (do inglês *e-science* ou *cyberinfrastructure*). As principais contribuições desta tese são:

i. Construtos iterativos na abordagem algébrica para *workflows* científicos centrados em dados com uma operação algébrica para iterações
ii. O conceito de linhagem de dados que possibilita o rastreamento, consulta e correlação dos dados de proveniência ao longo das iterações do *workflow*.
iii. Um modelo de execução dinâmico que apoia os ciclos dinâmicos através de ativações de avaliação e um modelo em cubo para a linhagem de dados
iv. O algoritmo Interação-Alfa que permite que os dados de entrada do *workflow* sofram alterações em tempo de execução.
v. O algoritmo Interação-Ômega que permite que a configuração do *workflow* seja alterada em tempo de execução
vi. A implementação do módulo iterativo e dos algoritmos de interação na máquina paralela de execução de *workflows* Chiron e a implementação de uma API denominada Achilles para inserir eventos de interação na proveniência do *workflow* sem gerar inconsistências nos resultados
vii. Uma avaliação experimental extensa utilizando um *workflow* dinâmico para a análise de quantificação de incertezas. Executamos o *workflow* em um cluster de 640-cores, mostrando os benefícios da nossa abordagem do ponto de vista do usuário.

Esta tese também envolve um conjunto de trabalhos publicados associados ao dinamismo na execução de *workflows*, à representação algébrica de *workflows* científicos e a execução de experimentos com apoio da proveniência de dados. Dentre estas publicações, podemos destacar:

i. DIAS, J., Ogasawara, E., Oliveira, D., Porto, F., Coutinho, A. and Mattoso, M. (2011). Supporting Dynamic Parameter Sweep in Adaptive and User-Steered Workflow. In *6th Workshop on Workflows in Support of Large-Scale Science*. , WORKS '11. ACM.

ii. Ogasawara, E., DIAS, J., Oliveira, D., Porto, F., Valduriez, P. and Mattoso, M. (2011). An Algebraic Approach for Data-Centric Scientific Workflows. *Proc. of VLDB Endowment*, v. 4, n. 12, p. 1328–1339.

iii. Ocaña, K. A. C. S., Oliveira, D., DIAS, J., Ogasawara, E. and Mattoso, M. (7 dec 2011). Optimizing Phylogenetic Analysis Using SciHmm Cloud-based Scientific Workflow. In *2011 IEEE Seventh International Conference on e-Science (e-Science)*. IEEE.

iv. Guerra, G., Rochinha, F., Elias, R., Oliveira, D., Ogasawara, E., DIAS, J., Mattoso, M. and Coutinho, A. L. G. A. (2012). Uncertainty Quantification in Computational Predictive Models for Fluid Dynamics Using Workflow Management Engine. *International Journal for Uncertainty Quantification*, v. 2, n. 1, p. 53–71.

v. Horta, F., DIAS, J., Ocaña, K. A. C. S., De Oliveira, D., Ogasawara, E. and Mattoso, M. (2012). Using Provenance to Visualize Data from Large-Scale Experiments (Abstract). In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*.

vi. Ogasawara, E., DIAS, J., Silva, V., Chirigati, F., Oliveira, D., Porto, F., Valduriez, P. and Mattoso, M. (2013). Chiron: A Parallel Engine for Algebraic Scientific Workflows. *Concurrency and Computation*, v. 25, n. 16, p. 2327–2341.

vii. Ocaña, K. A. C. S., Oliveira, F., DIAS, J., Ogasawara, E. and Mattoso, M. (2013). Designing a parallel cloud based comparative genomics workflow to improve phylogenetic analyses. *Future Generation Computer Systems*, v. 29, n. 8, p. 2205–2219.

viii. Santos, I., DIAS, J., Oliveira, D., Ogasawara, E., Ocaña, K. and Mattoso, M. (2013). Runtime Dynamic Structural Changes of Scientific Workflows in Clouds. In *Proceedings of the International Workshop on Clouds and (eScience) Applications Management - CloudAM*.

ix. DIAS, J., Ogasawara, E., Oliveira, D., Porto, F., Valduriez, P. and Mattoso, M. (2013). Algebraic Dataflows for Big Data Analysis. In *Proceedings of the IEEE International Conference on Big Data*.

x. Mattoso, M., DIAS, J., Oliveira, D., Ocaña, K., Ogasawara, E., Costa, F., Horta, F., Silva, V. and Araújo, I. (2013). User-Steering of HPC Workflows: State of the Art and Future Directions. In *Proceeding of the 2nd International Workshop on Scalable Workflow Enactment Engines and Technologies (SWEET'13)*.

1.3 Organização da tese

Esta tese está organizada em outros 8 capítulos. O Capítulo 2 apresenta os trabalhos relacionados a esta tese. O Capítulo 3 apresenta a fundamentação teórica necessária para melhor entendimento do restante da tese. O Capítulo 4 apresenta a abordagem algébrica para iterações com construtos iterativos, um novo operador algébrico denominado *Evaluate* e o conceito de linhagens para iterações. O Capítulo 5 discute os ciclos dinâmicos descrevendo recursos adicionais do modelo de execução e os algoritmos para interação. O Capítulo 6 apresenta os detalhes da implementação dos ciclos dinâmicos no Chiron e da plataforma Proteus. O Capítulo 7 apesenta os *workflow*s utilizados como estudos de caso e o Capítulo 8 apresenta a nossa avaliação experimental. O Capítulo 9 conclui a tese, discute as contribuições alcançadas e possíveis trabalhos futuros.

Capítulo 2 Trabalhos Relacionados

No contexto de experimentos científicos computacionais em larga escala, não existe hoje uma solução de apoio à execução interativa de um workflows científico por meio de iterações com um modelo de execução dinâmico considerando diferentes tipos de interação entre o usuário e o workflow. Sendo assim, os trabalhos encontrados na literatura estão relacionados com a base das soluções desenvolvidas não sendo possível analisar abordagens relacionadas à execução dinâmica interativa de workflows científicos. Este capítulo discute um conjunto de trabalhos que se relacionam com o tema desta tese em função de alguma característica em comum. Dentre estas características, existem dois tópicos principais que são discutidos em outros trabalhos e são relevantes para esta tese: (1) abordagens para execução iterativa, discutidas na seção 2.1 e (2) abordagens interativas para *workflows*, discutidas na seção 2.2.

2.1 Abordagens Para Execução Iterativa

A execução iterativa em algoritmos computacionais existe em diversas linguagens de programação. Também podemos destacar as metodologias recursivas para iterações (Bancilhon e Ramakrishnan 1986) e iterações em consultas de banco de dados através da recursões em *datalog* (Ceri *et al.* 1989) ou outras linguagens de banco de dados (Valduriez e Danforth 1992; Graefe 1993; Oliveira *et al.* 2010c). Nesta tese, contudo, iremos destacar as metodologias iterativas para: (1) *workflow*s científicos e (2) *dataflows* utilizando MapReduce, que possuem um propósito similar.

Diversos SGWfC oferecem recursos para execução de experimentos científicos computacionais em ambientes distribuídos. Sistemas como Pegasus (Deelman *et al.* 2007), o Nimrod/K (Abramson *et al.* 2008), o Triana (Taylor *et al.* 2007) e o Turbine (Wozniak *et al.* 2012), dentre outros, distribuem a execução de tarefas de *workflows* em grades, redes P2P e clusters de computadores utilizando algoritmos eficientes e otimizados de escalonamento. Entretanto, poucos SGWfC oferecem apoio completo a *workflows* iterativos, pois suas máquinas de execução interpretam seus *workflows* na forma de um grafo direcionado acíclico (DAG, do inglês *directed acyclic graph*) (Bondy e Murty 2010). Essa limitação obriga que os *workflows* sejam especificados ou sem iterações ou com iterações passíveis de linearização do ciclo (Fahringer *et al.* 2005). Ou seja, se o cientista projeta um *workflow* com um ciclo, quando ele submete o

workflow para executar, a máquina de execução do *workflow* gera um plano de execução na forma de um DAG. As atividades dentro do ciclo são executadas repetidas vezes em sequência, se houver dependência entre as iterações, ou são executadas em paralelo se não houver dependências. Essa abordagem funciona muito bem para ciclos cujo tamanho é conhecido *a priori* (ciclos com contagem). É conhecido que ciclos com contagem e sem dependências podem ser representados como múltiplos DAG e ciclos com contagem e dependências podem ser linearizados como um DAG (Laszewski *et al.* 2007). Entretanto, ciclos que são executados até que uma dada condição dinâmica seja satisfeita (ciclos condicionais) não podem ser desenrolados na forma de um DAG (Elmroth *et al.* 2010).

Consideramos como abordagem iterativa as ferramentas que oferecem apoio aos diferentes tipos de iteração, sejam elas ciclos com contagem ou ciclos condicionais. Nas seções 2.1.1 e 2.1.2 iremos destacar as abordagens relevantes tanto no domínio de *workflows* científicos quanto ferramentas iterativas dentro do paradigma *MapReduce*.

2.1.1 Abordagens de *Workflows* Científicos

Existem SGWfC que realizam a execução local do *workflow*, *i.e.* sem fazer a paralelização da execução em um ambiente remoto, e permitem diversos tipos de iteração incluindo ciclos condicionais (Altintas *et al.* 2004; Hull *et al.* 2006; Taylor *et al.* 2007; Pradal *et al.* 2008). Embora algum deles possam instanciar serviços remotos, o controle sobre o ciclo condicional permanece na máquina de execução local. Na extensão do nosso conhecimento, todas as implementações disponíveis para ciclos em *workflows* científicos são baseadas em um fluxo de controle. Isto significa que há um elemento no *workflow*, que não é uma atividade para processar dados, que decide quando o ciclo deve começar e quando o mesmo deve terminar. Não há nada de errado com uma abordagem baseada em um fluxo de controle, porém ela dificulta o paralelismo dinâmico pois requer sincronismo a cada iteração. Entretanto, em uma iteração com dinamismo, podem haver fluxos de dados em estados diferentes da execução, impossibilitando o sincronismo.

A execução paralela de ciclos condicionais em *workflows* científicos pode ser feita através do SGWfC Turbine (Wozniak *et al.* 2012). O Turbine é um dos SGWfC mais escaláveis entre seus pares, tendo sido reportados estudos de desempenho com até 30K núcleos de computação paralela. O Turbine interpreta um *workflow* descrito em

uma linguagem *script* de programação paralela e distribuída chamada Swift (Wilde *et al.* 2011). A Swift oferece uma diretiva *for each* para ciclos com contagem e uma diretiva *iterate* para ciclos condicionais. O critério de parada pode ser um parâmetro de controle ou um contador. A definição do *workflow* no Turbine, usando a Swift, é fortemente acoplada ao plano de execução do *workflow*, pois é o script que dita como o *workflow* é executado. Portanto, qualquer mecanismo para adaptar o *workflow* precisa estar programado no script e não pode ser ajustado dinamicamente durante a execução do *workflow*. Esta característica limita a possibilidade de interagir com o *workflow* em tempo de execução, pois depois que o Turbine interpreta o script e gera o plano de execução para o *workflow*, não há nada que o cientista possa fazer para alterar a especificação *workflow* sem interromper a execução. Além disso, a proveniência no Turbine é extraída de arquivos de *log* após a execução do *workflow* de forma que o cientista não pode interagir com a proveniência e com os resultados parciais para tomar decisões e modificar o *workflow*. Esta característica torna o apoio à análises de proveniência em tempo de real impraticáveis. Consequentemente, o cientista não pode interagir com o *workflow* e/ou melhorar o experimento dinamicamente.

2.1.2 Abordagens utilizando MapReduce

Muitos usuários utilizam o paradigma de programação funcional *MapReduce* (Dean e Ghemawat 2008) para realizar análises em grandes volumes de dados (análises de *big data* (Bertino *et al.* 2011)) em ambientes de processamento de alto desempenho (PAD). Eles precisam programar as funções *Map* e *Reduce* e o arcabouço oferece o paralelismo de dados nativamente. O Hadoop é a implementação de código aberto mais popular do arcabouço MapReduce. Entretanto, o Hadoop não oferece recursos para modelar fluxos de dados complexos (Olston *et al.* 2008b) como os *workflows* científicos. Uma forma de programar fluxos de dados em análises *big data* é usando da linguagem Pig Latin (Olston *et al.* 2008b), que combina os estilos de programação MapReduce e SQL provendo operações nativas para carregar, processar, filtrar, agrupar e armazenar dados no Hadoop. Pig, a máquina que implementa e interpreta a linguagem Pig Latin, considera otimizações da lógica do fluxo de dados (Olston *et al.* 2008a) incluindo o posicionamento otimizado de operações de filtro e a união de dois filtros em uma única função Map. Essas otimizações favorecem operações nativas no Pig Latin (como filtros e agrupamentos). Entretanto, quando o usuário precisa utilizar código customizado, o Pig Latin requer o uso de funções definidas pelo usuário (UDF, *user defined functions*

em inglês), que podem encobrir o comportamento do programa utilizado (Kwon *et al.* 2012).

No processo de distribuição de tarefas e alocação de recursos, o Pig confia a execução ao Hadoop para gerenciar as tarefas MapReduce. Infelizmente, o plano de execução do Hadoop não obtém vantagem das características declarativas do Pig Latin, oferecendo poucas oportunidades de otimização do fluxo de dados em tempo de execução. Embora existam diversos esforços focados em melhorar o desempenho do Hadoop (Floratou *et al.* 2011; Herodotou *et al.* 2011; Dittrich *et al.* 2012), eles focam apenas em melhorar o modo operacional do Hadoop, modificando a forma na qual os dados são armazenados e acessados além de ajustar parâmetros internos do Hadoop. Como o Hadoop não tem consciência do fluxo de dados e do histórico das execuções (por não ter dados de proveniência), ele não pode otimizar a execução obtendo vantagens na movimentação de dados e de acordo com o comportamento da execução das tarefas.

Ainda assim, existem abordagens dentro do paradigma MapReduce que oferecem apoio a iterações. O Twister (Ekanayake *et al.* 2010) e o HaLoop (Bu *et al.* 2010) são abordagens iterativas que estendem o Hadoop para permitir que os usuários explorem repetidamente suas funções Map e Reduce iterativamente até que uma dada condição é alcançada. Já o Stratosphere (Ewen *et al.* 2012) possui recursos nativos mais sofisticados para apoiar iterações em fluxo de dados. Ele explora o conteúdo das funções Map e Reduce programadas pelo usuário utilizando o conceito de micro-passos e otimiza a execução das tarefas. Com o apoio das funções de análise estática de código, nativas do Stratosphere, o modelo iterativo quebra as funções Map e Reduce em um fluxo de dados de tarefas menores e otimiza a execução aplicando técnicas de otimização que evitam que determinadas tarefas (trechos do algoritmo) sejam executadas de maneira desnecessária. Esta técnica mostrou-se muito eficiente em ciclos condicionais com uma caraterística incremental.

O Twister, o HaLoop e a abordagem iterativa do Stratosphere são complementares ao nosso trabalho. Eles focam na execução do processo iterativo como parte de um algoritmo implementado no fluxo de dados MapReduce. Diferentemente, nossa abordagem é projetada para *workflow*s de mais alto nível, onde os algoritmos e outras aplicações científicas são tratadas como caixas pretas. Estas aplicações podem ser código legado/complexo ou solucionadores de métodos numéricos muito

otimizados, todos difíceis de recodificar ou decompor. Algumas classes de algoritmos são difíceis de programar dentro do paradigma MapReduce (Srirama *et al.* 2012), portanto pode ser uma boa alternativa mantê-los como caixas pretas e otimizar o processo experimental como um todo ao invés de otimizar a execução paralela do algoritmo internamente. Entretanto, também podemos explorar o Twister, o HaLoop ou o Stratosphere dentro do nosso *workflow* como uma atividade do experimento. Desta forma, poderíamos enriquecer a execução global do experimento acrescentando proveniência em tempo real e o apoio a interações entre o usuário e o *workflow*.

As abordagens *MapReduce* iterativas existentes não são interessantes para o desenvolvimento de *workflows* dinâmicos por que elas estabelecem um plano de execução fixo baseado na especificação das atividades programadas nas funções Map e Reduce. Mesmo que algumas abordagens realizem otimizações no escalonamento de tarefas e no acesso aos dados em tempo de execução, se o cientista desejar realizar mudanças em parâmetros ou na estrutura do *workflow*, ele precisa recompilar as suas funções e disparar a execução novamente.

2.2 Abordagens Interativas para *Workflows*

Como já foi discutido anteriormente, existem muitos SGWfC que apoiam a execução em larga escala de *workflows* científicos computacionais com planos de execução bem controlados (Deelman *et al.* 2007) e recursos de interoperabilidade (Alqaoud *et al.* 2010). Entretanto, muitas questões referentes a *workflows* dinâmicos e a interatividade com cientistas ainda estão em aberto. Na extensão do nosso conhecimento, não há SGWfC que apoie *workflows* dinâmicos em larga-escala. Fora do escopo de SGWfC, pode-se estabelecer um relacionamento entre o trabalho proposto nesta tese e a otimização dinâmica de consultas (Jarke e Koch 1984), que traçam o melhor plano de execução da consulta na medida em que as operações algébricas sobre os dados produzem resultados. Porém, depois que a consulta é submetida para execução, uma mudança nos dados da consulta, por exemplo em um valor de um predicado de seleção, não é permitido. Algumas abordagens tratam a dinamicidade em *workflows* de negócio utilizando tratamento de exceções (Kammer *et al.* 2000), redes de Petri (Ellis *et al.* 1995) ou sistemas de bancos de dados ativos (Dąbrowski *et al.* 2010). Tais abordagens foram projetadas para atividades relacionadas a processos de negócio, não para programas científicos, os quais são tipicamente centrados em dados.

Alguns sistemas de *workflow*s científicos tratam adaptações em *workflow*s utilizando agentes autonômicos (Lee *et al.* 2007; Paton *et al.* 2009). Porém, eles realizam uma execução local, i.e. na máquina do cientista, que mapeia tarefas para execução remota utilizando serviços disponíveis em grades ou na nuvem. O *workflow* passa por uma fase de compilação pois há uma separação entre a fase de especificação na máquina local e a fase de execução em um ambiente remoto. Durante a execução do *workflow*, componentes autonômicos adaptam a execução do *workflow* conforme o cientista realiza modificações na especificação do *workflow*. O adaptação autonômica pausa e dispara serviços novamente em função das modificações considerando o último estado do *workflow* e sobrescrevendo, consequentemente, o que foi executado anteriormente. Embora essa abordagem permita alterações na especificação do *workflow* em tempo de execução, apenas os resultados do último estado do *workflow* são armazenados. Logo, o cientista pode ter dificuldade para avaliar como o experimento evoluiu e comparar resultados de explorações diferentes. Ele pode realizar uma modificação e depois observar que não foi bem sucedido e querer voltar ao estado anterior. Nesta situação ele precisa mudar o *workflow* novamente e os agentes autonômicos irão reexecutar os serviços necessários desperdiçado o que já havia sido processado.

Existem outros trabalhos que propõem recursos específicos envolvendo a dinamicidade em *workflows*. Missier *et al.* (2010) apoia a definição de gatilhos que tratem mudanças dinâmicas em *workflow* do SGWfC Taverna. Bowers *et al.* (2006) e Samak *et al.* (2011) apresentam recursos dinâmicos pata tratar falhas em Kepler e Pegasus, respectivamente em um processo similar ao tratamento de exceções em linguagens de programação. As abordagens mencionadas oferecem recursos importantes para *workflows* dinâmicos. Porém, todas as abordagens se baseiam em regras pré-programadas no *workflow*. Elas não apoiam mudanças realizadas no *workflow* quando ele já está em execução no ambiente remoto.

Recentemente, outros trabalhos mais direcionado a ambientes em larga escala propuseram soluções voltadas às questões envolvendo *workflows* dinâmicos. O portal de Ciência WorkWays (Nguyen e Abramson 2012) é um portal Web interativo que permite ao usuário inserir dados e exportar dados de um *workflow* que é mantido sob execução contínua. O WorkWays utiliza o conceito de *workflows* como serviços, logo, especificações estáticas de *workflows* são mantidas e executadas quando o cientista

interage com o portal e envia um novo dado para ser processado. Essa abordagem restringe a elaboração de *workflows* dinâmicos pois obriga um *workflow* a ser implementado como um serviço. Muitos *workflows* são custosos para serem mantidos como um serviço, além de possuir uma configuração volátil onde diversos parâmetros precisam ser ajustados dependendo do dado de entrada. Nesse caso, é importante que o cientista possa interagir com a especificação do *workflow* e não somente com os dados de entrada. Além disso, é importante que o cientista possa determinar o ambiente de execução do *workflow*, uma vez que ele pode demandar PAD.

O OpenMole (Reuillon *et al.* 2013) é um outro sistema de *workflows* projetado para modelos de simulação que precisem de contínua adaptação e melhoria. Ele permite que os cientistas troquem o *software* de uma dada atividade do *workflow* de forma transparente. Entretanto, a configuração do *workflow* permanece estática em uma linguagem específica de domínio (DSL, em inglês, *Domain Specific Language*), que é uma extensão da linguagem de programação Scala[1]. Neste sentido, a abordagem do OpenMole torna-se similar ao Turbine, porém com a possibilidade de trocar um programa por outro durante o experimento. O cientista não pode interagir com a configuração do *workflow* alterando parâmetros da execução. Se ele desejar fazer este tipo de interação, terá que embutir os parâmetros nos seus programas e fazer a troca da atividade.

As abordagens WorkWays e OpenMole são complementares ao trabalho proposto nesta tese. Entretanto, nossa abordagem tira vantagem dos benefícios proporcionados pela álgebra de *workflows*, como as possibilidades de otimização, e pela proveniência de dados integrada ao modelo de execução, que permite consultas aos resultados do experimento de forma estruturada. Cientistas podem se beneficiar de consultas complexas aos resultados ao longo das iterações e comparar resultados obtidos após diferentes interações com o *workflow.* O conceito de linhagem que propomos mantém os dados de proveniência consistentes com o processo iterativo e com as interações feitas durante a execução do *workflow*.

---

[1] http://www.scala-lang.org/

Capítulo 3 Fundamentação Teórica

Neste capítulo discutimos um conjunto de fundamentos necessários para o melhor entendimento desta tese. Na seção 3.1 discutimos os experimentos científicos baseados em simulação e o seu ciclo de vida. Na seção 3.2 discutimos os *workflows* científicos e na seção 3.3 apresentamos a representação que adotamos nesta tese através da álgebra de *workflows* científicos. Na seção 3.4 falamos de proveniência de dados e do modelo que adotamos nesta tese. Na seção 3.5 discutimos os *workflows* dinâmicos apresentando o subgrupo de recursos que estamos abordando. Na seção 3.6 discutimos o processo de interação entre o usuário e o *workflow*. Na seção 3.7 discutimos o modelo de regras ativas para banco de dados, que servem de inspiração para a implementação da abordagem dinâmica no Chiron. Finalmente, na seção 3.8 discutimos os diferentes tipos de iteração.

3.1 Experimentos Científicos Baseados em Simulação

Existem diferentes tipos de experimentos científicos. Os mais tradicionais são os *in vivo*, e os *in vitro*. No primeiro, os estudos são conduzidos com os objetos de estudo em seu próprio ambiente natural. Já no segundo, os objetos de estudo estão em um ambiente controlado. Os avanços nos sistemas computacionais vêm apoiando novas formas de experimentos baseados em computadores, chamados de *in virtuo* e *in silico* (Travassos e Barros 2003). Experimentos *in virtuo* são caracterizados pelos objetos de estudo reais participando interativamente do experimento em um ambiente simulado pela infraestrutura computacional, enquanto nos experimentos *in silico*, tanto os objetos de estudo quanto o ambiente são simulados por computadores. Nesta tese, propomos uma solução interativa para experimentos computacionais *in silico*. Computacionalmente custosos, estes experimentos envolvem o processamento massivo de dados, sendo oportuna a paralelização da execução do experimento.

Experimentos *in silico* são comuns em diversas áreas de pesquisa, tais como a meteorologia (Gannon *et al.* 2007), bioinformática (Dávila *et al.* 2008; Ocaña *et al.* 2011b, 2011a, 2012) e exploração de petróleo (Carvalho 2009; Oliveira *et al.* 2009). Tais experimentos são considerados de larga escala pelo aspecto exploratório de diversos fatores e conjunto de dados. Normalmente, tais experimentos consomem muitos recursos computacionais e podem demorar semanas, ou mesmo meses para

executar. A complexidade destes experimentos torna difícil o controle manual sobre eles. Os princípios de um experimento científico exigem que estes sejam sistemáticos e bem documentados para garantir sua reprodutibilidade. Portanto, o controle e a visão evolutiva que o cientista precisa ter do seu experimento é muito importante para o sucesso deste. Mattoso *et. al.* (2010b) propõe um ciclo de vida do experimento em larga escala composto por três fases como mostra a Figura 1.

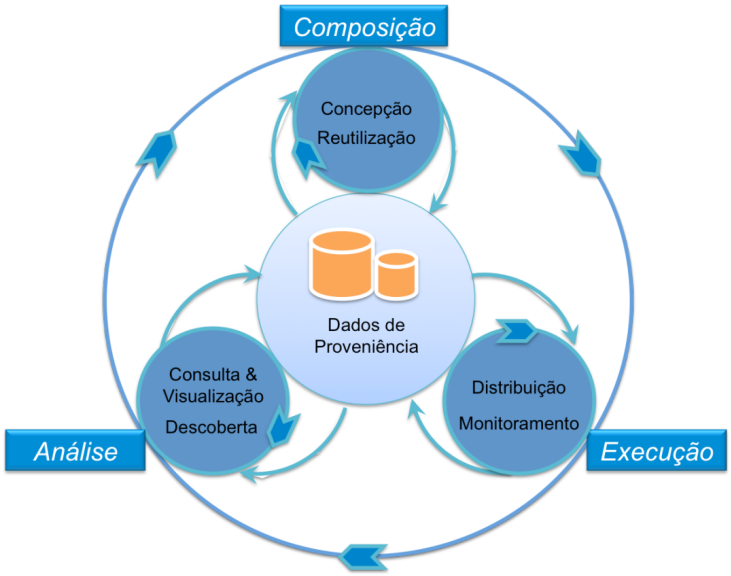

Figura 1: Ciclo de vida do experimento científico segundo Mattoso et.al. (2010b)

Na fase de composição, o cientista modela o experimento, definindo o encadeamento de atividades, os parâmetros utilizados, os dados de entrada e os resultados esperados. É na fase de composição que o cientista irá conceber o *workflow* científico. Ele também poderá reutilizar componentes ou *workflows* desenvolvidos anteriormente. Na fase de execução, o *workflow* é instanciado no ambiente de PAD através de uma máquina de execução de *workflows*. Ao executar o *workflow* em um ambiente de PAD, as diversas instâncias das atividades do *workflow* serão distribuídas para serem executadas em paralelo nos recursos disponíveis. A máquina de execução do *workflow* irá também monitorar o processo para detectar problemas e verificar o término das atividades, dando continuidade à execução do *workflow*. Na fase de análise, o cientista irá consultar e visualizar os resultados obtidos para verificar a sua hipótese. Ele também pode explorar o dados obtidos e os dados de proveniência com o objetivo de descobrir alguma correlação, comportamento ou conteúdo relevante. Baseado em suas conclusões, o experimento pode chegar ao fim ou o cientista pode decidir dar

continuidade ao ciclo pois o experimento ainda não convergiu para o resultado esperado.

A proveniência permeia todas as etapas do ciclo de vida de um experimento científico pois ela guarda o registro completo de tudo o que acontece no experimento Na fase da composição, registra-se a proveniência prospectiva, enquanto na fase de execução registra-se a proveniência retrospectiva. Na fase de análise, a proveniência é consultada, porém pode-se também registrar os resultados da análise na base de proveniência para futuras consultas e comparações com novos resultados.

3.2 *Workflows* Científicos

O termo *workflow* é originalmente relacionado à processos de automação de escritórios (Aalst e Hee 2002). Na seção 2.2 nos referimos à este tipo de *workflow* como *workflows* de negócio, para distingui-los dos *workflows* científicos. Em um *workflow* de negócio, o principal objetivo é sistematizar um processo de criação, armazenamento, compartilhamento e revisão de documentos e informações em uma empresa. Desta forma, evita-se o intercâmbio de documentos em papel, reduzindo gastos com impressão e otimizando o tempo de troca de informação entre os envolvidos no processo (Aalst e Hee 2002; Aalst *et al.* 2003).

Os *workflows* científicos estendem esse conceito para ambientes científicos (Deelman *et al.* 2009) como forma de abstrair e possivelmente sistematizar o processo experimental, ou ao menos parte dele. Alguns autores preferem chamar os *workflows* científicos de pipelines científicos ou simplesmente *dataflows* (Wozniak *et al.* 2012). O termo *dataflow* é adequado aos *workflows* científicos por que a natureza deles é centrada em dados, em função do grande volume de dados que flui entre uma atividade do *workflow* e outra. Em contrapartida, no *workflow* de negócio, há necessariamente um fluxo de controle. Embora o fluxo de dados faça parte do *workflow* científico, caracterizá-lo como um *dataflow* pode ser simplista, uma vez que *workflows* científicos estão associados a diversos outros recursos para facilitar o gerenciamento de experimentos, incluindo recursos para gerência de configuração (Callahan *et al.* 2006; Ogasawara *et al.* 2009b), visualização científica (Callahan *et al.* 2006; Horta *et al.* 2013) e proveniência (Altintas *et al.* 2006; Oliveira *et al.* 2012). Ainda assim, usar uma semântica de *dataflows* para representar *workflows* científicos pode trazer vantagens do ponto de vista do gerenciamento da execução iterativa. Mosconi e Porta (2000), por

exemplo, fazem um levantamento das abordagens iterativas em *dataflows* e apresentam construtos iterativos para linguagens visuais de *dataflow*. Os construtos apresentados por Mosconi e Porta servem como base para os construtos iterativos desta tese no contexto da álgebra de *workflows* científicos.

3.3 Álgebra de *Workflows* Científicos

Nesta tese, utilizamos uma álgebra centrada em dados para representar um *workflow* científico. Ao utilizar uma álgebra para representar um processo, busca-se uniformizar os elementos envolvidos e agregar informações semânticas que facilitem a automatização de processos de otimização. Em bancos de dados relacionais, a álgebra relacional (Özsu e Valduriez 2011) é o elemento chave no processo de otimização e processamento de consultas. Ao utilizar uma álgebra para representar *workflows*, uniformizamos os dados na forma de relações e adicionamos semântica às atividades que processam esses dados. Dessa forma, a máquina de *workflow* pode interpretar o *workflow* e conhecer o comportamento do fluxo de dados

A abordagem algébrica para *workflows* científicos (Ogasawara *et al.* 2011) é uma extensão da álgebra relacional, onde os conjuntos de dados são estruturados como relações. Operações algébricas (ϕ) são associadas às atividades do *workflow* agregando a informação de como a atividade consome e produz dados. Para uma dada relação de saída *T* com esquema $\mathcal{I}$ podemos escrever

$$T \leftarrow \phi(Y, \{\omega\}, \{R\})$$

onde $\{R\} = \{R_1, R_2, \ldots, R_n\}$ é o conjunto de relações de entrada com esquemas $\{\mathcal{R}_i, \ldots, \mathcal{R}_n\}$, $\{\omega\} = \{\omega_1, \omega_2, \ldots, \omega_n\}$ é o conjunto opcional de operandos adicionais e *Y* é a atividade do *workflow*. Nós representamos $Y <\{\mathcal{R}_i, \ldots, \mathcal{R}_j\}, \mathcal{I}, UC>$ como uma atividade que consome um conjunto de relações compatíveis com os esquemas $\{\mathcal{R}_i, \ldots, \mathcal{R}_n\}$ e executam uma dada unidade computacional (UC) para produzir uma relação de saída compatível com o esquema $\mathcal{I}$. É importante observar que a UC pode ser um software legado ou de terceiros, incapaz de compreender a representação relacional dos dados. Uma operação algébrica

$$\phi \in \{SplitMap, Map, Reduce, Filter, SRQuery, MRQuery\}$$

expressa o comportamento da atividade *Y* em relação ao consumo dos dados da relação de entrada e produto dos dados da relação de saída. Este metadado é importante pois vai

permitir ao modelo de execução ir da representação abstrata da atividade *Y* consumindo a relação *R* para as diversas instâncias de execução da UC de *Y* em paralelo.

3.3.1 Map

A operação Map está associada a atividades que, para cada tupla de entrada, produzem uma única tupla de saída. Este cenário é muito comum como no exemplo a seguir:

$$\begin{Bmatrix}(1,'Hello\ John'),\\(2,'Hello\ Mary')\end{Bmatrix} \leftarrow Map\left(H,\begin{Bmatrix}(1,'John'),\\(2,'Mary')\end{Bmatrix}\right)$$

Onde:

$$H\left\langle\begin{matrix}\{id:integer, name:String\},\\\{id:integer, hello:String\},\\'Hello.exe'\end{matrix}\right\rangle.$$

No exemplo, a tupla de entrada ('John') é consumida pelo programa *Hello.exe* e produz uma tupla de saída ('Hello John'). Possivelmente em paralelo, uma outra instância de *Hello.exe* consome a tupla 'Mary' para produzir a tupla 'Hello Mary'.

Esta definição de Map é um pouco diferente da definição no paradigma MapReduce. No MapReduce, a função Map pode produzir uma ou diversas saídas. Na álgebra de *workflows*, separamos a situação onde a atividade produz uma única tupla do caso onde diversas tuplas são produzidas. Dessa forma, ampliamos a semântica das atividades, trazendo mais oportunidades para técnicas de otimização.

3.3.2 SplitMap

A operação SplitMap está associada a atividades que, para cada tupla de entrada, produz um conjunto de tuplas de saídas. Ela está relacionada a métodos de fragmentação e decomposição. Normalmente, a tupla de entrada tem uma referência a um arquivo. Este arquivo é fragmentado durante a execução da atividade e cada fragmento é referenciado separadamente na tupla de saída como o seguinte exemplo:

$$\begin{Bmatrix}(1,1,'pic01.jpg'),\\(1,2,'pic02.jpg'),\\(1,3,'pic03.jpg')\end{Bmatrix} \leftarrow SplitMap(Z,\{(1,'pics.zip')\})$$

Onde

$$Z\left\langle \begin{matrix} \{zid: integer, zip: FileRef\}, \\ \{zid: integer, pid: integer, pic: FileRef\}, \\ 'unzip.sh' \end{matrix} \right\rangle.$$

No examplo, o script hipotético unzip.sh consome o arquivo compactado pics.zip e gera três arquivos de imagens armazenados separadamente na relação de saída. O conteúdo dos arquivos não é armazenado nas relações, somente as referências para os arquivos.

3.3.3 Reduce

A operação Reduce está associada a atividades que agregam dados. Para um conjunto de tuplas de entrada, ela gera uma única saída. As tuplas são agrupadas por um conjunto de atributos de agregação definidos como um operando adicional G_a. Os resultados são agregações horizontais das tuplas de entrada. Por exemplo:

$$\begin{Bmatrix} ('big', 2), \\ ('data', 3) \end{Bmatrix} \leftarrow Reduce\left(W, \{'word'\}, \begin{Bmatrix} ('big', 1), \\ ('big', 1), \\ ('data', 1), \\ ('data', 1), \\ ('data', 1) \end{Bmatrix} \right)$$

Aonde:

$$W\left\langle \begin{matrix} \{word: String, count: Integer\}, \\ \{word: String, total: integer\}, \\ 'WordCount.jar' \end{matrix} \right\rangle$$

No exemplo, a operação Reduce agrupa a relação de entrada pelo campo '*word'* definido no esquema e executa, para cada grupo, um programa Java *WordCount.jar* que soma o valor do atributo *count*. A soma é armazenada no campo *total* da relação de saída.

3.3.4 Filter

A operação Filter é utilizada quando uma dada atividade avalia se um dado de entrada é adequado para continuar no *dataflow* ou se ele deve ser descartado. Basicamente, uma tupla de entrada vai para a saída se e somente se a atividade associada ao Filter aprova esta tupla baseado no critério do filtro. Originalmente, Ogasawara et al. (2011) definem o *Filter* de forma que, se uma tupla de entrada não é aprovada pela atividade, ela é descartada e não aparece no conjunto de dados de saída. Nesta tese entretanto,

utilizamos uma semântica funcional, de forma que, se a tupla de entrada não atende ao critério do filtro, uma tupla nula é produzida. Por exemplo:

$$\begin{Bmatrix}(1,'email.txt'),\\ null\\ (3,'email.txt')\end{Bmatrix} \leftarrow Filter\left(S, \begin{Bmatrix}(1,'email.txt'),\\ (2,'email.txt'),\\ (3,'email.txt')\end{Bmatrix}\right)$$

Where:

$$S\left\langle \begin{matrix}\{id: integer, email: FileRef\},\\ \{id: integer, email: FileRef\},\\ 'isSpam.py'\end{matrix}\right\rangle.$$

No exemplo acima, o script *isSpam.py* avalia que a tupla com id 2 não atende um dado critério, portanto ela não é encaminhada para a relação de saída.

3.3.5 SRQuery e MRQuery

Tanto a operação SRQuery quanto a MRQuery utilizam expressões da álgebra relacional como a UC da atividade ao invés de programas ou scripts. Isto significa que ela processam consultas tradicionais da álgebra relacional sobre as relações de entrada e inserem o resultado na relação de saída. A operação SRQuery deve ser utilizada para operações unárias, i.e., seleções e projeções, enquanto a MRQuery deve ser utilizada para operações binárias, i.e., uniões, diferenças de conjuntos, produtos cartesianos e junções. No exemplo seguinte de uma SRQuery,

$$\begin{Bmatrix}(1,'r.data'),\\ (4,'r.data',)\end{Bmatrix} \leftarrow SRQuery\left(Q, \begin{Bmatrix}(1,'r.data',0.25),\\ (2,'r.data',0.55),\\ (3,'r.data',0.83),\\ (4,'r.data',0.10)\end{Bmatrix}\right)$$

onde

$$Q\left\langle \begin{matrix}\{id: integer, result: FileRef, error: double\},\\ \{id: integer, result: FileRef\},\\ '\pi_{id,result}(\sigma_{error<0.50}(R))'\end{matrix}\right\rangle,$$

a saída contém apenas os dados do campo *result* da relação de entrada onde o campo *error* é menor que 0.50. Além disso, a saída é uma projeção[2] que armazena apenas o id e a referência ao arquivo na relação de saída. Diferente do exemplo acima, a operação MRQuery consome mais de uma relação de entrada e produz uma única relação de saída. Por exemplo:

$$\begin{Bmatrix}('A','model.o'),\\('B','model.o')\end{Bmatrix} \leftarrow MRQuery\left(J, \begin{Bmatrix}(1,'A'),\\(2,'B')\end{Bmatrix}, \begin{Bmatrix}(1,'model.o'),\\(2,'model.o')\end{Bmatrix}\right)$$

onde

$$J\left\langle \begin{matrix} \begin{Bmatrix}[id: integer, modelName: String],\\ [id: integer, modelFile: FileRef]\end{Bmatrix}', \\ \{modelName: String, modelFile: FileRef\}, \\ '\pi_{modelName,modelFile}(R_1 \bowtie_{id} R_2)' \end{matrix} \right\rangle$$

No exemplo, as relações de entrada R_1 e R_2 são agregadas através de uma junção pelo atributo *id*. Novamente, apenas uma projeção dessa junção é armazenada na relação de saída. A Tabela 1 sumariza as operações algébricas disponíveis na álgebra de *workflows* e apresenta a razão de dados produzidos por dados consumidos.

Tabela 1. Operações algébricas para *workflows*.

Operação	Operando Executado	Razão
Map(*H*, *R*)	Programa	1:1
SplitMap(*Z*, *R*)	Programa	1:m
Reduce(*W*, G_a, *R*)	Programa	n:1
Filter(*S*, *R*)	Programa	1:[0,1]
SRQuery(*Q*, *R*)	Expr. da Álgebra Relacional	n:m

[2] *Projeção é uma operação unária da álgebra relacional de bancos de dados (Elmasri e Navathe 2010) que retorna as tuplas da relação projetada restringindo o conjunto de atributos. Na prática, ela filtra colunas da relação.*

MRQuery(J, $\{R_i, \ldots, R_n\}$)	Expr. da Álgebra Relacional	n:m

3.3.6 Representação do *Workflow*

Baseado nos operadores apresentados, é possível representar um *workflow* através de um conjunto de operações encadeadas através de suas relações. Esta representação é muito semelhante à representação tradicional na álgebra relacional. O exemplo a seguir é um *workflow* hipotético de três atividades.

$$T_1 \leftarrow SplitMap(Y_1, R_1)$$
$$T_2 \leftarrow Map(Y_2, T_1)$$
$$T_3 \leftarrow Reduce(Y_3, 'x', T_2)$$

3.4 Proveniência de Dados

Embora já tenhamos definido e discutido a proveniência de dados anteriormente, vamos aqui ressaltar alguns conceitos relevantes para os capítulos seguintes. Registrar a proveniência de um dado pode ser definido como o registro da origem do dado e o processo pelo qual este dado chegou a um banco de dados (Buneman *et al.* 2001). No contexto da Ciência em larga escala, Simmhan et al. (2005) definem proveniência como as informações que ajudam a determinar o histórico de um dado produzido, começando por suas origens. Segundo Freire et al. (2008), em experimentos científicos, a proveniência nos ajuda a interpretar e compreender resultados: ao examinar a sequência de etapas que levou a um resultado, podemos ter a percepção da cadeia de raciocínio utilizada em sua produção, verificar que o experimento foi realizado de acordo com procedimentos aceitáveis, identificar as entradas do experimento e, em alguns casos, reproduzir os resultados.

Em alguns trabalhos, o termo linhagem é utilizado como sinônimo de proveniência (Simmhan *et al.* 2005). Neste tese, entretanto, o termo linhagem é utilizado como parte dos construtos iterativos na álgebra de *workflows*. O conceito de linhagem foi inspirado na proveniência, porém nós utilizamos o termo com outro propósito. O objetivo da proveniência é mais geral, pois registra o histórico dos dados do *workflow* como um todo. O objetivo da linhagem é identificar todos os dados envolvidos na iteração do *workflow* com um identificador do seu ancestral. Dessa forma, para um determinado dado de entrada, é possível verificar os seus descendentes a cada iteração. Embora distintos, proveniência e linhagem (como descrita nesta tese)

compartilham da relação entre o dado e a sua origem. Por tal motivo, a linhagem contribui para o registro de proveniência facilitando as consultas ao longo das iterações do *workflow*. Na prática, estamos utilizando a proveniência ativamente na execução do *workflow* e não apenas como um registro histórico, uma vez que a linhagem tem um papel decisivo no controle implícito do ciclo dinâmico.

O armazenamento da proveniência pode ser feito de diferentes formas (Altintas *et al.* 2006; Bowers *et al.* 2008; Moreau *et al.* 2008; Gadelha *et al.* 2011; Costa *et al.* 2013). Nesta tese, nós utilizamos o modelo de proveniência do Chiron (Ogasawara *et al.* 2013) que segue o modelo de referência W3C PROV como descrito em Costa et al. (2013). O modelo W3C PROV (Moreau e Missier 2011) representa conceitos de alto nível os quais permitem que representações da proveniência específicas de um domínio ou aplicação possam ser traduzidas em um modelo de dados. Ele possui três elementos principais como ilustra a Figura 2: entidades, atividades e agentes. Entidades representam coisas com aspectos fixos, sejam elas físicas, digitais, conceituais ou de outro tipo. Entidades podem ser reais ou imaginárias. Uma atividade é algo que ocorre em um período de tempo e age sobre entidades ou em conjunto com entidades; esse processo pode incluir o consumo, processamento, transformação, modificação, realocação, utilização ou geração de entidades. Um agente é algo que tem alguma forma de responsabilidade sobre uma atividade acontecendo, pela existência de uma entidade, ou pela atividade de um outro agente.

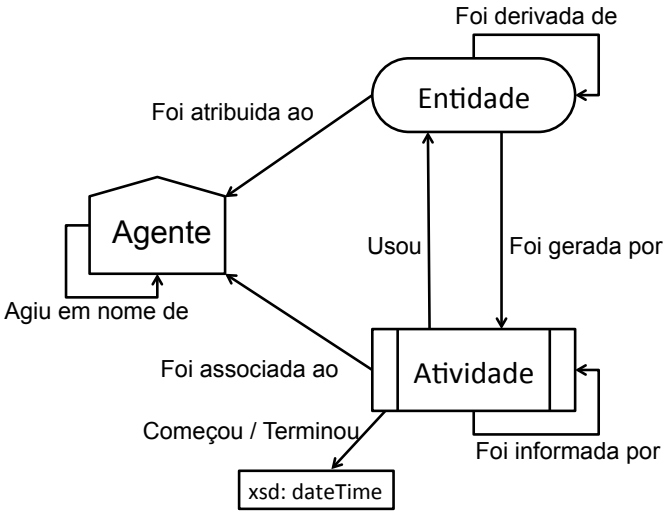

Figura 2. Relacionamentos entre os elementos do PROV (Moreau e Missier 2011).

Para o modelo de proveniência em tempo real do Chiron, realizamos uma extensão no modelo PROV específica para *workflows*, denominada PROV-Wf (Costa *et*

al. 2013). Neste modelo, os dados de entrada, dados de saída e os dados intermediários do *workflow* são referenciados na proveniência como membros de uma determinada relação. As relações são entidades derivadas de um esquema, que é membro de uma determinada atividade. Na prática, cada relação do *workflow* torna-se uma relação (tabela) no banco de dados de proveniência. Essa propriedade é importante pois estabelece uma conformidade com a álgebra de *workflows* científicos, onde os dados do *workflow* são representados de forma uniforme como relações. Essa característica aproxima o modelo da proveniência prospectiva do *workflow* do modelo de execução do *workflow*, facilitando o monitoramento da execução e a análise dos resultados. A Figura 3 exemplifica como o modelo da proveniência reflete o modelo do *workflow*. No exemplo, a relação *ISolver* é consumida pela atividade *Solver* e os resultados armazenados na relação *OSolver*. Em seguida, estes resultados são propagados para a relação *IMerge* como uma projeção da relação *OSolver* e assim sucessivamente. As relações na proveniência se relacionam por meio de chave estrangeira. Portanto, se o cientista quiser consultar algum resultado produzido, ou acompanhar a execução da atividade *Stats*, basta ele consultar a relação *OStats* na proveniência. Se ele também desejar saber qual entrada da relação *ISolver* produziu uma determinada saída do atributo do experimento *kin_energy_large* presente na relação *OStats*, ele poderá descobrir realizando junções entre as relações.

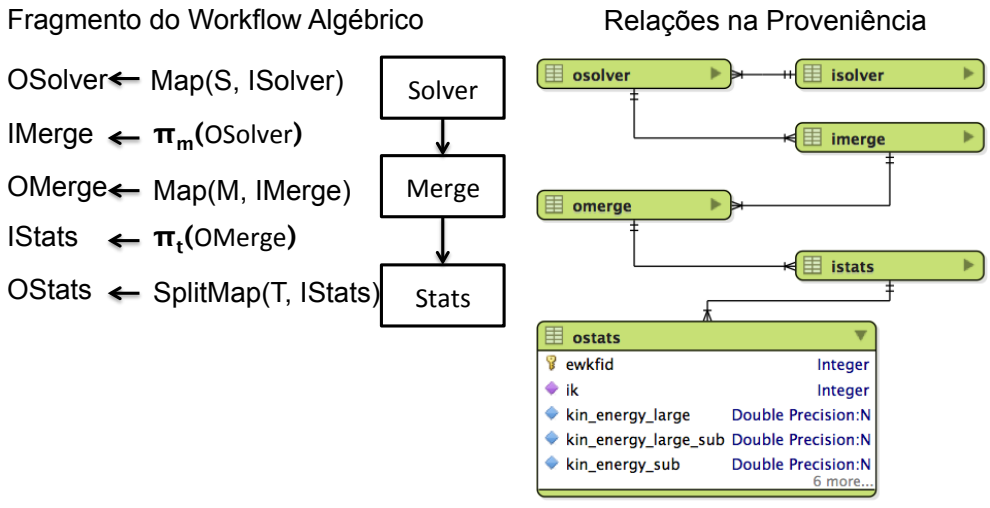

Figura 3. Relação entre o modelo algébrico do *workflow* com as relações na proveniência do PROV-Wf.

3.5 *Workflows* Dinâmicos

Hoje em dia, tanto o ambiente de desenvolvimento quanto o ambiente de execução de experimentos científicos são distribuídos. A colaboração entre centros de pesquisa faz com que cientistas com equipamentos distribuídos geograficamente precisem trabalhar no desenvolvimento do mesmo experimento. Além disso, ao executar tal experimento, eles precisam utilizar recursos computacionais distribuídos como grades computacionais, clusters e nuvens computacionais. Neste contexto, a utilização de *workflows* estáticos é pouco adequada. A ideia de apoiar *workflows* dinâmicos, adaptativos e conduzidos pelo usuário busca possibilitar e acelerar a metodologia científica distribuída e colaborativa através de rápida reutilização, melhoria e exploração de modelos acompanhados de contínua adaptação (Gil *et al.* 2007). Neste cenário, a reprodução do experimento pode ser ainda mais complexa. Portanto, o desafio é desenvolver mecanismos para criar, gerenciar e capturar *workflow* dinâmicos para possibilitar a reutilização e reprodução de resultados significantes em experimentos.

Um cenário muito comum em experimentos científicos é a utilização de resultados obtidos com um *workflow* inicial para tomar decisões a respeito de uma nova análise mais complexa realizada em seguida. Cientistas precisam compor *workflows* de forma dinâmica para analisar os resultados dos passos iniciais antes de tomar uma decisão e dar continuidade aos passos seguintes da análise. As decisões a respeito do experimento também podem ser desencadeadas a partir de eventos externos, que podem alterar a estrutura ou a semântica dos *workflows*. Dependendo do evento ocorrido, o *workflow* vai adotar um comportamento diferente. Dois *workflows* também podem afetar um ao outro através do compartilhamento de resultados, assumindo um comportamento dinâmico conforme eles respondem a eventos disparados nas execuções um do outro.

Experimentos em larga escala envolvem um grande número de cientistas e técnicos. As metodologias executadas são demoradas e muitas vezes necessitam de intervenção humana e condução dinâmica ao longo do processo experimental. Apoiar cientistas no complexo processo exploratório dos experimentos envolvendo *workflows* dinâmicos é um desafio (Gil *et al.* 2007). Cientistas precisam de um sistema de apoio a decisões que acomode suas necessidades ao buscar informações para compreender os

mais complexos processos envolvidos em todas as etapas de um experimento. É necessário um mecanismo eficiente de consultas com uma interface que permita facilmente que o cientista navegue, faça buscas, revise, e compreenda a informação. Simplificar o processo exploratório também requer formas inovadoras e escaláveis que permitam ao cientista manipular seus *workflows*, explorando fatias do espaço de parâmetros, e comparar resultados de diferentes configurações.

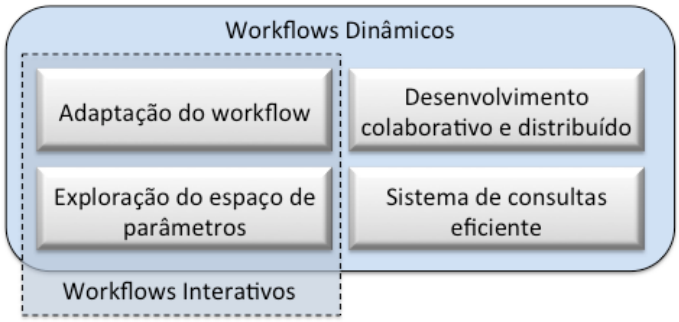

Figura 4. Principais requisitos de *workflows* dinâmicos.

Como discutido anteriormente, os *workflows* dinâmicos possuem quatro características principais conforme ilustra a Figura 4. Para possibilitar o desenvolvimento de *workflows* dinâmicos, um SGWfC precisa apoiar o desenvolvimento colaborativo e distribuído. Isso significa que cientistas de diferentes grupos de pesquisa precisam compartilhar dados de entrada, processos e resultados. Para tal, os *workflows* compartilhados devem ser interoperáveis (Korkhov *et al.* 2011; Plankensteiner *et al.* 2011). A entrada e ajustes dos dados também pode ocorrer de forma distribuída, vinda de diferentes equipamentos distribuídos geograficamente em diferentes centros de pesquisa que colaboram entre si. As análises precisam ser realizadas de maneira colaborativa, logo os dados de proveniência precisam ser disponibilizados a todos os colaboradores. Para consultar estes dados, os cientistas precisam de um sistema de consultas que permita que eles naveguem pelos dados e façam consultas *ad-hoc* para realizar analises sofisticadas, incluindo a comparação de resultados obtidos com diferentes configurações do *workflow*. Os resultados também precisam ser exibidos em diferentes ambientes de visualização, incluindo ambientes de visualização científica com painéis de monitores (*tiled wall displays*). Estes ambientes requerem uma interface específica para que sejam utilizados de forma simples pelo cientista pois, normalmente, utilizam um conjunto de aplicativos (Byungil Jeong *et al.* 2010) para gerenciar a renderização de imagens através do conjunto de monitores. Integrar tais ambientes ao mecanismo de análise do *workflow* pode exigir a

programação de módulos específicos para o aplicativo que controla o ambiente de visualização. Embora o desenvolvimento colaborativo e o sistema de consultas eficiente para *workflows* científicos sejam de grande importância, eles não fazem parte do objetivo desta tese.

A adaptação do *workflow* possibilita recursos como intervenção humana e condução dinâmica do experimento em tempo real. Os *workflows* também precisam se adaptar em resposta a eventos externos como resultados de outros *workflows*, dados gerados por equipamentos e decisões tomadas pelos cientistas. Além disso, os cientistas precisam explorar o espaço de parâmetros dividindo o espaço em fatias endereçáveis. Em seguida, os cientistas podem tomar decisões como descartar dados de entrada ou comparar resultados de diferentes fatias. A exploração do espaço de parâmetros também exige uma forma simples de consolidar os dados de proveniência e resultados da execução de forma que o cientista possa realizar a análise da execução como um todo e não da execução das fatias de forma individual. Tanto a adaptação do *workflow* quanto a exploração do espaço de parâmetros são características associadas a interação do cientista com o *workflow* especificamente. Desta forma, classificamos essa subdivisão dos *workflows* dinâmicos como *workflows* interativos como ilustra a Figura 4.

Os *workflows* interativos possuem características dinâmicas, mas não contemplam recursos para desenvolvimento colaborativo e não estão focados em um sistema de consultas dos dados de proveniência. Entretanto, para atender aos requisitos necessários de um *workflow* interativo, muitos aspectos de consulta e visualização dos dados de proveniência precisam ser levados em consideração, uma vez que a condução dinâmica e a interatividade dependem do processo de monitoramento, visualização e análise dos resultados parciais para então efetuar-se modificações no *workflow* (Mattoso *et al.* 2013).

3.6 Condução de *Workflows* pelos Usuários

Embora o dinamismo seja necessário em muitos experimentos, como não existem ferramentas disponíveis para gerenciar *workflows* de forma dinâmica, muitos cientistas conduzem manualmente a execução do experimento de forma estática. A abordagem de gerência manual estática envolve a modelagem do *workflow* científico tradicional, o qual é executado com uma configuração predeterminada. Se o resultado final obtido não é satisfatório, o cientista reexecuta todo o *workflow* com uma nova configuração ou

ajustando os dados de entrada. Mesmo que parte da execução do experimento tenha obtido bons resultados, como a gerência do que pode ser aproveitado e o que deve ser descartado é complexa, o cientista normalmente opta por reexecutar todo o *workflow* ajustando ou corrigindo o trecho indesejado. Se, durante a execução do *workflow*, o cientista nota alguma inconformidade ou resultado parcial inesperado, ele pode interromper o *workflow* para realizar ajustes e ressubmeter a execução com a nova configuração. Porém, ele precisará reexecutar todo o *workflow*, desperdiçando grande parte dos resultados já obtidos.

A abordagem de gerência manual estática é simples, mas pouco eficiente principalmente para análises com uso intensivo de dados e processamento numérico. Análises deste tipo podem demorar semanas ou até meses para terminar. Nestes casos, reexecutar um *workflow* completamente pode ser inviável. Como um experimento naturalmente envolve a realização de ajustes após obter resultados parciais, a abordagem estática tende a gerar desperdício no uso de recursos computacionais, mesmo que se interrompa o *workflow* antes do seu término para se fazer os ajustes.

Uma alternativa à abordagem de gerência manual estática é a execução em etapas. Nesta abordagem, um experimento que deveria estar modelado na forma de um único *workflow*, é modelado como vários *workflows* menores que são executados aos poucos, por etapas. Entre estas etapas, como o experimento foi fragmentado, o cientista pode fazer modificações nos dados de entrada e parâmetros da execução das próximas etapas *workflows*. Ainda assim, muitas vezes, o cientista é obrigado a descartar resultados indesejados ou reexecutar etapas. Ainda que esta abordagem aprimore a gerência manual estática, ela descaracteriza a modelagem do experimento como um *workflow*, pois ao invés de ter um único *workflow* para realizar a análise, o cientista passa ter um conjunto de *workflows*. Sob a perspectiva da proveniência, os dados deste conjunto de *workflows* podem ficar descorrelacionados, obrigando o cientista a analisá-los manualmente em conjunto, criando, por exemplo, visões (Elmasri e Navathe 2010). Além disso, a gerência separada do conjunto de *workflows* é, naturalmente, mais trabalhosa e propensa a erros.

Baseado em nossas experiências recentes apoiando cientistas na execução de seus experimentos (Ogasawara *et al.* 2009a, 2011; Oliveira *et al.* 2011; Guerra *et al.* 2012), a falta de recursos dinâmicos em *workflows* tornou-se uma questão crítica em experimentos envolvendo métodos iterativos. Muitos *workflows* científicos

computacionalmente intensivos são baseados em métodos que precisam de alguma forma de iteração. Dentre eles encontramos experimentos envolvendo problemas de otimização (Kelley 1999), quantificação de incertezas (Ma e Zabaras 2009; Guerra *et al.* 2012) e métodos de redução de base para soluções numéricas de condução de calor (Coutinho *et al.* 1989). Todos eles envolvem passos iterativos para encontrar respostas aproximadas para o problema.

É possível utilizar algoritmos genéticos (Bäck 1996) para otimizar o projeto de dutos (do inglês *risers*) de extração de petróleo de reservatórios em águas profundas (Lima *et al.* 2005). Neste contexto, a função de avaliação do problema de otimização pode ser composta por programas relacionados às análises estruturais dos dutos de petróleo. A Figura 5 apresenta o exemplo onde um conjunto de cromossomos é analisado por uma função de avaliação composta por duas atividades F e T que realizam a análise de flexibilidade e de tensão dos dutos, respectivamente. Em seguida, o algoritmo genético utiliza os resultados da avaliação para realizar a seleção de indivíduos (S), o cruzamento (C) e a mutação (M) nos cromossomos gerando uma nova geração de indivíduos. Após algumas iterações, o algoritmo pode convergir para o indivíduo com a melhor aptidão. Entretanto, o melhor indivíduo pode ter convergido para um mínimo ou máximo local. Os cientistas podem querer explorar o atual resultado antes de interromper a iteração de forma que possam tentar novas mutações em alguns indivíduos para manter o algoritmo em execução por mais tempo prevenindo a convergência prematura.

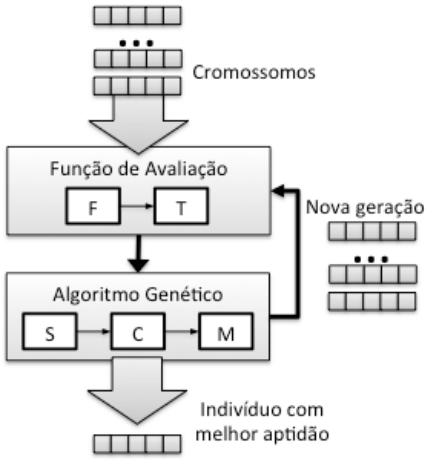

Figura 5. Exemplo de algoritmo genético.

O algoritmo genético é um bom exemplo onde cientistas se beneficiam do uso de explorações dinâmicas de parâmetros em *workflows* científicos. A utilização de

SGWfC para gerenciar experimentos já oferece uma visão estruturada do experimento e coleta nativa de proveniência. Adicionalmente, um *workflow* dinâmico pode permitir que cientistas conduzam o *workflow* ajustando parâmetros para obter uma resposta mais refinada ou definitiva. No exemplo da Figura 5, o núcleo do algoritmo genético está destacado da função de avaliação, já que são compostos de diferentes atividades no *workflow* (F, T, S, C, e M). Esta abordagem é mais flexível pois os cientistas podem facilmente alternar os programas da função de avaliação ou mudar o método de seleção (atividade S) do algoritmo. Em um *workflow* dinâmico, tais mudanças poderiam ser feitas durante a execução do experimento. Além disso, para possibilitar a condução de *workflows* pelos cientistas, os dados de proveniência precisam estar disponíveis para consultas durante a execução. Por exemplo, se o número de gerações do algoritmo é fixado inicialmente em 500 e durante a execução o cientista analisa os dados parciais consultando a base de proveniência e decide que o indivíduo atual com melhor aptidão já é satisfatório, o cientista pode interromper o ciclo iterativo do *workflow* antes que ele atinja as 500 iterações. A ideia deste exemplo pode ser estendida para outras abordagens como o método adaptativo de colocação de *grids* esparsas (Ma e Zabaras 2009; Guerra *et al.* 2012), uma alternativa às simulações com o método de Monte Carlo para a quantificação de incertezas. Neste caso, o número máximo de amostras pré-configuradas para a execução pode interromper a simulação prematuramente. Para prevenir este cenário, os cientistas podem interagir com o *workflow* para ajustar o parâmetro.

Outra família de problemas relevantes para *workflows* dinâmicos é a classe de experimentos envolvendo modelos de ordem reduzida MOR (do inglês *Reduced-Order Models, ROM*) para solucionar sistemas de equações lineares em problemas de otimização de topologia (Wang *et al.* 2007) e simulações envolvendo dinâmica de fluidos computacional (Amsallem e Farhat 2008). Estes modelos buscam a representação matemática mais simples que captura o comportamento dominante de um sistema a ser simulado. Os métodos de redução tipicamente trocam um conjunto de equações por outro conjunto com uma quantidade consideravelmente menor de incógnitas, utilizando uma transformação de coordenadas (Coutinho *et al.* 1989). Para gerar uma matriz de transformação, normalmente é utilizado um método que calcula o subespaço de Krylov, e o algoritmo de Lanczos (Heath 2002) é o membro mais simples desta classe de métodos. O algoritmo de Lanczos é baseado na construção iterativa de

uma base ortonormal no subespaço de Krylov para uma dada matriz. O algoritmo calcula os autovalores aproximados – também chamados de valores de Ritz – da matriz e as aproximações Rayleigh-Ritz dos seus auto-vetores. O número de valores de Ritz calculados está associado ao número de iterações do algoritmo. Para produzir mais valores de Ritz, o algoritmo precisa reiterar para dar continuidade à sequência de Krylov. Entretanto, não é eficiente computar todos os valores de Ritz utilizando o algoritmo de Lanczos, já que é bem conhecido que a sequência truncada já proporciona aproximações muito boas dos autovalores, principalmente para aqueles nas extremidades do espectro. Portanto, o número de iterações do algoritmo de Lanczos precisa ser dinamicamente ajustado pelo cientista durante a execução do experimento.

Uma característica interessante dos MOR é que eles podem ser reutilizados em problemas de domínios similares. Praks et al. (2006) aplicaram uma técnica denominada reciclagem de subespaços de Krylov para sequências de sistemas lineares, baseados no fato de que mudanças pequenas em um sistema de equações causam pequenas mudanças em seu MOR. Como uma simulação é baseada em sistemas de equações lineares para cada passo de tempo, e estes sistemas são similares ao longo do tempo, o subespaço de Krylov produzido durante a solução do primeiro sistema pode ser reciclado como entrada para o próximo sistema, e assim sucessivamente, fazendo a iteração continuar por toda a sequência de sistemas lineares. Esta abordagem é aplicada em diferentes experimentos como propagação de trincas (Parks *et al.* 2006), otimização de topologias (Wang *et al.* 2007) e tomografia difusiva (Kilmer e De Sturler 2006). Adicionalmente, Amsallem e Farhat (2008) propõem uma reutilização mais extensiva de MOR, onde eles mantem um banco de dados de MOR previamente calculados. Quando há uma nova simulação para ser realizada, eles usam um método de interpolação para adaptar MOR armazenados e produzir um novo modelo. Os resultados da interpolação servem como entrada para o novo experimento a ser executado. Esta abordagem é muito interessante do ponto de vista dos *workflows* dinâmicos e da proveniência, uma vez que ela toma a vantagem de resultados previamente calculados para melhorar novos experimentos. Um *workflow* dinâmico deve oferecer métodos de reutilização e exploração rápidos acompanhados de adaptação contínua e de melhorias (Gil *et al.* 2007).

Outra importante classe de problemas que podem se beneficiar das características dinâmicas em *workflows* são os experimentos envolvendo genômica

comparativa (Miller *et al.* 2004). A genômica comparativa é uma área de bioinformática que visa inferir relações de estrutura e função do genoma em várias espécies. Ela é também um campo crítico para a genômica funcional (área da biologia molecular) dado que permite que os cientistas acessem informação sobre genes, suas proteínas resultantes e o papel desempenhado por estas proteínas em processos bioquímicos do organismo (Zvelebil e Baum 2007). Apesar da genômica comparativa estar atraindo muita atenção da comunidade científica na última década (Thornton e DeSalle 2000; Lander 2001; Kudtarkar *et al.* 2010), este campo de pesquisa ainda é considerado um desafio.

Métodos de identificação por similaridade e de alinhamento múltiplo de sequências (AMS) são aplicações importantes usadas em experimentos de genômica comparativa. A comparação por similaridade tem sido o método mais utilizado na detecção de homologias (Pruitt *et al.* 2009). Estes algoritmos de comparação baseiam-se principalmente em técnicas de programação dinâmica, como o descrito por Smith e Waterman em 1981 (Smith 1981). Algumas ferramentas populares, como os pacotes BLAST (Altschul *et al.* 1990) e FASTA (Pearson 1990) são implementações desses algoritmos. Porém, outros pacotes, tais como o HMMER (Eddy 1998), o SAM (Hughey e Krogh 1996) e o THMM (Qian e Goldstein 2004) utilizam métodos baseados no algoritmo de modelos ocultos de Markov (HMM, do inglês *Hidden Markov models*). Este algoritmo utiliza um tipo especial de modelo probabilístico, denominado perfil HMM (pHMM), que tem demonstrado uma alta eficiência na detecção de homólogos distantes (sequências com similaridade muito pequena) (Park *et al.* 1998; Gough *et al.* 2001; Madera e Gough 2002; Wistrand e Sonnhammer 2005). Os pacotes hMMER, SAM e THMM criam pHMM a partir de um conjunto de sequências homólogas (de gene ou enzimas) e avaliam o quanto outras sequências se adaptam ao perfil.

O AMS é considerado o passo mais importante em muitas áreas da bioinformática. A eleição de um programa de AMS que demonstre um bom desempenho, tanto computacional como na identificação de sítios conservados entre as sequências, produz efeitos positivos na obtenção do melhor alinhamento das sequências biológicas (Park *et al.* 1998; Wallace *et al.* 2005). Espera-se que as sequências de um AMS tenham uma relação evolutiva pela qual compartilham uma linhagem e descendam de um ancestral comum. Sendo assim pode-se inferir a homologia e prosseguir com análises filogenéticas ou evolutivas para avaliar as origens evolutivas

compartilhadas pelas sequências. Existem inúmeros programas de AMS baseados em diferentes algoritmos usados na bioinformática, entre os mais usados estão ClustalW (Thompson *et al.* 1994), Kalign (Lassmann e Sonnhammer 2005), MAFFT (Katoh e Toh 2008), Muscle (Edgar 2004), ProbCons (Do *et al.* 2005), e T-Coffee (Notredame 2010).

Um dos critérios que precisa ser levado em consideração na genômica comparativa e em outros experimentos envolvendo o AMS, é o *e-value,* que é o valor esperado de erros por consulta (Madera e Gough 2002). No contexto de uma comparação entre sequências, espera-se que ocorra ao acaso uma média de *E* falsos positivos por modelo que adote um *e-value* melhor que *E*. Porém, o valor ideal do *e-value* pode variar para cada gene de entrada na comparação. É difícil para o cientista estabelecer *a priori* um *e-value* que atenda as suas necessidades para cada um dos genes de todo o conjunto de dados de entrada. Na prática, o que muitos cientistas fazem é fixar um *e-value* médio que traga (baseados na literatura científica, experimentos prévios ou testes) resultados satisfatórios das comparações, porém esta abordagem gera uma sobrecarga na etapa de análise, pois uma série destas sequências nos resultados pode precisar ser descartada por ser considerada como falsos positivos. Neste contexto, o cientista poderia conduzir a execução do *workflow* ajustando o *e-value* conforme o necessário para cada execução, visando produzir resultados de maior qualidade (*i.e.* só incluindo verdadeiros positivos).

Atividades relacionadas aos processos de alinhamento de sequência e detecção de homólogos, quando refinadas pelo cientista (mediante o ajuste do *e-value*), podem gerar resultados otimizados. Estes resultados otimizados, ao serem usados em uma análise mais complexa realizada em seguida (por ex. filogenia ou evolução) economiza recursos, tornando o *workflow* de filogenia (ou evolução) mais eficiente. Do ponto de vista biológico, o esforço do cientista ao conduzir parte do *workflow* ajustando os valores do *e-value* é compensado pela qualidade dos resultados finais.

Métodos para computar ROM, problemas de otimização e experimentos na bioinformática são importantes exemplos que precisam de iteração e características dinâmicas em *workflows*. Todos esses problemas (e possivelmente muitos outros) podem tirar vantagem da adaptação de *workflows* e da condução do experimento com explorações do espaço de parâmetro. O dinamismo acoplado pode melhorar o controle, gerenciamento e a coleta de proveniência em experimentos em larga escala.

3.7 Modelo de Regras Ativas

Estabelecemos uma ligação entre a execução de *workflows* e a álgebra relacional de banco de dados através da álgebra de *workflows*. Sendo assim, uma forma intuitiva de entender interações entre o usuário e o *workflow* em execução é através de regras ativas, popularmente conhecidas como gatilhos (ou, em inglês, *triggers*). De forma geral, regras de banco de dados ativos consistem em um conjunto de *eventos*, uma *condição* opcional e uma *ação* (Baralis e Widom 2000). Tais regras também são conhecidas como regras Evento-Condição-Ação (ECA). Uma regra ativa é disparada, isto é, elegível para avaliação, quando qualquer evento, especificado no conjunto de eventos da regra, acontece. Se a regra incluir uma condição, tal condição precisa ser verdadeira para que a ação seja executada. Segundo a abordagem algébrica para regras de banco de dados ativos de Baralis e Widom (2000), eventos tais como uma inserção (*ins*), remoção (*del*) e atualização (*upd*) em uma relação *R* podem ser vistos como operações algébricas.

Uma operação de inserção é representada pela expressão relacional E_{ins} tal que E_{ins} produz as tuplas (seja uma tupla constante ou tuplas resultantes de uma expressão algébrica) que são inseridas na relação *R*. Após a inserção, o novo estado de *R* é dado por $R \cup E_{ins}$. Portanto, o esquema de E_{ins} deve coincidir com o esquema de *R*. A operação de remoção é representada pela expressão relacional E_{del} tal que E_{del} produz as tuplas que serão removidas. O esquema de E_{del} também deve coincidir com o esquema de *R*. Após a remoção, o novo estado de *R* é dado por $R \,\overline{\ltimes}\, E_{del}$, onde $\overline{\ltimes}$ é uma anti-semi-junção[3] (*antisemijoin*) (Baralis e Widom 2000). A operação de atualização é representada pela expressão relacional E_{upd} cujo esquema é dado como *schema*(*R*) $\cup$ A'_u, onde o atributo A'_u armazena os novos valores para os atributos atualizados A_u. Se o valor do atributo $a \in A_u$ é atualizado, então o novo valor de *a* é atribuído a $a' \in A'_u$. A operação de atualização só ocorre em tuplas já presentes na relação sendo atualizada, logo $\pi_{schema(R)} E_{upd} \subseteq R$. E_{upd} é dada como

$$E_{upd} = \mathcal{E}[A'_{u1} = expr_1]\mathcal{E}[A'_{u2} = expr_2] \ldots \mathcal{E}[A'_{un} = expr_n]E_c,$$

[3] Segundo Baralis e Widom (2000) uma anti-semi-junção expressa sub-consultas negativas (por ex. 'não existe em') e é definida como $E_1 \,\overline{\ltimes}_P\, E_2 = E_1 - (E_1 \ltimes_P E_2)$ tal que $\ltimes_P$ é uma junção pelo atributo *P*.

onde E_c é uma expressão que produz as tuplas que são atualizadas (ou seja, a condição de seleção da operação de atualização). O esquema de E_c deve coincidir com o esquema de R. $\mathcal{E}[A'_{ui} = expr_i]$ avalia a expressão $expr_i$ para cada tupla de E_c e atribui o resultado ao novo atributo A'_{ui}. Em seguida o novo estado de R é dado conforme a expressão

$$\left(R \,\overline{\ltimes}\, E_{upd}\right) \cup \alpha_{A'_u;A_u}(\pi_{A_r;A'_u} E_{upd})$$

onde $\overline{\ltimes}$ é uma anti-semi-junção (*antisemijoin*), $\alpha_{A_1;A_2}$ renomeia o atributo A_1 para A_2 e $A_r = schema(R) - A_u$ assim como definido por Baralis e Widom (2000).

3.8 Tipos de Iteração

Algumas abordagens foram propostas no passado para dar apoio a iterações em linguagens de programação (Aho *et al.* 2006), em workflows (Ekanayake *et al.* 2010; Wozniak *et al.* 2012) e em linguagens de programação centradas em dados. FAD (Danforth e Valduriez 1992) é uma linguagem de programação funcional para banco de dados projetada para facilitar a otimização e a execução paralela, e oferece uma operação de segunda ordem *whiledo* para iterações sobre chamadas de funções de primeira ordem. Mosconi et al. (2000) faz uma revisão sobre um conjunto de abordagens existentes para iterações em linguagens de programação visual centradas em dados e discute o conjunto mínimo de características necessárias que possibilitam iterações em um fluxo de dados.

De acordo com Elmoroth et al. (2010), existem três tipos de iterações em *workflows* científicos: (i) ciclos com contagem sem dependência entre as iterações, também conhecidos como varredura de parâmetros, aonde uma dada computação é aplicada para cada entrada do ciclo; (ii) ciclos com contagem e com dependência, aonde os dados produzidos na iteração *k* é utilizada em *k+1*, mas o número de iterações é conhecido antes da execução do ciclo; e (iii) ciclos condicionais, também conhecidos como iterações sem contagem, ou iteração sequencial, aonde o ciclo é interrompido quando uma dada condição é atingida (por ex. uma estrutura em loop do tipo "*while...do*").

Utilizando os exemplos discutidos na seção 3.6, um experimento de genômica comparativa sem necessidades dinâmicas pode ser executado na forma de um ciclo com contagem sem dependência entre as iterações, ou seja, uma varredura de parâmetros. Nesse caso, cada dado de entrada pode ser processado de forma independente através do

workflow ao longo do experimento. Já o algoritmo de Lanczos pode ser implementado como um ciclo com contagem e com dependências pois o cálculo dos autovalores de uma iteração *k* depende dos resultados da iteração *k-1*. Ainda assim, pode-se estabelecer um valor fixo para o número total de iterações. Já um algoritmo genético é não determinístico, logo deve ser construído como um ciclo condicional que irá iterar o processo até que o algoritmo convirja. Embora tais cenários possam ser implementados com abordagens tradicionais para iteração, se houver uma necessidade dinâmica nestes experimentos, o cientista precisará executar o mesmo *workflow* diversas vezes alterando manualmente a configuração do *workflow* ou os dados de entrada.

Baseado nestes trabalhos, nesta tese propomos uma álgebra iterativa centrada em dados, com construtos iterativos, uma operação algébrica iterativa e um modelo de execução dinâmico. Nós permitimos ao *workflow* ser um grafo direcionado, com ciclos. Nós propomos um quarto tipo de iteração para *workflows* que chamamos ciclos dinâmicos como uma extensão dos ciclos condicionais com o apoio à interações no fluxo de dados através da condução dinâmica de *workflows*.

Capítulo 4 Abordagem algébrica para Iterações

Para dar apoio a *workflows* dinâmicos, propomos construtos iterativos na especificação da álgebra de *workflows* baseados em um conjunto mínimo de características necessárias para possibilitar iterações em fluxos de dados. A álgebra de *workflows* apoia de forma inerente os ciclos com contagem e sem dependências (i.e., varredura de parâmetros). Nós seguimos os construtos iterativos de Mosconi e Porta (2000) para construir uma abordagem iterativa centrada em dados, inspirada em linguagens funcionais para programação de bancos de dados (Danforth e Valduriez 1992). Para dar apoio a ciclos dinâmicos, combinamos nossa abordagem iterativa com um modelo de execução dinâmico discutido no Capítulo 5. Para apoiar ciclos, a álgebra de *workflows* deve satisfazer as seguintes condições: (1) o ciclo é capaz de começar, (2) o ciclo é capaz de iterar e (3) o ciclo é capaz de parar. Nós permitimos que o *workflow* seja um grafo direcionado, podendo haver ciclos, através de uma operação denominada *Evaluate* descrita na seção 4.1. Também introduzimos o conceito de átomo e definimos os conceitos de ativação e *workflow* utilizando o novo conceito de átomos. Para demonstrar que a operação *Evaluate* satisfaz a condição (3), introduzimos o conceito de linhagens na seção 4.2. A linhagem também possibilita consultas aos resultados do *workflow* ao longo das iterações através de um modelo em cubo que propomos.

A álgebra de *workflows* é centrada e conduzida por dados (do inglês *data-driven* (Johnston *et al.* 2004)), *i.e.*, uma atividade do *workflow* é disparada assim que dados de entrada estão disponíveis. Entretanto, como a entrada de uma atividade é uma relação, o conceito de qual dado de entrada é necessário para disparar a atividade pode ficar obscuro. Dependendo da operação associada à atividade, a forma de consumo de dados também é diferente. Por exemplo, se a operação *Map* está associada à atividade, ela consome uma tupla independentemente. Já uma atividade associada à operação *Reduce* consome um conjunto de tuplas de uma só vez. A relação de entrada pode também conter muitas tuplas, portanto diversas instâncias da atividade (ativações) podem ser disparadas em paralelo. Para formalizar a unidade mais fina de dados necessária para executar uma instância da atividade, nós definimos o conceito de átomo de dados.

Definição 1. Um átomo de dados a_i de uma relação R é um fragmento horizontal de R de forma que $a_i \leftarrow \sigma_{F_i}(R)$ para $1 < i < w$ onde F é o predicado de fragmentação ou predicado mintermo (Özsu e Valduriez 2011). O átomo tem as seguintes propriedades:

(i) ele não pode ser decomposto; (ii) para operações *Map*, *SplitMap* e *Filter*, cada átomo a_i é uma única tupla da relação *R*; (iii) Para a operação *Reduce*, cada a_i pode conter um conjunto de tuplas de acordo com o conjunto de atributos de agregação em G_A; (iv) Para as operações *SRQuery* e *MRQuery*, o átomo depende da consulta associada à operação; (v) O conjunto de átomos de entrada de uma relação *R* é denotado como *Atoms*(*R*).

A propriedade (iv) da Definição 1 não dá muitos detalhes sobre o átomo de relações de entrada de atividades associadas às operações *SRQuery* e *MRQuery*. É difícil predizer o tamanho do átomo neste caso porque as consultas das atividades podem conter agregações e junções em suas expressões da álgebra relacional. Para definir o átomo, é necessário decompor a consulta associada às operações *SRQuery* e *MRQuery* para compreender a forma que a relação é lida.

Definição 2. Uma ativação de uma atividade, ou simplesmente ativação, é um objeto autocontido que contém toda informação necessária, *i.e.*, quais programas invocar e quais dados acessar para executar uma atividade em qualquer núcleo de um nó computacional de um computador paralelo. Ativações contém o grão mais fino dos dados necessário para uma atividade executar (Bouganim *et al.* 1996), *i.e.*, um átomo a_i da relação de entrada *R* que possibilita a execução da atividade. Uma ativação tem as seguintes propriedades: (i) Assim que é criada, uma ativação está pronta para executar; (ii) ela consome um átomo a_i de tuplas; (iii) Uma ativação, quando disparada, sempre produz tuplas; (iv) A taxa de consumo e produto de tuplas varia de acordo com a operação associada à atividade que originou a ativação: *Map* (1:1), *SplitMap* (1:m), *Reduce* (n:1), *Filter* (1:1), *SRQuery* and *MRQuery* (n:m); (v) A relação de saída de uma atividade é composta pelo conjunto de tuplas produzido por todas as suas ativações.

Ativações procedem em três etapas: instrumentação dos dados de entrada, invocação do programa e extração dos dados de saída. A instrumentação dos dados de entrada extrai os valores do átomo de entrada e os prepara para a invocação do programa, configurando os valores dos parâmetros de entrada de acordo com o tipo de dado esperado. A invocação do programa dispara e monitora a execução do programa na máquina. A extração dos dados de saída agrega os valores da saída do programa executado e constrói a saída da ativação.

Definição 3. Um *workflow* científico centrado em dados é feito apenas de atividades associadas a operações algébricas, atributos adicionais das operações e suas respectivas

relações de entrada e saída. Além disso, um *workflow* científico centrado em dados é caracterizado pelas seguintes propriedades: (i) Os dados fluem através das relações do *workflow* e são transformados pelas atividades do *workflow*; (ii) A relação de entrada de uma atividade pode ser a relação de saída de outra atividade ou outra relação definida previamente; (iii) Uma atividade pode ter uma ou mais relações de entrada, na qual os dados de entrada são inseridos; (iv) Uma atividade pode ter uma ou mais relações de saída, nas quais seus dados de saída são inseridos; (v) Uma atividade dispara, *i.e.*, gera ativações, assim que um átomo esteja disponível em suas relações de entrada e este ainda não tenha sido consumido anteriormente; (vi) Uma vez que uma atividade foi disparada, é necessário que ela receba mais átomos para que ela dispare novamente; (vii) A relação de saída de uma atividade é a união da saída de todas as suas ativações; (viii) Uma atividade, uma vez disparada, sempre produz algum dado através de sua relação de saída; (ix) Se duas ou mais atividades recebem a mesma relação de saída de uma outra atividade, cada atividade recebe uma instância separada e idêntica de tal relação como entrada, e produz os seus átomos de acordo com a operação algébrica associada.

Baseado na propriedade (ii) da Definição 3, dizemos que existe uma dependência de dados $S \overset{dep}{\longleftarrow} R$ entre duas relações R e S, *i.e.*, S depende de R, se e somente se existe uma expressão algébrica $S \leftarrow \alpha(R)$ que consome R para produzir S. A dependência de dados é transitiva, portanto $\{S \overset{dep}{\longleftarrow} R, T \overset{dep}{\longleftarrow} S\} \vDash T \overset{dep}{\longleftarrow} R$. A Figura 6 ilustra o exemplo de transitividade. Como prova, considere a expressão algébrica $T \leftarrow \beta(S)$. Como $S \leftarrow \rho(R)$, podemos substituir S na primeira expressão $T \leftarrow \beta(\rho(R))$. Portanto, considerando γ como a composição de β e ρ, podemos dizer que $T \leftarrow \gamma(R)$ e, consequentemente, $T \overset{dep}{\longleftarrow} R$.

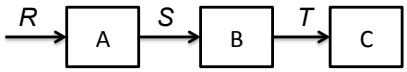

Figura 6. Exemplo de dependência de dados: *T* depende de *S* o qual depende de *R*

Definição 4. Um subgrafo de um *workflow* científico centrado em dados é dito iterativo se ao menos uma atividade *A* existe no subgrafo de forma que os dados de entrada de *A* dependem dos dados de saída de uma atividade *B* e os dados de entrada de *B* dependem dos dados de saída de *A*.

A definição assume um cenário simples como demonstrado na Figura 7.

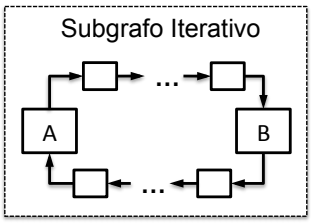

Figura 7. Um subgrafo iterativo simples.

Baseados nas definições 1-4, podemos agora discutir como garantimos as condições (1) o ciclo é capaz de começar, (2) o ciclo é capaz de iterar e (3) o ciclo é capaz de parar na álgebra de *workflows*.

Teorema 1. Em um *workflow* científico centrado em dados, só é possível iniciar um ciclo se ao menos uma atividade do subgrafo iterativo receber mais de uma relação para a mesma entrada.

Prova. Consideremos a Figura 7, a qual apresenta uma contradição ao Teorema 1. Nela, temos um subgrafo iterativo no qual nenhuma atividade recebe mais de uma relação para a mesma entrada. Tomemos, como exemplo, as atividades *A* e *B*. Como a entrada de *A* depende da saída de *B* e a entrada de *B*, por sua vez, depende da saída de *A*, se *A* não pode receber uma entrada inicial, ela não pode disparar, e isto é verdade para qualquer par de atividades do subgrafo. Logo, o ciclo não pode iniciar. Agora consideremos a Figura 8. Neste exemplo modificado, *A* pode receber uma entrada inicial de *C*. Isto faz com que a atividade A dispare, assim como, em algum momento, a atividade *B* dispare também. Como a entrada de *A* vem da saída de *B*, *A* irá disparar novamente e o processo iterativo tem continuidade. ■

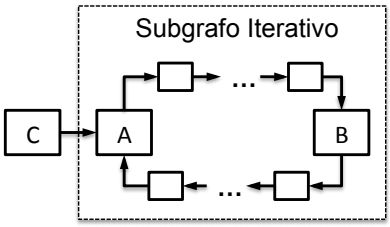

Figura 8. Subgrafo iterativo com uma entrada inicial.

O Teorema 1 diz que duas relações de entradas estão conectadas à mesma entrada de uma dada atividade. É diferente, por exemplo, de uma atividade associada à operação *MRQuery* que pode receber mais de uma relação de entrada, mas cada relação é uma entrada diferente. No caso levantado pelo Teorema 1, uma atividade espera receber uma entrada única, porém tal entrada pode vir como uma relação de entrada

inicial ou uma relação de entrada vinda do subgrafo iterativo. Isto significa que as duas relações de entrada esperada devem possuir um esquema que satisfaça as necessidades da atividade. Além disso, para cada subgrafo iterativo presente no *workflow*, é necessário que haja uma relação de entrada inicial equivalente. O Teorema 1 garante a condição (1) que um ciclo é capaz de começar em um *workflow* algébrico, desde que uma atividade do subgrafo iterativo receba duas relações para a mesma entrada.

Teorema 2. Em um *workflow* científico centrado em dados, iterações são sempre intermináveis.

Prova. Consideremos a Figura 8. No início, a atividade *A* dispara por que recebe uma entrada inicial da atividade *C*. Como a atividade sempre emite algum dado quando disparada (propriedade (viii) da Definição 3), a atividade *B*, a qual deve ter recebido dados de entrada, é disparada e produz dados de saída. Como a entrada de *A* depende da saída de *B*, atividade *A* dispara novamente e este processo nunca termina. ■

O Teorema 2 garante a condição (2) que, após iniciado, o ciclo é capaz de iterar. Entretanto, o ciclo nunca termina.

4.1 Operação de Avaliação Iterativa

De acordo com Mosconi e Porta (2000), uma iteração pode ter um fim se: (a) existe um mecanismo que impede ao menos uma atividade no subgrafo iterativo de disparar se dada condição é satisfeita; ou (b) ao menos uma atividade do subgrafo iterativo é impossibilitada de produzir todo o seu dado de saída, se alguma condição é satisfeita ao disparar-se a atividade. O caso (a) impede uma atividade do *workflow* de disparar. Se uma atividade *Y* do *workflow* não dispara, todas as atividades seguintes das quais os dados de entrada dependem da saída de *Y* serão incapazes de disparar a não ser que haja um sinalizador (*token*) para dar continuidade ao fluxo de dados (Mosconi e Porta 2000). No nosso modelo relacional para *workflows*, utilizamos apenas átomos como sinais para disparar uma atividade. Portanto, acreditamos que o caso (b) é melhor para a álgebra de *workflow* iterativos por que se uma atividade do *workflow* possui duas relações de saída e decide inserir os dados de saída em uma relação ou na outra, a atividade está somente conduzindo o fluxo de dados baseada em uma avaliação interna que é feita quando a atividade é disparada.

As operações algébricas existentes, por exemplo, *Map*, *Reduce*, *SplitMap* e *Filter*, não permitem a definição de duas relações de saída diferentes para uma mesma atividade nem duas relações para a mesma entrada. Portanto, definimos uma nova operação algébrica que apoia os teoremas 1 e 2.

Definição 5. *Evaluate* é uma operação algébrica que é associada à atividades que possuem um comportamento booleano, *i.e.*, os resultados de uma ativação de uma atividade associada à *Evaluate* indica se uma determinada condição é verdadeira ou falsa. A operação *Evaluate* possui as seguintes propriedades: (i) Uma atividade associada a operação *Evaluate* recebe duas relações R_{init} e R_{loop} para a mesma entrada $R_E = R_{init} \cup R_{loop}$ onde $schema(R_{loop}) \supseteq schema(R_{init})$; (ii) O átomo padrão para uma atividade associada à operação *Evaluate* é uma única tupla da relação R_E; (iii) Se um predicado de fragmentação ou predicado mintermo *F* é dado como operando à *Evaluate*, define-se o átomo de entrada como $a_i \leftarrow \sigma_{F_i}(R_E)$ para $1 < i < w$; (iv) Uma atividade associada à operação *Evaluate* produz duas relações de saída T_{true} e T_{false} de forma que o $schema(T_{true}) \subseteq schema(R_{loop})$ e $schema(T_{false}) \subseteq schema(R_{loop})$; (v) *Evaluate* requer a definição de uma função de avaliação ε para decidir se a saída de uma ativação é verdadeira ou falsa; (vi) Se ε é avaliado como verdadeiro, a saída é escrita na relação T_{true}. Do contrário é escrita na relação T_{false}; (vii) $T_{true} \cap T_{false} = \varnothing$, o que significa que apenas uma relação é de fato escrita por ativação.

Uma atividade *Y* associada à operação *Evaluate* é representada como:

$$\{T_{true}, T_{false}\} \leftarrow Evaluate(Y, \varepsilon, R_E)$$

O operando adicional e opcional *F* também pode ser necessário porque uma avaliação pode considerar uma única tupla de entrada, mas também pode precisar avaliar um conjunto de tuplas juntas. Portanto, a taxa de dados consumidos por dados produzidos em uma operação *Evaluate* é n:m. Baseado nos teoremas e definições apresentadas anteriormente, podemos afirmar que em um *workflow* científico centrado em dados, só é possível implementar um comportamento iterativo se uma atividade associado à operação *Evaluate* inicia o subgrafo iterativo. Um exemplo simples é apresentado na Figura 9, onde a atividade *E* é associada à operação *Evaluate*. Os dados de entrada da relação R_{init} são avaliados pela atividade *E* e, se ela os avaliar como verdadeiro, os dados de saída são emitidos a T_{true} e dispara o subgrafo iterativo. Do contrário, os dados de saída são emitidos para T_{false} e disparam o subgrafo de saída. Fazendo uma analogia

com o operador funcional *whiledo*(⟨*loop_fcn*⟩, ⟨*exit_fcn*⟩, ⟨*start_action*⟩) (Danforth e Valduriez 1992), na Figura 9, o subgrafo iterativo seria a *loop_fcn,* o subgrafo de saída seria a *exit_fcn* e a atividade em si associada à operação Evaluate seria a *start_action*.

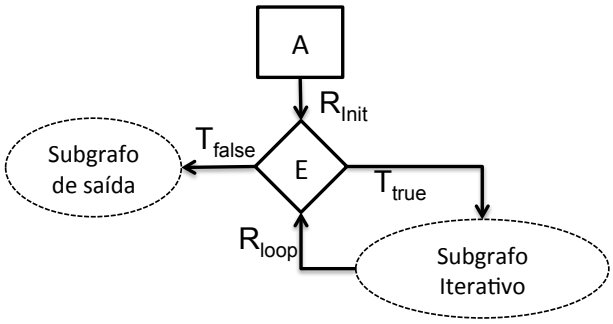

Figura 9. Exemplo de um *workflow* iterativo.

A operação *Evaluate* atende às condições (1) e (2) de que um ciclo é capaz de iniciar e iterar, respectivamente. Embora tenhamos a intuição de que o ciclo iniciado pela operação *Evaluate* também terá um fim, atendendo a condição (3), precisamos de uma maneira de consultar o estado do fluxo de dados no subgrafo iterativo. Baseado nesta consulta, podemos decidir se o ciclo chegou ao fim ou não. Para tal, propomos o conceito de linhagem para iterações na seção 4.2 e apresentamos a prova de que a linhagem associada ao operador *Evaluate* atende à condição (3) de que um ciclo na álgebra de workflows pode ter um fim.

4.2 Linhagem para Iterações

O conceito de linhagem é chave para gerenciar os ciclos em nossa abordagem e abre oportunidades para análise de resultados ao longo das iterações. Nesta seção, nós definimos o conceito de linhagem dando continuidade aos formalismos apresentados na seção 4.1. Além da definição, nós provamos como a nossa abordagem algébrica atende a condição (3), de que o ciclo pode ter um fim, utilizando o conceito de linhagem. Nós também propomos um modelo em cubo de dados para as linhagens para prover acesso estruturado aos dados de proveniência relacionados ao longo da iteração do *workflow*. O cubo de dados é importante para consultas mas também viabiliza os algoritmos de interações descritos na seção 5.3 do Capítulo 5.

Consideremos um *workflow W* com um subgrafo iterativo iniciado por uma atividade *E* associada à operação *Evaluate*, como o exemplo da Figura 9. O subgrafo iterativo pode conter diferentes tipos de atividades e o aspecto dos dados que retornam

para a atividade E através de R_{loop} podem ser diferentes após cada avaliação. Novos átomos podem ter sido produzidos no subgrafo iterativo e os dados existentes sofrem alteração no fluxo de dados. A atividade de avaliação remove átomos da iteração quando avalia o resultado para falso, direcionando os resultados para a relação T_{false}. A única coisa que todos os átomos compartilham enquanto estão dentro do subgrafo iterativo no *workflow* é que eles foram todos derivados de um átomo de entrada inicial $a_i \in Atoms(R_{init})$. Esta propriedade possibilita manter o controle sobre os átomos que ainda estão fluindo no subgrafo iterativo seguindo o átomo que originou aquela iteração. Portanto, definimos que todos os átomos originados de um dado ancestral são marcados como pertencentes a uma dada linhagem como explicada a seguir. Ainda que o conceito de linhagem seja inspirado na proveniência de dados, neste trabalho nós usamos linhagem como um conceito mais específico no *workflow* envolvendo os átomos e seus ancestrais dentro de um subgrafo iterativo.

Definição 6. Uma linhagem $\lambda_k(a_i) \mid a_i \in Atoms(R_{init})$ é um conjunto possivelmente vazio de átomos originados de uma átomo a_i após k avaliações positivas durante uma iteração. Linhagens têm as seguintes propriedades: (i) Há sempre uma linhagem para cada átomo de R_{init}; (ii) Se um átomo é avaliado como falso, ele sai da linhagem, porque, baseado na propriedade (vi) da Definição 5, o resultado obtido com tal átomo é inserido em T_{false} e sai do subgrafo iterativo; (iii) $\bigcup_{i=1}^{w} \lambda_k(a_i) = R_E^k$, o que significa que a entrada para a k-ésima avaliação, denotada como R_E^k é sempre a união de todas as atuais linhagens; (iv) $\lambda_0(a_i) = a_i$, o que significa que antes da primeira avaliação, a linhagem de um átomo é somente o próprio átomo, portanto $R_E^0 = R_{init}$; (v) Se $\lambda_k(a_i) = \varnothing$ então $\lambda_{k+1}(a_i) = \varnothing$, o que significa que se uma linhagem está vazia em uma dada avaliação, na próxima ela permanecerá vazia.

Teorema 3. Se uma atividade associada à operação *Evaluate* dá início a um subgrafo iterativo que recebe um número finito de átomos em sua relação de entrada R_{init} e pode, para cada linhagem $\lambda_k(a_i) \mid a_i \in Atoms(R_{init})$, avaliar para falso em um dado k, a iteração sempre tem um fim.

Prova. O teorema diz que após um certo número de k avaliações, a atividade avalia para falso para uma dada linhagem $\lambda_k(a_i) \mid a_i \in Atoms(R_{init})$. Portanto, baseado na propriedade (ii) da Definição 6, o teorema assume que, após k avaliações uma dada linhagem $\lambda_k(a_i)$ estará vazia, *i.e.*, todos os átomos que já pertenceram a $\lambda_k(a_i)$ já foram

avaliados como falso. Com este pressuposto, podemos dizer que existe uma função $f(a_i) = b_i$, $a_i \in Atoms(R_{init})$ a qual, para cada átomo a_i , retorna um número finito e mínimo b_i de avaliações tal que $\lambda_{b_i}(a_i) = \emptyset$. Podemos também dizer que existe sempre um valor máximo $b_{max} = \max(f(a_1), \ldots, f(a_w))$ tal que $\lambda_{b_{max}}(a_i) = \emptyset \; \forall \; a_i \in Atoms(R_{init})$. Usando a propriedade (iii) da Definição 6, $\bigcup_{i=1}^{w} \lambda_{b_{max}}(a_i) = R_E^{b_{max}} = \emptyset$. Portanto, existe sempre um número finito de avaliações $k=b_{max}$ após as quais a relação de entrada R_E da atividade associada à operação *Evaluate* fica vazia. Sendo assim, de acordo com a propriedade (vi) da Definição 3 e assegurados pela propriedade (v) da Definição 6, após $k=b_{max}$ avaliações, a iteração tem um fim. ■

O Teorema 3 garante a condição (3) que um ciclo é capaz de terminar. Isto completa os construtos necessários para iterações centradas em dados na álgebra de *workflows*. Entretanto, o conceito de linhagem proporciona outros recursos no contexto dos ciclos dinâmicos uma vez que relaciona dados através das iterações. Esta propriedade pode facilitar a comparação de dados entre linhagens ou o acompanhamento da evolução de uma linhagem de dados ao longo da execução do experimento.

Gerenciar dados ao longo das iterações pode ser difícil uma vez que eles adquirem um comportamento de séries temporais. Dessa forma, representar a linhagem na forma de um cubo de dados (Gray *et al.* 1997; Morfonios e Ioannidis 2008) provê um acesso uniforme e estruturado aos dados envolvidos em um subgrafo iterativo do *workflow*. Uma atividade associada à operação *Evaluate* tem duas relações de entrada, portanto ela gera ativações quando há um átomo disponível em R_{init} ou R_{loop}. Se o átomo a_i vem de R_{init}, nós dizemos que $a_i = \alpha_i$ é um átomo-α, e ele origina uma nova linhagem $\lambda_0(\alpha_i)$. Se o átomo a_i vem de R_{loop}, ele já pertence a uma dada linhagem $\lambda_k(\alpha_i)$ onde α_i é o seu k-ésimo ancestral que originou a linhagem. Por uma questão de simplicidade, definimos o conjunto $A=\{\alpha_1, \ldots, \alpha_n\} \mid \alpha_i \in Atoms(R_{init})$.

Considerando o Teorema 3, para aferir se um ciclo chegou ao fim, para todo $\alpha_i \in A$, a linhagem $\lambda_k(\alpha_i)$ deve estar vazia. Logo é importante armazenar os dados da linhagem de forma que possam ser facilmente consultados ao longo das iterações. A linhagem para cada átomo-α muda a cada iteração e pode conter átomos de diferentes relações que estão envolvidas no subgrafo iterativo. Consideremos o conjunto $P = \{R_1, \ldots, R_n\} \mid R_i \overset{dep}{\longleftarrow} R_E$ de relações que dependem de uma atividade Y_E associada à

operação *Evaluate* e um outro conjunto $\mathrm{P}_{False} = \{R_1, \ldots, R_n\} \mid R_i \overset{dep}{\longleftarrow} T_{False}$ de relações dependentes da saída T_{False} da atividade Y_E. Como $T_{False} \in \mathrm{P}$, temos que $\mathrm{P}_{False} \subset \mathrm{P}$. Portanto, o conjunto de relações envolvidas no subgrafo iterativo é $R_{\circlearrowleft} = \mathrm{P} - \mathrm{P}_{False}$. As relações $R_i \in R_{\circlearrowleft}$ estão contém os átomos pertencentes às linhagens.

Com o conjunto A de átomos-α, o conjunto $R_{\circlearrowleft}$ de relações envolvidas no subgrafo iterativo e o número k de iterações, é possível definirmos um cubo de dados $\Lambda(\alpha, R, k) \mid \alpha \in A, R \in R_{\circlearrowleft}, k \in \mathbb{N}$. Para um dado valor de k, é possível detalhar o cubo de dados para descobrir o conjunto de átomos de uma relação R_i como $Atoms(R_i)=\Lambda(\alpha, R_i, k)\ \forall\ \alpha \in \mathrm{A}$ ou a linhagem de um átomo-α como $\lambda_k(\alpha_i)= \Lambda(\alpha_i, R, k)\ \forall\ R \in R_{\circlearrowleft}$. Portanto, para verificar se um ciclo terminou após k iterações, é possível detalhar o cubo e verificar se $\Lambda(\alpha, R, k)=\varnothing\ \forall\alpha \in \mathrm{A}, \forall R \in R_{\circlearrowleft}$.

Na Figura 10, mostramos um exemplo da visualização de um cubo dos dados da linhagem. Destacamos em vermelho um conjunto $Atom(R_i)$ da primeira iteração e, em amarelo, uma linhagem $\lambda_0(\alpha_i)$. A parte azul é o fecho transitivo de um átomo ao longo de k iterações. Também podemos observar que uma linhagem fica vazia na penúltima iteração exibida na figura e mais duas linguagens ficam vazias na última iteração. Ainda que estejamos definindo o cubo utilizando as relações envolvidas na iteração ($R_{\circlearrowleft}$), é possível detalhar o cubo para campos específicos destas relações. Como estes campos podem conter metadados do *workflow*, tais como parâmetros explorados e resultados chave obtidos, é possível que os cientistas realizem análises em tempo real do *workflow* ao longo das iterações fazendo consultas ao cubo de dados de linhagem.

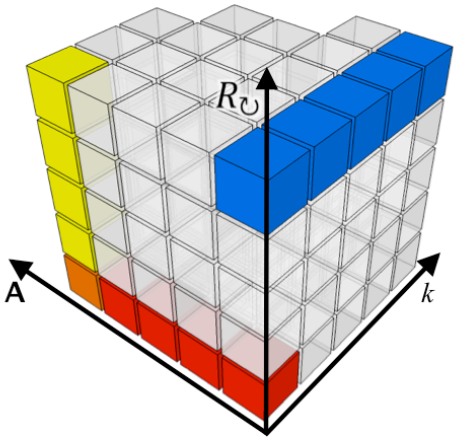

Figura 10. Visão em cubo dos dados de linhagem.

Capítulo 5 Modelo de Execução para Ciclos Dinâmicos

O modelo de execução proposto nesta tese estende conceitos introduzidos por Ogasawara et. al. (2011), porém utiliza novas estruturas para gerenciar o *workflow* dinâmico. Um novo tipo de ativação é utilizado para apoiar as atividades associadas com o operador iterativo. Os dados de proveniência são estruturados na forma de um cubo de dados que permite consultas aos resultados ao longo das interações no experimento através do conceito de linhagem. Também propomos nesta tese dois algoritmos que tratam interações entre o cientista e o *workflow*. A máquina de execução iterativa e o modelo de execução dinâmico, incluindo os algoritmos de interação, foram implementados para esta tese como componentes adicionais na máquina de execução de *workflows* Chiron (Ogasawara *et al.* 2013). Durante a execução, o Chiron consulta a base de proveniência em busca de novos eventos interativos registrados. Ao capturá-los, ele os processa aplicando os algoritmos propostos.

Nosso modelo de execução para ciclos dinâmicos é projetado para execução paralela em clusters com muitos nós e processadores com diversos núcleos. A iteração centrada em dados é uma boa base para ciclos dinâmicos porque a iteração é implícita no fluxo de dados e não explícita em uma estrutura de controle. Por exemplo, cada dado novo inserido na relação de entrada gera novas ativações e, consequentemente, novas linhagens no ciclo. Um aspecto do modelo de execução está associado ao escalonamento e distribuição de ativações aos nós de execução como discutido na seção 5.1. Como introduzimos a nova operação algébrica *Evaluate*, fazemos uma extensão do conceito de ativação para acomodar as necessidades da etapa de avaliação como apresentado na seção 5.2. Outro aspecto importante do nosso modelo de execução é como tratar as interações que os cientistas fazem durante a execução do *workflow*. Portanto, na seção 5.3, propomos dois algoritmos para tratar interações durante a execução do *workflow* utilizando o cubo de dados de linhagens.

5.1 Escalonamento e Distribuição

A base do nosso modelo de execução é como escalonar e distribuir as ativações para execução em múltiplos núcleos dos nós de um computador paralelo. Baseados na Definição 3, um *workflow* algébrico W tem um conjunto de atividades $Y=\{Y_1,\ldots Y_n\}$; cada atividade Y_i está associada a uma operação ϕ_i, a um conjunto ω_i de atributos

adicionais, a um conjunto de relações de entrada R_i e a um conjunto de relações de saída T_i. Baseados na propriedade (v) da Definição 3, desde que hajam átomos disponíveis na relação de entrada de uma atividade, novas ativações x_i são produzidas e adicionadas ao conjunto X de ativações. Cada ativação x_i também está associada a uma dada atividade $Act(x_i)=Y_i$. Originalmente, para produzir novas ativações, o modelo de execução avalia, para cada atividade, se suas dependências de dados foram satisfeitas, *i.e.*, se as atividades ou ativações anteriores no *workflow* já foram executadas. Esta avaliação é feita baseada nos registros de proveniência. Podemos classificar essa abordagem como uma abordagem proativa (uma abordagem *get*), ou seja, o modelo de execução avalia o estado atual da execução para obter novas ativações. Já a abordagem para iterações em *dataflows* apresentada no Capítulo 4 é dirigida por dados (do inglês *data-driven*). Portanto, ela é naturalmente reativa (uma abordagem *put*), uma vez que novas ativações são adicionadas à fila de escalonamento na medida que novos átomos são inseridos na relação de entrada da atividade.

A abordagem reativa é interessante para o modelo iterativo e dinâmico pois ela não se prende ao direcionamento do *workflow*. Se há um dado novo disponível para uma atividade, ela dá continuidade ao fluxo de dados. Desta forma, ela é capaz de responder naturalmente a eventos dinâmicos de inserção de dados e iteração do *workflow*. Para possibilitar a abordagem reativa, acrescentamos o conceito de gatilhos no modelo de execução. Quando uma ativação termina sua execução e as tuplas de saída são registradas na proveniência, um novo gatilho (*Trigger*) é disparado. Consideremos uma atividade Y que escreve dados em uma relação de saída O_Y com esquema de dados $\mathfrak{D}_Y$. Uma segunda atividade Z possui uma relação de entrada I_Z com esquema $\mathfrak{I}_Z$ tal que $I_Z \overset{dep}{\longleftarrow} O_Y$. Quando algum dado é inserido na relação O_Y, um gatilho Tr_1 é disparado. O gatilho Tr_1 é definido como uma regra Evento-Condição-Ação (Baralis e Widom 2000) na qual o evento do gatilho são os dados inseridos na relação O_Y, a condição (C) é sempre verdadeira e a ação executada é a inserção (E_{ins}), em I_Z, dos dados inseridos em O_Y dada uma projeção $\pi_{\mathfrak{I}_Z}(O_Y)$. Cada tupla de O_Y possui uma chave primária (*key*), que é propagada para I_Z. Utilizando a linguagem de Baralis e Widom (2000), temos que:

$$Tr_1 = ins\ O_Y$$

$$E_{ins} = \mathcal{E}[I_Z.key = O_z.key]\pi_{\mathfrak{I}_Z}(O_Y) \bar{\ltimes} I_Z$$

Quando os dados são inseridos em I_Z, novas ativações de Z podem ser produzidas de acordo com as propriedade (v) e (vi) da Definição 3 do Capítulo 4.

Considerando X como a relação que guarda o conjunto de ativações do *workflow* e $\phi(Z, I_Z)$ é a função que produz o conjunto de ativações da atividade Z baseado nos dados presentes em sua relação de entrada I_Z. Para dar apoio às propriedade (v) e (vi) da Definição 3 no Capítulo 4, definimos um outro gatilho (Tr_2) que dispara quando dados são inseridos em I_Z. A condição (C) também é sempre verdadeira e a ação executada é a inserção (E_{ins}), em X, das ativações geradas a partir dos átomos inseridos em I_Z. Na linguagem de Baralis e Widom (2000), temos que:

$$Tr_2 = ins\ I_Z$$

$$E_{ins} = \phi(Z, I_Z) \overline{\ltimes} X$$

O gatilho Tr_2 alimenta a relação X com novas ativações para serem processadas. O modelo de execução pode, então, escalonar e distribuir as ativações $x_i \in X$. O modelo de execução tem duas políticas de escalonamento e duas estratégias de distribuição (Ogasawara *et al.* 2011), as quais estão relacionadas à maneira na qual as ativações $x_i \in X$ são escalonadas e distribuídas para os nós de execução. A política de escalonamento síncrona (ou *First-Activity-First*) escalona as ativações de uma dada atividade se e somente se todas as ativações das atividade precedentes já tiverem terminado, i.e., existe uma barreira de sincronização após cada atividade. A política assíncrona (ou *First-Tuple-First*) escalona *pipelines* de ativações respeitando a dependência de dados entre as ativações. A respeito das estratégias de distribuição, a estratégia estática distribui conjuntos fixos de ativações para os nós de execução. A quantidade de ativações dentro do conjunto é fixo e pode estar relacionada ao número total de ativações dividido pela quantidade de nós de execução disponíveis. A estratégia de distribuição dinâmica envia um única ativação por requisição.

5.2 Ativações de Avaliação

Diferentes da definição original de ativações na álgebra de *workflow*, ativações associadas à operação *Evaluate* precisam decidir se um dado de saída corresponde a verdadeiro ou falso. Logo, definimos uma especialização do conceito de ativação para apoiar a etapa de avaliação utilizando a função de avaliação ε. Ativações seguem um procedimento de execução em três etapas: instrumentação, execução e extração (Ogasawara *et al.* 2011). Baseados nas propriedades (ii) e (iii) da Definição 2, cada ativação x_i consome um átomo de entrada a_i e produz uma saída. Baseados na propriedade (v) da Definição 2, a saída é inserida em uma relação de saída da atividade

Act(x_i). Entretanto, de acordo com as propriedades (v) e (vi) da Definição 5, se uma ativação x_i pertence a uma atividade associada à operação *Evaluate*, existem dois possíveis destinos (T_{True} and T_{False}) para os dados de saída e a função de avaliação ε é utilizada para decidir a saída correta.

A *ativação de avaliação* herda todas as propriedades da ativação original. Adicionalmente, ela possui uma quarta etapa denominada avaliação, que marca o átomo de saída com um sinalizador booleano obtido através de ε. A função de avaliação ε deve ser provida antes da execução do *workflow*. A função pode ser uma expressão lógica ou um programa ou script customizado definido pelo cientista. Por exemplo, considere que as relações de saída T_{True} e T_{False} possuem esquema $\mathcal{T}_{True}$ e $\mathcal{T}_{False}$, respectivamente. A saída possui um esquema $\mathcal{Q} = \{f_j, \ldots, f_m\} \subseteq \mathcal{T}_{True} \cup \mathcal{T}_{False}$. Logo:

$$\varepsilon = \bigwedge_{p_i \in P} p_i \ , P = \{p_i \mid p_i = f_j \ \theta \ Valor\} \ 1 \leq i \leq n$$

onde $\theta \in \{ =, <, \neq, \leq, >, \geq \}$ e *Valor* é comparável com f_j. Como o resultado de ε é verdadeiro ou falso, a etapa de avaliação marca a saída com o sinalizador correto. Se ε for um programa ou script, ele deve ser capaz de ler a saída da ativação e produzir um resultado booleano. Baseado na sinalização Booleana, é possível inserir os resultados na relação de saída correta.

5.3 Algoritmos de Interação

A condução dinâmica de *workflows* está fortemente relacionada a como os cientistas interagem com os dados. Estes dados podem ser os dados de entrada que estão explorando ou os parâmetros configurados no *workflow*. Os dados de proveniência também podem ser analisados para auxiliar os cientistas a tomarem decisão a respeito da interação. Para conduzir a execução do *workflow*, cientistas precisam monitorar e analisar dados disponíveis e decidir se devem interferir ou não no *workflow* em tempo de execução. Nós consideremos que a condução dinâmica deve ocorrer no contexto de um ciclo dinâmico. Isto não restringe o nosso modelo dado que qualquer *workflow* pode ser envolvido por um ciclo dinâmico para possibilitar a condução dinâmica.

5.3.1 Tipos de Interação

Consideramos dois tipos de interações no *workflow*: (s^{α}) para mudanças feitas em átomos-α; e (s^{ω}) para mudanças feitas no conjunto ω de atributos das atividades. O tipo s^{α} ocorre quando cientistas trocam os dados de entrada ou modificam seu conteúdo, tal como mudar uma combinação de parâmetros que estão explorando. O tipo s^{ω} ocorre quando cientistas querem ajustar um atributo da operação associada à atividade, como o atributo de agregação de uma atividade associada à operação *Reduce* ou a definição da função de avaliação (ε) de uma atividade associada à operação *Evaluate*. Interações s^{α} afetam somente os dados modificados enquanto interações s^{ω} afetam o fluxo de dados por completo.

5.3.2 Algoritmo Interação-Alfa

Nós denotamos uma interação $s^{\alpha} < \alpha_{old}, \alpha_{new}, k_s, \delta >$ como uma interação que modifica um átomo-α α_{old} para α_{new} na iteração k_s. O atributo opcional de *profundidade* (δ | δ > 0) indica o número de iterações pelas quais a interação deve ser mantida. O valor padrão para δ é infinito (δ = +∞), o que significa que a interação é permanente. Todas as interações são eventos assíncronos mantidos em uma lista *S*. O Algoritmo 1 mostra como os eventos s^{α} são tratados no modelo de execução. Consideremos que uma iteração k nunca é incrementada antes que todos os $s^{\alpha} \in S$ ocorridos naquela iteração tenham sidos tratados. Se s^{α} ocorreu na iteração atual ($k = k_s$), o algoritmo suspende a linhagem de α_{old} (linha 4). Quando a linhagem é suspensa, todas as ativações em execução com átomos daquela linhagem terminam sua execução. Entretanto, novas ativações da linhagem de α_{old} não são criadas. Se a linhagem do átomo-α α_{new} não existir no cubo, o algoritmo cria uma nova linhagem (linha 9). O algoritmo também verifica se a profundidade (δ) está usando o valor padrão infinito. Caso verdadeiro, ele remove a interação da lista por que ele é considerada permanente (linha 11). Por questões de proveniência, uma lista de todas as interações ocorridas é mantida. Se a interação não é permanente, após δ iterações o algoritmo reverte a interação, suspendendo a linhagem α_{new} (linha 14) e reiniciando a linhagem α_{old} (linha 16). Este tipo de recurso é importante porque cientistas podem não ter certeza se uma dada mudança vai melhorar os resultados. Portanto, eles avaliam uma modificação por algumas iterações e comparam os resultados. Se eles gostarem das mudanças, eles

podem torná-las permanentes. Se o processo já tiver sido revertido, eles precisam apenas realizar uma nova interação s^{α} e o algoritmo irá reiniciar a linhagem α_{new}. Interações s^{α} também pode ser utilizadas para adicionar novos dados de entrada no *workflow* se um valor nulo for usado para α_{old}.

Algoritmo 1 **Alfa Steering**

```
1   for each s^α in S
2     if k = k_s then
3       call cube.getLineage with α_old, k returning λ
4       set status of λ to suspended
5       if cube lineage (α_new, k) exists then
6         call cube.getLineage with α_new, k returning λ
7         set status of λ to running
8       else
9         call cube.addLineage with α_new, k
10      if δ = +∞ then
11        call removeFromS with s^α
12    if k = k_s + | δ | then
13      call cube.getLineage with α_new, k returning λ
14      set status of λ to suspended
15      call cube.getLineage with α_old, k_s returning λ
16      set status of λ to running
17      call removeFromS with s^α
```

5.3.3 Algoritmo Interação-Omega

Denotamos uma interação $s^{\omega} < \omega_{old}, \omega_{new}, k_s, \delta >$ como uma interação que altera um conjunto de atributos ω_{old} do *workflow W* para um novo conjunto ω_{new} durante a iteração k_s. O conjunto ω pode conter diversos atributos das atividades do *workflow* e uma mudança em qualquer atributo de ω produz uma interação s^{ω} com o novo ω. Por predefinição, o valor da profundidade δ é +∞, o que significa que a interação s^{ω} é permanente. Entretanto, diferente das interações s^{α}, nas interações s^{ω}, δ pode assumir valores negativos (interações para trás) ou valores positivos (interações para frente). Como interações s^{ω} afetam a configuração do *workflow*, cientistas podem querer explorar a nova configuração começando por uma iteração passada. Portanto, quando δ < 0, a interação realiza uma reversão (*rollback*) da execução do *workflow*. Como é

importante manter os registros de todas as explorações do *workflow*, reversões, na verdade, criam ramos (*branches*) nas linhagens. O Algoritmo 2 descreve como eventos s^{ω} são tratados. Quando uma interação s^{ω} ocorre ($k = k_s$), o algoritmo verifica se já existe um ramo suspenso com a dada configuração ω_{new} para o *workflow W* (linha 4). Se um ramo como a nova configuração ω_{new} já existe, mas está suspenso, ele se torna o ramo ativo (line 6) através da sub-rotina *switchBranch* (line 25). Do contrário, ele muda a configuração do *workflow* (linha 8) e cria um novo ramo. Se s^{ω} é uma interação para trás (linha 9), o algoritmo cria o novo ramo baseado nas linhagens passadas de $k_{rollback}$ (linha 10). Se s^{ω} é uma interação para frente, ele cria um ramo baseado nas atuais linhagens. Interações para frente com δ não infinitos devem ser revertidas após k_s + δ iterações (linha 15). O algoritmo faz a reversão chamando a sub-rotina *switchBranch* utilizando ω_{new} como a atual configuração e ω_{old} como a nova.

Algoritmo 2 **Omega Steering**

```
1   for each s^ω in S
2       if δ = 0 set δ to +∞ end if
3       if k = k_s then
4           if λ_suspended[ω_new] of W exists then
5               obtain ω from W
6               call switchBranch with ω, ω_new, k
7           else
8               set ω of W to ω_new
9               if δ < 0 then
10                  set k_rollback to k + δ
11                  call branch with ω_old, k_rollback
12                  call removeFromS with s^ω
13              else
14                  call branch with ω_old, k
15      if k = k_s + δ then
16          call switchBranch with ω_new, ω_old, k
17          call removeFromS with s^ω
18  subroutine branch (ω_old, k)
19      call cube.getLineages with k returning lineages
20      set λ_suspended[ω_old] of W to lineages
21      for each λ in lineages
```

```
22          set status of λ to suspended
23          call cube.copyLineage with λ returning λ_new
24          set status of λ_new to running
25  subroutine switchBranch (ω_current, ω_new, k)
26      call cube.getLineages with k returning lineages
27      set λ_suspended [ω_current] of W to lineages
28      for each λ in lineages
29          set status of λ to suspended
30      obtain lineages from λ_suspended [ω_new] of W
31      for each λ in lineages
32          set status of λ to running
33      set λ_suspended [ω_new] to null
```

Estes dois algoritmos são projetados para apoiar a condução dinâmica de *workflows*. Eles também são capazes de funcionar em conjunto de forma que ambas as interações possam ser feitas durante a execução. Cientistas podem ramificar a execução do *workflow* para uma nova configuração, e então adicionar novos dados para serem explorados, e em seguida, realizar novas ramificações ou retornar para uma configuração anterior. A combinação das iterações centradas em dados como nosso modelo de execução, as ativações de avaliação e o modelo de linhagens dá apoio aos ciclos dinâmicos, os quais acreditamos serem importantes para a definição e a execução de *workflows* dinâmicos.

Capítulo 6 Implementação dos Ciclos Dinâmicos

Nós implementamos a iteração centrada em dados e nosso modelo de execução dinâmico no Chiron (Ogasawara *et al.* 2013). O Chiron é uma máquina de execução de *workflows* centrada em dados e projetada para a álgebra de *workflows* (Ogasawara *et al.* 2011). Portanto, ele representa dados de entrada e saída como relações e associa operações algébricas às atividades. Ele também implemente as quatro estratégias de execução discutidas na seção 5.1 para escalonar e distribuir ativações. Entretanto, originalmente o Chiron permitia somente a utilização de DAG como *workflows*. Para fazer com que o Chiron suportasse ciclos dinâmicos, desenvolvemos o novo operador *Evaluate*, as novas ativações de avaliação e fizemos mudanças necessárias em alguns métodos do modelo de execução como descrito na seção 6.1. Também modificamos o modelo de dados da proveniência no Chiron para apoiar a linhagem nas iterações como descrito na seção 6.2. Um módulo dentro do Chiron foi desenvolvido para dar suporte aos dois tipos de interações descritos na seção 5.3. As interações são inseridas no banco de dados de proveniência e tratados pelo Chiron em tempo de execução utilizando os algoritmos Interação-Alfa e Interação-Omega. Para definir e inserir os eventos de interação na proveniência do Chiron, o cientista pode utilizar a interface de programação de aplicação (API) *Achilles*, descrita na seção 6.3.4.

Da perspectiva do cientista, programar uma API pode não ser o ideal para interagir com o *workflow*. Além disso, a interação envolve a etapa de monitoramento da execução, a análise dos resultados parciais para, então, e a intervenção na execução do *workflow* (Mattoso *et al.* 2013). Para prover uma interface mais usual e amigável para o cientista e que, ao mesmo tempo, permeie as diversas etapas do processo interativo entre o usuário e o *workflow*, propomos também nesta tese a plataforma Proteus descrita na seção 6.3. A plataforma Proteus abriga um conjunto de ferramentas destinadas a tornar mais simples a composição, execução e análise de *workflows* científicos, permitindo também a interação com os *workflows*. Proteus é um portal Web pelo qual o cientista pode se conectar a uma base de proveniência e, através dela, interagir com seus *workflows* através de diversos módulos disponíveis. Estes módulos são aplicações Web fruto de trabalhos em parceria com alunos de mestrado e graduação do grupo de pesquisa da professora Marta Mattoso, dos quais alguns já renderam publicações (Horta *et al.* 2012, 2013; Santos *et al.* 2013).

6.1 Ciclos Dinâmicos no Chiron

A classe que implementa originalmente o escalonamento de *workflows* DAG no Chiron é chamada *EWorkflow*. Esta classe possui um método chamado *evaluateDependencies* que analisa, ao longo da execução, quais atividades do *workflow* estão prontas para executar, de acordo com a estratégia de execução adotada. Originalmente, esta classe implementa a abordagem proativa do modelo de execução. Para reproduzir o comportamento reativo descrito na seção 5.1, implementamos um novo modo de operação no Chiron.

Para construir o novo modo de operação sem desfazer o anterior, definimos uma interface Java chamada *CEngine* que estabelece um contrato, o qual, todas as classes que a estendem devem cumprir. No caso do Chiron, a *CEngine* obriga a todas as classes que a estendem possuir os métodos: *evaluateDependencies*, que atualiza o status da execução, e; o método de escalonamento chamado *iterateExecution*, que retorna um conjunto, possivelmente unitário, de ativações para execução. A classe *EWorkflow* tornou-se uma extensão da interface *CEngine*. Criamos uma outra classe chamada *EDataflow*, que também estende a *CEngine*. A classe *EDataflow* utiliza uma abordagem reativa para decidir quais ativações podem ser executadas, baseada no modelo de execução descrito na seção 5.1.

A Figura 11 apresenta o diagrama de classes simplificado, em UML (Fowler 2004), do Chiron. As classes destacadas em azul foram desenvolvidas para dar apoio a ciclos dinâmicos, embora outras classes originais do Chiron também tenham sofrido alterações. Quando dados são escritos em uma relação de saída, dispara-se um gatilho Tr_1 criando uma nova instância da classe *Trigger* na classe *EDataflow*. A classe *Trigger* associa os dados inseridos a uma dada atividade que pode ser disparada em função daquela inserção. O método *evaluateDependencies* processa os gatilhos de maneira assíncrona, realizando a ação de inserção nas relações, que propaga os dados da relação de saída para relação de entrada da atividade engatilhada. A inserção dos dados na relação de entrada dispara o gatilho Tr_2, que insere a atividade engatilhada *EActivity* em uma lista na classe *EDataflow* chamada *runnableActivities*. Quando os processos do Chiron requisitam novas ativações para consumir, se não houver mais ativações prontas para execução, a classe *EDataflow* instancia uma atividade da lista *runnableActivities* e chama o seu método *generateActivations*. Este método implementa a ação do gatilho

Tr_2, produzindo um conjunto de ativações baseadas nos dados inseridos na relação de entrada. Dependendo da operações associada à atividade engatilhada, a produção de ativações é diferente pois o tamanho do átomo pode variar. Portanto, a classe *EActivity* utiliza o método *generateActivations* da sua operação (*CActivity*) associada. Como a classe *CActivity* é abstrata, a implementação do método depende da instância real da classe, que pode ser um *Map*, um *SplitMap*, um *Reduce*, um *Filter*, um *MRQuery* ou um *Evaluate*.

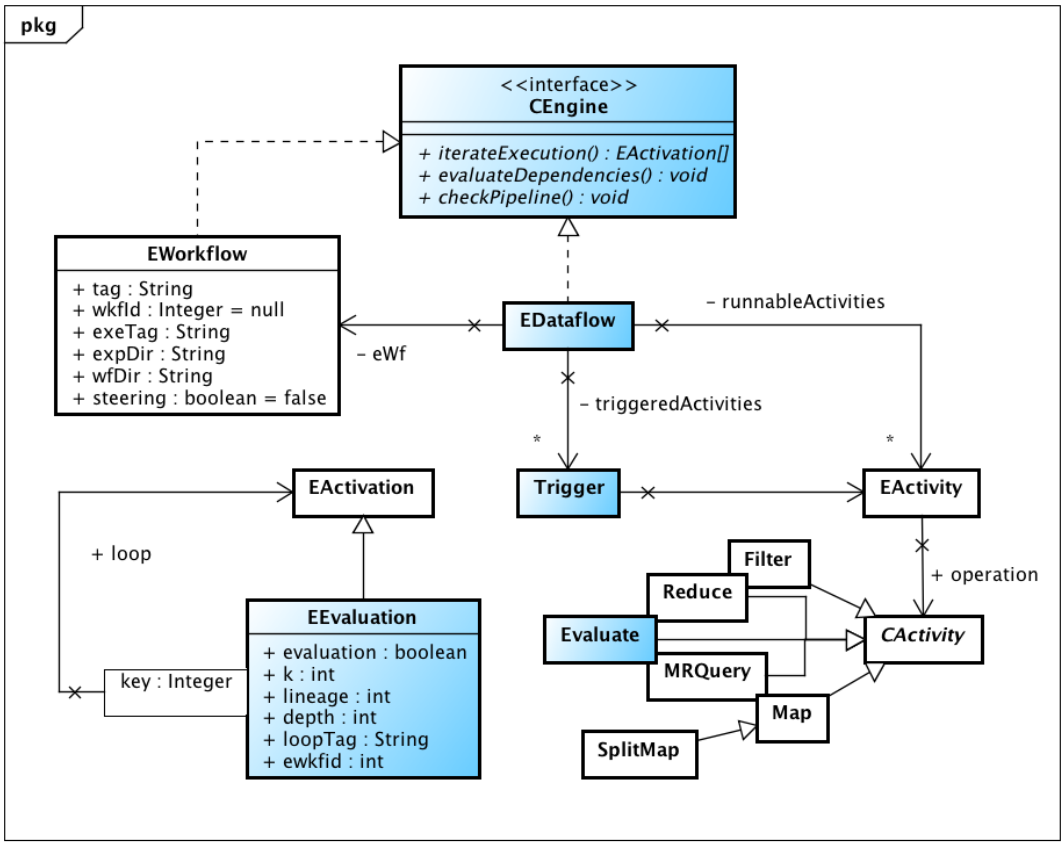

Figura 11. Diagrama simplificado de classes do Chiron com novas classes para ciclos dinâmicos destacadas em azul.

A classe *Evaluate* estende a classe CActivity, porém ela possui duas relações de entrada e duas relações de saída para possibilitar a definição de um subgrafo iterativo no *workflow*, conforme discutido no Capítulo 4. A cada vez que o método *generateActivations* da classe *Evaluate* é chamado, os dados das relações de entrada são unidos em uma relação intermediária $R_E = R_{init} \cup R_{loop}$ como descrito na propriedade (i) da Definição 5 do Capítulo 4. As ativações são geradas a partir dos átomos presentes na relação R_E. É importante ressaltar que uma atividade associada ao operador *Evaluate* é engatilhada sempre que um dado é inserido tanto em sua relação de entrada R_{init}

quanto em sua relação R_{loop}. Já a saída de uma atividade associada à operação *Evaluate* é feita em apenas uma de suas relações de saída (ou em T_{True} ou em T_{False}). Dessa forma, ela irá disparar apenas um gatilho Tr_1.

As atividades associadas à operação *Evaluate* possuem uma função de avaliação que dita se o conteúdo produzido deve ser interpretado como verdadeiro ou falso. Essa decisão direciona o fluxo de dados pela relação T_{True} ou pela relação T_{False}. Dessa forma, implementamos uma especialização da classe *EActivation* no Chiron chamada *EEvaluation*, como mostra a Figura 11. Essa classe adiciona a etapa de avaliação ao processo de ativação. Além disso, ela possibilita o pipeline de ciclos. Uma classe *EEvaluation* pode receber um átomo de entrada e a sequência de ativações que correspondem a um ciclo completo da iteração. Ela executa todas as ativações do ciclo e realiza a avaliação do resultado. Se a avaliação for verdadeira, ela reitera o ciclo criando dinamicamente um novo conjunto de ativações. O processo se repete até que a avaliação retorne falso. Neste momento, ela retorna toda a sequência de ativações como concluídas. Alternativamente, um atributo de profundidade (*depth*) é ajustado para designar por quantas iterações a classe *EEvaluation* fará o pipeline do ciclo. O ajuste da profundidade é importante para prevenir o escalonamento de ciclos infinitos se a definição do *workflow* estiver errada. Além disso, a profundidade é uma variável passível de otimização dinâmica de acordo com o tamanho do ciclo e tempo de execução de uma iteração.

Os ciclos dinâmicos no Chiron também incluem a implementação dos algoritmos Interação-Alfa e Interação-Ômega como descritos nas seções 5.3.2 e 5.3.3, respectivamente. Eles foram implementados nas classes *sAlfa* e *sOmega* apresentadas na Figura 12. Estas classes estendem a classe abstrata *SEvent* que guarda o conjunto comum de atributos para todos os eventos de interação, como a iteração k_s que o evento ocorreu e o valor da *profundidade* (delta, δ) do evento. Eventos *sAlfa* guardam a referência ao átomo alfa original (*alfaOld*) e a referência ao novo átomo alfa (*alfaNew*). Já os eventos *sOmega* registram a referência à configuração anterior do *workflow* (*omegaOld*) e à configuração nova (*omegaNew*). Durante a execução do *workflow*, a classe *ESteering* é instanciada para executar. Ela consulta a proveniência em busca de novos eventos de interação. Para cada evento registrado na proveniência, cria-se uma nova instância de *SEvent*, que pode ser uma classe *sAlfa* ou *sOmega* e dispara-se o algoritmo através do método *run*(). Se o valor da profundidade não for infinito, os

eventos são mantidos em memória na lista *runningEvents* para serem desfeitos após (k_s +δ) iterações.

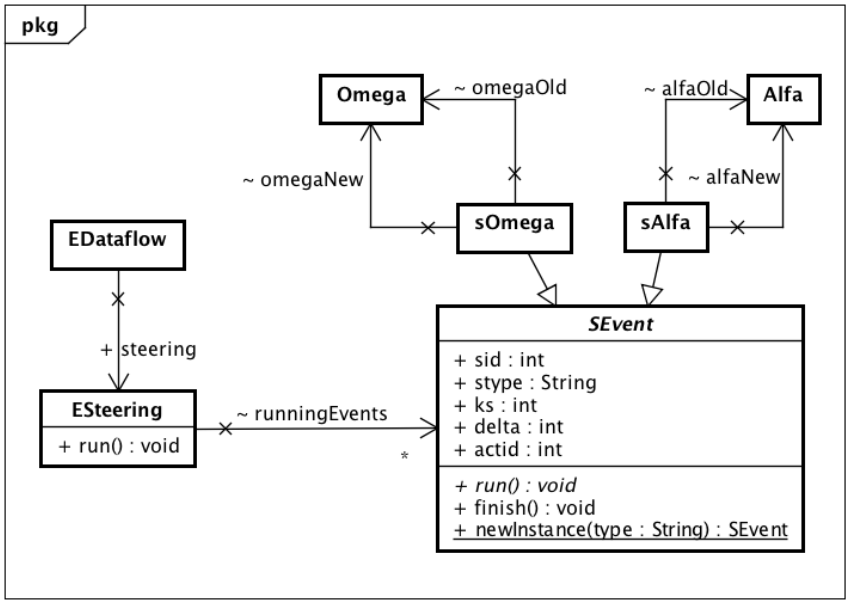

Figura 12. Diagrama de classes da implementação dos algoritmos de interação.

6.2 Modelo de Proveniência

A proveniência do Chiron é armazenada em um banco de dados relacional. Portanto, acrescentamos um conjunto de atributos e novas entidades para implementar os conceitos de linhagem e os eventos de interação. O conceito de linhagem só está presente em *workflows* dinâmicos, que contém iterações. Na prática, ele está vinculado a *workflows* que possuem alguma atividade associada à operação *Evaluate*. A relação R_{init} de uma atividade associada à operação *Evaluate* contém os átomos alfa da interação. Logo, quando os dados da relação R_{init} são propagados para a relação R_E, eles recebem uma chave única para designar sua linhagem e um atributo *k* que é igual a zero, inicialmente. Os valores da linhagem e do *k* são propagados como chaves estrangeiras para as demais relações pertencente ao ciclo do *workflow*. Ao final de uma iteração, quando um dado é propagado da relação R_{loop} para a relação R_E, o dado da linhagem é propagado como chave estrangeira, porém o valor de *k* é incrementado. Esse processo continua para um dado átomo alfa até que sua linhagem seja avaliada como falsa. Neste momento, a saída da atividade associada ao operador *Evaluate* é inserida na relação T_{false}, a qual não possui nem o registro da linhagem nem o de *k*, pois está fora do ciclo.

Na nossa atual implementação, o cubo de dados de linhagem está planificado no modelo relacional da proveniência do Chiron como exemplifica a Figura 13. Pode-se observar que cada relação do subgrafo iterativo do *workflow* (no exemplo, são as relações T1, T2 e T3) contém uma fatia do cubo, para todo átomo-α (λ) e para todo *k*. A chave única (λ) que designa a linhagem de um átomo-α da relação R_{init} é propagada para demais relações do subgrafo iterativo que um contém um átomo originado daquele átomo-α. Para construir o cubo de dados, é necessário realizar junções entre as tabelas agrupando as tuplas pela chave λ e pelo valor de *k*.

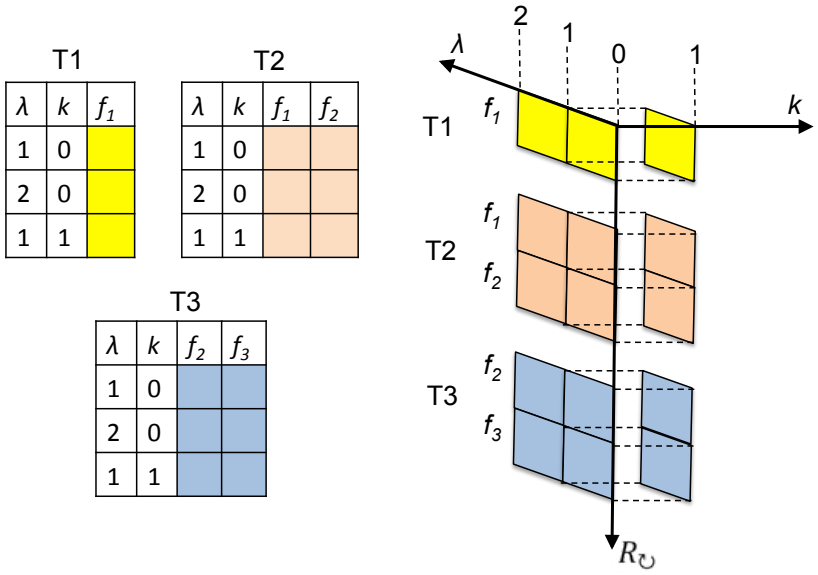

Figura 13. Exemplo da estrutura planificada do cubo no modelo relacional.

Embora o cubo de linhagens esteja planificado e segmentado nas relações do subgrafo iterativo do *workflow*, esta estrutura não dificulta as operações sobre o cubo. Na prática, o Chiron não precisa realizar a reconstrução completa do cubo durante a execução do *workflow*. As operações com o cubo presentes nos algoritmos Interação-Alfa e Interação-Beta operam sobre fatias do cubo e tiram proveito das operações comuns em bancos de dados relacionais tais como inserção, copia e atualização de dados. Entretanto, pode-se pensar na utilização de outras estruturas de bancos de dados que funcionem nativamente sobre cubo de dados, tais como SciDB (Cudre-Mauroux *et al.* 2009). A vantagem de manter o cubo planificado em um banco de dados relacional é minimizar o custo de armazenamento do cubo. Independente do cubo de linhagens, os dados de proveniência já são armazenados nas relações presentes no banco de dados. A estrutura proposta para o cubo apenas referencia/indexa esses dados utilizando a chave da linhagem (λ) e pelo valor de *k*. Portanto, o custo adicional do cubo pode ser

considerado desprezível no âmbito da proveniência. Mesmo quando ocorre uma interação ômega e parte dos dados do cubo são duplicados para a criação de um ramo na execução, podemos equiparar esse custo adicional de armazenamento ao custo de uma nova exploração do workflow, a qual o cientista teria que realizar manualmente em um ambiente sem apoio à interações. É importante destacar que, se o cientista realizar a nova exploração manualmente, ele pode reexecutar trechos do workflow e deixar de aproveitar dados já obtidos, aumentando o custo de armazenamento total do experimento.

Podemos estabelecer um paralelo entre os dados de entrada que o cientista deseja explorar e as linhagens. Sendo assim, digamos que o cientista deseja explorar um conjunto $\Lambda=\{\alpha_1, \alpha_2, ..., \alpha_n\}$ de átomos-α de entrada. Podemos imaginar uma função de custo $f(\alpha, \omega)$ que indica um custo computacional (custo de processamento numérico e custo de armazenamento, por exemplo) para processar toda a linhagem de um átomo-α ao longo das iterações de um workflow com uma dada configuração ω. Logo, o custo para processar todo o conjunto Λ será $F(\Lambda, \omega) = \sum_{i=1}^{n} f(\alpha_i, \omega) \mid \alpha_i \in \Lambda$. Se o cientista explorar um conjunto $\Omega=\{\omega_1, \omega_2, ..., \omega_m\}$ de configurações do workflow, o custo total do experimento será $F_E(\Lambda) = \sum_{j=1}^{m} F(\Lambda, \omega_j) | \omega_j \in \Omega$. Se o cientista executar o experimento com uma abordagem manual, o custo do experimento será, necessariamente, de pelo menos $F_E(\Lambda)$. Entretanto, ao utilizar a abordagem interativa, o cientista pode optar por realizar o mudança da configuração do workflow após o término do processamento de um conjunto de átomos-α. Dessa maneira, a interação ômega irá duplicar um conjunto Λ^* de átomos-α para ser processado na nova configuração ω_{new} tal que $|\Lambda^*| \leq |\Lambda|$. Portanto, podemos concluir, em teoria, que o custo da abordagem interativa é menor ou igual o custo da abordagem manual, ou seja, $F_E^*(\Lambda) \leq F_E(\Lambda)$.

Para permitir o registro dos eventos de interação com o *workflow*, estendemos o modelo de proveniência do Chiron para registrar eventos de interação em uma entidade denominada *esteering*, que registra as propriedades de um evento de interação, como o seu identificador único, o valor de k_s, o valor da profundidade (δ), o tipo do evento e o seu estado. Duas outras entidades (*ealfa* e *eomega*) registram características específicas dos eventos de interação alfa e ômega. O Chiron consulta a relação *esteering* para

buscar por novos eventos de interação. Se houver um evento novo, ele consulta as relações *ealfa* ou *eomega* para obter as propriedades detalhadas do evento.

6.3 Ciclos Dinâmicos na Plataforma Proteus

O Chiron é uma máquina de execução de *workflows*. Ele foi projetado para executar em ambientes remotos com disco compartilhado, tais como clusters. O SciCumulus (Oliveira *et al.* 2010a) tem o Chiron em seu núcleo e pode executar *workflows* na nuvem. Para utilizar o Chiron ou o SciCumulus, o usuário deve dispará-los através de uma interface de linha de comando. A definição do *workflow* e os parâmetros da execução são descritos em arquivos XML. Esta abordagem pode não ser a mais adequada para um cientista não habituado a ambientes de PAD, do ponto de vista da usabilidade. Este tipo de problema já foi tratado anteriormente em grades computacionais, com a utilização de portais (Romano *et al.* 2007; Jacob *et al.* 2009) para dar apoio a aplicações científicas ou submissão de *workflows*. Portais sempre foram aliados de ferramentas de processamento distribuído, para facilitar a execução de experimentos. Nguyen e Abramson (2012) extrapolam o conceito de portais científicos para tratar interatividade na execução de experimentos científicos baseados em *workflows*. Seguindo esta linha, acreditamos que um portal Web pode oferecer recursos para dar apoio a interações em todas as etapas do ciclo de vida do experimento. A interatividade com experimentos científicos pode exigir o monitoramento da execução, a análise dos resultados, da proveniência e a modificação de algum atributo do experimento (Mattoso *et al.* 2013). Porém, programar essas interações ou intervir na execução por meio de ferramentas de linha de comando pode ser pouco intuitivo para o cientista.

A plataforma *Proteus* foi projetada para abrigar, em um único ambiente, um conjunto de ferramentas necessárias para facilitar a condução dinâmica de experimentos pelo usuário. O objetivo da Proteus é oferecer ferramentas interativas para composição, execução e análise de experimentos científicos através de um portal Web. A Figura 14 ilustra a estrutura da Proteus. Ao conectar no portal, o cientista autentica no ambiente e estabelece uma sessão de usuário. A partir deste momento ele pode estabelecer conexões com suas bases de proveniência e carregar diferentes módulos. Na Proteus, as bases de proveniência estabelecem o elo entre o cientista e seus experimentos. Sob o comando do cientista, os módulos da Proteus interagem com as base de proveniência

para intervir na composição ou execução do experimento. Ademais, os módulos podem oferecer diferentes ferramentas para análise de resultados e da proveniência. O desenvolvimento da plataforma Proteus e dos seus módulos é resultado do trabalho de parceria entre alunos de iniciação científica, mestrado e doutorado do grupo da professora Marta Mattoso.

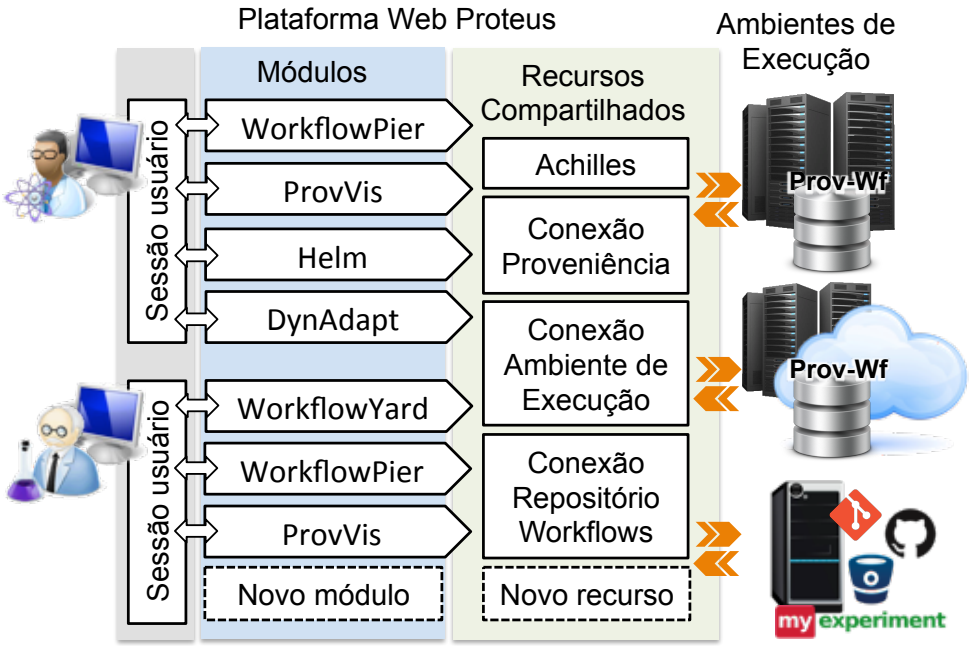

Figura 14. Estrutura da plataforma Proteus.

Os módulos para interação com experimentos científicos compartilham de diversas características. Eles requerem recursos e ferramentas para conectar com a base de proveniência, conectar com o ambiente remoto de execução, conectar com repositórios de *workflows* e intervir em um experimento. Por tal motivo, a plataforma Proteus possui um conjunto de recursos compartilhados que podem ser utilizados pelos módulos disponíveis para interagir com um experimento. A Figura 14 mostra os principais componentes compartilhados oferecidos pela Proteus. É importante destacar que os componentes de conexão com a proveniência são projetados para interagir com bases de proveniência modeladas seguindo o PROV-Wf (Costa *et al.* 2013). Nas seções seguintes iremos discutir os principais módulos em desenvolvimento para a plataforma Proteus. Os módulos *Workflow*Yard e *Workflow*Pier são responsáveis pela composição e disparo de *workflow*s, respectivamente, como descrito na seção 6.3.1. O Prov-Vis oferece recursos de visualização através da proveniência como descrito na seção 6.3.2. Já o DynAdapt oferece recursos para alterar a estrutura do *workflow* em tempo de execução através da troca de uma atividade do *workflow* por outra como descrito na

seção 6.3.3. A API Achilles descrita na seção 6.3.4 é a interface para definir e disparar eventos interativos durante a execução do Chiron. A Achilles é manipulada pelo módulo Helm descrito na seção 6.3.5.

6.3.1 Composição e Disparo de *Workflow*s

Ferramentas de composição e o disparo de *workflow*s são recursos disponíveis na grande maioria dos SGWfC (Altintas *et al.* 2004; Callahan *et al.* 2006; Hull *et al.* 2006). Eles oferecem ferramentas visuais semelhantes a ferramentas de programação visual (Mosconi e Porta 2000), onde o cientista arrasta componentes para uma área de trabalho e configura o fluxo de dados entre eles. Alguns SGWfC oferecem recursos para o controle de versão (Callahan *et al.* 2006) para registrar a evolução do experimento. Outra abordagem interessante para a reutilização de *workflow*s são as linhas de experimentos (Ogasawara *et al.* 2009b; Oliveira *et al.* 2010b) onde o experimento é modelo como um *workflow* conceitual. Cada atividade da linha de experimento pode possuir um conjunto de variabilidades que implementam aquela atividade. A partir da linha de experimento, o cientista pode derivar *workflow*s concretos fazendo a opção pelas variabilidades desejadas. A abordagem das linhas de experimento é complementar as abordagens existentes de SGWfC, uma vez que as linhas de experimento têm potencial para derivar *workflow*s para diferentes SGWfC.

A grande maioria das ferramentas de composição de *workflow* se preocupam com a modelagem da estrutura do *workflow*, *i.e.*, suas atividades e os enlaces de dados. Entretanto, um *workflow* projetado para executar em ambientes remotos como clusters e nuvens requer que outras propriedades e atributos sejam definidos para o *workflow*. Por exemplo, ao executar um *workflow* em um cluster ou nuvem, o usuário precisa definir o ambiente de trabalho, uma estrutura de diretórios que comporte a definição do seu *workflow*, com todas os programas necessários, bibliotecas e arquivos de configuração. Muitos ambientes de PAD utilizam gerenciadores de recursos e escalonadores como o PBS (Bayucan *et al.* 2000) ou o SGE (Gentzsch 2001). Estas ferramentas exigem a definição de um arquivo de configuração específico para a submissão de tarefas na fila de execução para utilizar os recursos distribuídos. Estes arquivos de configuração não pertencem à definição do *workflow* diretamente, mas estão associados ao experimento e precisam ser considerados na etapa de composição do seu ciclo devida. Estes arquivos e a estrutura de diretórios podem mudar ou sofrer ajustes, como por exemplo, a troca de

bibliotecas ou o aumento da demanda por mais recursos computacionais para o experimento. Para disparar um *workflow* em um ambiente de execução remoto e distribuído, a ferramenta de disparo precisa estar ciente de todo o conjunto de configurações e estrutura de diretórios do ambiente de destino, para que possa fazer a submissão corretamente.

Para gerenciar a configuração do *workflow* e do ambiente que ele requer para executar, podemos utilizar ferramentas de controle de versão como o Git (Loeliger e McCullough 2012). Um repositório Git pode atuar não somente como um repositório para registrar as versões do *workflow* e seu ambiente, mas também como um intermédio para implantar facilmente um novo ambiente de execução remoto. O cientista pode definir o *workflow*, o ambiente e as configurações em sua máquina local ou na interface Web e enviar as informações – estruturas de diretórios e arquivos – para o repositório. Quando ele decidir executar o *workflow*, bastará ir ao ambiente de execução e trazer as informações do repositório implantando-o no ambiente remoto para, então, submeter o *workflow* para execução. Ferramentas de gerência de configuração como o Git oferecem recursos para implantação de um novo repositório de forma simples e que pode ser facilmente automatizado.

No contexto da Proteus, utilizamos o Git como ferramenta para implantar *workflow*s em ambientes remotos. Como efeito colateral positivo, mantemos o ambiente do experimento sob um sistema de controle de versão. Vale a pena ressaltar que quando o cientista constrói o ambiente do experimento em sua máquina local ou no ambiente Web, ele não está configurando o ambiente para ser executado na máquina local, ele já realiza a configuração pensando no ambiente remoto. Se ele decidir, mais tarde, alterar o ambiente de execução, ele pode fazer ajustes no repositório local, enviar as alterações para o repositório como uma nova versão do experimento e, então, implantar no novo ambiente. Se ele tiver vários ambientes de execução (por ex. cluster e nuvem) ele pode manter os dois ambientes configurados no mesmo repositório e marcá-los com etiquetas (ou *tags*) no Git (Loeliger e McCullough 2012). Futuramente, além de repositórios Git, podemos estender a Proteus para interagir com outros repositórios para reutilização de *workflow*s como o myExperiment (De Roure e Goble 2007).

O módulo *WorkflowYard*, na Proteus, é o responsável pela composição do experimento, incluindo a definição do *workflow* e a definição do ambiente. O *Workflow*Yard ainda está em desenvolvimento e é um trabalho em conjunto com os

alunos de iniciação científica Lucas Carneiro e Kaique Rodrigues sob nossa orientação. No *WorkflowYard*, o cientista fornece o seu repositório Git destinado à configuração do ambiente do *workflow*. Em seguida, ele pode registrar as etiquetas do Git, ou uma determinada revisão do repositório como favoritos para execução do *workflow*. Além disso, o *WorkflowYard* permite a composição gráfica de *workflow*s seguindo a especificação algébrica do Chiron (Ogasawara *et al.* 2013). O usuário pode definir as atividades e as relações do *workflow* de forma interativa. Ao final do processo, ele pode exportar o arquivo XML do *workflow* ou registrá-lo na base de proveniência escolhida.

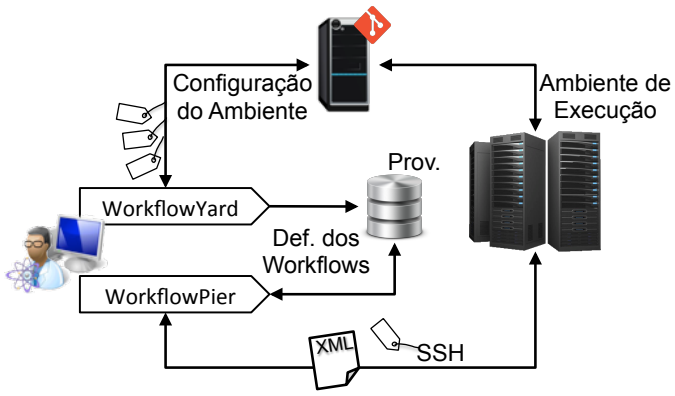

Figura 15. Funcionamento dos módulos de composição e disparo do workflow.

Após definir o *workflow* e configurar um ambiente de execução através do repositório, o cientista precisa disparar o *workflow* no ambiente remoto. No contexto da Proteus, o cientista pode utilizar o módulo *WorkflowPier*, também desenvolvido em conjunto com os alunos de iniciação científica Lucas Carneiro e Kaique Rodrigues sob nossa orientação. A Figura 15 ilustra o relacionamento entre os módulos *WorkflowYard* e o *WorkflowPier*. O *WorkflowPier* permite que o cientista cadastre ambientes de execução e configure uma forma de acesso (tal como SSH) para esse ambiente. O módulo também lista os *workflows* cadastrados na base de proveniência e as configurações do ambiente definidas como favoritas no *WorkflowYard* para aquele *workflow*. Dessa forma, o cientista pode selecionar uma determinada revisão do repositório de um *workflow* para implantar no ambiente de execução de destino. Através do *WorkflowPier*, o cientista também pode configurar um conjunto de parâmetros da execução do Chiron como: a estratégia de execução, o diretório no qual será implantado o *workflow*, o diretório onde deverá ser executado o experimento e uma etiqueta para identificar aquela execução, dentre outros. É através do *WorkflowPier* que o cientista também aponta os dados de entrada do *workflow*, *i.e.*, os arquivos que contém as relações de entrada. Ele também precisa apontar o comando ou script que dispara o

workflow no ambiente, por exemplo o comando para inserir o experimento na fila do cluster. Com essas informações o *WorkflowPier* pode se conectar ao ambiente de execução, implantar o repositório na revisão escolhida, adicionar um arquivo XML com a configuração da execução do Chiron e, então, disparar a execução. A partir do disparo, o cientista poderá acompanhar a execução através da proveniência do *workflow* utilizando o Prov-Vis.

6.3.2 Prov-Vis

A visualização pode dar a apoio a análises complexas de *workflow*s científicos. Cientistas precisam de ferramentas de navegação para examinar dados enriquecidos com proveniência de forma que possam realizar filtragens de dados e transferir apenas dados selecionados. Desta forma, eles podem transferir do ambiente de execução remoto apenas resultados relevantes, indo rapidamente ao conteúdo que eles precisam analisar. Acompanhar os resultados parciais da execução permite que os cientistas monitorem e analisem a execução do *workflow*. Sendo assim, cientistas podem ficar conscientes do atual estado do *workflow*; identificar e resolver problemas durante a execução; se recuperar de falhas e erros; reduzir o tempo gasto ao processar dados de baixa qualidade; e finalmente, cortar tempo de execução e reduzir gastos financeiros evitando transferir dados inúteis de ambientes remotos como clusters e nuvens.

Depois de coletar dados relevantes para análise, cientistas podem escolher a melhor ferramenta de visualização para cada caso. Existem diversos tipos de estruturas de dados para representar um resultado produzido. Alguns resultados também precisam ser agregados e consolidados antes da visualização. Portanto, mesmo depois de obter todas as saídas produzidas desejadas, os cientistas algumas vezes precisam manipular cada arquivo para formatá-lo e abri-lo em softwares de terceiros, movimentando os resultados para um ambiente de visualização. Esta tarefa é complexa e propensa a erros dado que um experimento pode ter uma vasta quantidade de resultados.

Para enfrentar as necessidades de visualização e condução dinâmica do experimento, o Prov-Vis (Horta *et al.* 2012, 2013) oferece um ambiente para a seleção e filtragem eficiente de dados, para posterior consolidação e transferência de dados relevantes necessários para análises preliminares. Esta solução é baseada em ações de visualização definidas pelo usuário tais como: (i) associar um comportamento a um tipo de arquivo; (ii) fazer uso de ferramentas específicas de visualização e estatística; (iii)

enriquecer resultados com dados de proveniência, e (iv) engatilhar alguma ação em um ambiente de visualização dedicado. O Prov-Vis é um trabalho em conjunto com o aluno de mestrado Felipe Horta (Horta *et al.* 2012, 2013).

Após transferir os dados de saída do experimento, cientistas podem precisar visualizar diversos resultados, buscando estabelecer comparações – principalmente em cenários de exploração de parâmetros – ou simplesmente analisar imagens altamente detalhadas e simulações enriquecidas com dados de proveniência. Em alguns casos, ambientes de visualização simples, restritos por telas comuns de computadores, não são suficientes para analisar dados tão sensíveis. Para casos como este, Prov-Vis utiliza tecnologias para painéis com múltiplas telas (do inglês, *tiled wall displays*) ampliando a experiência de visualização do usuário.

O Prov-Vis exibe dados de proveniência em uma aplicação Web interativa, que possibilita a navegação através dos dados em uma interface amigável e multi-plataforma. Além disso, o Prov-Vis faz-se valer de um serviço Web, uma aplicação remota que integra os resultados selecionados no cliente Web com um ambiente de visualização como os painéis com múltiplas telas. Esta aplicação remota provê uma interface que é acessada remotamente em tempo de execução e interpreta os resultados para serem visualizados – baseado na configuração do usuário – com softwares de terceiros para visualização. Os cientistas podem tirar proveitos de tecnologias disponíveis para painéis de visualização como o TACC Display Cluster (Texas Advanced Computing Center 2012), o SAGE (Byungil Jeong *et al.* 2010), CGLX (Doerr e Kuester 2011) e o Paraview (ParaView 2011).

6.3.3 DynAdapt

O DynAdapt (Santos *et al.* 2013) é uma ferramenta para apoiar mudanças na estrutura de *workflow*s em execução através da proveniência. Baseado em um conjunto de programas que podem ser utilizados para uma atividade do *workflow* (*i.e.* variabilidades), o DynAdapt escolhe o mais adequado utilizando um modelo de custo ponderado. Alternativamente, os cientistas podem interferir e optar por uma outra variabilidade para executar, caso eles não estejam satisfeitos com a atividade escolhida pelo escalonador do *workflow*. O DynAdapt é um trabalho em conjunto com o aluno de mestrado Igor Araújo dos Santos (Santos *et al.* 2013). O DynAdapt é projetado para ser independente de SGWfC, portanto sua arquitetura possui alguns componentes que se

sobrepõe com componentes disponíveis na Proteus. A Figura 16 mostra a arquitetura do DynAdapt, a qual é composta por quatro componentes: (i) o componente de composição, que é responsável por definir os parâmetros do DynAdapt e os programas alternativos que podem ser invocados (*i.e.* a variabilidade de uma atividade); (ii) Repositório de proveniência o qual contém informações a respeito da estrutura do *workflow* e suas execuções passadas que podem ser utilizadas como entrada para as mudanças estruturais; (iii) o componente de adaptação que é responsável por estabelecer uma interface com o a máquina de execução do *workflow* para realizar as mudanças e (iv) o componente rendez-vous que é responsável para prover proveniência em tempo real para o componente de adaptação quando ela não é oferecida nativamente pelo SGWfC.

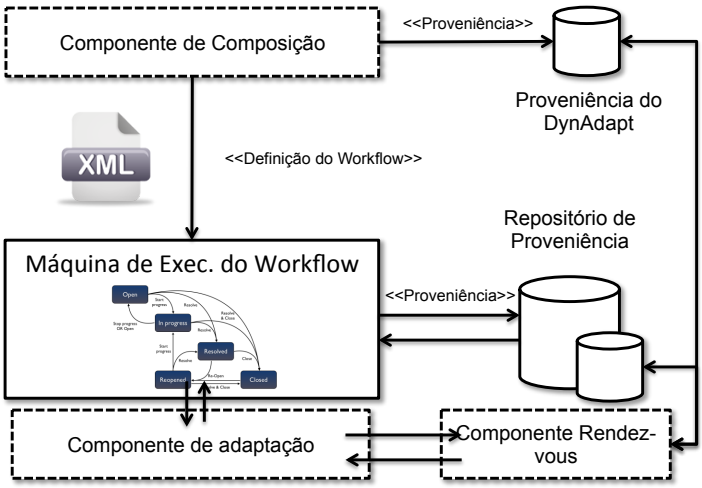

Figura 16. Arquitetura do DynAdapt.

Para ajudar a escolher o programa mais adequado que implementa uma atividade específica, o DynAdapt utiliza um modelo de custo ponderado que associa, para cada programa alternativo de uma atividade, um conjunto de fatores para serem avaliados. O tempo de execução, o custo financeiro e a qualidade dos resultados são exemplos de fatores que podem ser associados a um conjunto específicos de atividades alternativas. Baseado nestes fatores, o DynAdapt escolhe a melhor alternativa baseado em um dado critério. Para apresentar o modelo de custo ponderado, é importante apresentar aqui algum formalismo. Um *workflow* científico pode ser definido como um grafo $W(Y, Dep)$. O conjunto Y de vértices em W contém todas as atividades a serem executadas em paralelo e as arestas (Dep) são associadas a dependências de dados entre as atividades de Y. Desta forma, $Y = \{Y_1, Y_2, \ldots, Y_k\}$. Cada atividade Y_k pode ter um conjunto de programas alternativos $PA = \{pa_1, pa_2, \ldots, pa_m\}$. Cada atividade Y_k consome um

conjunto $R=\{R_1, ..., R_n\}$ de relações que seguem um esquema $\{\mathcal{R}_1, ..., \mathcal{R}_n\}$ e produz uma relação T com esquema $\mathcal{T}$. Desta forma, para cada relação de entrada R_{ki} de Y_k que segue um esquema $\mathcal{R}_{ki}$, $\forall\, pa \in \{pa_1, ..., pa_m\}$ tem uma relação de entrada R_x com esquema $\mathcal{R}_x$ de forma que $\mathcal{R}_x \subseteq \mathcal{R}_{ki}$. Isto significa que toda a variabilidade de uma atividade Y_k precisa ser capaz de consumir o mesmo conjunto de relações de entrada de Y_k.

O modelo de custo ponderado é inspirado no trabalho de Boeres *et al.* (2011) *apud* Oliveira *et al.* (2012). Cada programa alternativo é avaliado seguindo d fatores diferentes $F=\{f_1, f_2, ..., f_d\}$. Cada fator é associado com a função que calcula o valor v_d do fator. Desta forma, cada programa alternativo é associado com $\{v_1, ..., v_d\}$ valores, um para cada um dos d fatores. Cada fator também é associado a um peso p_d de forma que $\sum p_d = 1$. Para cada programa alternativo pa_x de uma atividade, temos que calcular um valor na forma $f(pa_x) = p_1.v_1 + p_2.v_2 + ... + p_d.v_d$. Como para cada fator é associado um peso p_d, quanto maior o valor de p_d, mais valorizado é o fator para calcular o valor de $f(pa_x)$. Sendo assim, os cientistas podem estabelecer um critério definindo pesos para cada atividade do *workflow*. Para uma mesma atividade Y_k, cada programa alternativo pode apresentar diferentes valores para os fatores, porém os pesos são os mesmos. Se os cientistas estão interessados em resultados de maior qualidade, por exemplo, eles simplesmente aumentam o peso deste fator, do contrário podem utilizar os valores default. A ideia é que os cientistas possam ajustar estes pesos tanto antes quanto durante a execução do *workflow*. Baseados nos valores dos fatores e seus pesos, o DynAdapt escolhe o programa mais apropriado para uma dada atividade.

O componente de composição é responsável para prover uma interface para cientistas definirem as variabilidades do *workflow* e ajustar o modelo de custo ponderado. O cientista pode conduzir a execução do *workflow* mudando os pesos e valores dos fatores da função de custo. Baseado no valor retornado pela função de custo, o componente de adaptação (detalhado a seguir) decide o programa mais adequado para ser escalonado. Alternativamente, cientistas podem escolher uma atividade arbitrariamente para substituir o programa atualmente sendo escalonado para execução. O componente de composição é executado primeiro, antes de instanciar o *workflow*. Ele modifica a definição do *workflow* acrescentando todos os metadados associados a possíveis mudanças tais como os programas alternativos e os fatores do modelo de custo de cada atividade. Existem duas formas de implementar o componente

de composição: (i) submeter um arquivo adicional de configuração do *workflow* ou (ii) estender o esquema do arquivo de definição do *workflow*.

A primeira solução é não intrusiva e requer um arquivo XML adicional para ser submetido em conjunto com a definição do *workflow*. O DynAdapt guarda uma referência para o SGWfC a ser usado, o *workflow* a ser executado e suas atividades. Para cada atividade, cientistas definem uma ou mais alternativas para serem invocadas, se necessário. Os cientistas precisam definir a linha de comando e o caminho de diretórios do programa. A segunda solução é intrusiva e requer mudanças no arquivo de definição do SGWfC. Isto só é possível quando os cientistas podem modificar a estrutura do arquivo de definição, como é o caso da Proteus.

O componente de adaptação é responsável por consultar os valores dos fatores e seus pesos e eleger o programa alternativo para ser executado. Quando o programa é escolhido, o agente de adaptação muda a estrutura das instâncias do *workflow* que estão em execução. O componente de adaptação é invocado quando um dos seguintes eventos é detectado: (i) um novo *workflow* é registrado ou atualizado no repositório, (ii) os cientistas chamam o DynAdapt explicitamente para mudar uma atividade arbitrariamente, ou (iii) novos pesos são registrados no base de proveniência. Se o cientista opta por uma dada atividade arbitrariamente, o agente de adaptação substitui a atividade em execução no plano atual pela atividade escolhida. Do contrário, para cada atividade alternativa, o componente de adaptação realiza o seguinte cálculo: $C = \sum_{\forall d} p_d . v_d$ tal que p_d é o peso e v_d é o valor do fator f_d. Este componente elege o programa alternativo que apresenta o melhor valor de *C*. É importante ressaltar que, como o DynAdapt utiliza a abordagem algébrica para *workflow*s (Ogasawara *et al.* 2011), quando o DynAdapt troca uma atividade, ele precisa garantir que a atividade alternativa pode consumir a mesma relação de entrada como esquema $\mathcal{R}$ e produzir uma saída com esquema $\mathcal{I}$. Isto significa que todos os programa alternativos podem usar o mesmo conjunto de parâmetros e escrever a mesma saída. Esta compatibilidade é verificada no nível de esquema. Ainda que isso pareça restritivo, podemos relaxar a definição atestando que, para uma dada atividade, se uma de suas atividades alternativa pa_x tem um esquema de entrada $\mathcal{K}$ e uma outra segue o esquema $\mathcal{G}$, nós podemos definir o esquema de entrada como $\mathcal{R} \supseteq \mathcal{K} \cup \mathcal{G}$. Para o esquema de saída $\mathcal{I}$, o argumento é análogo.

O DynAdapt precisa acessar os dados de proveniência do *workflow* em execução para realizar as mudanças no plano de execução. Entretanto, muitos SGWfC não permitem consultar a proveniência em tempo de execução. Portanto, o componente rendez-vous é um componente opcional responsável para prover proveniência em tempo real para o DynAdapt. Além disso, o componente também interage com o componente de adaptação para trocar informações sobre as mudanças feitas no plano de execução do *workflow*. No Proteus, a proveniência pode ser acessada em tempo real, logo, o componente rendez-vous é desnecessário.

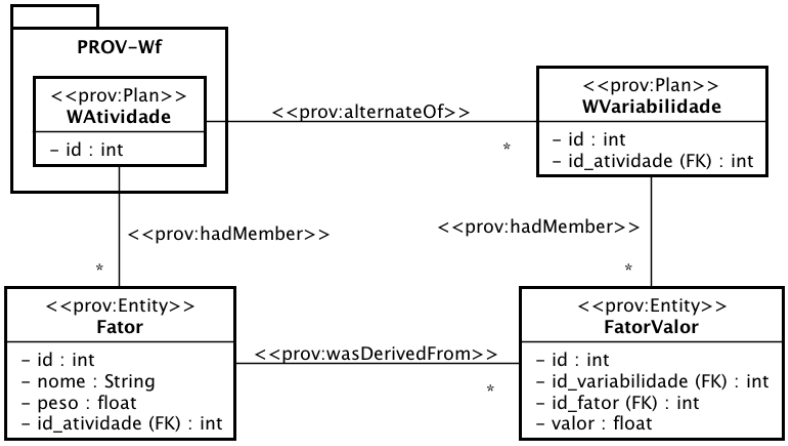

Figura 17. Fragmento da proveniência do DynAdapt.

O modelo de proveniência do DynAdapt é uma extensão do PROV-Wf (Costa *et al.* 2013). A extensão consiste na inclusão da entidade que presenta uma alternativa para uma atividade específica do *workflow*, chamada variabilidade, inspirado no conceito de linhas de produto de software e linhas de experimento (Northrop 2002; Ogasawara *et al.* 2009b). Entidades representando o modelo de fatores ponderado também foram incluídas. O modelo de proveniência do DynAdapt pode ser separado da base de proveniência do *workflow*. Na Proteus, utilizamos uma única base de proveniência. Na Figura 17, a entidade *WVariabilidade* representa um programa alternativo de uma atividade, *i.e.* uma implementação diferente de uma *WAtividade* do PROV-Wf. A entidade *Fator* é um aspecto que deve ser considerado durante a execução do *workflow*, tal como o tempo total de execução, qualidade dos dados gerados, etc. Um fator tem um peso associado, a função que calcula o seu valor e um nome. A entidade *FatorValor* representa a relação entre uma *WVariabilidade* e um *Fator* caracterizando um relacionamento através de um valor. Para cada fator associado com uma *WAtividade*, cada *WVariabilidade* é associada através de um *FatorValor* para estabelecer um ranking

de programas alternativos. Por exemplo, se uma atividade alternativa β tem o valor 8.1 para um valor fictício de "Qualidade", enquanto uma alternativa γ tem o valor 5.4, isto significa que β apresenta mais qualidade que γ.

6.3.4 Achilles

Para possibilitar a interação com *workflow*s durante a execução, definimos uma interface de programação de aplicação (API) denominada Achilles. O objetivo da Achilles é permitir a definição de eventos de interação α e ω. A Figura 18 apresenta o diagrama de classes UML da Achilles. A classe abstrata *SteeringEvent* representa um evento de interação entre o cientista e o *workflow*. Suas duas especializações, *Alfa* e *Omega*, representam interações do tipo α e ω, respectivamente. Uma interação do tipo α referencia um átomo-α original (*alfaOld*) e sua versão modificada após a interação (*alfaNew*). Um átomo mapeia um conjunto de campos e seus valores. Uma interação do tipo ω referencia um conjunto de operandos da configuração original do *workflow* (*omegaOld*) e a configuração nova (*omegaNew*).

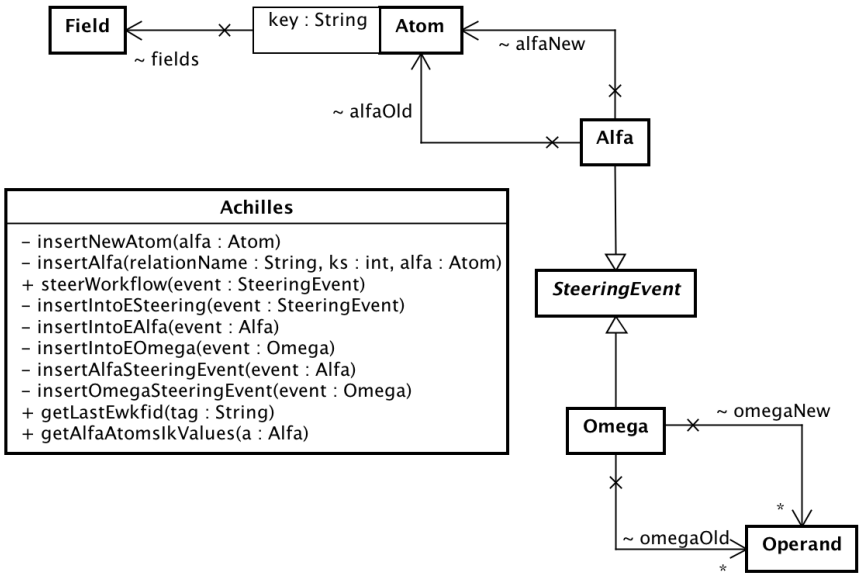

Figura 18. Diagrama de classes UML da API Achilles.

A classe Achilles é a classe principal da API e contém os principais métodos para definição de eventos interativos. O usuário deve criar uma instância de um evento de interação (SteeringEvent) e então submetê-lo através do método publico *steerWorkflow* da classe Achilles. Os métodos internos da Achilles inserem o evento na base de proveniência em um estado definido como pronto (*READY*, em inglês). Durante a execução, o Chiron captura estes eventos e os processa através dos algoritmos de

Interação-Alfa e Interação-Omega definidos na seção 5.3. A conexão com a base de proveniência é feita pelo componente de conexão na Proteus. Alternativamente, a conexão pode ser definida no construtor da classe Achilles.

6.3.5 Helm

Como a Achilles é uma interface de programação, ela pode não ser a melhor ferramenta para um cientista utilizar ao realizar a interação com os seus *workflow*s. Desta forma, o módulo *Helm* tem como objetivo oferecer uma interface Web para definição de eventos interativos. O *Helm* ainda está em desenvolvimento e é um trabalho em conjunto com o aluno de graduação Fernando Pinheiro do grupo da Professora Marta Mattoso.

Ao abrir o *Helm* na Proteus, o cientista escolhe o *workflow* e a instância deste *workflow* em execução com a qual ele deseja interagir. Selecionado o *workflow*, ele pode decidir entre duas opções: (i) alterar os dados de entrada do *workflow* ou (ii) alterar a configuração do *workflow*. Se ele desejar alterar os dados de entrada, o Helm exibe o conjunto de átomos-α do *workflow* para o cientista. Em seguida, o cientista pode editar o átomo-α ou adicionar novos átomos. Ao final das edições o cientista pode salvar as edições. Neste momento, o *Helm* cria os eventos de interação na Achilles e os submete. Se o cientista optar pela segunda opção, o *Helm* exibe a lista de operandos do *workflow* que podem ser editados. Novamente, o cientista pode modificar os operando e salvar as edições. O *Helm* cria os eventos de interação e o submete através da Achilles. É importante ressaltar que o *Helm* não exige que o cientista conheça a especificação de eventos α e ω ou dos algoritmos Interação-Alfa e Interação-Omega. O cientista interage com os dados de entrada ou com a configuração de *workflow* de forma transparente e o *Helm* submete os eventos para serem tratados pelo Chiron em tempo de execução.

Capítulo 7 Estudos de Caso: *Workflow*s Interativos

Este capítulo apresenta um conjunto de *workflows* que utilizamos como estudos de caso na avaliação experimental desta tese. Todos estes *workflows* apresentam características dinâmicas, ou seja, que demandam ajustes ao longo de sua execução. Na seção 7.1 apresentamos o *workflow* Lanczos (Dias *et al.* 2011) que implementa um conhecido algoritmo para cálculo de autovalores e autovetores. O algoritmo de Lanczos é iterativo e o número de iterações pode ser ajustado de acordo com a necessidade do cientista. Na seção 7.2 discutimos o *workflow* SciPhy (Ocaña *et al.* 2011b) para análises filogenéticas. No SciPhy, o cientista pode precisar alterar o método de alinhamento múltiplo de sequências dinamicamente durante a execução. Na seção 7.3, apresentamos um *workflow* de quantificação de incertezas (QI). Neste o *workflow*, a confiabilidade desejada nos resultados pode variar, afetando também o número de iterações do experimento.

7.1 *Workflow* Lanczos

Antes de implementar os ciclos dinâmicos, realizamos uma avaliação preliminar do potencial da adaptabilidade em métodos iterativos. Queríamos avaliar se o resultado de um método iterativo poderia ser melhorado se o usuário intervisse na solução do problema enquanto o *workflow* era executado. Para tal, exploramos uma prova de conceito utilizando um algoritmo iterativo envolvendo subespaços de Krylov para cálculo de autovalor. Nosso exemplo é baseado em um cenário de modelo de ordem reduzida (MOR). Como explicado na seção 3.6., MOR usa uma representação matemática mais simples que captura características dominantes do problema original para resolvê-lo mais rapidamente. Para reduzir o problema para um modelo mais simples, o método de redução tipicamente substitui um conjunto de equações por outro com menos incógnitas, utilizando uma transformação de coordenadas. Um método de redução muito comum utiliza o algoritmo de Lanczos (Coutinho *et al.* 1989) para gerar coordenadas de transformação. O algoritmo de Lanczos resolve um problema de autovalor para uma dada matriz simétrica e o algoritmo é muito popular pois ele converge rapidamente para os autovalores aproximados em ambos os extremos do espectro da matriz (Heath 2002). Como a maioria dos problemas não precisam de todos os autovalores de uma matriz, eles podem utilizar, normalmente, uma sequência

truncada interrompendo o algoritmo de Lanczos em uma dada iteração. Para otimizar a execução do algoritmo de Lanzos, cientistas precisam avaliar quando interromper a iteração do Lanczos. Uma medida de erro é normalmente utilizada para fazer essa avaliação (Coutinho *et al.* 1989).

A Figura 19 representa o *workflow* científico para o algoritmo de Lanczos. Ele representa uma varredura de parâmetros dinâmica que executa o *workflow* para *n* matrizes de entrada, resolvendo, portanto, *n* problemas de autovalor simétricos diferentes. A primeira atividade carrega a matriz M_1 para ser consumida pela atividade 'Núcleo Lanczos'. A atividade 'Carrega Matriz' lê as matrizes armazenadas em uma estrutura de dados adequada. A atividade 'Núcleo Lanczos' é equivalente a uma única iteração do algoritmo como descrito em Heath et al. (2002). A cada iteração *i*, o núcleo produz dois parâmetros de saída: α_i e β_i. Estes parâmetros são utilizados para compor a matriz quadrada tridiagonal T_i onde $\alpha_1, \alpha_2,\ldots,\alpha_i$ compõem a diagonal principal, e $\beta_1,\beta_2,\ldots,\beta_{i-1}$ compõem a subdiagonal e superdiagonal de T_i. Durante o processo iterativo, T_i torna-se maior conforme novos valores de α e β são calculados pelo núcleo do algoritmo de Lanczos. A próxima atividade do *workflow* utiliza T_i para calcular os autovalores aproximados, também chamados de valores de Ritz para aquela iteração. Baseado nestes resultados, é possível estimar um erro, ou seja, quanto dos atuais valores de Ritz desviam do valor real dos autovalores da matriz M de entrada. Esta estimativa é feita pela última atividade do *workflow*.

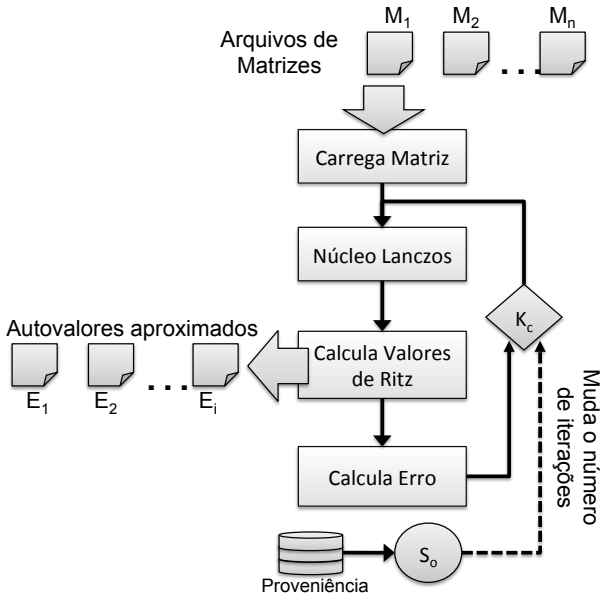

Figura 19. O *Workflow* Lanczos (Dias *et al.* 2011).

Este *workflow* não utilizou a abordagem iterativa do Chiron. Portanto, ele possui estruturas de controle externas que, na prática, instanciam o Chiron repetidas vezes. A estrutura de controle K_C controla as repetidas execuções do Chiron seguindo um parâmetro fixo do número de iterações necessárias para o método. O ponto de interação S_0 é outra estrutura de controle que checa a variação do erro na base de proveniência e, baseado neste valor, pode ajustar o parâmetro armazenado em K_C para um valor ótimo. Por exemplo, depois de consumir vinte matrizes 100×100 por 100 iterações (produzindo, portanto todos os autovalores das matrizes), o ponto de interação S_0 pode engatilhar uma regra que decide, baseado no erro, que o algoritmo já produz resultados satisfatórios depois de 10 iterações. Sendo assim, o número de iterações pode ser ajustado para 10. O *workflow* consome as matrizes restantes economizando 90 iterações por matriz. 10 iterações pode não ser o valor ótimo para todos os problemas de autovalor. Portanto, o número de iterações pode ser ajustado dinamicamente durante a execução.

Nossa prova de conceito avalia o tempo economizado ao utilizar a abordagem ajustando o número de iterações utilizando matrizes simétricas aleatórias. Baseado na literatura, é possível decidir quando truncar a sequência produzida pelo Lanczos (em torno de $\sqrt{N}$ para matrizes $N \times N$ (Heath 2002)). Entretanto, dependendo do problema, os cientistas podem precisar de maior confiabilidade ou simplesmente um número maior de autovalores, por exemplo.

7.2 *Workflow* SciPhy

Análises filogenéticas é um dos muitos campos da bioinformática que tem como objetivo comparar centenas de diferentes genomas para identificar a similaridade evolucionária entre diferentes organismos. Diversos tipos de aplicações da bioinformática são utilizadas neste campo, como o Alinhamento Múltiplo de Sequências (AMS) e Eleição de Modelo Evolucionário, os quais estão crescendo em escala e complexidade. Gerenciar experimentos filogenéticos está longe de ser trivial, um vez que suas atividades são intensivas computacionalmente e geram grandes quantidades de dados. Este tipo de experimentos podem ter o apoio de *workflows* científico. Especialmente em *workflows* de análises filogenéticas, cientistas realizam um conjunto específico de atividades para produzir uma série de árvores filogenéticas, as quais são

utilizadas para inferir relacionamentos evolucionários entre genes de espécies divergentes.

SciPhy (Ocaña *et al.* 2011b) é um exemplo de *workflow* de análise filogenética. O SciPhy realiza uma varredura de parâmetros onde todas as atividades são executadas para cada arquivo de um dado conjunto de entrada. O SciPhy possui quatro atividades: (1) alinhamento múltiplo de sequências (AMS), (2) conversão do alinhamento, (3) eleição do modelo evolucionário, e (4) construção das árvores filogenéticas. Estas atividades executam as seguintes ferramentas da bioinformática respectivamente: programas para AMS (MAFFT, Kalign, ClustalW, Muscle e ProbCons), Readseq, ModelGenerator, e RAxML. O SciPhy é apresentado resumidamente na Figura 20.

Uma execução paralela típica do SciPhy pode durar alguns dias em ambientes de execução paralela. Atualmente, a execução nestes ambientes, como nuvens de computadores, foca na otimização do escalonamento das atividades ou nas políticas de distribuição de dados, o que é essencial. Entretanto, baseado na condução do cientista, analisando resultados parciais do *workflow* em execução, mudanças dinâmicas do programas associados com as atividades podem ser feitas para melhorar o desempenho ou melhorar a qualidade dos resultados.

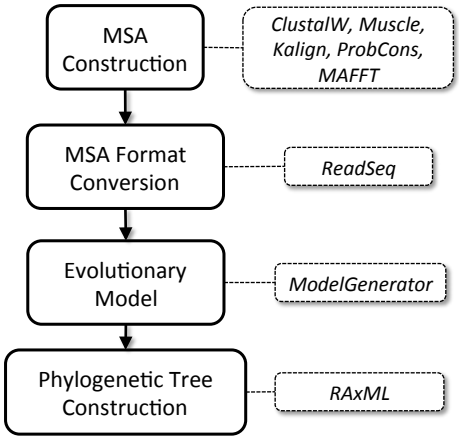

Figura 20. *Workflow* SciPhy como definido em Ocaña et. al. (2011b)

Por exmplo, para a atividade de AMS, existem diversas implementações, tais como MAFFT, ProbCons, Muscle, ClustaW e Kalign. Estes programas podem ser considerados intercambiáveis, mas a qualidade de seus resultados depende do características dos dados. Cada alternativa para a atividade pode apresentar diferentes desempenho e qualidade dos resultados. Alguns *workflows* incluem todas as alternativas para o AMS na estrutura do *workflow* por que a decisão da melhor escolha não pode ser

feita antes da execução. Entretanto, durante a execução, baseado na análise de resultados parciais, o cientista pode perceber que a alternativa atual não é adequada para esse conjunto de dados. Não há propósito em continuar a usar tal programa, uma vez que o cientista pode ter uma ideia melhor de qual alternativa pode ser melhor, ao invés de executar todas os possíveis programas de AMS. Por exemplo, vamos considerar que a execução do SciPhy comece utilizando o MAFFT para a atividade de AMS. Após 12 horas de execução o cientista recebe uma notificação da máquina de execução do *workflow* dizendo que a execução irá demorar mais tempo do que o esperado. Alguma ação pode ser feita para atender às restrições do cientista. Pode ser interessante, permitir ao cientista trocar o programa da atividade de AMS (por ex. utilizar o ProbCons ao invés de usar o MAFFT, baseado na sugestão do cientista) para obter um alinhamento com uma melhor relação custo benefício. Nas abordagens existentes, quando o programa associado a uma atividade é escolhido, ele não pode ser alterado durante a execução do *workflow*.

7.3 Quantificação de Incertezas

Sistemas de engenharia complexos são normalmente baseados em modelos matemáticos sofisticados para executar experimentos computacionais de engenharia. Estes experimentos utilizam simulações numéricas para analisar cenários do mundo real. Quanto mais confiável for a simulação, maior a capacidade do experimento em avaliar o objeto de estudo. A confiabilidade de simulações numéricas pode ser medida por métodos de quantificação de incertezas (QI) (Guerra *et al.* 2012). A ideia geral dos métodos de QI é explorar um modelo utilizando entradas distribuídas de forma estocástica e, então, combinar um resultado médio e seu desvio padrão. Os resultados obtidos podem dat ao cientista a consciência da fragilidade do modelo explorado. O tamanho da exploração é associado a uma precisão pré-definida pelos cientistas. Para o método de colocação estocástica com grids esparsas (CEGE) (Xiu e Hesthaven 2005) que utilizamos, o tamanho da exploração cresce em níveis de interpolação. Quanto maior o nível de interpolação, maior é a exploração, e, consequentemente, mais instâncias da simulação precisam ser executadas.

A quantidade de dados produzidos em um único experimento cresce exponencialmente com o aumento do nível de interpolação como apresentado na Figura 21 (os eixos verticais estão em escala logarítmica). A figura mostra quatro cenários com

o tamanho do modelo de entrada variando de 1 milhão de tetraedros (1M) até 17 milhões (17M). Em todos os cenários, a simulação é configurada para armazenar apenas 1% dos resultados calculados. Estes cenários foram executados em um cluster com armazenamento dedicado, de alta velocidade e compartilhado entre os nós. Os resultados na Figura 21(a) consideram apenas duas variáveis (duas dimensões) com incertezas enquanto na Figura 21(b) o nível de interpolação é fixado no nível 5 e varia-se a dimensão do problema. Portanto, não é razoável iniciar o experimento com um alto valor para o nível de interpolação. Tipicamente, os cientistas iniciam o experimento com um nível de interpolação mais baixo. Se a confiabilidade dos resultados não atendem a um dado critério estipulado pelo cientista, ele executa o experimento novamente em um nível de interpolação mais alto. Este tipo de mudança poderia ser feito em tempo de execução se o experimento é modelado como um *workflow* dinâmico.

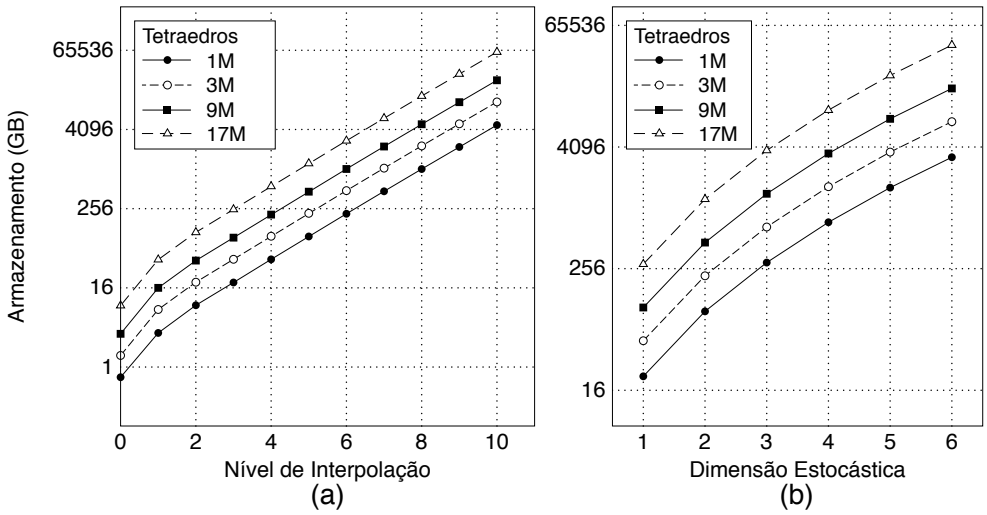

Figura 21. Crescimento do volume de dados em função do (a) nível de interpolação e da (b) dimensão do problema.

A Figura 22 mostra como os cientistas tipicamente realizam um experimento de QI modelado como um *workflow* dinâmico em alto nível. Eles começam criando a malha baseados em dados de geometria que representam um lugar ou objeto físico (Atividade 1). A Atividade 2 cria o conjunto de explorações, replicando a malha e atribuindo um conjunto de condições iniciais para cada exploração. A quantidade de réplicas está relacionada ao nível de interpolação inicial. Por exemplo, considerando duas variáveis com incertezas (i.e. duas dimensões), o nível de interpolação 5 requer 145 réplicas (chamadas pontos de colocação), enquanto o nível de interpolação 6 requer 321 pontos de colocação. As condições iniciais variam seguindo uma distribuição estocástica. A Atividade 3 particiona cada malha inicializada para ser consumida pelo

solver[4] paralelo na Atividade 4. Há uma execução do *solver* paralelo para cada malha de entrada. O *solver* simula um dado fenômeno e produz diversas saídas capturadas ao longo de um intervalo de tempo.

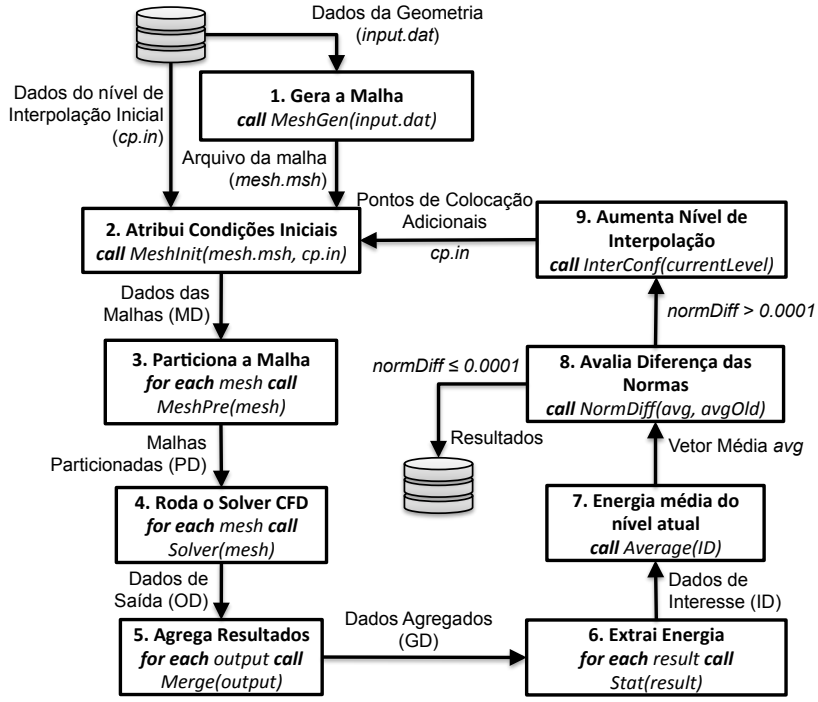

Figura 22. *Workflow* conceitual de Quantificação de Incertezas.

A Atividade 5 agrega a saída do *solver*, criando portanto um único arquivo de resultado. A Atividade 6 extrai o vetor de energia cinética de cada saída. Portanto, a saída da atividade 6 contém um vetor para cada ponto de colocação. A Atividade 7 calcula a média ponderada dos vetores, considerando todos os vetores obtidos até então. A Atividade 8 avalia a convergência da energia cinética calculando a diferença entre as normas do atual vetor médio e do vetor médio do nível de interpolação anterior (se houver um). Se a diferença de normas é menor que um limite, a iteração é interrompida por que o nível de interpolação atual é suficiente. Do contrário, a Atividade 9 aumenta o nível de interpolação, estabelecendo os pontos de colocação adicionais. Baseada nos novos pontos, a Atividade 2 inicializa novos réplicas da malha e a iteração continua.

[4] Solver, do inglês, é um termo genérico dado a um programa de computador ou biblioteca que soluciona um problema matemático. O solver CFD, por exemplo, soluciona um sistema de equações no decorrer do tempo com o intuito de modelar eventos da dinâmica de fluidos computacional (CFD).

Como caso de uso para um *workflow* de QI iterativo, nós apresentamos uma análise estocástica de um modelo de grande interesse para geologia submarina, conhecido como correntes de turbidez. Correntes de turbidez são tradicionalmente definidas como fluxos de sedimento por gravidade, onde sedimentos estão suspensos na água por turbulências no fluido. Este mecanismo de formação geológica é de grande importância na formação de reservatórios de petróleo ao longo do tempo geológico. Por esta razão, este problema é de grande interesse aos geólogos. Devido a comportamento estocástico das propriedades do fluido e as condições iniciais e de contorno, experimentos envolvendo correntes de turbidez precisam considerar uma análise estocástica como melhor mecanismo para analisar o comportamento do modelo.

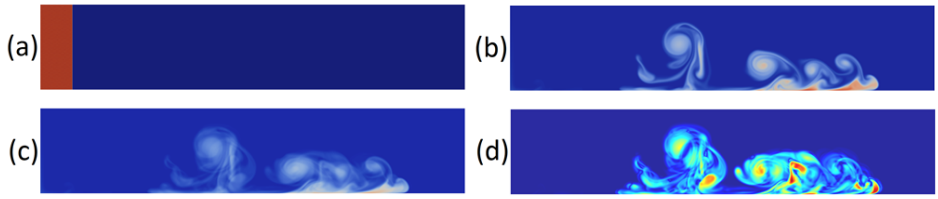

Figura 23. (a) Condição inicial heterogênea; (b) evolução da concentração; (c) média da concentração e (d) desvio padrão da concentração no final da execução.

Com este objetivo, um modelo de referência utilizado por cientistas é a configuração *lock-exchange*. Neste modelo, dois fluidos imiscíveis com diferentes densidades são confinados em uma caixa retangular tridimensional. No começo, ambos os fluidos estão em repouso e separados por uma barreira sólida como ilustra a Figura 23(a). Neste caso, a condição inicial do fluido pesado é considerada heterogênea e representada através de uma distribuição que depende de dois parâmetros estocásticos. Na Figura 23(b), podemos ver a evolução da corrente de turbidez para um par de parâmetros selecionado aleatoriamente. Já na Figura 23(c) e (d) podemos ver os momentos estatísticos, média e desvio padrão respectivamente, obtidos utilizando o método SC.

7.3.1 *Workflow* Manual

Para avaliar a abordagem dos ciclos dinâmicos, executamos duas versões do *workflow* de QI e discutimos as vantagens da abordagem dinâmica do ponto de vista do usuário. O primeiro *workflow* (manual) é um *workflow* tradicional não-iterativo como mostrado na Figura 24. O números apresentados nas atividades correspondem às atividades descritas no seção 7.3. Também mostramos as operações algébricas associadas a cada

atividade. O *workflow* recebe os parâmetros de geometria para a malha de entrada e um dado nível de interpolação inicial. O cientista executa o *workflow* manualmente para o nível especificado. Se os resultados não atenderem ao critério de confiabilidade, ele executa o *workflow* manual novamente com um nível de interpolação mais alto.

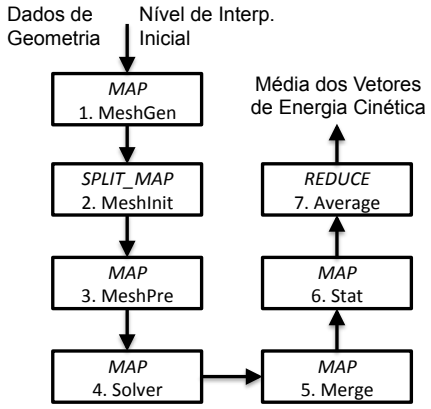

Figura 24. *Workflow* de QI manual

7.3.2 *Workflow* Dinâmico

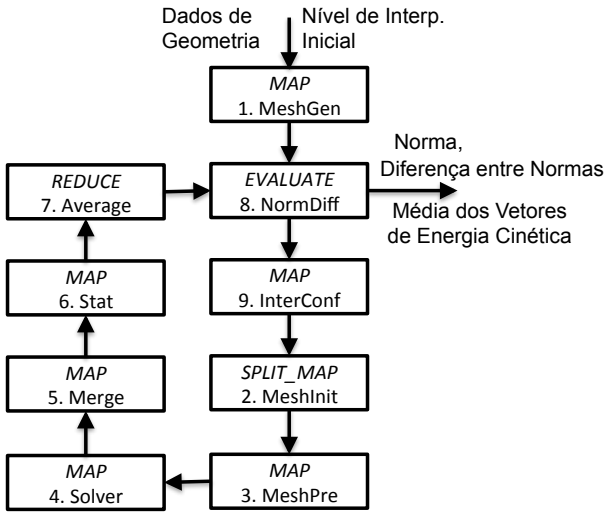

Figura 25. *Workflow* de QI iterativo (e dinâmico).

Uma versão alternativa (dinâmica) utiliza ciclos dinâmicos para explorar as características iterativas das análise de QI como apresentado na Figura 25. O *workflow* utiliza a operação *Evaluate* associada a uma atividade que verifica se a diferença entre as normas dos vetores obtidos é menor que o limiar definido inicialmente como 0.001. Com o *workflow* dinâmico, os cientistas não precisam executar o *workflow* manualmente diversas vezes. Eles iniciam o *workflow* com um dado nível de interpolação e o *workflow* aumenta o nível de interpolação automaticamente, se

necessário. Um aspecto importante do *workflow* de QI é que algumas explorações de um nível de interpolação podem ser reutilizadas pelo próximo nível de interpolação. Entretanto, como é muito trabalhoso configurar a próxima execução do *workflow* manual para utilizar resultados obtidos previamente, os cientistas normalmente preferem executar o *workflow* manual por completo. Utilizando o *workflow* dinâmico, fica mais simples reutilizar tais explorações por que elas já estão no contexto do *workflow* e são automaticamente referenciadas pela máquina de execução do *workflow*.

Capítulo 8 Avaliação Experimental

Este capítulo apresenta um conjunto de avaliações experimentais envolvendo a interação com *workflow*s científicos. Para avaliar a hipótese desta tese, na seção 8.1 avaliamos as melhorias obtidas no tempo de execução de alguns experimentos ao possibilitar interações entre o cientista e o *workflow*. Apresentamos uma abordagem inicial sem ciclos dinâmicos com o algoritmo de Lanczos, a execução do *workflow* SciPhy utilizando o DynAdapt e a avaliação dos ciclos dinâmicos com o *workflow* de quantificação de incertezas. Na seção 8.2 apresentamos uma análise a respeito da sobrecarga introduzida pela abordagem dinâmica quando comparada a abordagem manual sem ciclos dinâmicos. Na seção 8.3 discutimos as possibilidades de análises de proveniência e dos resultados utilizando o cubo de dados de linhagem como ferramenta. Na seção 8.4 avaliamos o comportamento dos algoritmos de interação implementados no Chiron em função do número de eventos ocorridos em um instante de tempo.

8.1 Economia do Tempo de Execução

A hipótese desta tese é que a interação dinâmica com *workflow*s traz melhorias no tempo de execução do experimento. Antes de elaborar a abordagem de ciclos dinâmicos, realizamos experimentos preliminares para justificar a proposta desta hipótese utilizando um algoritmo iterativo e a versão original do Chiron (*i.e.* sem ciclos dinâmicos) como descreve a seção 8.1.1. Para simular um comportamento iterativo no Chiron original, utilizamos um script de controle que executava o Chiron repetidas vezes em um ciclo com contagem e com dependências. Ao longo da execução, adaptamos a contagem do ciclo para valores ótimos. Na seção 8.1.2 avaliamos os ganhos ao trocar a estrutura do *workflow* dinamicamente utilizando DynAdapt e o *workflow* SciPhy no Chiron original. Na seção 8.1.3 utilizamos a abordagem de ciclos dinâmicos no Chiron para implementar um *workflow* de quantificação de incertezas que ajusta o valor do nível de interpolação automaticamente baseado na diferença entre a norma dos vetores médios da energia cinética entre as iterações do *workflow*. Exploramos o dinamismo na troca da função de avaliação ε do *workflow*.

8.1.1 *Workflow* Lanczos

Este estudo compara o tempo gasto para aplicar o algoritmo de Lanczos em um conjuntos de matrizes utilizando duas abordagens: (i) varredura de parâmetros tradicional (VP Tradicional) e (ii) varredura de parâmetros dinâmica (VP Dinâmica). Na VP tradicional, o algoritmo itera N vezes para encontrar todos os N autovalores das matrizes $N \times N$. Nós aplicamos ambas as abordagens em dois conjuntos de matrizes contendo 128 matrizes. O primeiro conjunto contém matrizes 256×256 (N=256) e o segundo conjunto possui matrizes 512×512 (N=512). No cenário de VP dinâmico, uma interação com *workflow* ocorre após o processamento de 16 matrizes iniciais do conjunto. O número 16 foi utilizado por que a execução foi feita em 16 núcleos de um cluster SGI Altix ICE 8200, com processadores Intel Xeon 5355 com frequência de 2.66GHz. Os resultados são apresentados na Figura 26. A interação economizou 75% do tempo no caso das matrizes de tamanho N=256 e 78% nas matrizes com tamanho N=512. No caso onde N=256, a interação reconfigurou o *workflow* para truncar a série produzida pelo algoritmo na 37ª iteração enquanto no caso onde N=512 decidiu-se truncar a série após 46 iterações do algoritmo. Estes bons resultados são esperados uma vez que o processo de interação economiza 219 iterações (256-37) para 112 matrizes (128-16) no caso onde N=256 e 466 iterações (512-46) para 112 matrizes no caso onde N=512.

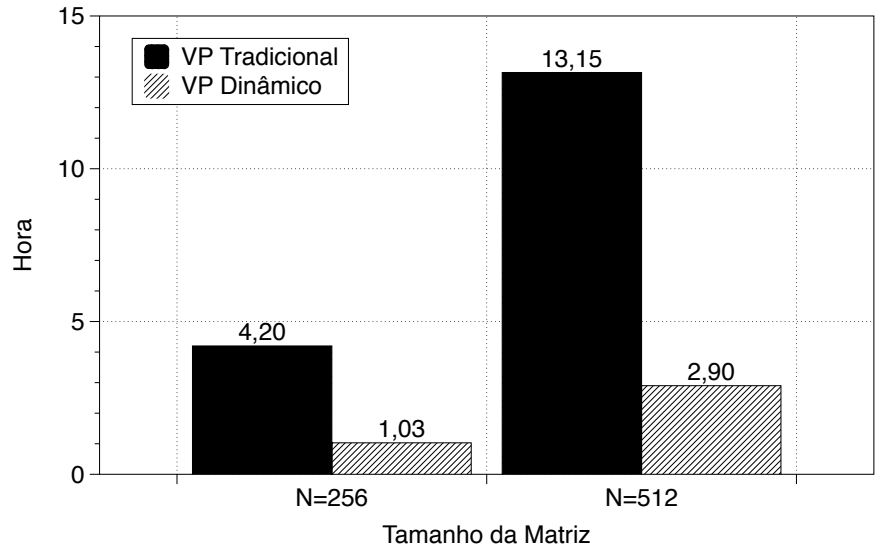

Figura 26. Economia do tempo de execução ao realizar interações com o workflow.

Em um cenário mais realístico, cientistas não executariam o algoritmo de Lanczos sem truncar a série, uma vez que é bem conhecido que ele não é eficiente neste

caso (Heath 2002). Mesmo assim, os cientistas podem escolher de forma arbitrária um número que ainda seja muito grande. Portanto, em um segundo estudo, utilizamos um conjunto de 128 matrizes com tamanho 1024×1024. Inicialmente, o algoritmo está definido para iterar 180 vezes. Entretanto, a interação pode refinar o valor após o consumo das primeiras 16 matrizes, da mesma forma que o primeiro estudo. Os resultados estão apresentados na Figura 27.

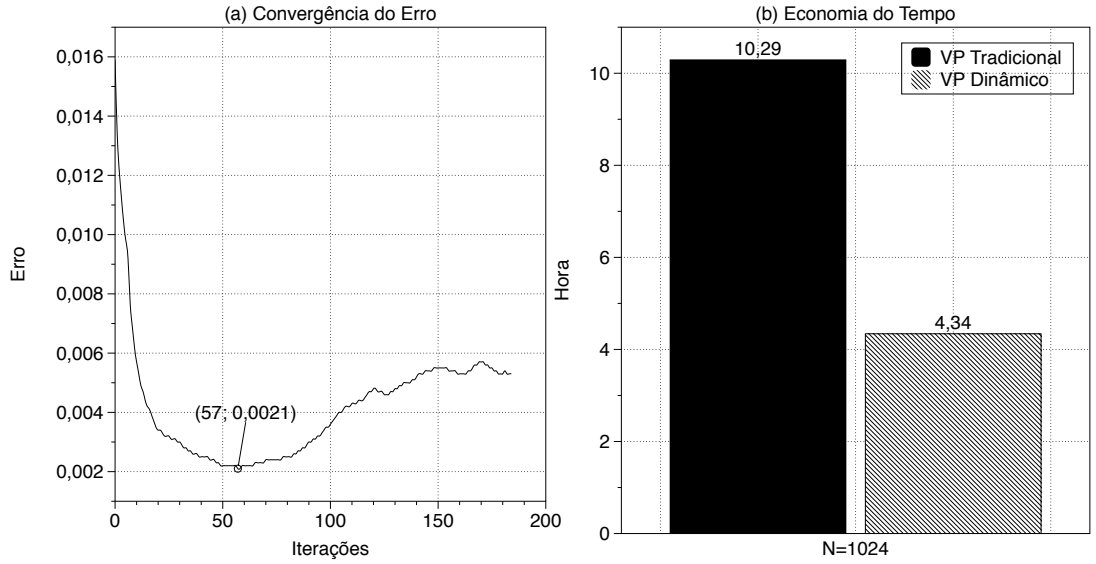

Figura 27. (a) Convergência do erro no segundo estudo e (b) o tempo economizado com o processo interativo.

A Figura 27(a) mostra a variação da média do erro das primeiras 16 matrizes durante 180 iterações. O erro está convergindo para algum lugar próximo de 0,005. Entretanto, os valores mais acurados dos autovalores são obtidos até a iteração 57, como ilustra a Figura 27(a). Sendo assim, o VP dinâmico altera para 57 o número de iterações. Na Figura 27(b) é possível verificar que o processo interativo salvou mais de 6 horas no tempo de processamento. Os resultados obtidos são positivos no sentido que experimentos de larga escala podem economizar tempo utilizando abordagens interativas, ajustando alguns parâmetros do *workflow* em tempo de execução.

8.1.2 *Workflow* SciPhy com o DynAdapt

Para realizar esta avaliação, nós utilizamos a máquina de execução de *workflow*s SciCumulus utilizando os serviços da Amazon AWS (Amazon EC2 2010). A Amazon oferece diferentes tipos de máquinas virtuais para execução (VM). Cada VM tem uma configuração diferente de memória e CPU. Neste estudo utilizamos um único tipo de instância: m1.large – 7.5 GB RAM, 850 GB de armazenamento e 2 núcleos de

processamento. Cada VM instanciada utiliza processadores Intel Xeon com quatro núcleos de processamento de 2.33GHz de frequência. Cada VM utiliza o sistema operacional Linux Cent OS 5 (64-bit) e foi configurada com os softwares e bibliotecas necessárias, tais como as aplicações que o *workflow* SciPhy utiliza, por exemplo ProbCons e ClustaW. A imagem virtual (AMI IDs: ami-6e1a8907) é armazenada na nuvem e o SciCumulus cria um cluster virtual para executar o experimento baseado nesta imagem. Em termos de software, todas as instâncias, não importando o seu tipo, possuem os mesmos programas e configurações.

Nós exploramos a variabilidade da atividade de AMS no SciPhy como descrito na seção 7.2. O SciPhy utiliza um conjunto de dados de arquivos multi-fasta como entrada contendo sequencias de proteínas extraídas da base de dados RefSeq versão 48. Este conjunto de dados possui 200 arquivos multi-fasta de aminoácidos e cada arquivo é constituído de uma média de 10 sequências. O programa MAFFT versão 6,857 foi registrado como programa inicial para a atividade de AMS e os seguintes programas foram registrados como alternativos: ClustalW v.2.1, Kalign v.1.04, Muscle v. 3.8.31, and ProbCons v. 1.12.

A ideia central do experimento apresentado nesta seção é alterar dinamicamente o programa que implementa a atividade AMS do *workflow* para obter melhor desempenho e reduzir os custos financeiros. Para este experimento, não consideramos o critério de qualidade, ainda que isso fosse relativamente simples uma vez que temos o parâmetro *e-value* na bioinformática que indica a qualidade dos resultados. Para a atividade AMS, nós associamos 3 diferentes fatores como apontado por Oliveira *et al.* (2012): desempenho, custo financeiro e qualidade. Desta forma, o valor C para cada programa alternativo é calculado como $C = p_1.v_{performance} + p_2.v_{financialCost} + p_3.v_{quality}$ onde $p_3 = 0$ pois não consideramos o critério de qualidade.

O cientista inicia a execução do *workflow* original utilizando o MAFFT. Durante a execução do *workflow*, ele analisa resultados preliminares e verifica que o desempenho e o custo financeiro não estão atendendo suas necessidades. O cientista, então, requisita ao DynAdapt que mude a estrutura do *workflow* baseado na função de custo (não arbitrariamente). O DynAdapt realiza a mudança, de acordo com os fatores e a função de custo, de forma que as instâncias em execução do *workflow* são imediatamente afetadas pela adaptação estrutural. Em todos os casos, o MAFFT foi substituído pelo programa Muscle. Para melhor avaliar a abordagem proposta e analisar

a diferença no tempo total da execução e custos financeiros, analisamos quatro diferentes cenários: (i) processamento de todos os dados de entrada utilizando o MAFFT, (ii) mudança do MAFFT pelo Muscle quando 10% dos arquivos são processados, (iii) mudança do MAFFT pelo Muscle quando 50% dos arquivos foram processados e (iv) mudança do MAFFT pelo Muscle quando 90% dos arquivos foram processados. Para cada um destes cenários nós comparamos o tempo de execução do experimento e o custo financeiro do *workflow*.

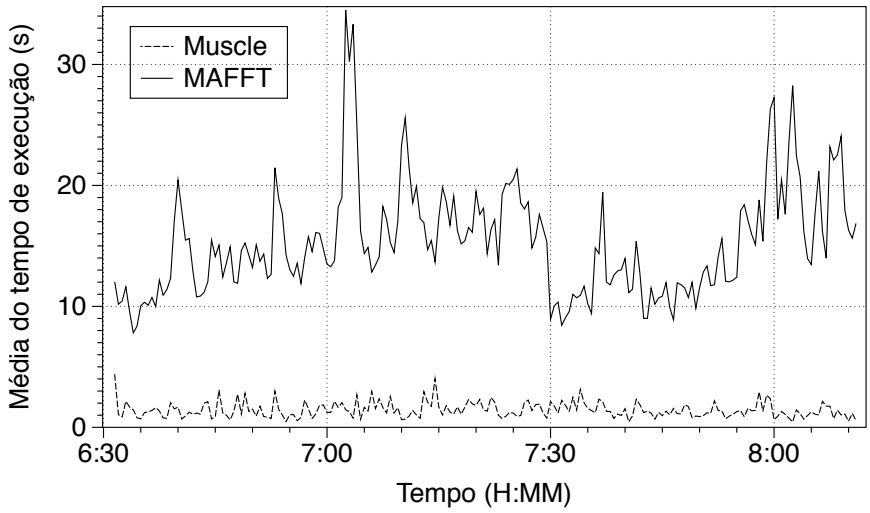

Figura 28. Tempo de execução médio das tarefas do Muscle *versus* MAFFT.

Nós analisamos por que o MAFFT foi sempre substituído pelo Muscle na execução do *workflow*. O MAFFT utiliza um método de alinhamento L-INS-I que foca na acurácia ao invés do desempenho. Cada execução do MAFFT é, em média, 20 vezes maior que uma execução do Muscle como apresentado na Figura 28. Na Figura 28, o eixo *x* representa o tempo de execução do *workflow* e o eixo *y* representa o tempo de execução médio das tarefas terminadas naquele tempo específico. Note que na Figura 28 nós não consideramos todas as execuções da tarefas do MAFFT.

Após esta primeira análise, nós executamos os três cenários restantes com o DynAdapt. Os resultados estão condensados na Figura 29, que mostra claramente quando o MAFFT é substituído pelo Muscle em tempos específicos da execução do *workflow* pois a média da execução das tarefas decresce consideravelmente para cada um dos cenários (10% - linha preta, 50% - linha vermelha tracejada e 90% linha azul pontilhada). A mudança dinâmica impacta diretamente no desempenho total. Há uma diferença de 40 minutos entre trocar o MAFFT pelo Muscle com 10% e com 90% dos dados processados. Ao mesmo tempo, o DynAdapt insere uma sobrecarga desprezível

na execução. Na verdade, apenas quando há uma mudança estrutural no *workflow*, faz-se necessário reexecutar algumas tarefas que estavam suspensas quando a mudança aconteceu.

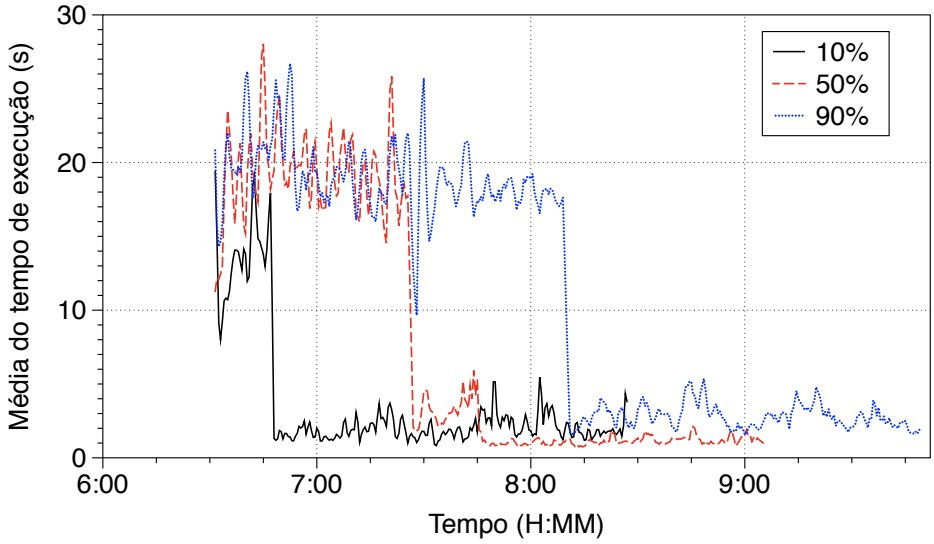

Figura 29. Mudanças dinâmicas no SciPhy.

Uma segunda análise foi feita do ponto de vista do usuário para comparar o tempo de execução total e o custo financeiro da execução do *workflow*. Ainda que o SciPhy seja composto por algumas atividades, nós focamos no tempo e no custo financeiro da atividade de AMS somente. Sem o DynAdapt, cientistas têm poucas opções que não abortar a execução do *workflow* e começar novamente com a nova estrutura do *workflow*. Portanto, nós comparamos todos os cenário mencionados anteriormente utilizando somente o MAFFT (o programa original configurado para o *workflow*) e utilizando somente o Muscle (o programa escolhido para substituir o MAFFT). Quando o cientista executa o *workflow* utilizando apenas o MAFFT, cerca de 20 horas de tempo total de execução foi necessário para processar 200 arquivos de entrada multi-fasta enquanto demora-se 8 horas ao utilizar apenas o Muscle. Os cientistas têm duas opções para realizar mudanças: utilizar o DynAdapt ou reiniciar o *workflow* manualmente. A Figura 30 apresenta o tempo total de execução utilizando o DynAdapt e utilizando a reexecução manual para cada um dos cenários. Em geral, existe uma diferença de cerca de 14% de diferença entre a abordagem DynAdapt e a abordagem manual. Este resultado pode implicar em um tempo absoluto considerável quando o experimento cresce tanto em complexidade e volume de dados processados. Este ganho de desempenho é esperado pois o DynAdapt não requer a reexecução do *workflow* pela máquina de execução do *workflow*, poupando o tempo de implantação e

inicialização, especialmente em ambientes distribuídos como nuvens. O DynAdapt reduziu o tempo de execução de 20 horas no pior caso (utilizando somente o MAFFT) para 8 horas quando trocou-se o MAFFT pelo Muscle depois que 10% dos dados foram processados, o que representa 40% em redução do tempo total de execução. O tempo total de execução do DynAdapt é ligeiramente superior ao tempo total ao utilizar somente o Muscle. Com relação aos custos financeiros, como o tempo de execução foi reduzido e, na Amazon AWS paga-se por hora de processamento, o DynAdapt também reduziu os custos financeiros como mostra a Figura 30.

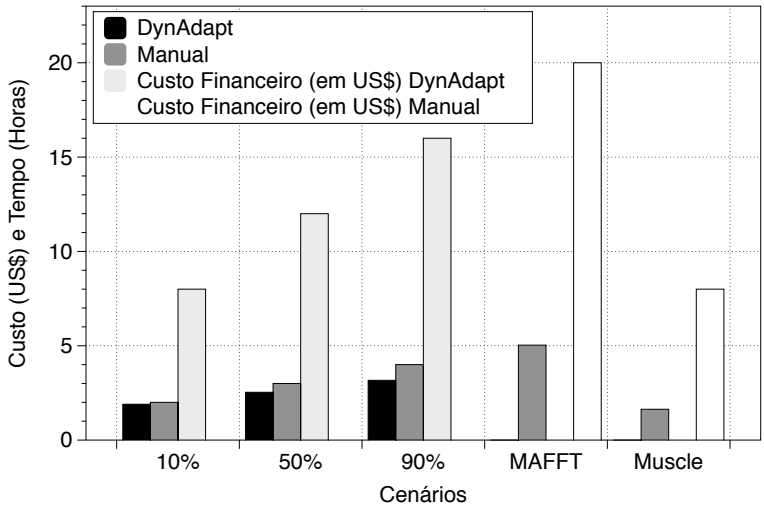

Figura 30. Comparação de tempo de execução e custo financeiro.

8.1.3 Quantificação de Incertezas

Esta avaliação experimental explora o *workflow* de quantificação de incertezas descrito na seção 7.3. Nós executamos as duas versões do *workflow* de QI (com e sem ciclos dinâmicos) descritas nas seções 7.3.1 e 7.3.2 e discutimos as vantagens da abordagem dinâmica do ponto de vista do usuário. Nós consideramos que o usuário realiza uma interação do tipo s^{ω} durante a execução. Para ambos os *workflow*s de QI, o critério de parada é a diferença entre as normas dos vetores de energia cinética entre dois níveis de interpolação subsequentes. Inicialmente, o critério é configurado para uma diferença normalizada menor que 0,001. Com o *workflow* manual, após a execução de pelo menos dois níveis de interpolação, se os resultados obtidos não atendem o critério definido, o cientista executa o *workflow* manual novamente com um nível de interpolação mais alto. Com o *workflow* dinâmico, os cientistas iniciam o *workflow* em um dado nível de interpolação e o *workflow* automaticamente aumenta o nível, se necessário. Se eles

precisarem alterar o critério de parada (*i.e.*, a função de avaliação ε), eles realizam uma interação s^{ω} e o ciclo dinâmico se adapta.

Ambos os *workflow*s foram executados em um cluster com 384 processadores Intel Xeon X5650 com frequência de 2,67GHz, 256 processadores Intel Xeon X5355 com frequência de 2,66GHz, 1,28 TBytes de memória RAM distribuída e 72TBytes de armazenamento NAS. Os *workflow*s foram explorados em dois cenários: completo e curto. No cenário completo, os cientistas realizam a análise de QI começando no nível de interpolação 0. No cenário curto, antes da execução do *workflow* de QI, o cientista consulta os dados de proveniência de execuções anteriores e conclui que o nível de interpolação 3 é melhor para iniciar o processo, uma vez que a confiabilidade dos resultados abaixo do nível 3 é em geral inadequado. Nós apresentamos os resultados no cenário curto pois no longo prazo ele é mais realista. Em ambos os casos o critério de parada é configurado inicialmente como $\varepsilon = normDiff < 0.001$. Este limite pode fazer o experimento executar até os níveis 6 ou 7. Entretanto, analisando os resultados do nível de interpolação 4, os cientistas podem decidir que eles podem mudar o critério do erro para $\varepsilon = normDiff < 0.01$. Na abordagem manual, nada é alterado por que o cientista é o responsável por calcular a norma dos vetores e controlar o processo iterativo. Na abordagem dinâmica, entretanto, mudar o ε causa uma interação s^{ω}, a qual é tratada pelo algoritmo Interação-Omega no Chiron.

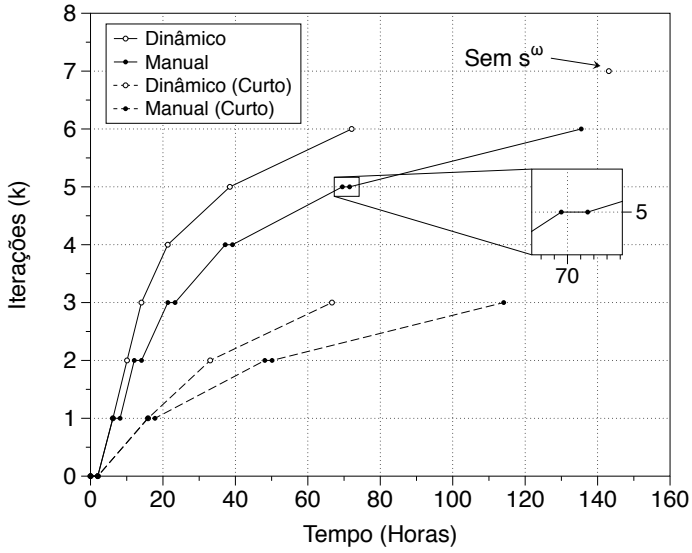

Figura 31. Tempo de execução das abordagens manual e dinâmica.

O benchmark *lock-exchange* que nós utilizamos no *workflow* de QI requer o nível de interpolação 5 (6 iterações) para alcançar o critério estabelecido (*normDiff* <

0.01). Sendo assim, o *workflow* manual no cenário completo requer que o cientista analise e reinicie o *workflow* 5 vezes. Os pontos de reinício podem ser visto na Figura 31; nós ampliamos um ponto de reinício na iteração 5. Consideramos que o tempo que o cientista gasta para implantar o *workflow* e dispará-lo como uma constante. A abordagem dinâmica requer que os cientistas iniciem o *workflow* apenas uma vez. Um aspecto importante do *workflow* de QI é que diversas explorações de um dado nível de interpolação podem ser reutilizadas nos níveis seguintes. Entretanto, como é trabalhoso configurar a execução seguinte do *workflow* manual para utilizar resultados obtidos previamente, os cientistas costumam optar por rodar o *workflow* manual sem reutilizar resultados anteriores. Utilizando a abordagem dinâmica, é mais simples reutilizar estar explorações por que elas já estão no contexto do *workflow* e são simplesmente referenciados pela máquina de execução do *workflow*. A Figura 31 apresenta o resultado para o cenário completo. A abordagem dinâmica economiza mais de 2 dias e meio do tempo de execução. Esta economia é esperada já que o *workflow* dinâmico reutiliza execuções de níveis de interpolação anteriores e também economiza o tempo de implantação e disparo do *workflow*. Nós também apresentamos, na Figura 31, quando a abordagem dinâmica iria parar se a interação s^{ω} não for processada. A interação com o *workflow* em tempo de execução economizou quase 3 dias de tempo de execução.

A Figura 31 também apresenta os resultados para o cenário curto. No cenário curto, os cientistas iniciam ambos os *workflow* no nível 3 e ele itera até o nível 5 (3 iterações). Este cenário reduz o número de iterações e, consequentemente, o número de vezes que os cientistas precisam reiniciar o *workflow* com um nível de interpolação mais elevado. Portanto, ele naturalmente melhora a abordagem manual. Contudo, a abordagem dinâmica ainda economiza praticamente 2 dias de tempo de execução. A diferença de desempenho entre o cenário completo e curto não é maior por que os pontos de colocação dos níveis de interpolação 0, 1 e 2 são incluídos para serem processados no nível 3. A diferença é que, no cenário completo, eles são processados de forma incremental, enquanto no cenário curto eles são processados com se pertencessem ao nível 3. Além disso, a quantidade de pontos nos níveis 0, 1 e 2 é muito menor que em níveis mais altos, uma vez que a quantidade de pontos cresce exponencialmente.

8.2 Análise da Sobrecarga da Abordagem Dinâmica

O nível de interpolação necessário para um problema é desconhecido antes da execução do *workflow*. Entretanto, os cientistas podem sempre escolher iniciar o *workflow* de QI em um nível alto para garantir uma alta confiabilidade em uma única execução. Por exemplo, no caso de uso do *lock-exchange*, eles podem ter um bom palpite e iniciar o *workflow* manual no nível 5. Porém, escolher níveis mais altos para o nível de interpolação pode resultar no desperdício de recursos computacionais e de armazenamento. Por exemplo, ao invés de iniciar o *workflow* no nível 5, os cientistas podem optar pelo nível 8, o qual também atende os requisitos de confiabilidade mas pode consumir muito mais tempo de execução e armazenamento para acabar. Nestes casos, a abordagem dinâmica é mais eficiente. Utilizando resultados do *workflow* de QI, associamos o nível que o cientista escolhe para iniciar o *workflow* com o nível adequado que o problema requer e a eficiência do *workflow* dinâmico iniciado no nível de interpolação 3. A Figura 32 apresenta os resultados para os níveis de interpolação de 5 a 8. As linhas horizontais são os tempos constantes para executar o *workflow* manual nos níveis 5, 6, 7 e 8 em uma única execução. Na abordagem dinâmica, o tempo de execução cresce de maneira adaptativa de acordo com o nível de interpolação mais adequado.

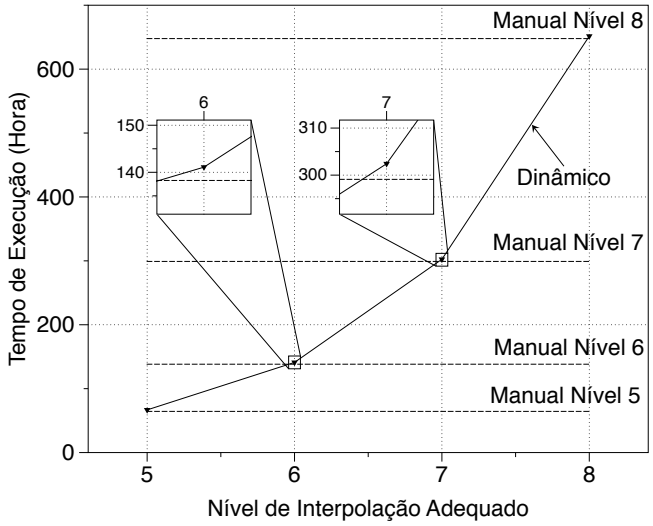

Figura 32. Eficiência da abordagem dinâmica comparada com a abordagem manual.

Se o cientista sabiamente escolhe o nível de interpolação mais adequado para iniciar o *workflow* manual, tem-se naturalmente a melhor opção. Contudo, a diferença no tempo entre a abordagem manual e dinâmica é pequena e torna-se menos significante

conforme aumentamos o nível de interpolação. Na Figura 32, nós ampliamos os resultados para os níveis 6 e 7. No nível 6. A abordagem dinâmica é 2,82 horas mais lenta enquanto no nível 7 é 3,32 horas mais lenta. É importante ressaltar que, ainda que a diferença no tempo cresça, quando comparamos com o tempo total de execução, a diferença torna-se menos significante como mostra a Tabela 2. A abordagem dinâmica aumento o tempo de execução em 2,05% no nível 6 mas aumento apenas 1,11% no nível 7 e 0,59% no nível 8. Acreditamos que essa sobrecarga é aceitável uma vez que a abordagem dinâmica pode poupar muitas horas na execução se o nível de interpolação não for escolhido corretamente no *workflow* manual como mostra a Tabela 2. Por exemplo, se o nível mais adequado é o 6, mas o cientista inicia o *workflow* manual no nível 7, a abordagem dinâmica toma 141 horas para executar ao invés das 300 horas da abordagem manual. Neste caso, o *workflow* dinâmico economiza 52,83% de tempo de execução. No maior cenário, se o cientista inicia o *workflow* no nível 8 ao invés do 5, a abordagem dinâmica pode economizar até 24,2 dias (89,69% do tempo de execução). Ainda que este comportamento esteja associado com o problema de quantificação de incertezas, a mesma situação ocorre em outros problemas que consomem dados de forma incremental ou problemas com requisitos de adaptabilidade ou auto-ajuste tais como problemas de otimização não determinísticos e experimentos com alinhamento múltiplo de sequências na bioinformática.

Tabela 2. Economia de tempo da abordagem dinâmica tendo como base a escolha manual do nível de interpolação.

Escolha Manual	**Nível Adequado**			
	5	*6*	*7*	*8*
5	**-3,63%**	-	-	-
6	51,70%	**-2,05%**	-	-
7	77,67%	52,83%	**-1,11%**	-
8	89,69%	78,22%	53,32%	**-0,59%**

8.3 Análise da Proveniência Através do Cubo de Dados

Durante a execução do *workflow*, os cientistas precisam seguir os resultados para avaliar a convergência do experimento. Baseados nesta análise, eles podem ajustar o critério de parada através do ajuste de ε. Se o experimento é modelado como um *workflow*

dinâmico, os dados do experimento podem ser analisados através do cubo de dados de linhagem. Na análise de QI, por exemplo, armazenamos as normas dos vetores em uma relação do *workflow* que pertence a $R_{\circlearrowleft}$. Portanto, podemos selecionar as normas calculadas para todas as linhagens e visualizar a distribuição das normas ao longo das iterações do *workflow* como mostram os diagramas de caixa (*boxplots*) (Devore 2011) na Figura 33. Baseado neste resultado, podemos observar que as normas calculadas na primeiras iterações são diversas e com um baixo nível de confiabilidade em função da alta variância. Entretanto, nas iterações 4 e 5, a distribuição das normas converge e é mais confiável com uma variância menor.

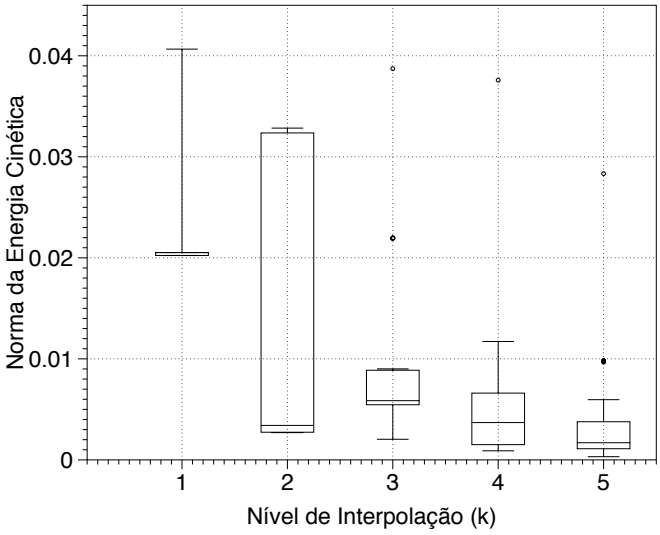

Figura 33. A distribuição das normas para todas as linhagens ao longo das iterações obtidas pelo cubo de dados da linhagem.

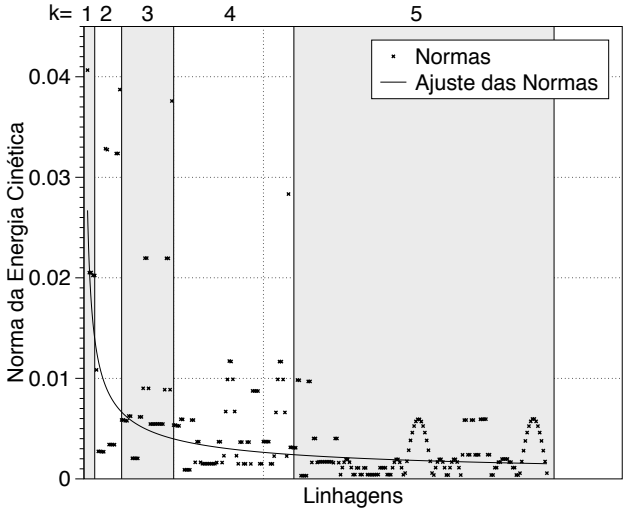

Figura 34. Valores das normas dos vetores para todas as linhagens do experimento.

Na Figura 33, nós visualizamos o cubo de dados da perspectiva das iterações (*k*). Outra análise possível é mostrada na Figura 34 onde nós visualizamos a perspectiva das linhagens. Neste gráfico, é mais fácil compreender por que a confiabilidade de níveis mais altos é melhor. Nos níveis de interpolação mais baixos, observam-se menos pontos e eles estão mais esparsos. Nos níveis mais altos, observam-se mais amostras e elas estão menos esparsas.

8.4 Análise do Comportamento dos Algoritmos de Interação

Outro aspecto importante dos *workflow*s dinâmicos é a performance dos algoritmos de interação. Quando os cientistas interagem com o *workflow* em tempo real, a API Achilles armazena o evento na base de proveniência. O Chiron trata o evento em tempo de execução utilizando os algoritmos de Interação Alfa ou Ômega, ambos descritos na seção 5.3. Para avaliar o desempenho dos algoritmos, nós instrumentamos o Chiron para armazenar o tempo gasto para processar cada evento de interação. Em seguida, nós programamos algumas interações em série para ocorrer em diferentes tempos da execução do *workflow*.

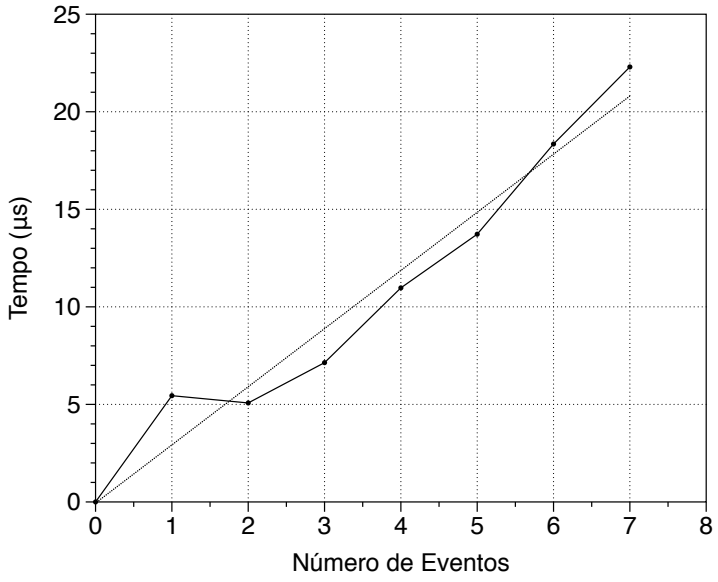

Figura 35. Desempenho do algoritmo Interação-Alfa.

A Figura 35 apresenta o desempenho do algoritmo Interação-Alfa. Mesmo quando o Chiron processa 7 eventos em seguida, o tempo gasto com o algoritmo é menor que 25 microssegundos. Como o algoritmo executa rapidamente, ele é sensível a pequenas flutuações de desempenho. Esta avaliação mostra um comportamento linear do algoritmo de Interação-Alfa.

O comportamento do algoritmo Interação-Ômega também é linear, como mostra a Figura 36. Como o algoritmo demora mais para executar, as flutuações no desempenho são difíceis de notar. O tempo total para processar 7 eventos em sequência é menor que 1 milissegundo, um tempo insignificante quando comparamos com o tempo de execução do *workflow* que pode demorar dias ou até mesmo semanas. Além disso, 7 eventos s^{ω} em sequência não é muito realista uma vez que isso representaria o cientista alterando a configuração do *workflow* diversas vezes rapidamente sem esperar pelo retorno dos resultados produzidos. Normalmente, o Chiron irá processar apenas um evento s^{ω} por vez, no contexto da execução de um *workflow*.

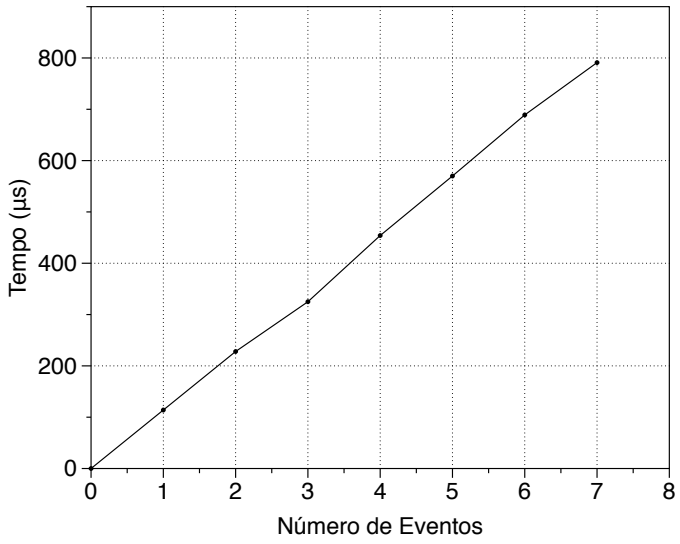

Figura 36. Desempenho do algoritmo Interação-Omega.

Nossa avaliação experimental mostra que os ciclos dinâmicos melhoram a execução de experimentos que precisam de adaptações e análises constantes dos resultados produzidos. O *workflow* dinâmico pode automatizar parte da avaliação experimental e os algoritmos de interação Alfa e Ômega possibilitam a interação com os dados de entrada e com a configuração do *workflow*. Ambos os algoritmos de interação tiveram um bom desempenho, executando em menos de 1 milissegundo no cenário do *workflow* de QI. Além disso, o modelo de cubo de dados para linhagem possibilitam consultas complexas através dos dados do experimento para analisar os resultados produzidos ao longo das iterações do *workflow*.

Capítulo 9 Conclusões

*Workflow*s dinâmicos são sujeitos a adaptação e refinamento contínuos, logo é necessário que o usuário interaja com o *workflow* em tempo de execução. Como a interação entre o cientista e o seu experimento é um processo iterativo, dar apoio a iterações dinâmicas em SGWfC é um importante requisito. As abordagens existentes para iteração em *workflow*s (Ekanayake *et al.* 2010; Wozniak *et al.* 2012) não são centradas em dados nem apoiam interações dinâmicas entre o usuário e o *workflow*. Nesta tese, discutimos construtos iterativos para álgebra de *workflow*s científicos (Ogasawara *et al.* 2011) inspirados por linguagens de programação para fluxo de dados em bancos de dados (Danforth e Valduriez 1992) e linguagens visuais de fluxo de dados (Mosconi e Porta 2000). Propomos uma operação algébrica para apoiar iterações em *workflow*s dinâmicos. Desta forma, o processo iterativo torna-se passível de modelagem na álgebra de *workflow*s. Nós introduzimos o conceito de linhagem na iteração de forma que o gerenciamento dos dados do *workflow* seja consistente com o processo iterativo e com possíveis mudanças dinâmicas. A linhagem possibilita consultas aos dados da execução através de um cubo de dados. Os cientistas podem navegar e interagir com os dados do *workflow* e suas configuração em tempo de execução através de dois algoritmos de interação. O algoritmo Interação-Alfa permite que os cientistas interajam com os dados de entrada do *workflow*, modificando parâmetros de entrada, selecionando novos dados para execução ou removendo entradas indesejadas. Já o algoritmo Interação-Ômega permite alterar a configuração do *workflow* gerando ramificações na execução.

Para avaliar a nossa hipótese de que a interação com *workflows* melhora a eficiência da execução, realizamos uma avaliação experimental utilizando diferentes *workflows*, incluindo um *workflow* dinâmico para um experimento de quantificação de incertezas. Para tal, desenvolvemos um módulo iterativo dentro do Chiron, a nossa máquina de execução de *workflow*s científicos que suporta a abordagem algébrica. Executamos os experimentos em um cluster de 640 processadores e também utilizando uma nuvem de computadores na infraestrutura da Amazon AWS. Apresentamos os benefícios da nossa abordagem do ponto de vista do usuário. Os resultados mostram melhoras de até 24 dias no tempo de execução em um experimento de quantificação de incertezas. Além disso, as consultas através do cubo de dados de linhagem permitem

que o cientista analise os resultados ao longo das iterações do *workflow*. Em um cenário onde nós privilegiamos a abordagem não-iterativa, a sobrecarga introduzida por nossa abordagem dinâmica é bem pequena (menos que 0,6% em um cenário grande) quando comparada ao tempo total de execução do experimento. O tempo de execução dos algoritmos de interação também é pequeno (menor que 1 milissegundo) e o desempenho cresce linearmente em relação ao número de eventos.

9.1 Contribuições Alcançadas

Podemos destacar as contribuições alcançadas nesta tese sob a perspectiva do ciclo de vida de um experimento científico (Mattoso *et al.* 2010b). Para a fase de composição, possibilitamos a concepção de experimentos na forma de um *workflow* iterativo através do operador algébrico *Evaluate*. Ao permitir ciclos na álgebra de *workflow*s, possibilitamos a modelagem de algoritmos iterativos e de experimentos com necessidades adaptativas ou de ajuste fino. Além disso, os módulos de composição e disparo de *workflow*s na plataforma Proteus facilitam a modelagem e implantação de *workflow*s em ambientes de PAD. Na fase de execução, utilizamos um modelo de execução dinâmico que apoia os ciclos dinâmicos com ativações de avaliação e um modelo em cubo para a linhagem de dados. Além disso, dois algoritmo possibilitam a interação entre o usuário e o *workflow* em tempo de execução. Através da plataforma Proteus, o usuário pode utilizar o módulo Helm para realizar interações alfa e ômega e o módulo DynAdapt para trocar atividades do *workflow*. Para a fase de análise, o modelo em cubo para linhagem permite a elaboração de consultas aos dados do experimento ao longo das iterações. O usuário pode realizar as consultas através do módulo Prov-Vis da plataforma Proteus.

A interação entre o usuário e o *workflow* em ambientes de PAD está associada a três principais questões envolvendo a gerência do *workflow*: monitoramento da execução, análise dos resultados parciais e interferência dinâmica na execução (Mattoso *et al.* 2013). Sob esta perspectiva também oferecemos recursos para o monitoramento e análise da execução através do módulo Prov-Vis combinado com os recursos da linhagem de dados e os algoritmos de interação através do módulo Helm e DynAdapt para possibilitar a interferência dinâmica na execução.

Realizamos também uma avaliação experimental envolvendo diferentes *workflow*s. Um *workflow* iterativo de quantificações de incertezas foi projetado para as

análises utilizando o modelo de referência *lock-exchange* e pode ser reutilizado para outros experimentos de quantificação de incertezas.

9.2 Trabalhos Futuros

A abordagem interativa proposta nesta tese pode ser avaliada como um passo inicial na direção dos *workflow*s dinâmicos como descritos em Gil *et al.* (2007). Ainda existem desafios envolvendo o desenvolvimento colaborativo e distribuído de *workflow*s científicos, bem como a necessidade de sistemas de consultas mais eficientes para análises dos dados do experimento e dos dados de proveniência. Considerando apenas a interação entre o usuário e o *workflow*, também existem alguns desafios em aberto. Na etapa do monitoramento, a maioria dos recursos já estão presentes em máquinas de *workflow* existentes que possibilitam a consulta à base de proveniência em tempo de execução (Oliveira *et al.* 2010a; Ogasawara *et al.* 2013) e notificação de eventos científicos (Pintas *et al.* 2012). Neste sentido, as abordagens de monitoramento disponíveis poderiam apenas melhorar com um melhor apoio na construção de consultas complexas e novas formas de visualização de dados, possivelmente tirando vantagem da linhagem de dados como proposto nesta tese e do relacionamento entre dados de proveniência e dados do experimento.

Para a etapa de análises dos dados parciais, podemos destacar o desafio ao transferir dados entre o ambiente de execução remoto, tal como um cluster ou nuvem, e o ambiente de análise. Embora os SGWfC possam lidar com mecanismos para paralelização e consulta de proveniência, o planejamento cuidadoso das transferências de dados é necessário, especialmente quando faz-se necessário um grande tráfico de dados na rede. Quando cientistas estão executando na nuvem, é inviável transferir mais que alguns gigabytes através de uma conexão comum com a Internet. Ainda que existem alguns mecanismos de alto desempenho para transferências como o gridFTP (Kourtellis *et al.* 2008), novos mecanismos para melhorar a eficiência da transferência de dados são efetivamente necessários.

Quando não é possível transferir dados do ambiente remoto, pode-se realizar análises *in situ*, de forma que a análise seja realizada remotamente no ambiente de execução. Desta forma, se os cientistas puderem realizar análises preliminares na nuvem ou no cluster, eles podem realizar a transferência apenas de uma pequena porção dos dados produzidos que são de interesse. Após realizar as análises, o cientista pode decidir

se deve fazer uma intervenção na execução do *workflow* ou não. Portanto, é importante prover ferramentas de apoio à tomada de decisão, por exemplo, ferramentas para transformar, correlacionar e analisar dados para facilitar a tarefa dos cientistas ao fazer a decisão adequada para o experimento. As ferramentas podem tirar vantagem dos dados de proveniência para enriquecer os resultados visualizados.

Para a interferência dinâmica em *workflow*s, esta tese cobre os principais desafios existentes (Mattoso *et al.* 2013), uma vez que possibilita a troca dinâmica de atividades através do DynAdapt e a alteração dinâmica de dados de entrada e da configuração do *workflow* através dos algoritmos Interação-Alfa e Interação-Ômega. Entretanto, o objetivo final da intervenção na execução do *workflow* é otimização do experimento como um todo. Ao invés de otimizar simplesmente um algoritmo ou o escalonamento do *workflow*, os cientistas precisam otimizar o experimento. Para tal, eles precisam realizar o ajuste-fino dos parâmetros como fatores de convergência, taxas de filtro, margens de erro ou nível de refinamento de um modelo. A melhor escolha para estes parâmetros só é descoberta durante a execução do experimento e pode ser inviável especificar estes parâmetros *a priori*. Para otimizar o experimento, o *workflow* precisa ser conduzido pelo cientista, que é o conhecedor do domínio do problema e pode fazer as melhores escolhas para refinar a execução. Embora esta tese ofereça ferramentas que apoiem este tipo de abordagem, ainda existem oportunidades em aberto para tratar o processo como um todo. Uma oportunidade em aberto é o desenvolvimento de ferramentas que permitam ao cientista explorar o cubo de linhagens de diferentes formas, podendo visualizar os resultados de maneira eficiente, realizar consultas analíticas e estabelecer correlações entre as diferentes explorações do experimento.

Referências bibliográficas

Aalst, W. Van der and Hee, K. Van (2002). *Workflow Management: Models, Methods, and Systems*. The MIT Press.

Aalst, W. M. P. Van der, Dongen, B. F. Van, Herbst, J., Maruster, L., Schimm, G. and Weijters, A. J. M. M. (2003). Workflow mining: a survey of issues and approaches. *Concurrency and Computation: Practice and Experience*, v. 47, n. 2, p. 237–267.

Abramson, D., Bethwaite, B., Enticott, C., Garic, S. and Peachey, T. (jun 2011). Parameter Exploration in Science and Engineering Using Many-Task Computing. *IEEE Trans. Parallel Distrib. Syst.*, v. 22, n. 6, p. 960–973.

Abramson, D., Enticott, C. and Altinas, I. (2008). Nimrod/K: towards massively parallel dynamic grid workflows. In *Proc. of International Conference for High Performance Computing, Networking, Storage and Analysis*. . IEEE Press.

Aho, A. V., Lam, M. S., Sethi, R. and Ullman, J. D. (2006). *Compilers: Principles, Techniques, and Tools*. 2. ed. Addison Wesley.

Ailamaki, A. (2011). Managing scientific data: lessons, challenges, and opportunities. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. , SIGMOD '11. ACM.

Ailamaki, A., Kantere, V. and Dash, D. (jun 2010). Managing scientific data. *Communications of the ACM*, v. 53, n. 6, p. 68–78.

Alqaoud, A., Taylor, I. and Jones, A. (1 jan 2010). Scientific workflow interoperability framework. *International Journal of Business Process Integration and Management*, v. 5, n. 1, p. 93–105.

Altintas, I., Barney, O. and Jaeger-Frank, E. (2006). Provenance Collection Support in the Kepler Scientific Workflow System. *Provenance and Annotation of Data*. LNCS. Springer Berlin. v. 4145p. 118–132.

Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B. and Mock, S. (2004). Kepler: an extensible system for design and execution of scientific workflows. In *Scientific and Statistical Database Management*.

Altschul, S. F., Gish, W., Miller, W., Myers, E. W. and Lipman, D. J. (oct 1990). Basic Local Alignment Search Tool. *Journal of Molecular Biology*, v. 215, n. 3, p. 403–410.

Amazon EC2 (2010). Amazon Elastic Compute Cloud (Amazon EC2). . http://aws.amazon.com/ec2/.

Amsallem, D. and Farhat, C. (2008). Interpolation Method for Adapting Reduced-Order Models and Application to Aeroelasticity. *AIAA Journal*, v. 46, p. 1803–1813.

Bäck, T. (1996). *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford, UK: Oxford University Press.

Bancilhon, F. and Ramakrishnan, R. (jun 1986). An Amateur's Introduction to Recursive Query Processing Strategies. *SIGMOD Rec.*, v. 15, n. 2, p. 16–52.

Baralis, E. and Widom, J. (Setembro 2000). An algebraic approach to static analysis of active database rules. *ACM Trans. Database Syst.*, v. 25, n. 3, p. 269–332.

Bayucan, A., Henderson, R. L. and Jones, J. P. (2000). *Portable Batch System Administration Guide*. Mountain View, CA, USA: Veridian Systems.

Berriman, G. B., Deelman, E., Good, J., Jacob, J. C., Katz, D. S., Laity, A. C., Prince, T. A., Singh, G. and Su, M.-H. (2007). Generating Complex Astronomy Workflows. *Workflows for e-Science*. Springer. p. 19–38.

Bertino, E., Bernstein, P., Agrawal, D., Davidson, S., Dayal, U., Franklin, M., Gehrke, J., Haas, L., Halevy, A., Han, J., Jadadish, H. V., Labrinidis, A., Madden, S., Papokonstantinou, Y., Patel, J., *et al.* (2011). Challenges and Opportunities with Big Data. Cyber Center Publications.

Boeres, C., Sardiña, I. and Drummond, L. (Agosto 2011). An efficient weighted bi-objective scheduling algorithm for heterogeneous systems. *Parallel Computing*, v. 37, n. 8, p. 349–364.

Bondy, A. and Murty, U. S. R. (2010). *Graph Theory (Graduate Texts in Mathematics)*. 2008. ed.

Bouganim, L., Florescu, D. and Valduriez, P. (1996). Dynamic load balancing in hierarchical parallel database systems. In *Proceedings of the 22nd International Conference on Very Large Databases*. . Morgan Kaufmann Publishers Inc.

Bowers, S., Ludascher, B., Ngu, A. H. H. and Critchlow, T. (2006). Enabling ScientificWorkflow Reuse through Structured Composition of Dataflow and Control-Flow. In *Proceedings of the 22nd International Conference on Data Engineering Workshops*. . IEEE Computer Society.

Bowers, S., McPhillips, T. M. and Ludescher, B. (2008). Provenance in collection-oriented scientific workflows. *Concurrency and Computation: Practice and Experience*, v. 20, n. 5, p. 519–529.

Bu, Y., Howe, B., Balazinska, M. and Ernst, M. D. (sep 2010). HaLoop: efficient iterative data processing on large clusters. *Proc. VLDB Endow.*, v. 3, n. 1-2, p. 285–296.

Buneman, P., Khanna, S. and Tan, W. (2001). Why and Where: A Characterization of Data Provenance. *International Conference on Database Theory*, p. 316–330.

Buyya, R., Broberg, J. and Goscinski, A. M. (2011). *Cloud Computing: Principles and Paradigms*. 1. ed. Wiley.

Byungil Jeong, Leigh, J., Johnson, A., Renambot, L., Brown, M., Jagodic, R., Sungwon Nam and Hyejung Hur (jun 2010). Ultrascale Collaborative Visualization Using a Display-Rich Global Cyberinfrastructure. *IEEE Computer Graphics and Applications*, v. 30, n. 3, p. 71–83.

Callahan, S. P., Freire, J., Santos, E., Scheidegger, C. E., Silva, C. T. and Vo, H. T. (2006). VisTrails: visualization meets data management. In *SIGMOD International Conference on Management of Data*. . ACM.

Carvalho, L. O. M. (2009). Application of Scientific Workflows in the Design of Offshore Systems for Oil Production (in Portuguese). COPPE - Federal University of Rio de Janeiro , Civil Engineering Department.

Ceri, S., Gottlob, G. and Tanca, L. (1989). What you always wanted to know about Datalog (and never dared to ask). *Knowledge and Data Engineering, IEEE Transactions on*, v. 1, n. 1, p. 146–166.

Costa, F., Oliveira, D., Ocaña, K., Ogasawara, E. and Mattoso, M. (2012). Enabling Re-Executions of Parallel Scientific Workflows Using Runtime Provenance Data. In: 4th International Provenance and Annotation Workshop.

Costa, F., Silva, V., De Oliveira, D., Ocaña, K., Ogasawara, E., Dias, J. and Mattoso, M. (2013). Capturing and Querying Workflow Runtime Provenance with PROV: A Practical Approach. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*. , EDBT '13. ACM.

Coutinho, A. L. G. A., Landau, L., Wrobel, L. C. and Ebecken, N. F. F. (1989). Modal solution of transient heat conduction utilizing Lanczos algorithm. *International Journal for Numerical Methods in Engineering*, v. 28, n. 1, p. 13–25.

Cudre-Mauroux, P., Kimura, H., Lim, K.-T., Rogers, J., Simakov, R., Soroush, E., Velikhov, P., Wang, D. L., Balazinska, M., Becla, J., DeWitt, D., Heath, B., Maier, D., Madden, S., Patel, J., *et al.* (aug 2009). A demonstration of SciDB: a science-oriented DBMS. *Proceedings of the VLDB Endowment*, v. 2, n. 2, p. 1534–1537.

Dąbrowski, M., Drabik, M., Trzaska, M. and Subieta, K. (2010). *Dynamic Changes of Workflow Processes*.

Danforth, S. and Valduriez, P. (feb 1992). A FAD for data intensive applications. *IEEE Transactions on Knowledge and Data Engineering*, v. 4, n. 1, p. 34–51.

Dávila, A. M. R., Mendes, P. N., Wagner, G., Tschoeke, D. A., Cuadrat, R. R. C., Liberman, F., Matos, L., Satake, T., Ocaña, K. A. C. S., Triana, O., Cruz, S. M. S., Jucá, H. C. L., Cury, J. C., Silva, F. N., Geronimo, G. A., *et al.* (2008). ProtozoaDB: dynamic visualization and exploration of protozoan genomes. *Nucleic Acids Research*, v. 36, n. Database issue, p. D547–D552.

De Roure, D. and Goble, C. (2007). myExperiment – A Web 2.0 Virtual Research Environment. *In International Workshop on Virtual Research Environments and Collaborative Work Environments.*,

Dean, J. and Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, v. 51, n. 1, p. 107–113.

Deelman, E., Gannon, D., Shields, M. and Taylor, I. (2009). Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, v. 25, n. 5, p. 528–540.

Deelman, E., Mehta, G., Singh, G., Su, M.-H. and Vahi, K. (2007). Pegasus: Mapping Large-Scale Workflows to Distributed Resources. *Workflows for e-Science*. Springer. p. 376–394.

Devore, J. L. (2011). *Probability and Statistics for Engineering and the Sciences*. USA: Brooks/Cole, Cengage Learning.

Dias, J., Ogasawara, E., Oliveira, D., Porto, F., Coutinho, A. and Mattoso, M. (2011). Supporting Dynamic Parameter Sweep in Adaptive and User-Steered Workflow. In *6th Workshop on Workflows in Support of Large-Scale Science*. , WORKS '11. ACM.

Dias, J., Ogasawara, E., Oliveira, D., Porto, F., Valduriez, P. and Mattoso, M. (2013). Algebraic Dataflows for Big Data Analysis. In *Proceedings of the IEEE International Conference on Big Data*.

Dittrich, J., Quiané-Ruiz, J.-A., Richter, S., Schuh, S., Jindal, A. and Schad, J. (2012). Only aggressive elephants are fast elephants. *Proceedings of the VLDB Endowment*, v. 5, n. 11, p. 1591–1602.

Do, C. B., Mahabhashyam, M. S. P., Brudno, M. and Batzoglou, S. (1 feb 2005). ProbCons: Probabilistic consistency-based multiple sequence alignment. *Genome Research*, v. 15, n. 2, p. 330 –340.

Doerr, K.-U. and Kuester, F. (2011). CGLX: A Scalable, High-Performance Visualization Framework for Networked Display Environments. *IEEE Transactions on Visualization and Computer Graphics*, v. 17, n. 3, p. 320–332.

Eddy, S. R. (1998). Profile Hidden Markov Models. *Bioinformatics*, v. 14, n. 9, p. 755–763.

Edgar, R. C. (1 mar 2004). MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, v. 32, n. 5, p. 1792 –1797.

Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S.-H., Qiu, J. and Fox, G. (2010). Twister: A Runtime for Iterative MapReduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing - HPDC '10*.

Ellis, C., Keddara, K. and Rozenberg, G. (1995). Dynamic change within workflow systems. . ACM New York, NY, USA.

Elmasri, R. and Navathe, S. (2010). *Fundamentals of Database Systems*. 6. ed. Addison Wesley.

Elmroth, E., Hernández, F. and Tordsson, J. (Fevereiro 2010). Three fundamental dimensions of scientific workflow interoperability: Model of computation, language, and execution environment. *Future Generation Computer Systems*, v. 26, n. 2, p. 245–256.

Ewen, S., Tzoumas, K., Kaufmann, M. and Markl, V. (jul 2012). Spinning fast iterative data flows. *Proc. VLDB Endow.*, v. 5, n. 11, p. 1268–1279.

Fahringer, T., Prodan, R., Rubing Duan, Nerieri, F., Podlipnig, S., Jun Qin, Siddiqui, M., Hong-Linh Truong, Villazon, A. and Wieczorek, M. (13 nov 2005). ASKALON: a Grid application development and computing environment. In *6th IEEE/ACM International Workshop on Grid Computing*. . IEEE.

Floratou, A., Patel, J. M., Shekita, E. J. and Tata, S. (Abril 2011). Column-oriented storage techniques for MapReduce. *Proceedings of the VLDB Endowment*, v. 4, n. 7, p. 419–429.

Fowler, M. (2004). *UML distilled: a brief guide to the standard object modeling language*. Boston: Addison-Wesley.

Freire, J., Koop, D., Santos, E. and Silva, C. T. (2008). Provenance for Computational Tasks: A Survey. *Computing in Science and Engineering, v.10*, n. 3, p. 11–21.

Gadelha, L. M. R., Clifford, B., Mattoso, M., Wilde, M. and Foster, I. (jun 2011). Provenance management in Swift. *Future Generation Computer Systems*, v. 27, n. 6, p. 775–780.

Gannon, D., Plale, B., Marru, S., Kandaswamy, G., Simmhan, Y. and Shirasuna, S. (2007). Dynamic, Adaptive Workflows for Mesoscale Meteorology. *Workflows for e-Science*. Springer. p. 126–142.

Gentzsch, W. (2001). Sun Grid Engine: towards creating a compute power grid. . IEEE Comput. Soc.

Gil, Y., Deelman, E., Ellisman, M., Fahringer, T., Fox, G., Gannon, D., Goble, C., Livny, M., Moreau, L. and Myers, J. (2007). Examining the Challenges of Scientific Workflows. *Computer*, v. 40, n. 12, p. 24–32.

Goecks, J., Nekrutenko, A. and Taylor, J. (2010). Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, v. 11, n. 8, p. 86.

Gough, J., Karplus, K., Hughey, R. and Chothia, C. (2 nov 2001). Assignment of homology to genome sequences using a library of hidden Markov models that represent all proteins of known structure. *Journal of Molecular Biology*, v. 313, n. 4, p. 903–919.

Graefe, G. (1993). Query evaluation techniques for large databases. *ACM Computing Surveys*, v. 25, n. 2, p. 73–169.

Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M., Pellow, F. and Pirahesh, H. (1 mar 1997). Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. *Data Mining and Knowledge Discovery*, v. 1, n. 1, p. 29–53.

Guerra, G., Rochinha, F. A., Elias, R., De Oliveira, D., Ogasawara, E., Dias, J. F., Mattoso, M. and Coutinho, A. L. G. A. (2012). Uncertainty Quantification in Computational Predictive Models for Fluid Dynamics Using Workflow Management Engine. *International Journal for Uncertainty Quantification*, v. 2, n. 1, p. 53–71.

Heath, M. (2002). *Scientific computing : an introductory survey*. 2nd ed. ed. Boston: McGraw-Hill.

Herodotou, H., Lim, H., Luo, G., Borisov, N., Dong, L., Cetin, F. B. and Babu, S. (2011). Starfish: A Self-tuning System for Big Data Analytics. In *Proceedings of the 5th Conference on Innovative Data Systems Research*.

Horta, F., Dias, J., Elias, R., Oliveira, D., Coutinho, A. L. G. A. and Mattoso, M. (2013). Prov-Vis: Large-Scale Scientific Data Visualization Using Provenance (Abstract). In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*.

Horta, F., Dias, J., Ocaña, K. A. C. S., De Oliveira, D., Ogasawara, E. and Mattoso, M. (2012). Using Provenance to Visualize Data from Large-Scale Experiments (Abstract). In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*.

Hughey, R. and Krogh, A. (apr 1996). Hidden Markov models for sequence analysis: extension and analysis of the basic method. *Computer Applications in the Biosciences: CABIOS*, v. 12, n. 2, p. 95–107.

Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M. R., Li, P. and Oinn, T. (2006). Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, v. 34, n. 2, p. 729–732.

Jacob, J. C., Katz, D. S., Berriman, G. B., Good, J. C., Laity, A. C., Deelman, E., Kesselman, C., Singh, G., Su, M.-H., Prince, T. A. and Williams, R. (2009). Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking. *Int. J. Comput. Sci. Eng.*, v. 4, n. 2, p. 73–87.

Jarke, M. and Koch, J. (jun 1984). Query Optimization in Database Systems. *ACM Comput. Surv.*, v. 16, n. 2, p. 111–152.

Johnston, W. M., Hanna, J. R. P. and Millar, R. J. (mar 2004). Advances in dataflow programming languages. *ACM Comput. Surv.*, v. 36, n. 1, p. 1–34.

Kammer, P. J., Bolcer, G. A., Taylor, R. N., Hitomi, A. S. and Bergman, M. (2000). Techniques for Supporting Dynamic and Adaptive Workflow. *Computer Supported Cooperative Work (CSCW)*, v. 9, n. 3-4, p. 269–292.

Katoh, K. and Toh, H. (jul 2008). Recent developments in the MAFFT multiple sequence alignment program. *Briefings in Bioinformatics*, v. 9, n. 4, p. 286–298.

Kelley, C. T. (1999). *Iterative methods for optimization*. SIAM.

Kilmer, M. E. and De Sturler, E. (2006). Recycling Subspace Information for Diffuse Optical Tomography. *SIAM Journal on Scientific Computing*, v. 27, n. 6, p. 2140.

Korkhov, V., Krefting, D., Kukla, T., Terstyanszky, G. Z., Caan, M. and Olabarriaga, S. D. (2011). Exploring workflow interoperability tools for neuroimaging data analysis. In *Proceedings of the 6th workshop on Workflows in support of large-scale science*. , WORKS '11. ACM.

Kourtellis, N., Prieto, L., Iamnitchi, A., Zarrate, G. and Fraser, D. (2008). Data transfers in the grid: workload analysis of globus GridFTP. In *Proceedings of the 2008 international workshop on Data-aware distributed computing*. , DADC '08. ACM.

Kudtarkar, P., Deluca, T. F., Fusaro, V. A., Tonellato, P. J. and Wall, D. P. (2010). Cost-effective cloud computing: a case study using the comparative genomics tool, roundup. *Evolutionary Bioinformatics Online*, v. 6, p. 197–203.

Kwon, Y., Balazinska, M., Howe, B. and Rolia, J. (Agosto 2012). SkewTune in action: mitigating skew in MapReduce applications. *Proceedings of the VLDB Endowment*, v. 5, n. 12, p. 1934–1937.

Lander, E. S. (15 feb 2001). Initial sequencing and analysis of the human genome. *Nature*, v. 409, n. 6822, p. 860–921.

Lassmann, T. and Sonnhammer, E. L. L. (2005). Kalign--an accurate and fast multiple sequence alignment algorithm. *BMC Bioinformatics*, v. 6, p. 298.

Laszewski, G., Hategan, M. and Kodeboyina, D. (2007). Java CoG Kit Workflow. *Workflows for e-Science*. Springer. p. 340–356.

Lee, K., Sakellariou, R., Paton, N. W. and Fernandes, A. (2007). Workflow adaptation as an autonomic computing problem. In *Proceedings of the 2nd workshop on Workflows in support of large-scale science*. , WORKS '07. ACM.

Lima, B., Jacob, B. and Ebecken, N. (2005). A hybrid fuzzy/genetic algorithm for the design of offshore oil production risers. *International Journal for Numerical Methods in Engineering*, v. 64, n. 11, p. 1459–1482.

Loeliger, J. and McCullough, M. (2012). *Version Control with Git: Powerful tools and techniques for collaborative software development*. O'Reilly Media.

Ma, X. and Zabaras, N. (2009). An adaptive hierarchical sparse grid collocation algorithm for the solution of stochastic differential equations. *J. Comput. Phys.*, v. 228, p. 3084–3113.

Madera, M. and Gough, J. (1 oct 2002). A comparison of profile hidden Markov model procedures for remote homology detection. *Nucleic Acids Research*, v. 30, n. 19, p. 4321–4328.

Mattoso, M., Coutinho, A., Elias, R., Oliveira, D. and Ogasawara, E. (2010a). Exploring Parallel Parameter Sweep in Scientific Workflows. In *WCCM - World Congress on Computational Mechanics*.

Mattoso, M., Ocaña, K., Horta, F., Dias, J., Ogasawara, E., Silva, V., De Oliveira, D., Costa, F. and Araújo, I. (2013). User-steering of HPC workflows: state-of-the-art and future directions. In *Proceedings of the 2nd ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*. . ACM Press.

Mattoso, M., Werner, C., Travassos, G. H., Braganholo, V., Murta, L., Ogasawara, E., Oliveira, D., Cruz, S. M. S. Da and Martinho, W. (2010b). Towards Supporting the Life Cycle of Large-scale Scientific Experiments. *International Journal of Business Process Integration and Management*, v. 5, n. 1, p. 79–92.

Miller, W., Makova, K. D., Nekrutenko, A. and Hardison, R. C. (sep 2004). Comparative Genomics. *Annual Review of Genomics and Human Genetics*, v. 5, n. 1, p. 15–56.

Missier, P., Soiland-Reyes, S., Owen, S., Tan, W., Nenadic, A., Dunlop, I., Williams, A., Oinn, T. and Goble, C. (2010). Taverna, reloaded. In *Proceedings of the 22nd international conference on Scientific and statistical database management*. , SSDBM'10. Springer-Verlag.

Moreau, L., Freire, J., Futrelle, J., McGrath, R., Myers, J. and Paulson, P. (2008). The Open Provenance Model: An Overview. *Provenance and Annotation of Data and Processes*. p. 323–326.

Moreau, L. and Missier, P. (2011). The PROV Data Model and Abstract Syntax Notation. http://www.w3.org/TR/prov-dm/.

Morfonios, K. and Ioannidis, Y. (1 jul 2008). Supporting the data cube lifecycle: the power of ROLAP. *The VLDB Journal*, v. 17, n. 4, p. 729–764.

Mosconi, M. and Porta, M. (jul 2000). Iteration constructs in data-flow visual programming languages. *Computer Languages*, v. 26, n. 2–4, p. 67–104.

Nguyen, H. and Abramson, D. (2012). WorkWays: Interactive workflow-based science gateways. In *2012 IEEE 8th International Conference on E-Science (e-Science)*.

Northrop, L. M. (2002). SEI's software product line tenets. *IEEE Software*, v. 19, n. 4, p. 32–40.

Notredame, C. (mar 2010). Computing multiple sequence/structure alignments with the T-coffee package. *Current Protocols in Bioinformatics / Editoral Board, Andreas D. Baxevanis ... [et Al*, v. Chapter 3, p. Unit 3.8.1–25.

Ocaña, K. A. C. S., Oliveira, D., Dias, J., Ogasawara, E. and Mattoso, M. (7 dec 2011a). Optimizing Phylogenetic Analysis Using SciHmm Cloud-based Scientific

Workflow. In *2011 IEEE Seventh International Conference on e-Science (e-Science)*. . IEEE.

Ocaña, K. A. C. S., Oliveira, D., Ogasawara, E., Dávila, A. M. R., Lima, A. A. B. and Mattoso, M. (2011b). SciPhy: A Cloud-Based Workflow for Phylogenetic Analysis of Drug Targets in Protozoan Genomes. In: Norberto de Souza, O.; Telles, G. P.; Palakal, M.(Eds.). *Advances in Bioinformatics and Computational Biology*. Berlin, Heidelberg: Springer. v. 6832p. 66–70.

Ocaña, K. A. C. S., Oliveira, D. De, Horta, F., Dias, J., Ogasawara, E. and Mattoso, M. (2012). Exploring Molecular Evolution Reconstruction Using a Parallel Cloud-based Scientific Workflow. *Advances in Bioinformatics and Computational Biology*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. v. 7409p. 179–191.

Ocaña, K. A. C. S., Oliveira, F., Dias, J., Ogasawara, E. and Mattoso, M. (2013). Designing a parallel cloud based comparative genomics workflow to improve phylogenetic analyses. *Future Generation Computer Systems*, v. 29, n. 8, p. 2205–2219.

Ogasawara, E. (Dezembro 2011). Uma Abordagem Algébrica para Workflows Científicos com Dados em Larga Escala. Universidade Federal do Rio de Janeiro.

Ogasawara, E., Dias, J., Oliveira, D., Porto, F., Valduriez, P. and Mattoso, M. (2011). An Algebraic Approach for Data-Centric Scientific Workflows. *Proc. of VLDB Endowment*, v. 4, n. 12, p. 1328–1339.

Ogasawara, E., Dias, J., Silva, V., Chirigati, F., Oliveira, D., Porto, F., Valduriez, P. and Mattoso, M. (2013). Chiron: A Parallel Engine for Algebraic Scientific Workflows. *Concurrency and Computation*, v. 25, n. 16, p. 2327–2341.

Ogasawara, E., Oliveira, D., Chirigati, F., Barbosa, C. E., Elias, R., Braganholo, V., Coutinho, A. and Mattoso, M. (2009a). Exploring many task computing in scientific workflows. In *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*. . ACM.

Ogasawara, E., Paulino, C., Murta, L., Werner, C. and Mattoso, M. (2009b). Experiment Line: Software Reuse in Scientific Workflows. (M. Winslett, Ed.)In *Scientific and Statistical Database Management*. . Springer Berlin Heidelberg.

Oliveira, D., Cunha, L., Tomaz, L., Pereira, V. and Mattoso, M. (2009). Using Ontologies to Support Deep Water Oil Exploration Scientific Workflows. In *IEEE International Workshop on Scientific Workflows*.

Oliveira, D., Ocaña, K. A. C. S., Ogasawara, E., Dias, J., Gonçalves, J., Baião, F. and Mattoso, M. (sep 2013). Performance evaluation of parallel strategies in public clouds: A study with phylogenomic workflows. *Future Generation Computer Systems*, v. 29, n. 7, p. 1816–1825.

Oliveira, D., Ocaña, K., Baião, F. and Mattoso, M. (2012). A Provenance-based Adaptive Scheduling Heuristic for Parallel Scientific Workflows in Clouds. *Journal of Grid Computing*, v. 10, n. 3, p. 521–552.

Oliveira, D., Ocaña, K., Ogasawara, E., Dias, J., Baião, F. and Mattoso, M. (4 jul 2011). A Performance Evaluation of X-Ray Crystallography Scientific Workflow

Using SciCumulus. In *IEEE International Conference on Cloud Computing (CLOUD)*. . IEEE.

Oliveira, D., Ogasawara, E., Baião, F. and Mattoso, M. (2010a). SciCumulus: A Lightweight Cloud Middleware to Explore Many Task Computing Paradigm in Scientific Workflows. In *3rd International Conference on Cloud Computing*. , CLOUD '10. IEEE Computer Society.

Oliveira, D., Ogasawara, E., Seabra, F., Silva, V., Murta, L. and Mattoso, M. (2010b). GExpLine: A Tool for Supporting Experiment Composition. *Provenance and Annotation of Data and Processes*. Lecture Notes in Computer Science. Springer Berlin / Heidelberg. p. 251–259.

Oliveira, D., Porto, F., Giraldi, G., Schulze, B. and Pinto, R. C. G. (2010c). Optimizing the pre-processing of scientific visualization techniques using QEF. In *8th International Workshop on Middleware for Grids, Clouds and e-Science*. , MGC '10. ACM.

Olston, C., Reed, B., Silberstein, A. and Srivastava, U. (2008a). Automatic optimization of parallel dataflow programs. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*. , ATC'08. USENIX Association.

Olston, C., Reed, B., Srivastava, U., Kumar, R. and Tomkins, A. (2008b). Pig latin: a not-so-foreign language for data processing. In *Proceedings of SIGMOD*. . ACM.

Özsu, M. T. and Valduriez, P. (2011). *Principles of Distributed Database Systems*. 3. ed. New York: Springer.

ParaView (2011). *ParaView open-source, multi-platform data analysis and visualization application*.

Park, J., Karplus, K., Barrett, C., Hughey, R., Haussler, D., Hubbard, T. and Chothia, C. (11 dec 1998). Sequence comparisons using multiple sequences detect three times as many remote homologues as pairwise methods. *Journal of Molecular Biology*, v. 284, n. 4, p. 1201–1210.

Parks, M. L., De Sturler, E., Mackey, G., Johnson, D. D. and Maiti, S. (2006). Recycling Krylov Subspaces for Sequences of Linear Systems. *SIAM Journal on Scientific Computing*, v. 28, n. 5, p. 1651.

Paton, N., De Aragão, M. A. T., Lee, K., Fernandes, A. A. A. and Sakellariou, R. (2009). Optimizing utility in cloud computing through autonomic workload execution. *Bulletin of the Technical Committee on Data Engineering*, v. 32, n. 1, p. 51–58.

Pearson, W. R. (1990). Rapid and sensitive sequence comparison with FASTP and FASTA. *Methods in Enzymology*, v. 183, p. 63–98.

Pedram, M. (2009). Green computing: reducing energy cost and carbon footprint of information processing systems. In *Proceedings of the 19th ACM Great Lakes symposium on VLSI*. , GLSVLSI '09. ACM.

Pintas, J., Oliveira, D., Ocaña, K., Dias, J. and Mattoso, M. (2012). Monitoramento em Tempo Real de Workflows Científicos Executados em Paralelo em Ambientes Distribuídos. In *VI e-Science workshop*.

Plankensteiner, K., Montagnat, J. and Prodan, R. (2011). IWIR: a language enabling portability across grid workflow systems. In *Proceedings of the 6th workshop on Workflows in support of large-scale science*. , WORKS '11. ACM.

Pradal, C., Dufour-Kowalski, S., Boudon, F., Fournier, C. and Godin, C. (2008). OpenAlea: a visual programming and component-based software platform for plant modelling. *Functional Plant Biology*, v. 35, n. 10, p. 751–760.

Pruitt, K. D., Tatusova, T., Klimke, W. and Maglott, D. R. (jan 2009). NCBI Reference Sequences: current status, policy and new initiatives. *Nucleic Acids Research*, v. 37, n. Database issue, p. D32–D36.

Qian, B. and Goldstein, R. A. (22 sep 2004). Performance of an iterated T-HMM for homology detection. *Bioinformatics (Oxford, England)*, v. 20, n. 14, p. 2175–2180.

Reuillon, R., Leclaire, M. and Rey-Coyrehourcq, S. (oct 2013). OpenMOLE, a workflow engine specifically tailored for the distributed exploration of simulation models. *Future Generation Computer Systems*, v. 29, n. 8, p. 1981–1990.

Romano, P., Bartocci, E., Bertolini, G., De Paoli, F., Marra, D., Mauri, G., Merelli, E. and Milanesi, L. (2007). Biowep: a workflow enactment portal for bioinformatics applications. *BMC Bioinformatics*, v. 8, n. Suppl 1, p. 1–13.

Samak, T., Gunter, D., Goode, M., Deelman, E., Mehta, G., Silva, F. and Vahi, K. (2011). Failure prediction and localization in large scientific workflows. In *Proceedings of the 6th workshop on Workflows in support of large-scale science*. , WORKS '11. ACM.

Santos, I., Dias, J., Oliveira, D., Ogasawara, E., Ocaña, K. and Mattoso, M. (2013). Runtime Dynamic Structural Changes of Scientific Workflows in Clouds. In *Proceedings of the International Workshop on Clouds and (eScience) Applications Management - CloudAM*.

Simmhan, Y. L., Plale, B. and Gannon, D. (2005). A survey of data provenance in e-science. *ACM SIGMOD Record*, v. 34, n. 3, p. 31–36.

Smith, T. (mar 1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, v. 147, n. 1, p. 195–197.

Srirama, S. N., Jakovits, P. and Vainikko, E. (jan 2012). Adapting scientific computing problems to clouds using MapReduce. *Future Generation Computer Systems*, v. 28, n. 1, p. 184–192.

Strachey, C. (1967). *Fundamental Concepts in Programming Language*.

Taylor, I., Shields, M., Wang, I. and Harrison, A. (2007). The Triana Workflow Environment: Architecture and Applications. *Workflows for e-Science*. Springer. p. 320–339.

Texas Advanced Computing Center (2012). DisplayCluster. http://www.tacc.utexas.edu/tacc-projects/displaycluster, (accessed on Sep 13).

Thompson, J. D., Higgins, D. G. and Gibson, T. J. (11 nov 1994). CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, v. 22, n. 22, p. 4673–4680.

Thornton, J. W. and DeSalle, R. (2000). Gene family evolution and homology: genomics meets phylogenetics. *Annual Review of Genomics and Human Genetics*, v. 1, p. 41–73.

Travassos, G. H. and Barros, M. O. (2003). Contributions of In Virtuo and In Silico Experiments for the Future of Empirical Studies in Software Engineering. In *2nd Workshop on Empirical Software Engineering the Future of Empirical Studies in Software Engineering*.

Valduriez, P. and Danforth, S. (Agosto 1992). Functional sql (fsql), an sql upward-compatible database programming language. *Information Sciences*, v. 62, n. 3, p. 183–203.

Wallace, I. M., Orla, O. and Higgins, D. G. (15 apr 2005). Evaluation of Iterative Alignment Algorithms for Multiple Alignment. *Bioinformatics*, v. 21, n. 8, p. 1408–1414.

Wang, J., Crawl, D. and Altintas, I. (2009). Kepler + Hadoop: a general architecture facilitating data-intensive applications in scientific workflow systems. In *4th Workshop on Workflows in Support of Large-Scale Science*. . ACM.

Wang, S., Sturler, E. De and Paulino, G. H. (2007). Large-scale topology optimization using preconditioned Krylov subspace methods with recycling. *International Journal for Numerical Methods in Engineering*, v. 69, n. 12, p. 2441–2468.

Wilde, M., Hategan, M., Wozniak, J. M., Clifford, B., Katz, D. S. and Foster, I. (2011). Swift: A language for distributed parallel scripting. *Parallel Computing*, n. 37(9), p. 633–652.

Wistrand, M. and Sonnhammer, E. L. L. (2005). Improved profile HMM performance by assessment of critical algorithmic features in SAM and HMMER. *BMC Bioinformatics*, v. 6, p. 99.

Wozniak, J., Armstrong, T., Maheshwari, K., Lusk, E., Katz, D., Wilde, M. and Foster, I. (2012). Turbine: A distributed-memory dataflow engine for extreme-scale many-task applications. In *Proceeding of 1st International workshop on Scalable Workflow Enactment Engines and Technologies*.

Xiu, D. and Hesthaven, J. S. (2005). High-Order Collocation Methods for Differential Equations with Random Inputs. *SIAM Journal on Scientific Computing*, v. 27, n. 3, p. 1118–1139.

Zvelebil, M. and Baum, J. (2007). *Understanding Bioinformatics*. 1. ed. Garland Science.