An Algebraic Approach for Scientific Workflows with Large Scale Data

Eduardo Ogasawara

UMA ABORDAGEM ALGÉBRICA PARA WORKFLOWS CIENTÍFICOS COM DADOS EM LARGA ESCALA

Eduardo Soares Ogasawara

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, RJ – BRASIL

DEZEMBRO DE 2011

UMA ABORDAGEM ALGÉBRICA PARA WORKFLOWS CIENTÍFICOS COM DADOS EM LARGA ESCALA

Eduardo Soares Ogasawara

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 ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

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

_____
Prof. Patrick Valduriez, Dr.

_____
Prof. Fábio André Machado Porto, D.Sc.

_____
Prof. Jano Moreira de Souza, Ph.D.

_____
Prof[a]. Claudia Bauzer de Medeiros, Ph.D.

RIO DE JANEIRO, RJ - BRASIL
DEZEMBRO DE 2011

OGASAWARA, EDUARDO SOARES

Uma Abordagem Algébrica para Workflows Científicos com Dados em Larga Escala / Eduardo Soares Ogasawara – Rio de Janeiro: UFRJ/COPPE, 2011.

XIV, 98 p.: il.; 29,7 cm.

Orientadores: Marta Lima de Queirós Mattoso & Patrick Valduriez.

Tese (doutorado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2011.

Referências Bibliográficas: p. 99-111.

1. Workflows Científicos. 2. Álgebra de Workflows. 3. Modelo Relacional. I. Ogasawara, Eduardo Soares. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

*Ao meu pai Tsuneharu (*In memoriam*),*

a minha mãe Ana Maria, a minha esposa Ingrid,

a minha irmã Angélica e aos meus amigos,

por tudo o que representam em minha vida.

À Deus, por tornar tudo possível;

À minha família, por ser o meu pilar mestre;

À Ingrid, por ser o amor da minha vida;

À Marta Mattoso, por sua valiosa orientação e exemplo na minha vida acadêmica;

Ao Patrick Valduriez, pela precisa orientação;

Ao Fabio Porto, pela ajuda nos formalismos e algoritmos;

À Claudia Bauzer de Medeiros, Jano Moreira de Souza e Marco Antônio Casanova, por terem aceitado participar da banca do meu doutorado;

Ao Alexandre Assis, Álvaro Coutinho, Cláudia Werner, Fernanda Baião, Geraldo Xexéo, Geraldo Zimbrão, Guilherme Travassos, Leonardo Murta, Renato Elias, Vanessa Braganholo Murta, pelo apoio no meu doutorado;

Ao Daniel de Oliveira, Jonas Dias, Kary Ocaña, Anderson Marinho, Wallace Martinho pela amizade e parceira em pesquisa;

Ao Albino Aveleda, Gabriel Guerra, Mara Prata, Myriam Costa, Orlando Caldas, pelo apoio no NACAD/UFRJ;

Ao Fernando Chirigati, Pedro Cruz, Ricardo Busquet e, em especial, ao Vítor Silva, pela ajuda na implementação da minha tese;

À Ana Rabello, Cláudia Prata, Juliana Beltrami, Maria Mercedes, Natália Prata, Patrícia Leal, Solange Santos, Sônia Galliano, por sempre me ajudarem com as questões administrativas;

Ao Carlos Schocair, Eduardo Bezerra, Fábio Júnior, Fellipe Duarte, João Quadros, Jorge Abreu, Myrna Amorim, Rafael Castaneda, Renato Mauro, pelo apoio no CEFET/RJ;

À CAPES, pela concessão da bolsa de doutorado entre 2010 e 2011 e ao CNPq, pela concessão da bolsa DTI entre 2008 e 2009, que me permitiram realizar o trabalho,

Agradeço.

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

UMA ABORDAGEM ALGÉBRICA PARA *WORKFLOWS* CIENTÍFICOS

COM DADOS EM LARGA ESCALA

Eduardo Soares Ogasawara

Dezembro/2011

Orientadores: Marta Lima de Queirós Mattoso e Patrick Valduriez

Programa: Engenharia de Sistemas e Computação

Os *workflows* científicos emergiram como uma abstração básica para estruturar experimentos científicos baseados em simulações computacionais. Em muitas situações, estes *workflows* são intensivos, seja computacionalmente seja quanto em relação à manipulação de dados, exigindo a execução em ambientes de processamento de alto desempenho. Entretanto, paralelizar a execução de *workflows* científicos requer programação trabalhosa, de modo *ad hoc* e em baixo nível de abstração, o que torna difícil a exploração das oportunidades de otimização. Visando a abordar o problema de otimizar a execução paralela de *workflows* científicos, esta tese propõe uma abordagem algébrica para especificar o *workflow*, bem como um modelo de execução que, juntos, possibilitam a otimização automática da execução paralela de *workflows* científicos. A tese apresenta uma avaliação ampla da abordagem usando tanto experimentos reais quanto dados sintéticos. Os experimentos foram avaliados no Chiron, um motor de execução de *workflows* desenvolvido para apoiar a execução paralela de *workflows* científicos. Os experimentos apresentaram resultados excelentes de paralelização na execução de *workflows* e evidenciaram, com a abordagem algébrica, diversas possibilidades de otimização de desempenho quando comparados a execuções paralelas de *workflow* de modo *ad hoc*.

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

AN ALGEBRAIC APPROACH FOR DATA-CENTRIC SCIENTIFIC *WORKFLOWS*

Eduardo Soares Ogasawara

December/2011

Advisors: Marta Lima de Queirós Mattoso and Patrick Valduriez

Department: Systems and Computer Engineering

Scientific workflows have emerged as a basic abstraction for structuring and executing scientific experiments in computational simulations. In many situations, these workflows are computationally and data intensive, thus requiring execution in large-scale parallel computers. However, the parallelization of scientific workflows is low-level, *ad hoc* and labor-intensive, which makes it hard to exploit optimization opportunities. To address the problem of optimizing the parallel execution of scientific workflows, we propose an algebraic approach to represent the workflow and a parallel execution model that together enable the automatic optimization of the parallel execution of scientific workflows. We conducted a thorough validation of our approach using both real applications and synthetic data scenarios. The experiments were run in Chiron, a data-centric scientific workflow engine implemented to parallelize scientific workflow execution. Our experiments demonstrated excellent parallel performance improvements obtained and evidenced through our algebraic approach several optimization opportunities when compared to *ad hoc* workflow implementation.

Índice

Índice de Figuras

Índice de Tabela

Capítulo 1 - Introdução

A evolução da ciência da computação nas últimas décadas permitiu a exploração de novos tipos de experimentos científicos baseados em simulação assistida por computadores (Deelman et al. 2009). Com o aumento do desempenho dos computadores, foi possível aumentar também a complexidade dos modelos utilizados nos experimentos científicos. Ao longo do processo de experimentação, os cientistas necessitam desempenhar diversas atividades, e algumas delas estão relacionadas ao encadeamento de programas usados durante as simulações. Cada execução de programa pode produzir uma coleção de dados com determinada sintaxe e semântica. Esta coleção de dados pode ser usada como entrada para o próximo programa a ser executado no fluxo (Taylor et al. 2007a).

O encadeamento destes programas não é uma tarefa trivial e, em muitos casos, pode se tornar uma barreira técnica para a construção de modelos mais sofisticados ou a realização de análise de resultados (Gil et al. 2007a). Este encadeamento é comumente representado por meio de *workflows* científicos. O termo *workflow* científico é usualmente utilizado para descrever *workflows* (Hollingsworth 1995) em qualquer área da ciência, como, por exemplo, biologia, física, química, ecologia, geologia, astronomia e engenharias em geral (Cavalcanti et al. 2005). Estas áreas compartilham algumas características, como a necessidade de manipulação de grandes volumes de dados e demanda de alto poder computacional (Deelman et al. 2009).

Os Sistemas de Gerência de *Workflows* Científicos (SGWfC) são softwares que proveem a infraestrutura para configuração, execução e monitoramento de recursos de *workflows* científicos. Muitos SGWfC foram desenvolvidos (Deelman et al. 2009), como, por exemplo, VisTrails (Callahan et al. 2006), Kepler (Altintas et al. 2004), Taverna (Hull et al. 2006), Pegasus (Deelman et al. 2007), Swift (Zhao et al. 2007) e Triana (Taylor et al. 2007b). Cada um deles apresenta uma forma particular para representação de *workflows* e visa a atender a determinados tipos de aplicações e domínio. Estes SGWfC são responsáveis pela coordenação das chamadas a programas, seja para execução local ou para execução em ambientes distribuídos (Dantas 2005). À coordenação destas chamadas dá-se o nome de orquestração.

1.1 Paralelização de execução de *workflows* científicos

Durante a execução de um *workflow* científico, os cientistas podem precisar explorar o comportamento do modelo científico representado pelo *workflow* por meio do uso de diferentes valores de entrada (parâmetros ou dados distintos). Este comportamento é comumente conhecido como varredura de parâmetros (Samples et al. 2005, Walker e Guiang 2007). Neste cenário, muitas diferentes execuções de *workflows* (tentativas) precisam ser avaliadas. Em muitas situações, estes *workflows* científicos são intensivos,

seja computacionalmente seja quanto em relação à manipulação de dados (Yunhong Gu e Grossman 2008), requerendo a utilização de computadores paralelos para execução em larga escala (Ogasawara et al. 2009b).

Apesar de alguns SGWfC possuírem recursos para execução paralela, paralelizar um *workflow* de larga escala é uma tarefa difícil, *ad hoc* e trabalhosa. Com as soluções existentes, cabe aos cientistas (desenvolvedores dos *workflows*) decidir a ordem, dependências e estratégias de paralelização. Estas decisões, em muitos casos, restringem as oportunidades de otimização para execução do *workflow* científico que poderiam levar a melhorias significativas de desempenho (Ogasawara et al. 2011).

Pode-se ilustrar o problema por meio de uma aplicação crítica da Petrobras, que é usada recorrentemente como exemplo nesta tese. A extração de petróleo em águas ultraprofundas é feita por meio de uma estrutura tubular denominada *riser (Dantas et al. 2004, Rodrigues et al. 2007, Vieira et al. 2008)*, que leva o petróleo do ponto de extração até a plataforma. Manter e reparar os *risers* em águas ultraprofundas é difícil, custoso e crítico para o meio ambiente, uma vez que é necessário evitar derramamento de petróleo. Entender o comportamento dinâmico de um *riser* e sua expectativa de vida útil é crítico para a Petrobras. Desta forma, os cientistas precisam prever a fadiga de *risers* por meio de modelos científicos complexos que devem ser avaliados em diferentes cenários. Como apresentado na Figura 1, executar uma análise de fadiga de *risers* (AFR) requer um *workflow* complexo, composto por atividades tanto intensivas em dados quanto computacionalmente. Um *workflow* típico de análise de fadiga de *risers* (Mattoso et al. 2010) consome grande quantidade de arquivos de entrada contendo informações de *risers*, como malha de elementos finitos, vento, correnteza, estudos de caso (Asetti et al. 2003), e produz arquivos contendo análise de resultados para serem posteriormente avaliados pelos cientistas.

Estes *workflows* podem ser modelados como um grafo direcionado acíclico (do inglês, DAG) (Cormen et al. 2009) de atividades com arestas estabelecendo um fluxo de dados entre as atividades. Na Figura 1, cada retângulo indica uma atividade, as linhas sólidas representam parâmetros de entrada e saída que são transferidos pelas atividades e as linhas tracejadas são parâmetros de entrada e saída compartilhados como arquivos de dados. Para fins de simplicidade, pode-se assumir que todos os dados estão compartilhados entre todas as atividades por meio de um disco compartilhado (Dantas 2005). Durante uma única análise de fadiga de *risers*, um *workflow* pode ter cerca de 2.000 arquivos de entrada (aproximadamente 1 GB) e produzir cerca de 6.500 arquivos (22 GB) em um *workflow* de nove atividades, incluindo: extração dos dados de *risers*, análise estática dos *risers*, análise dinâmica dos *risers*, análise de tensão, análise de curvatura, agrupamento e compressão dos resultados. Algumas atividades, como, por exemplo, a análise dinâmica, são executadas repetidas vezes a partir de diferentes arquivos de entrada. Dependendo do refinamento da malha de elementos finitos e outras

informações dos *risers*, cada execução isolada pode levar minutos até a conclusão. O processamento sequencial deste *workflow* com um conjunto de dados pequeno invoca 1.432 atividades e pode levar 16 horas até a conclusão.

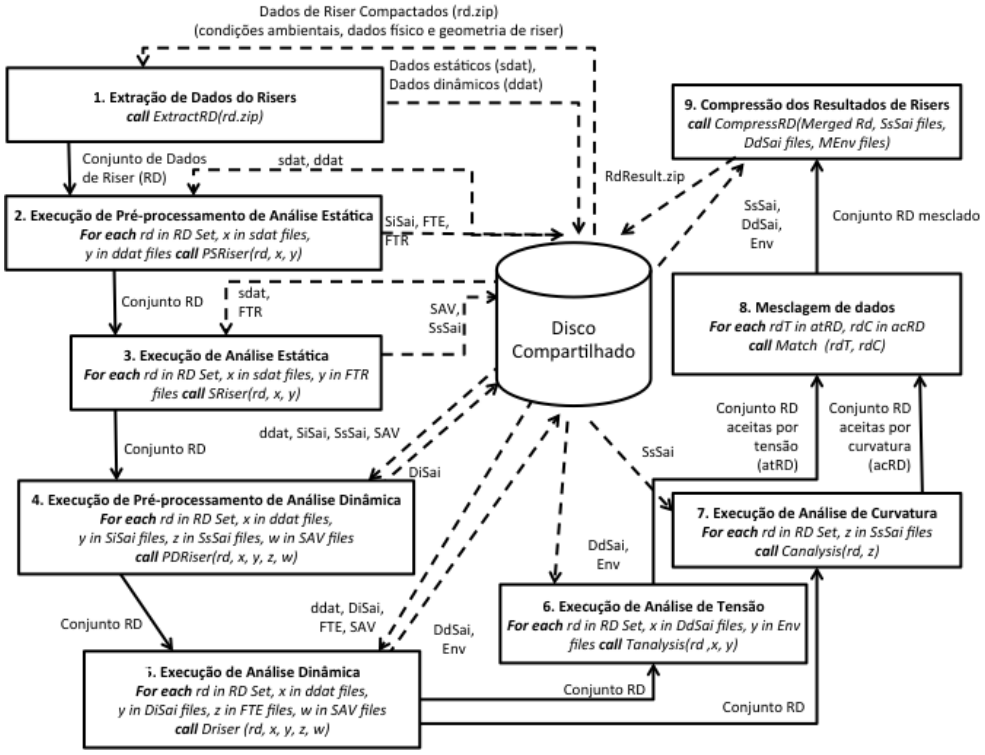

Figura 1 - *Workflow* para Análise de Fadiga de Risers (AFR).

Alguns SGWfC apoiam a execução paralela das atividades de *workflows* científicos. O Swift, por exemplo, permite que os cientistas possam especificar *workflows* paralelos por meio de uma linguagem de script (Wilde et al. 2009). Similarmente, implementações usando *MapReduce* (Dean e Ghemawat 2008), como, por exemplo, o Hadoop (Wang et al. 2009), possibilitam que algumas atividades sejam paralelizadas em conjunto durante a execução do *workflow*. Entretanto, estas soluções requerem que os cientistas tenham que especificar estratégias de paralelização programaticamente, o que limita as oportunidades para otimização automática da execução do *workflow*. A otimização da execução fica a cargo do escalonador, que distribui as atividades entre os nós computacionais de um ambiente de processamento de alto desempenho (PAD) a partir de uma codificação rígida feita pelos cientistas.

De modo a ilustrar as possibilidades de otimização, considere o cenário da Figura 1. Seria melhor deixar a atividade 2 consumir todos os dados de entrada antes de passar o processamento para a atividade 3 ou seria melhor iniciar imediatamente a execução da atividade 3 para cada dado produzido pela atividade 2? Esta é uma decisão difícil que, quando deixada a cargo dos cientistas, torna o desenvolvimento do *workflow* improdutivo. Além disto, para possibilitar que o SGWfC automaticamente decidisse pela melhor alternativa, seriam necessários alguns recursos de otimização da linguagem

de especificação do *workflow*. Por exemplo, se o SGWfC pudesse identificar na linguagem que a atividade 7 é seletora, *i.e.*, capaz de filtrar dados e identificar que os dados de entrada para ela não são modificados desde a atividade 3, então a execução da atividade 7 poderia ser antecipada e ter a sua execução posicionada entre as atividades 3 e 4. Isto poderia reduzir a quantidade de dados a serem processados pelas atividades 4 e 5, ambas computacionalmente intensivas. Nestes dois exemplos apresentados, podem-se perceber oportunidades de otimização na execução do *workflow*. Entretanto, as soluções existentes realizam a execução do *workflow* conforme especificado pelo usuário. Nota-se que faltam semântica e abstração para apoiar a otimização automática da execução do *workflow*.

1.2 Abordagem algébrica

A hipótese geral deste trabalho é que a representação de *workflows* por meio de uma abordagem algébrica para *workflows* científicos traz benefícios à otimização da execução paralela de *workflows* científicos se comparada com as abordagens atuais. No cenário de experimentos científicos em larga escala, trazer benefício implica fazer-se valer das expressões algébricas para gerar especificações equivalentes do *workflow*, mas que possuem desempenhos distintos. Técnicas de otimização permitem a escolha de uma expressão algébrica "ótima". Assim, a adoção de uma abordagem algébrica para especificar *workflows* científicos permite realizar a otimização da execução paralela de *workflows* científicos de modo sistemático, *i.e.*, de modo que o cientista não tenha que se preocupar com codificações paralelas, mas sim com a especificação do *workflow*.

De fato, com a realização de inúmeros experimentos de execução paralela em aplicações reais (Barbosa et al. 2009, Coutinho et al. 2010, 2011, Ogasawara et al. 2009a, 2009b, Silva et al. 2011b), foi possível observar características e necessidades comuns presentes nestas aplicações, bem como a identificação de padrões de consumo e produção de dados em paralelo. A experiência obtida durante a execução destes *workflows* viabilizou a elaboração de uma abordagem algébrica para *workflows* científicos.

Na proposta algébrica deste trabalho, as atividades em um *workflow* são regidas por operações algébricas que definem como as atividades devem consumir e produzir os seus dados. Nesta formalização, os dados são tratados de modo uniforme, ou seja, as atividades de um *workflow* apenas consomem e produzem conjunto de tuplas (relações). Desta forma, todas as operações algébricas operam em cima de relações, possibilitando uma representação declarativa do *workflow* na qual as atividades consomem e produzem relações. Esta representação facilita a geração de *workflows* científicos prontos para execução em paralelo. Para tanto, este trabalho apresenta o conceito de ativação de atividade (Ogasawara et al. 2011), inspirado em ativação de tuplas em banco de dados (Bouganim et al. 1996), que possibilita que a representação declarativa do *workflow* seja

executada de modo transparente em ambientes de PAD, como, por exemplo, *clusters*, grades computacionais e nuvens.

A partir desta proposta algébrica, os *workflows* podem ser automaticamente otimizados para execução paralela por meio de transformações algébricas. Estas transformações visam a alterar a dependência de dados necessários à execução das atividades. Cada transformação aplicada, apesar de garantir que o *workflow* produza o mesmo resultado, traz uma diferença em termos de custo computacional. Em outras palavras, expressões algébricas equivalentes produzem diferentes planos de execução do *workflow*. Pode-se avaliar o custo destes planos por meio de uma função de custo. Desta forma, a abordagem algébrica possibilita a visualização do problema de execução paralela de *workflow* de modo análogo à otimização de consultas em bancos de dados relacionais.

De modo a avaliar a abordagem algébrica foi projetado e desenvolvido o Chiron, um motor de execução de *workflows* elaborado a partir de uma linguagem declarativa capaz de executar *workflows* especificados por meio de operações algébricas. O Chiron foi utilizado para avaliar a adequação da abordagem algébrica como primitiva para paralelização da execução de *workflows*, bem como para avaliar o potencial de otimização algébrica no que tange à melhoria de desempenho.

Esta tese contextualiza sua pesquisa na área de "ciência apoiada pela computação" (do inglês: *e-science* ou *cyberinfrastructure*), contribuindo nesta área por apresentar:

- uma álgebra para *workflows* científicos com operações que trazem semântica às atividades;
- um modelo de execução para esta álgebra baseado no conceito de ativação de atividades, que possibilita a distribuição e paralelização das atividades de modo transparente;
- um conjunto de transformações algébricas e algoritmo de otimização capazes de potencializar a execução paralela de *workflows* científicos;
- um motor de execução de *workflows* paralelo, denominado Chiron, que explora o paralelismo de dados em *workflows*;
- uma avaliação extensiva baseada na utilização do Chiron, que apoia a abordagem algébrica. Esta avaliação foi constituída por experimentos reais da área de petróleo e bioinformática, como também por dados sintéticos, todos executados em computadores paralelos de 256 núcleos, o que possibilitou observar os benefícios da abordagem.

1.3 Organização da tese

Além desta introdução, esta tese está organizada em outros sete capítulos. O Capítulo 2 apresenta conceitos relacionados a experimentos científicos, *workflows* científicos e aos principais esforços realizados no apoio à composição, execução e análise. O Capítulo 3 apresenta os modelos de execução paralela de *workflows* científicos. O Capítulo 4, referente à abordagem proposta, detalha a álgebra de *workflows*, o modelo de execução e as técnicas de otimização para execução paralela. O Capítulo 5 apresenta o Chiron, um motor de execução de *workflow* desenvolvido para explorar paralelismo de dados em larga escala e utilizado para avaliar a abordagem algébrica. O Capítulo 6 apresenta a avaliação experimental da abordagem algébrica, contemplando a avaliação do modelo de execução e das heurísticas de otimização. O Capítulo 7 apresenta trabalhos relacionados. Finalmente, o Capítulo 8 conclui o documento apresentando os principais resultados alcançados e os desdobramentos para diversos trabalhos futuros na direção dos desafios da gerência de dados científicos.

Capítulo 2 - Experimentos científicos

Este capítulo aborda os principais conceitos relacionados a experimentos científicos, em particular aqueles baseados em simulação computacional. O capítulo detalha o ciclo de vida de um experimento científico, contextualizando o uso de *workflows* científicos para apoiá-los. Além disto, são apresentadas as principais formas de representação de *workflows* científicos.

2.1 Experimentos científicos baseados em simulação

Um experimento científico pode ser definido como um "teste sobre condições controladas, que é feito para demonstrar a verdade, examinar a validade de uma determinada hipótese ou determinar a eficácia de algo previamente não explorado" (Soanes e Stevenson 2003). Um experimento científico também pode ser definido como "uma situação, criada em laboratório, que visa a observar, sob condições controladas, o fenômeno de interesse" (Jarrard 2001). O termo "controle" é usado para indicar que existem esforços para reduzir, ao máximo possível, os erros que podem ocorrer por influência do meio onde o experimento é realizado (Jarrard 2001). Baseado nestas definições, pode-se concluir que um experimento científico está associado a um conjunto de ações controladas. Estas ações controladas incluem variações de testes e os resultados são usualmente comparados entre si com vistas a corroborar a aceitação ou rejeição de uma hipótese.

Experimentos científicos baseados em simulação, ou simplificadamente nesta tese, experimentos científicos, são aqueles em que os objetos em estudo são representados por modelos computacionais (Travassos e Barros 2003). Eles são comuns em diversas áreas de pesquisa, tais como dinâmica de fluidos computacional (do inglês, CFD) (Guerra et al. 2011, Ogasawara et al. 2009b), bioinformática (Coutinho et al. 2010, Digiampietri et al. 2007, Greenwood et al. 2003, Lemos et al. 2004, Ocaña et al. 2011a, Romano et al. 2007), agricultura (Fileto et al. 2003), visualização (Brandao et al. 2009, Callahan et al. 2006, Dávila et al. 2008, Oliveira et al. 2010c, Parker e Johnson 1995), prospecção de petróleo em águas profundas (Carvalho 2009, Martinho et al. 2009, Ogasawara et al. 2011, Oliveira et al. 2009a) e energia escura (Governato et al. 2010). Grande parte dos experimentos nestas categorias é considerada como de larga escala por sua natureza exploratória e pelo amplo conjunto de dados a ser processado, que demanda muitos recursos computacionais. Experimentos em larga escala demandam muitas horas de processamento em computadores paralelos. Pela grande quantidade de chamadas a programas e de número de nós computacionais envolvidos, gerenciar a execução paralela deste tipo de experimento se torna uma tarefa árdua (grande quantidade de dados manipulados, falhas de execução, duração do experimento). Este gerenciamento torna-se ainda mais complicado quando há a

preocupação de registro da execução para análise posterior e para tentar sua reprodutibilidade (Davidson e Freire 2008). Esse é um dos princípios de um experimento científico, isto é, toda a execução, bem como os dados consumidos e produzidos devem ser registrados.

Neste cenário, é comum a utilização de *workflows* para apoiar a condução do experimento. Os *workflows* científicos vêm sendo utilizados como uma abstração para modelar o fluxo de atividades e de dados. Nos *workflows* científicos, estas atividades são geralmente programas ou serviços que representam algum algoritmo ou método que deve ser aplicado ao longo do processo de experimentação (Barker e van Hemert 2008). A execução de um *workflow* pode ser vista como um conjunto de ações controladas do experimento. Os SGWfC são focados na gerência da execução destes *workflows*. Cada execução pode representar uma das tentativas conduzidas no contexto do experimento científico para avaliar uma hipótese. O histórico do experimento é, então, representado por todas as execuções distintas que compõem esse experimento científico. Entretanto, executá-las é apenas uma das etapas do ciclo de vida do experimento científico em larga escala. Apoiar o experimento científico significa apoiar o ciclo do experimento científico como um todo (Bose e Frew 2005, Goble et al. 2003, Hoffa et al. 2008, Livny et al. 1994).

2.2 Ciclo de vida do experimento científico

O termo ciclo de vida do experimento é geralmente usado para descrever uma sequência de passos realizados durante um determinado estudo. Ele denota um conjunto de tarefas através das quais os cientistas iteram durante um estudo experimental (Deelman et al. 2009). Este ciclo pode ser executado diversas vezes até que a hipótese do experimento seja confirmada ou refutada. Apesar de existirem várias propostas para ciclo de vida de experimentos (Oinn et al. 2004, Taylor et al. 2007a), de acordo com Mattoso et al. (2010), o processo de experimentação científica apresentado na Figura 2 pode ser resumidamente representado por um ciclo formado por três fases: composição, execução e análise.

A fase de composição (que é, em geral, a fase inicial do ciclo) é responsável por definir e modelar todo o experimento, estabelecendo o sequenciamento lógico das atividades, os tipos de dados de entrada, de parâmetros e de dados gerados. A composição é a fase que trata da definição, edição e manipulação dos *workflows*. Neste contexto, modelos científicos, métodos, algoritmos, programas, serviços e dados são os ativos usados pelo cientista ao longo do ciclo de vida do experimento (Mattoso et al. 2010). Esta fase pode ser decomposta em duas subfases: concepção e reutilização. A subfase de concepção é responsável pela especificação e modelagem do experimento (Martinho et al. 2009, Mattoso et al. 2009, Pereira e Travassos 2010). Durante a concepção, o objetivo é capturar as informações do experimento, ou seja, o protocolo do

experimento (Juristo e Moreno 2001), e modelá-lo como *workflows*. Em um cenário ideal, os cientistas deveriam ser capazes de elaborar uma definição abstrata do experimento e a partir dela conduzir escolhas de métodos específicos para apoiá-los na avaliação de uma hipótese (Mattoso et al. 2010, 2009). A subfase de reutilização é responsável por carregar as informações de execuções passadas para apoiar a criação, adaptação ou simples reutilização de *workflows* para uma nova execução do experimento (Ogasawara et al. 2009c, Oliveira et al. 2010b, 2008).

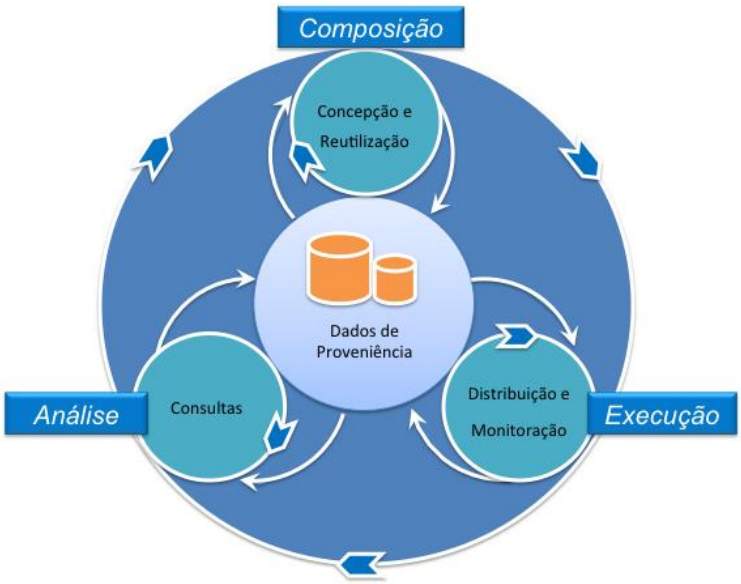

Figura 2 - Ciclo do experimento científico - Mattoso et al. (2010).

A fase de execução é responsável pela materialização do experimento, obtida a partir do resultado da fase de composição. Nesta fase de execução o *workflow* é instanciado de modo a ser executado em uma determinada infraestrutura computacional, definindo-se os dados exatos que devem ser utilizados como entrada e como valores de parâmetros, por meio de um SGWfC (Zhao et al. 2005). Durante esta fase, várias informações relacionadas à produção dos resultados do experimento precisam ser coletadas para posterior análise. Esta fase pode ser dividida em duas subfases: distribuição e monitoração (Mattoso et al. 2010). A distribuição engloba as ações necessárias para que se executem as atividades do *workflow*, dentre as quais destacam-se aquelas que devem ser executadas paralelamente em ambientes distribuídos (Abramson et al. 2008, Meyer et al. 2007, Ogasawara et al. 2010, 2009b, Oliveira et al. 2010a, Smanchat et al. 2009, Sonmez et al. 2010). A monitoração está relacionada à necessidade de se conferir o estado da execução do *workflow*, uma vez que esta pode ter longo período de duração (Marinho et al. 2009, McGough et al. 2007, Medeiros et al. 1995).

A fase de análise é responsável pelo estudo dos dados gerados provenientes das fases de composição e execução. A proveniência (Freire et al. 2008), também chamada de histórico do experimento (*i.e.* sua definição, seus dados produzidos, etc.), é essencial para se preservar os dados do experimento, determinar suas qualidades e formas de

reprodução. As consultas aos dados de proveniência podem incluir informações tanto da fase de composição quanto da fase de execução do experimento. Denomina-se proveniência prospectiva (Freire et al. 2008) o conjunto de informações da fase de composição correspondente à definição dos passos que precisam ser executados para se gerar os resultados do experimento. Denomina-se proveniência retrospectiva (Freire et al. 2008) o conjunto de informações da fase de execução referente aos passos efetivamente executados para a produção dos resultados. A partir dos dados de proveniência se obtém informações sobre a produção, a interpretação e a validação do resultado científico alcançado em um experimento (Marinho et al. 2009, 2010a, 2010b).

2.3 *Workflows* científicos

O termo *workflow* tem suas origens associadas ao processo de automação de escritórios. Essa tecnologia originou-se na década de 70, quando o foco era oferecer soluções voltadas para a geração e distribuição de documentos em uma organização, visando à redução da manipulação de documentos em papel (Hollingsworth 1995, Mattos et al. 2008). A definição de *workflows,* segundo *Workflow Management Coalition* (WfMC) (WfMC 2009), é: "A automação de um processo de negócio, completo ou apenas parte dele, através do qual documentos, informações ou tarefas são transmitidos de um participante a outro por ações, de acordo com regras procedimentais" .

Apesar de os *workflows* terem inicialmente surgido no ambiente de negócios (Aalst e Hee 2002), cada vez mais as ciências os utilizam. O termo *workflow* científico é usado para descrever *workflows* em algumas destas áreas da ciência, nas quais se compartilham as características de manipulação de grandes volumes de dados e de demanda por alto poder de processamento computacional (Deelman et al. 2009). Ao contrário dos *workflows* de negócios, que representam instâncias de tarefas bem definidas e que comumente se repetem como parte de uma estrutura de atividades de uma organização, os *workflows* científicos têm a característica de serem mais dinâmicos e frequentemente se tem a necessidade de adaptá-los para explorar novas técnicas, métodos ou modelos científicos (Barga e Gannon 2007). Outra importante característica é a diferença ou heterogeneidade de dados presentes em *workflows* científicos. Muitas atividades nestes *workflows* manipulam dados em formatos específicos para determinados programas científicos. O formato destes dados pode ser binário ou textual semiestruturado (Gil et al. 2007a).

2.4 Representação de *workflows* científicos

A representação de *workflows* científicos pode tomar diferentes formas e se apresentar em diferentes níveis de abstração (Mattoso et al. 2008). Normalmente, a geração ou composição de modelos de *workflows* é apoiada pela presença de ferramentas visuais que permitem a edição, gravação e execução destes sistemas. Adicionalmente, a

modelagem e o estudo de especificações formais garantem a integridade da ferramenta e a formalização do seu comportamento, visando à redução de possíveis ambiguidades e possibilitando verificações e análises formais (Mattoso et al. 2009).

Existe um conjunto de modelos matemáticos ou computacionais a partir do qual várias linguagens ou representações de *workflows* são elaboradas. Destacam-se as que são baseadas em grafos direcionados, Redes de Petri (Hoheisel e Alt 2007), Linguagem Unificada de Modelagem (do inglês, UML) (Fowler 2003) e Álgebra de Processos (Baeten et al. 2009).

2.4.1 Grafos

Uma das mais comuns representações de *workflows* científicos são as baseadas em grafos direcionados, que podem ser tanto acíclicos (do inglês, DAG) quanto cíclicos (do inglês, DCG). Quando representadas por grafos as atividades correspondem aos vértices e as dependências entre estas atividades correspondem às arestas. Apesar de muitos SGWfC usarem intrinsicamente grafos para a sua modelagem, o armazenamento destes grafos é feito por meio de uma linguagem própria (Deelman et al. 2009).

A maioria dos SGWfC adota a XML como linguagem para representação do *workflow* como um grafo. Este é o caso do Kepler (Altintas et al. 2004, Bowers et al. 2008b), Triana (Taylor et al. 2007b, 2003), Pegasus (Deelman et al. 2007, Gil et al. 2007b, Lee et al. 2008), VisTrails (Callahan et al. 2006, Scheidegger et al. 2008) e Taverna (Hull et al. 2006, Oinn et al. 2004, 2007). Em alguns casos, a representação é baseada na linguagem de processo de negócios (do inglês, BPEL), como, por exemplo, no caso do OMII-BPEL (Emmerich et al. 2005). A Figura 3 apresenta a representação de *workflow* no Kepler tanto como grafo quanto como XML.

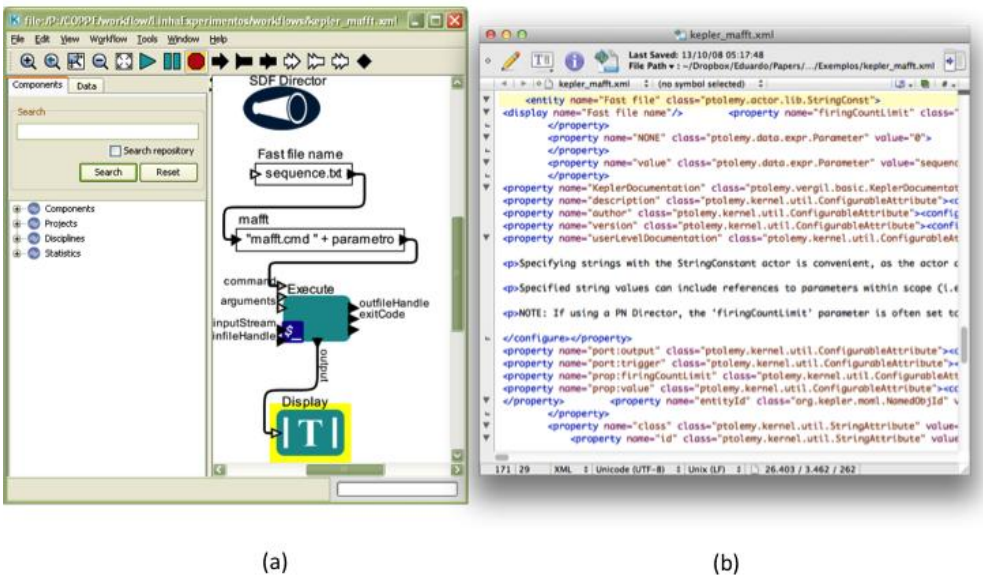

Figura 3 - Representação em grafo do Kepler (a); Visão equivalente em XML (b).

2.4.2 Redes de Petri

As redes de Petri podem ser interpretadas como um caso particular de grafos direcionados. Elas são comumente usadas para representar sistemas distribuídos discretos, podendo, ainda, ser usadas para representar processos (Hoheisel e Alt 2007). As redes de Petri servem para descrever processos distribuídos ao estenderem as máquinas de estados com concorrência. Nas redes de Petri os vértices são divididos em dois tipos: de localização e de transição. As arestas são direcionadas e sempre conectam vértices de tipos diferentes, o que faz com que o grafo seja bipartite.

Ao se usar as redes de Petri para se modelar a execução de um *workflow*, cada vértice pode armazenar um ou mais *tokens*. Diferentemente de sistemas mais tradicionais de processamento de dados, que comumente processam somente um único fluxo de *tokens* entrantes, os vértices de transição de redes de Petri podem consumir e produzir *tokens* de múltiplas localizações. Os vértices de transições agem em *tokens* de entrada por um processo denominado disparo. Quando uma transição é disparada, ela consome os *tokens* de suas posições de entrada, realiza alguma tarefa de processamento, e realoca um número específico de *tokens* nas suas posições de saída. Isso é feito atomicamente. Como os disparos são não-determinísticos, as redes de Petri são muito utilizadas para modelar comportamento concorrente em sistemas distribuídos.

Desta forma, no contexto de *workflows*, tem-se que um *workflow* termina quando não há nenhum *token* com possibilidade de execução. As redes de Petri são usadas em alguns sistemas, como *Grid Workflow Description Language* (G*Workflow*DL) (Alt et al. 2006) e *Grid Flow Description Language* (GFDL) (Guan et al. 2006). A Figura 4 apresenta um exemplo de redes de Petri para representar *workflows* científicos. Uma descrição bem detalhada sobre rede de Petri pode ser vista em Hoheisel e Alt (2007).

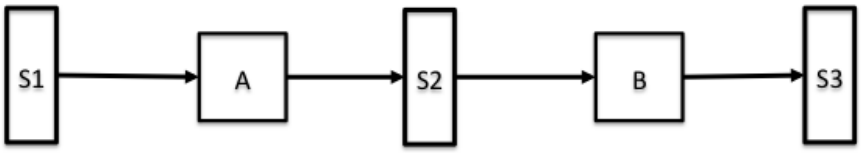

Figura 4 - Exemplo de uma rede de Petri (adaptado de Guan et al., (2006)).

2.4.3 UML

O modelo UML é a linguagem padrão para modelagem de *software* orientado a objetos. Um dos recursos de modelagem presentes na UML é o diagrama de atividades, que pode ser usado para explicitar as dependências entre as atividades, servindo, desta forma, para modelar *workflows*. O diagrama de atividades pode ser usado tanto para elicitação (Martinho et al. 2009) quanto para execução (Qin et al. 2007) de *workflows* científicos.

O sistema Askalon (Fahringer et al. 2005a) usa o diagrama de atividades como representação gráfica de *workflows*. Este diagrama de atividades é traduzido pela ferramenta para representação em XML (Fahringer et al. 2005b), que é usada como insumo para o motor de execução de *workflow* do Askalon. A Figura 5 apresenta um *workflow* modelado em diagrama de atividades no Askalon.

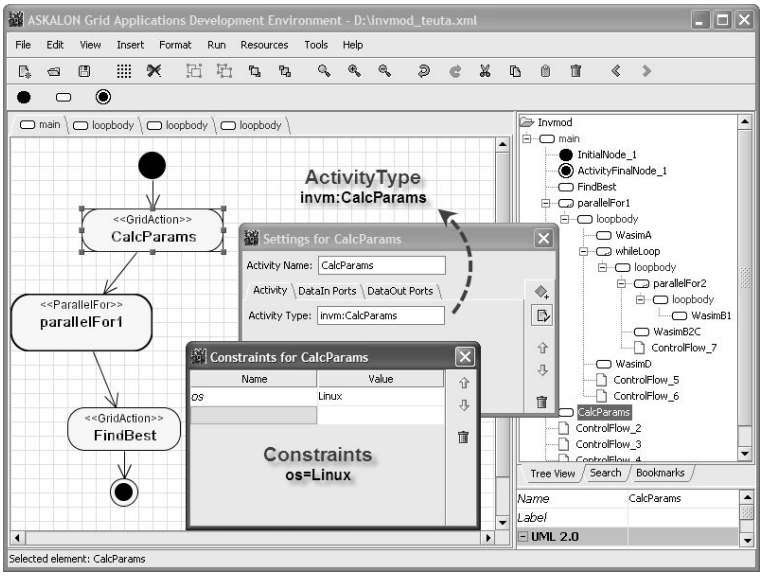

Figura 5 - Exemplo de *workflow* modelado em diagrama de atividades no Askalon (Qin et al. 2007).

2.4.4 Álgebra de Processos

A álgebra de processos pode ser entendida como um estudo do comportamento de sistemas paralelos ou distribuídos por meio de uma abordagem algébrica (Baeten 2005). O comportamento de um sistema pode ser descrito como o número de eventos ou ações que um sistema pode executar.

Por meio da álgebra de processos podem ser representados diversos padrões de *workflow* (Aalst et al. 2003), conforme resumido em Mattos et al. (2008) e apresentado na Tabela 1. Comumente, a álgebra de processos é usada para realizar verificação de *workflows* (Baeten 2005, Silva et al. 2010a). Em alguns casos, também pode ser usada para aperfeiçoar os processos por meio de transformações e análise algébricas (Braghetto 2006). A Figura 6 apresenta uma forma de representar um processo de modo algébrico.

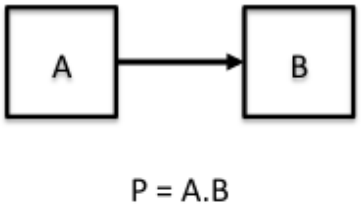

Figura 6 – Exemplo de álgebra de processos

Tabela 1 - Operações algébricas para Processos

Operação	Descrição
.	Indica uma sequência de expressões algébricas. Ex.: A.B indica que a atividade B é apta para execução após o término da atividade A.
||	Indica que duas expressões podem ser executadas em paralelo e independentemente. Ex.: C || D indica que as atividades C e D podem ser executadas em paralelo e independentemente.
+	Indica que uma das duas expressões participantes desta operação é executada. Ex.: E + F indica que ou a atividade E ou a atividade F é executada.

2.5 Modelos de execução de *workflows*

Paralelamente aos modelos de representação existem os modelos de execução. Os modelos de execução de *workflows* se organizam predominantemente em duas classes gerais: modelos baseados em fluxo de controle e baseados em fluxos de dados. Os dois modelos são semelhantes à medida que especificam interações entre atividades que fazem parte do *workflow*, mas se diferem na forma pela qual implementam estas interações (Deelman et al. 2009).

Os *workflows* baseados em fluxo de dados são projetados para apoiar aplicações orientadas a dados. As dependências representam o fluxo de dados entre as atividades do *workflow*, ou seja, entre o produtor e o consumidor. A maior parte dos *workflows* definidos por fluxo de dados é naturalmente simples e, ao contrário do caso de fluxos de controle, costuma se estruturar a partir de uma sequência de atividades, principalmente por meio de *pipelines*, que são trechos do *workflow* cujo dado produzido por uma atividade é consumido imediatamente por outra atividade subsequente no fluxo, conforme abordado por Casanova e Lemos (2007). Os *workflows* baseados em fluxo de controle têm as conexões entre as atividades representando a transferência do controle da execução de uma atividade para a subsequente. A partir dos fluxos de controle podem-se apoiar tanto sequências quanto estruturas mais sofisticadas, como condicionais e iterações (Shields 2007).

Alguns modelos de execução de *workflow* apresentam uma combinação híbrida de fluxos de controle e fluxos de dados. Estes modelos híbridos usam os dois tipos de dependências, mas comumente priorizam um dos modelos. Sistemas como Triana apresentam predominantemente fluxo de dados e, quando necessário, se fazem valer do fluxo de controle. Sistemas como o Kepler podem definir que um determinado sub-*workflow* apresente um modelo diferente do modelo do *workflow* principal. Alguns

modelos híbridos não apresentam nenhuma construção de controle, mas introduzem estruturas de controle de laços e condicionais por meio de componentes específicos que aceitam receber dados que servem apenas para apoiar o controle (Deelman et al. 2009).

Além da vertente do tipo de fluxo, existe também a caracterização segundo o tipo de dado que trafega pelos fluxos nos *workflows* científicos. Considerando a vertente do tipo de dado trafegado, há sistemas que apoiam o conceito de transferência de objetos e outros que apoiam o conceito de transferência de coleção de dados. Este último é o caso do modelo e projeto orientado a coleções (do Inglês, COMAD) (Bowers et al. 2008a), que possibilita o processamento de coleções aninhadas e filas de coleções de *tokens*. Os documentos em XML são decompostos em filas de *tokens* e processados por atividades específicas.

2.6 Considerações sobre *workflows* científicos

Observa-se que existem diferentes modelos de representação e execução para *workflows* científicos. No que tange aos modelos de representação, os baseados em grafos são os mais usados, sendo comumente representados em XML. Os modelos de rede de Petri e baseados em álgebra introduzem mecanismos para apoiar representações paralelas e distribuídas.

No que tange aos modelos de execução, tem-se predominantemente a adoção dos fluxos de controle e de dados. A fixação em apenas um dos tipos de fluxos pode limitar a capacidade de representação dos *workflows*. Associada à questão do fluxo no *workflow* tem-se a dimensão do tipo de informação que trafega pelos fluxos, que pode ser mensagens, tokens, objetos simples ou coleções de objetos.

A combinação dos modelos de representação e de execução faz com que se aumente ou não a complexidade das máquinas de execução de *workflows* científicos (Deelman et al. 2009). Cabe ressaltar que, independentemente do modelo escolhido, observa-se uma ausência de uniformidade quanto à caracterização dos dados e do tipo de dados manipulados nas atividades e nas relações de dependências entre elas.

Capítulo 3 - Modelos de execução paralela de *workflows* científicos

Experimentos em larga escala são computacionalmente intensivos, podendo envolver o processamento de muitas atividades e grande manipulação de dados. Inicialmente, para caracterizá-los, são apresentados na seção 3.1 cenários de experimentos científicos que demandam PAD, mais particularmente o caso de varredura de parâmetros. Considerando que normalmente a execução destes experimentos é feita de maneira paralela em recursos computacionais distribuídos, como *clusters*, grades computacionais ou nuvem, na seção 3.2 são detalhados estes tipos de ambientes.

Segundo Yu e Buyya (2005), é possível classificar os SGWfC em duas diferentes categorias: os SGWfC que executam os *workflows* em ambientes distribuídos, que, por simplificação, podem ser chamados de executores distribuídos, e aqueles que executam os *workflows* localmente no *desktop* dos cientistas (*i.e.*, o controle da execução é completamente local), que, por simplificação, podem ser chamados de executores locais. Na primeira categoria, têm-se os SGWfC como Pegasus (Deelman et al. 2007), Swift (Zhao et al. 2007), Askalon (Wieczorek et al. 2005) e Triana (Taylor et al. 2007b), que apoiam a execução distribuída e paralela. Na segunda categoria, têm-se os SGWfC como Kepler, VisTrails e Taverna, que focam na usabilidade, proveniência e semântica. No decorrer deste capítulo também são apresentadas abordagens existentes que dão apoio à execução destes experimentos em ambientes distribuídos, mais particularmente as nativas (seção 3.3) e híbridas (seção 3.4).

Finalmente, as questões abordadas na seção 3.5 deste capítulo apresentam as dificuldades para os cientistas executarem *workflows* em ambientes de PAD de forma transparente. Estes questionamentos conduzem à motivação para a abordagem algébrica.

3.1 Varredura de parâmetros

Conforme mencionado anteriormente, em função da natureza exploratória dos métodos científicos (Jarrard 2001), um experimento científico pode requerer a exploração de um determinado *workflow* usando diferentes parâmetros ou diferentes dados. Este tipo de cenário exploratório é um candidato para paralelização por uma técnica denominada varredura de parâmetros (Abramson et al. 2009a, 2009b, 2008, Samples et al. 2005, Smanchat et al. 2009, Walker e Guiang 2007).

Considere um *workflow* científico *Wf* contendo um conjunto de atividades $\{Y_1, Y_2, \ldots, Y_n\}$. Conforme apresentado em Ogasawara et al. (2009b), o paralelismo por varredura de parâmetros é caracterizado pela execução simultânea de uma atividade Y_i do *workflow Wf*, na qual cada execução da atividade é feita usando um diferente

conjunto de valores para os parâmetros de sua entrada. Exemplificando, suponha-se que uma atividade Y_i do *workflow* possua os parâmetros de entrada $\{pe_1, pe_2, pe_3\}$ e produza os seguintes parâmetros de saída $\{ps_1, ps_2\}$. Suponha-se, também, que o domínio para os valores destes parâmetros pe_1, pe_2 e pe_3 sejam, respectivamente, $D_{pe1} = \{x, x'\}$, $D_{pe2} = \{y, y'\}$ e $D_{pe3} = \{z, z'\}$. A atividade Y_i pode consumir os valores *x*, *y*, *z* dos parâmetros pe_1, pe_2 e pe_3, produzindo *k*, *w* para os parâmetros de saída ps_1 e ps_2, respectivamente, e uma outra instância pode consumir os valores *x'*, *y'* e *z'* , produzindo *k'*, *w'*.

A paralelização da varredura de parâmetros para uma determinada atividade Y_i do *workflow* é realizada alocando-se os valores dos parâmetros de entrada para serem executados independentemente em cada núcleo computacional em um ambiente de PAD. Estas configurações são formadas por meio do consumo ordenado destas combinações de parâmetros. Assim, seja $\{E_{1, ..., } E_m\}$ o conjunto dos valores de entrada para cada execução isolada, a paralelização da execução das atividades Y_i ocorre por meio de um particionamento dos *m* diferentes valores de parâmetros de entrada em *n* núcleos computacionais. Ao final da execução, um conjunto $\{S_{1, ..., } S_m\}$ de valores de parâmetros de saída é produzido. A Figura 7 apresenta uma esquematização deste tipo de paralelismo, no qual a mesma atividade Y_i é replicada em *n* diferentes nós.

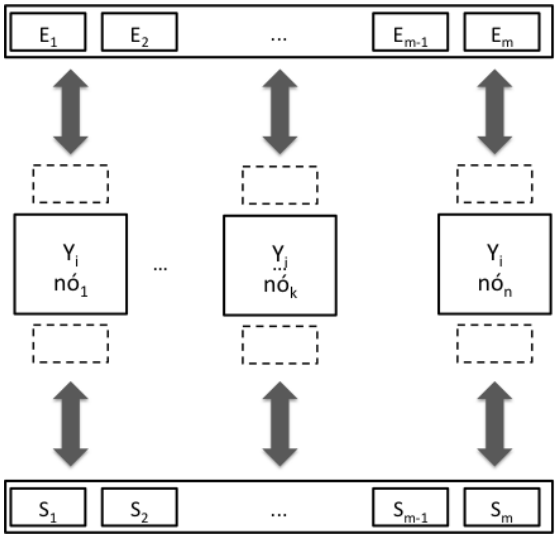

Figura 7 – Varredura de parâmetros em atividades de *workflows*

Como existem diferentes infraestruturas computacionais para executar varredura de parâmetros, comumente há a inclusão de atividades no *workflow* para se atender a exigências inerentes à infraestrutura na qual o *workflow* deve ser executado. Estas exigências podem estar relacionadas à necessidade de execução em ambientes heterogêneos, como *clusters*, grades ou nuvens (Dantas 2005, Vaquero et al. 2009). Por este motivo, algumas atividades são incluídas no *workflow* para cuidar do transporte de dados, da manipulação de elementos relacionados à segurança, bem como possibilitar o despacho e monitoração destas execuções nos diferentes tipos de ambientes (Ogasawara et al. 2010, 2009b, Oliveira et al. 2010a, Pinheiro et al. 2007, Taylor et al. 2007a). Estas

atividades relacionadas à infraestrutura são comumente diferenciadas para cada tipo de ambiente distribuído. Note-se que a responsabilidade pela obtenção do paralelismo fica toda a cargo dos cientistas e as abordagens variam de acordo com o SGWfC utilizado (Aalst et al. 2003, Barga et al. 2008).

3.2 Ambientes de processamento de alto desempenho

O desempenho de um *workflow* científico pode variar significativamente em diferentes ambientes computacionais. Atualmente consideram-se como ambiente de PAD *clusters* (Dantas 2005), grades computacionais (Foster e Kesselman 2004) e nuvens (Vaquero et al. 2009). Os *clusters* são um conjunto de nós computacionais usualmente homogêneos conectados por meio de uma rede de alto desempenho com baixa latência. Estes nós podem trabalhar como um único computador, de modo que as aplicações podem ser distribuídas e executadas em paralelo por vários nós para acelerar a execução. Um cenário de *cluster* é apresentado na Figura 8a. As aplicações distribuídas em um *cluster* dependem de um determinado tipo de escalonador (Bayucan et al. 2000, Staples 2006, Thain et al. 2002). A execução de *workflows* científicos neste ambiente requer um mecanismo eficiente de escalonamento que tire vantagem da estabilidade e das características de homogeneidade dos *clusters* para se ter um bom desempenho.

Uma estrutura em grade computacional combina vários recursos computacionais distribuídos (*hardware* e *software*) de múltiplos domínios. Assim, grades tendem a ser heterogêneas, fracamente acopladas e geograficamente dispersas. Grades comumente dependem de camadas especializadas responsabilizadas por distribuir as instâncias de programas em diversos computadores. A execução de *workflows* científicos em grades precisa se fazer valer de um escalonamento flexível e robusto para distribuir a aplicação nos recursos computacionais considerando-se a banda e a latência para transferência de dados (Thain et al. 2002, Yu et al. 2005, Yu e Buyya 2005). A Figura 8b apresenta uma visão simplificada de uma estrutura em grade.

As nuvens podem ser consideradas ambientes de PAD que possuem uma grande diversidade de recursos computacionais que são acessados virtualmente. Uma nuvem pode ser definida como uma camada conceitual que apresenta uma interface padrão para manipular todos os recursos na forma de serviços acessados através da *Web* (Vaquero et al. 2009). Esses serviços, por sua vez, são oferecidos sob demanda e a custos incrementais. Ao contrário do que acontece nas grades, onde os recursos ficam igualmente compartilhados, nas nuvens os recursos ficam dedicados a um único usuário quando ele requisita o serviço. Nas estruturas em nuvens, a execução de *workflows* científicos deve ser capaz de avaliar a relação entre tempo de execução, recursos instanciados, transferência de dados e custos financeiros (Oliveira et al. 2011a, Vaquero et al. 2009). A Figura 8c apresenta uma visão simplificada de estrutura em nuvem.

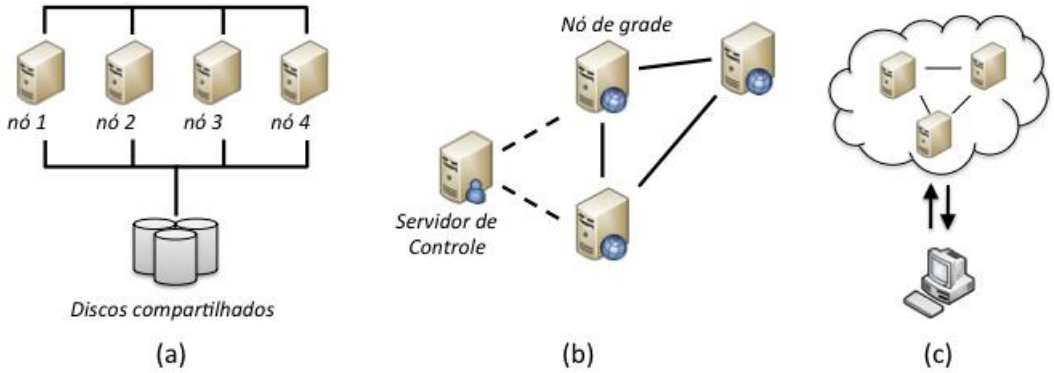

Figura 8 – Visão geral de *clusters* (a), grades (b) e nuvens (c)

Nestes ambientes computacionais existem diferentes arquiteturas para armazenamento de dados, sendo que estas arquiteturas impactam fortemente no desempenho da execução dos *workflows* científicos. Os tipos de arquitetura para armazenamento de dados podem ser divididos em arquiteturas de discos compartilhados e arquiteturas de discos não-compartilhados (Bouganim et al. 1996, Dantas 2005). As arquiteturas de discos não-compartilhados dependem exclusivamente dos discos locais. Desta forma, os dados precisam ser divididos de modo que os nós computacionais acessem apenas os dados direcionados especificamente a eles. Nas arquiteturas de discos compartilhados todos os nós podem ter acesso aos discos, deixando a distribuição da execução mais flexível. Na Figura 8a tem-se um exemplo de *clusters* acessando discos compartilhados.

3.3 Modelo de execução de *workflow* com paralelismo nativo

Na abordagem nativa enquadram-se os SGWfC paralelos, ou seja, SGWfC cujo motor de execução é específico para a execução em ambientes de PAD. Nesse caso, cada motor de execução possui uma forma específica de atender aos ambientes de PAD e de melhorar o desempenho da execução dos *workflows*. Existem diversos sistemas com estas características; nesta seção são apresentados dois dos mais conhecidos sistemas: Swift (Zhao et al. 2007) e Pegasus (Gil et al. 2007b). No caso do Swift (seção 3.3.1), seu motor de execução faz uso de primitivas próprias para otimizar as execuções (Laszewski et al. 2007). No Pegasus (seção 3.3.2), as dependências entre atividades e execuções são definidas por meio da construção de um grafo direcionado acíclico, o que permite verificar as melhores possibilidades de paralelização (Couvares et al. 2007).

3.3.1 Motor de execução de *workflow* paralelo do Swift

O Swift (Zhao et al. 2007) foi desenvolvido sobretudo para apoiar problemas de forte base matemática. Tem como foco a execução de *workflows* científicos em ambientes de PAD. Seu motor de execução, o Karajan (Laszewski et al. 2007), é paralelo, ou seja, ele é específico para esse tipo de execução, permitindo otimizações de escalonamento e

balanceamento de carga em tempo de execução. Duas atividades científicas que são independentes entre si, por exemplo, podem ser executadas paralelamente.

Apesar das vantagens relacionadas ao paralelismo em larga escala, não há interface gráfica para a definição dos *workflows*; os cientistas devem defini-los usando uma linguagem própria de *script* (*SwiftScript*). A linguagem oferece comandos como *ForEach* e *Iterate*, que são utilizados para controlar as primitivas de paralelismo. Além disso, o apoio à proveniência durante a execução é feito exclusivamente por arquivos de log. Estes arquivos de log só podem ser exportados para uma base de dados relacional após a execução completa do *workflow*. Estas características deixam este quesito insatisfatório para atender às demandas de monitoramento dos cientistas em experimentos científicos de larga escala.

3.3.2 Motor de execução de *workflow* paralelo do Pegasus

O Pegasus (Gil et al. 2007b), assim como o Swift, é um SGWfC paralelo, ou seja, é um sistema desenvolvido para executar *workflows* de diferentes domínios em ambientes de PAD, incluindo *clusters*, grades e nuvens. Ele permite a transformação de instâncias do *workflow*, que seriam os *workflows* com seus dados de entrada, em *workflows* executáveis, que seriam os *workflows* associados aos recursos de PAD, prontos para serem executados. Essa transformação envolve encontrar automaticamente os recursos apropriados para uma determinada instância de *workflow* e, eventualmente, sua reestruturação, de forma a melhorar o desempenho da execução (Deelman et al. 2007).

O motor de execução do Pegasus é dependente do Condor (Couvares et al. 2007), que provê uma interface para a submissão e execução de tarefas em ambientes de PAD, e é baseado no DAGMan (*Directed Acyclic Graph Manager*), um escalonador que automatiza a submissão e o gerenciamento de *workflows* científicos no Condor (Couvares et al. 2007). Ao contrário do Swift, o Pegasus possui uma interface gráfica para a definição dos *workflows* e para o monitoramento da execução. Entretanto, pode-se também definir os *workflows* diretamente por meio de sua linguagem de script.

3.4 Modelo de execução de *workflow* com paralelismo híbrido

Apesar da presença desses SGWfC paralelos, a maioria dos SGWfC existentes, como, por exemplo, Taverna, Kepler e VisTrails, possui motores de execução que não foram desenvolvidos para permitir a execução direta de *workflows* em ambientes de PAD. Muito embora não apoiem a execução em ambientes de PAD, os cientistas podem preferir usar um desses SGWfC, seja por já estarem acostumados a ele ou por precisarem usar alguma característica específica do SGWfC. Para viabilizar a execução de atividades destes *workflows* em ambientes de PAD é necessário que os cientistas paralelizem manualmente as atividades usando técnicas de paralelização, como

MapReduce (seção 3.4.1), ou usem uma camada especializada de paralelismo vinculada a esses SGWfC (seção 3.4.2).

3.4.1 Motor de execução de *workflow* sequencial e *MapReduce*

Uma implementação do modelo de programação *MapReduce* (Dean e Ghemawat 2008) associado a um motor de execução sequencial é um exemplo de modelo híbrido que pode ser adotado de modo a paralelizar uma ou mais atividades. Esse modelo de programação usa uma função de mapeamento que permite que a entrada possa ser processada em paralelo por diferentes unidades de processamento. Para isso, são criadas atividades de mapeamento, ou atividades *Map*, que são passíveis de serem paralelizadas. Ao final, uma função de agregação, denominada *Reduce*, é usada para permitir a integração dos resultados obtidos. Ambas as funções de mapeamento e de agregação devem ser definidas pelos cientistas, pois elas dependem da natureza das aplicações cujas execuções estão sendo paralelizadas, porém a paralelização e a execução em ambiente de PAD são feitas de forma automática.

Uma integração entre um SGWfC e o *MapReduce* pode ser encontrada em (Wang et al. 2009), na qual o Apache Hadoop (Hadoop 2010), uma implementação de código aberto do *MapReduce*, é acoplado ao Kepler. Nessa integração, um *subworkflow* é usado para gerar as atividades *Map* e realizar a agregação ao final da execução. As características do modelo de programação *MapReduce*, como distribuição e paralelização da execução e tolerância a falhas, são garantidas pelo Hadoop. Outra implementação de código aberto do *MapReduce* é o Disco (Disco 2011), um *framework* que também pode ser acoplado a um SGWfC para permitir a paralelização de atividades científicas.

3.4.2 Motor de execução de *workflow* sequencial e camada de paralelismo

Uma camada de paralelismo é uma ferramenta que, quando acoplada a um motor de execução sequencial, permite a execução de atividades de um *workflow* científico em ambientes de PAD. Essa camada de paralelismo não é conectada ao motor de execução dos SGWfC. Na verdade, o motor de execução enxerga a camada de paralelismo como uma atividade. Esta atividade, por sua vez, conecta-se ao ambiente de PAD e fica responsável pela paralelização de atividades científicas, assim como pelo seu gerenciamento e monitoramento. O SGWfC, dessa forma, não se preocupa em melhorar o desempenho da execução de atividades no ambiente de PAD, já que é a camada de paralelismo que se encarrega dessa funcionalidade. Com essa abordagem, as atividades do *workflow*, que não são demoradas, podem continuar rodando localmente. Ou seja, os cientistas podem escolher as atividades que são mais interessantes de serem executadas em ambientes de PAD, não havendo necessidade de o *workflow* completo ter sua execução paralelizada, ao contrário do que ocorre nos SGWfC paralelos existentes.

O *Hydra* (Ogasawara et al. 2009b) é um exemplo de camada de paralelismo baseada no paradigma de Muitas Tarefas Computacionais (do inglês, MTC) (Raicu et al. 2008), desenvolvida especificamente para ser acoplada a um SGWfC. Ele apoia o paralelismo de varredura de parâmetros. O *Hydra* possui um conjunto de componentes que facilita a transferência de dados para o ambiente de PAD, a execução paralela e a coleta de dados e de proveniência do ambiente de PAD para o SGWfC.

3.5 Considerações sobre os modelos de execução paralela de *workflows* científicos

Apesar de alguns SGWfC focarem na execução paralela, a paralelização de *workflows* em larga escala é ainda difícil, *ad hoc* e trabalhosa. Os cientistas precisam decidir a ordem de execução das atividades, dependências e estratégias de paralelização. Estas decisões dificultam as escolhas dentre as alternativas de paralelização. Além disso, uma vez feita a escolha, diversas oportunidades de otimização podem ser perdidas.

Para exemplificar as limitações intrínsecas ao estabelecimento da ordem envolvida na paralelização dos *workflows*, suponha-se a introdução do cenário de varredura de parâmetros para o *workflow* da Figura 1. Em linhas gerais, uma varredura de parâmetros comumente é implementada por meio de uma iteração sobre o conjunto de valores de parâmetro de entrada, do qual pode-se tomar como exemplo o *para todo* (do inglês, *ForEach*) apresentado por Graefe (1993). Existem diversas maneiras de se implementar a varredura de parâmetros. A Figura 9 apresenta duas formas diferentes de implementá-la. Na primeira (Figura 9.a), um conjunto de entradas é passado para cada atividade, na qual um comando *ForEach* é utilizado para realizar a varredura por atividade. Por outro lado, a Figura 9.b introduz uma atividade de varredura de parâmetros (PS) implementando uma iteração que envolve as atividades 2 e 3. Entretanto, embora as duas estratégias sejam comuns para se realizar a varredura de parâmetros nos SGWfC, a natureza imperativa destas linguagens de *workflow* associa uma implementação que não necessariamente leva a uma estratégia de execução ótima.

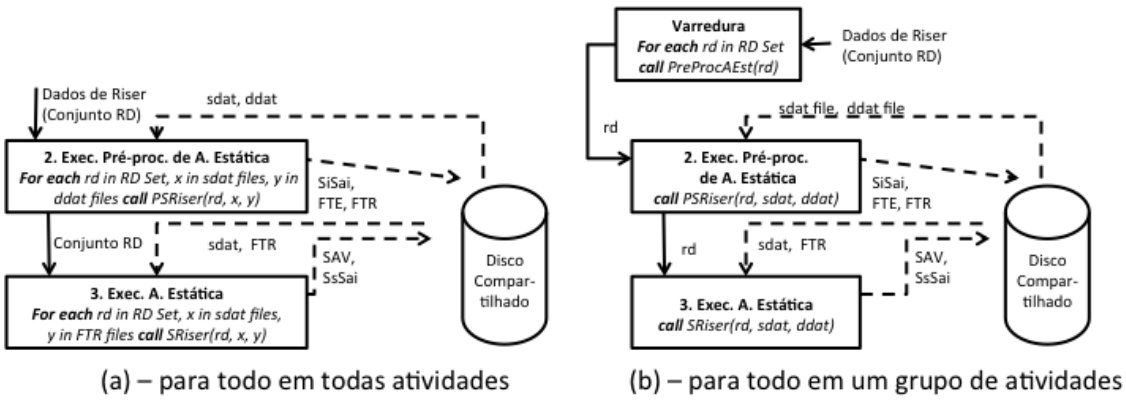

Figura 9 - Diferença nas abordagens para implementação de varredura de parâmetros

Capítulo 4 - Abordagem algébrica para *workflows* científicos

No Capítulo 3 foram apresentadas algumas restrições dos SGWfC existentes quanto ao apoio a cenários de varredura de parâmetros. De modo a abordar este problema, este capítulo apresenta uma proposta algébrica (Ogasawara et al. 2011), inspirada em décadas do já consolidado conhecimento de modelos de otimização de consultas em bancos de dados relacionais. Na seção 4.1 é apresentada uma álgebra para *workflows* científicos. A partir desta álgebra, é apresentado, na seção 4.2, um modelo de execução paralelo para a abordagem algébrica. Finalmente, na seção 4.3 é apresentada uma técnica de otimização para execução paralela.

4.1 Álgebra

Na proposta algébrica apresentada, os dados são uniformemente representados por meio de relações (como no contexto do modelo relacional) e as atividades de *workflows* são regidas por operações algébricas que possuem semântica sobre produção e consumo destes dados. A semântica presente nas operações algébricas possibilita a reescrita de *workflows*, gerando expressões equivalentes, porém, com desempenho de paralelismo distinto. Da mesma forma que no modelo relacional, a navegação por esse espaço de solução, formado por expressões equivalentes de *workflows*, permite a busca por expressões eficientes. Essa busca, orientada por um modelo de custos de execução paralela, viabiliza, assim, a otimização automática da execução paralela. Considera-se, nesta álgebra, que os parâmetros de entrada e saída de um *workflow* compõem uma unidade de dados equivalente às tuplas no modelo relacional. Neste contexto, atividades consomem e produzem relações, as quais são definidas como um conjunto de tuplas de dados primitivos (inteiro, real, alfanumérico, data, etc.) ou complexos (referência a arquivos).

Cada relação *R* possui um esquema $\mathcal{R}$, com os seus respectivos tipos de atributos. Considerando-se o exemplo da Figura 1, pode-se especificar que R_2 é uma relação de entrada para a atividade de pré-processamento estático (*IPSRiser*). O esquema da relação R_2 pode ser declarado como $\mathcal{R}_2$ = *(RID: Inteiro, Study: Alfanumérico; SDat: RefArquivo, DDat: RefArquivo)* e a ligação entre a relação e seu respectivo esquema pode ser especificada como $R_2(\mathcal{R}_2)$. As relações têm as seguintes propriedades:

- possuem atributos chave;
- possuem um conjunto de atributos com nome, tipo e restrições;
- Dada uma relação $R(\mathcal{R})$, representa-se *atr(R)* como um conjunto de atributos de $\mathcal{R}$ e *key(R)* como o conjunto contendo os atributos chave de $\mathcal{R}$.

De modo análogo à álgebra relacional, relações podem ser manipuladas por operações de conjunto: união ($\cup$), interseção ($\cap$) e diferença (-), desde que os seus esquemas sejam compatíveis (aridade da relação e domínio de cada atributo) (Elmasri e Navathe 2006, Silberschatz et al. 2010). Desta forma, considerando $R(\mathcal{R})$ e $S(\mathcal{R})$, então ($R \cup S$), ($R \cap S$) e ($R - S$) são expressões válidas. Novamente em analogia à álgebra relacional, pode-se atribuir uma relação a variáveis de relações temporárias. A operação de atribuição ($\leftarrow$) possibilita esta representação. Desta forma, $T \leftarrow R \cup S$ realiza uma cópia temporária de $R \cup S$ para variável de relação T, que pode ser reutilizada posteriormente em uma outra expressão algébrica.

Na abordagem algébrica deste trabalho, a definição de atividade compreende uma unidade computacional (UC), *i.e.*, um programa ou expressão da álgebra relacional, um conjunto de esquemas de relações de entrada e um esquema de relação de saída. A notação $Y <\{\mathcal{R}_i, .., \mathcal{R}_j\}, \mathcal{I}, UC>$ representa uma atividade Y que consome um conjunto de relações que são compatíveis com $\{\mathcal{R}_i, .., \mathcal{R}_j\}$ e executa a UC para produzir uma relação compatível com $\mathcal{I}$. A Tabela 2 apresenta um exemplo de uma relação de entrada para a atividade *IPSRiser* do *workflow* de AFR (Figura 1), cujo esquema é $\mathcal{R}_2$.

Tabela 2 - Exemplo de relação de entrada para atividade IPSRiser

RID	CaseStudy	sdat	ddat
1	U-125	U-125S.DAT	U-125D.DAT
1	U-127	U-127S.DAT	U-127D.DAT
2	U-129	U-129S.DAT	U-129D.DAT

As atividades do *workflow* são regidas por operações algébricas que especificam a razão de consumo e produção entre as tuplas. Esta característica possibilita um tratamento uniforme para as atividades, viabilizando a realização de transformações algébricas. A álgebra inclui seis operações (resumidos na Tabela 3): *Map*, *SplitMap*, *Reduce*, *Filter*, *SRQuery* e *MRQuery*. Cabe salientar que a maioria das operações algébricas consome uma única relação. Apenas a operação *MRQuery* consome um conjunto de relações. As primeiras quatro operações são usadas para apoiar atividades que executam programas. As duas últimas operações são usadas para executar atividades que processam expressões da álgebra relacional, sendo úteis para realizar filtragem e transformação de dados. O *SRQuery* consome uma única relação, enquanto o *MRQuery* consome um conjunto de relações. Para estas duas operações, assume-se a existência embutida de um processador de expressões da álgebra relacional.

Formalmente, pode-se definir *Op(Y)* como uma função que indica a operação algébrica que rege uma determinada atividade *Y*. Os possíveis valores para *Op(Y)* são as

seis operações algébricas citadas. Isto significa que se uma atividade *Y* é regida por uma operação *Map*, então *Op(Y) = Map*.

Tabela 3 – Resumo das operações algébricas

Operação	Tipo de atividade operado	Operandos adicionais	Resultado	Razão de consumo/produto de tuplas
Map	programa	Relação	Relação	1:1
SplitMap	programa	Atributo do tipo FileRef, Relação	Relação	1:m
Reduce	programa	Conjunto de atributos para grupamento, Relação	Relação	n:1
Filter	programa	Relação	Relação	1:(0-1)
SRQuery	Exp. da álgebra relacional	Relação	Relação	n:m
MRQuery	Exp. da álgebra relacional	Conjunto de Relações	Relação	n:m

Ademais, assume-se que os valores dos atributos em tuplas na álgebra são imutáveis. Ou elas são atribuídas como entrada para um *workflow* ou são produzidas durante a execução de uma atividade. Uma vez configurado o valor da tupla, ele não pode ser alterado. O comportamento dos valores das tuplas é similar ao comportamento das variáveis finais em Java.

4.1.1 Operação de mapeamento (*Map*)

Uma atividade *Y* é dita ser regida por uma operação *Map* quando uma única tupla é produzida na relação de saída *T* para cada tupla consumida na relação de entrada *R*. A operação *Map* pode ser representado como:

$$T \leftarrow Map(Y, R)$$

A Figura 10 ilustra a operação *Map* usando o exemplo do *workflow* de AFR. Considere-se a variável de tupla *r* referente à primeira tupla da relação de entrada *R* para uma atividade *PSRiser*. Neste caso, *r*[‘RID’] = ‘1’ e *r*[‘Study’] = ‘U-125’ são alguns dos valores expressados pela tupla *r*, que é consumida pela atividade *Y* e que produz a primeira tupla *t* da relação de saída *T* para atividade *PSRiser*. Assim, *t*[‘FTR’] = ‘U-125S.FTR’ e *t*[‘FTE’] = ‘U-125S.FTE’ são alguns dos valores expressados pela tupla *t*.

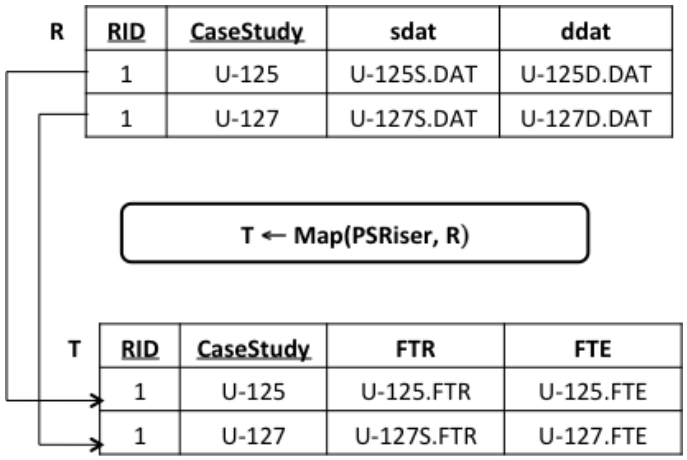

Figura 10 - Exemplo de operação *Map*

4.1.2 Operação de fragmentação de dados (*SplitMap*)

Uma atividade *Y* é dita ser regida pela operação *SplitMap* quando ela produz um conjunto de tuplas na relação de saída *T* para cada tupla na relação de entrada *R*. Esta decomposição é possível somente se o esquema $\mathcal{R}$ para a relação de entrada *R* tiver um atributo *a* do tipo *RefArquivo* para ser usado para fragmentação. Para cada tupla da relação de entrada, divide-se em várias partes o arquivo associado ao atributo especificado para fragmentação nas tuplas de saída. O *SplitMap* é representado como:

$$T \leftarrow SplitMap(Y, a, R).$$

Cabe salientar que os critérios para fragmentação são definidos pelo programa vinculado à atividade. A Figura 11 ilustra a operação *SplitMap*. Considere *r* uma variável de tupla referente à primeira tupla da relação de entrada *R* para a atividade *extractRD*. Neste caso, *r* [‘RID’] = ‘1’ e *r*[‘RdZip] = ‘Project1.zip’. Esta tupla é consumida pela atividade *extractRD*, que produz as duas primeiras tuplas t_1 e t_2 na relação de saída *T*. Assim, t_1[‘Study’] = ‘U-125’ e t_2[‘Study’] = ‘U-127’ são alguns dos valores expressados nas tuplas.

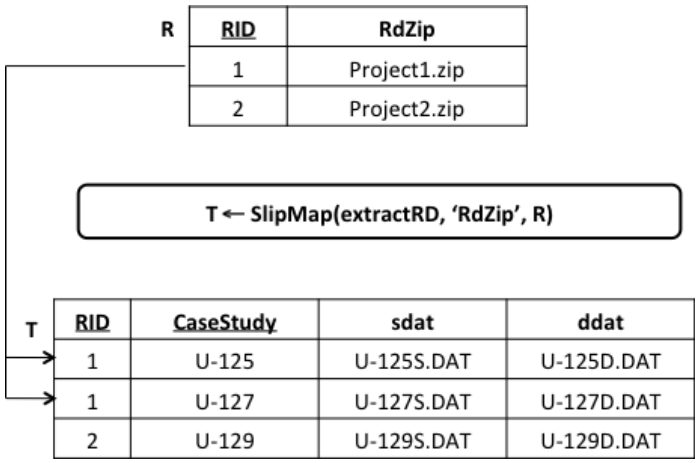

Figura 11 - Exemplo de operação *SplitMap*

4.1.3 Operação de redução (*Reduce*)

Uma atividade *Y* é dita ser regida pela operação *Reduce* quando ela produz um única tupla *t* na relação de saída *T* para cada subconjunto de tuplas na relação de entrada *R*. As tuplas de *R* são agrupadas por um conjunto de *n* atributos da relação *R*, que podem ser representados por $g_A=\{G_1, ..., G_n\}$. O conjunto g_A pode ser usado para agrupar *R* em subconjuntos (partições), *i.e.*, g_A estabelece critérios para particionamento horizontal da relação *R*. A operação *Reduce* executa a atividade *Y* consumindo cada partição de cada vez e produzindo uma tupla agregada *t* para cada partição. A operação *Reduce* é representada como:

$$T \leftarrow Reduce(Y, g_A, R).$$

A Figura 12 apresenta a operação *Reduce*. Ela retrata uma relação de entrada *R* que contém um atributo de agrupamento *RID*, que é usado como critério para o estabelecimento de uma partição horizontal sobre *R*. Neste exemplo, tem-se duas tuplas (r_1 e r_2) para partição, na qual *RID*=1. r_1['MEnv'] = 'u-125.ENV' e r_2 ['MEnv'] = 'u-127.ENV' são alguns exemplos de valores para estas tuplas, que são usadas para produzir a tupla de saída *t* da relação *T*, de modo que *t*['RID'] = 1, *t*['RdResultZip'] = 'ProjectResult1.zip'.

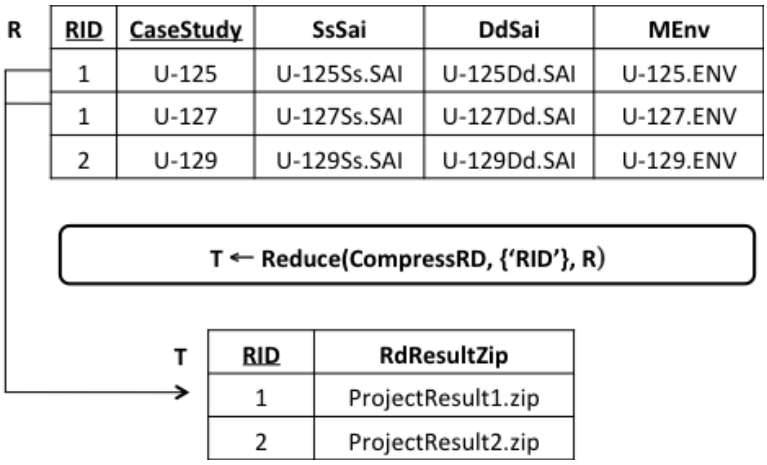

Figura 12 - Exemplo de operação *Reduce*

4.1.4 Operação de filtro (*Filter*)

Uma atividade *Y* é dita ser regida pela operação *Filter* quando cada tupla consumida na relação de entrada (*R*) puder ser copiada para a relação de saída (*T*), desde que a atividade *Y* a aceite. Além disso, uma propriedade importante da operação *Filter* é que o esquema da relação de entrada é exatamente igual ao da relação de saída. A operação executa a atividade *Y* *n* vezes, na qual *n* é o número de tuplas presentes em *R*, produzindo *m* tuplas de saída em *T*, na qual $m \leq n$. A operação de filtro é representada como:

$$T \leftarrow Filter(Y, R).$$

Cabe destacar que o critério de filtragem é definido pelo programa vinculado à atividade. A Figura 13 apresenta um exemplo da operação de filtro.

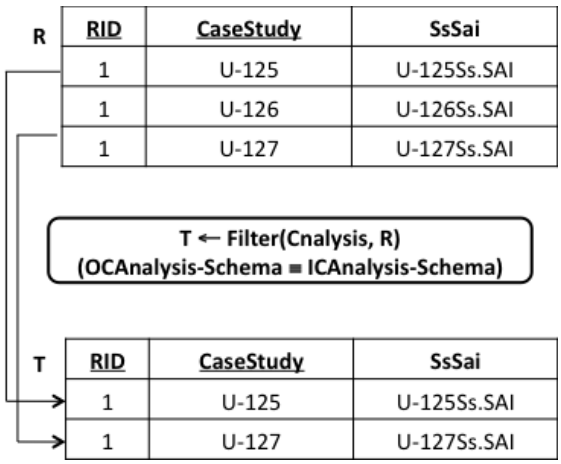

Figura 13 - Exemplo de operação *Filter*

4.1.5 Operação de consulta a relação única (*SRQuery*)

Uma atividade *Y* é dita ser regida pela operação *SRQuery* sempre que consumir uma relação *R* por meio de uma expressão de álgebra relacional (*qry*) para produzir uma relação de saída *T*. A operação *SRQuery* é representada como:

$$T \leftarrow SRQuery(qry, R).$$

Esta operação é útil para a transformação ou filtragem de dados. Além disso, quando o esquema da relação de entrada for exatamente igual ao da relação de saída, então a operação *SRQuery* assume também a semântica do *Filter*.

A Figura 14 mostra a operação *SRQuery*, na qual o primeiro argumento é uma expressão de álgebra relacional $(\pi_{RID,\ Study,\ SsSai,\ Curvature}(\sigma_{Curvature>1}(R)))$ usada para consumir a relação de entrada *R* e produzir a relação de saída *T*.

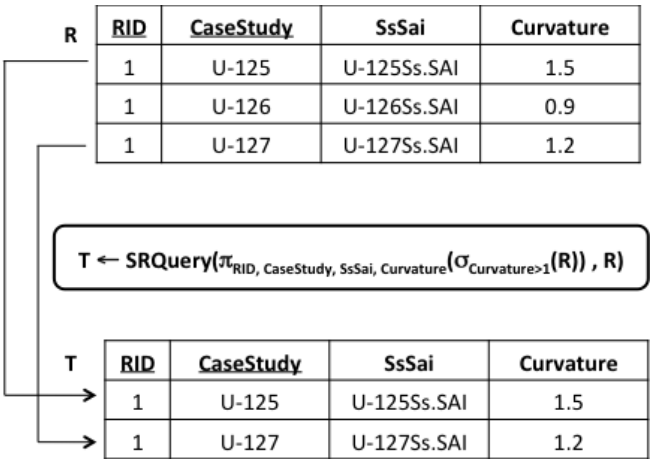

Figura 14 - Exemplo de operação *SRQuery*

4.1.6 Operação de consulta a relações múltiplas (*MRQuery*)

Uma atividade *Y* é dita ser regida pela operação *MRQuery* sempre que consumir um conjunto de relações $\{R_1, ..., R_n\}$ por meio de uma expressão de álgebra relacional (*qry*) para produzir uma única relação de saída *T*. Esta é a única atividade que consome mais de uma relação. Algebricamente, a operação *MRQuery* é representada como:

$$T \leftarrow MRQuery(qry, \{R_1, ..., R_n\})$$

Esta operação é útil para a transformação de dados ou filtragem de dados sobre as relações múltiplas. Além disso, vale mencionar que tanto o *SRQuery* quanto o *MRQuery* são computacionalmente não custosos quando comparados às demais operações algébricas.

A Figura 15 mostra a operação *MRQuery*, na qual o primeiro argumento é uma expressão de álgebra relacional $(\pi_{RID, Study, SsSAI, DdSAI, Env} (S \bowtie R))$ utilizada para consumir o conjunto de relações de entrada (*R* e *S*) e produzir a relação de saída *T*.

S

RID	CaseStudy	SsSai
1	U-125	U-125Ss.SAI
1	U-127	U-127Ss.SAI
2	U-129	U-129.SAI

R

RID	CaseStudy	DdSai	Env
1	U-125	U-125Dd.SAI	U-125.ENV
1	U-127	U-127Dd.SAI	U-127.ENV
2	U-129	U-129Dd.SAI	U-129.ENV

$OMerge \leftarrow MRQuery(\pi_{RID,CaseStudy,SsSAI,DdSAI, Env} (S \bowtie R), \{S, R\})$

T

RID	CaseStudy	SsSai	DdSai	Env
1	U-125	U-125Ss.SAI	U-125Dd.SAI	U-125.ENV
1	U-127	U-127Ss.SAI	U-127Dd.SAI	U-127.ENV
2	U-129	U-129.ENV	U-129Dd.SAI	U-129.ENV

Figura 15 - Exemplo de operação *MRQuery*

4.1.7 Representação de *workflows* científicos

A uniformidade obtida pela adoção de uma abordagem relacional como modelo de dados possibilita a expressão de *workflows* científicos como uma composição de expressões algébricas. Esta informação pode ser ilustrada pelo *workflow* AFR da Figura 1 expressado algebricamente na Figura 16.

Quando a saída de uma atividade for totalmente consumida por uma única atividade, é possível reescrever a expressão algébrica de uma forma mais concisa, como, por exemplo, $T_2 \leftarrow Map(PSRiser, SplitMap(ExtractRD, R_1))$.

$T_1 \leftarrow$ SplitMap(ExtractRD, R_1)
$T_2 \leftarrow$ Map(PSRiser, T_1)
$T_3 \leftarrow$ Map(SRiser, T_2)
$T_4 \leftarrow$ Map(PDRiser, T_3)
$T_5 \leftarrow$ Map(DRiser, T_4)
$T_6 \leftarrow$ Filter(Tanalysis, T_5)
$T_7 \leftarrow$ Filter(Canalysis, T_5)
$T_8 \leftarrow$ MRQuery(Match, {T_6, T_7})
$T_9 \leftarrow$ Reduce(CompressRD, T_8)

Figura 16 - Representação algébrica do *workflow* RFA como expressões algébricas

4.2 Modelo de execução

Esta subseção apresenta o modelo de execução paralelo para a abordagem algébrica. Inicialmente é apresentado o conceito de ativação de atividade como uma unidade básica para escalonamento de *workflows*, cuja estrutura possibilita que qualquer atividade possa ser executada por qualquer núcleo em um ambiente de PAD. A flexibilidade trazida pelo conceito de ativação serve de base para um modelo de execução, a ser apresentado no restante desta seção.

4.2.1 Ativação de atividade

Uma ativação de atividade, ou resumidamente ativação, é um objeto autocontido que possui todas as informações necessárias, como qual unidade computacional executar e qual dado acessar, de modo a viabilizar a execução de uma atividade em qualquer núcleo. A ativação é inspirada no conceito de ativação em banco de dados (Bouganim et al. 1996). As ativações podem consumir e produzir conjunto de tuplas; entretanto, uma ativação contém a unidade mínima de dados necessários para que, ainda assim, a execução da atividade seja realizada. A razão entre o consumo e o produto de tuplas em uma ativação varia de acordo com a operação algébrica que rege aquela atividade (Tabela 3). A saída de uma atividade é composta por um conjunto de tuplas produzidas por todas as ativações.

As execuções das ativações podem ser divididas em três etapas: instrumentação da entrada, invocação de programa e extração de saída. A instrumentação da entrada extrai os valores das tuplas de entrada e prepara para a execução do programa, configurando os valores das entradas dos parâmetros de acordo com os tipos de dados esperados. A invocação de programa despacha e monitora o programa associado à atividade. A extração de saída coleta os dados da saída do programa executado, transformando-os nas tuplas de saída.

A Figura 17 ilustra a ativação de atividade Y consumindo tuplas de entrada *{r}* da relação *R* e produzindo a saída de tuplas *{t}* da relação *T* para T $\leftarrow \alpha(Y, R)$, na qual α é uma operação que rege a atividade Y. Conforme mencionado anteriormente, Y pode ser um programa ou uma expressão algébrica a ser executada por um processador de consultas.

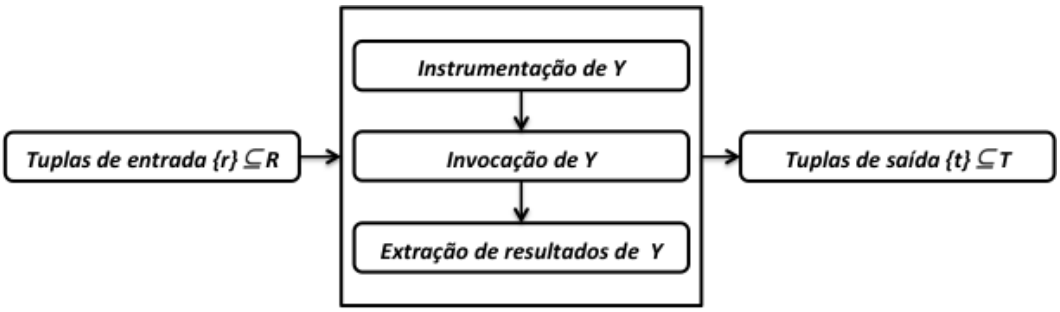

Figura 17 - Exemplo de ativação da operação *Map*

As ativações possuem quatro estados básicos: *Enfileirada*, *Executando*, *Esperando* e *Executada*. Uma ativação está *Enfileirada* quando ela já tem todos os seus dados prontos e está apenas aguardando a sua distribuição para execução em um determinado núcleo de um nó computacional. Uma ativação está *Executando* quando está alocada a um determinado nó e executando uma das três etapas da ativação (instrumentação, invocação e extração) em um determinado núcleo. Uma ativação está no estado *Executada* quando ela termina a sua execução. Uma ativação está no estado *Esperando* quando ela precisa que outras ativações terminem a sua execução antes que ela possa ir para o estado *Enfileirada*.

4.2.2 Escalonamento de *workflows*

Alguns conceitos básicos são apresentados nesta seção para a apresentação do algoritmo de execução de *workflows*, em particular os conceitos de dependência de atividades, dependência de ativações e escalonamento de *workflows*.

Um *workflow W* contempla um conjunto de atividades $\{Y_1, \ldots, Y_n\}$. Dada uma atividade $Y_i \mid (1 \leq i \leq n)$, seja R uma relação pertencente ao conjunto de relações de entrada $\{R_1, \ldots, R_m\}$ consumida pela atividade Y_i, tal que $R \in \{R_1, \ldots, R_m\}$, então $R \in Input(Y_i)$. Adicionalmente, seja T uma relação de saída produzida por uma atividade Y_i, então $T = Output(Y_i)$.

Uma dependência entre duas atividades Y_j, Y_i é expressada como $Dep(Y_j, Y_i) \leftrightarrow \exists\ R_{k'} \in Input(Y_j) \mid R_{k'} = Output(Y_i)$. Adicionalmente, um fragmento de *workflow*, ou resumidamente fragmento, é um subconjunto F de atividades do *workflow W*, de modo que F é um conjunto unitário ou $\forall\ Y_j \in F, \exists\ Y_i \in F, i \neq j \mid (Dep(Y_i, Y_j)) \vee (Dep(Y_j, Y_i))$. A Figura 18.a apresenta um exemplo de *workflow* composto por quatro atividades particionadas em três fragmentos.

Dado um *workflow W*, um conjunto $X=\{x_1, \ldots, x_k\}$ de ativações é criado para execução. Cada ativação x_i pertence a uma atividade particular Y_j, que pode ser representada como $Act(x_i) = Y_j$. Cada ativação x_i consome um conjunto de tuplas de entrada $InputX(x_i)$ e produz um conjunto de tuplas de saída $OutputX(x_i)$. Pode-se também estabelecer uma dependência entre duas ativações x_j, x_i, expressada como

$DepX(x_j, x_i) \leftrightarrow \exists r \in InputX(x_j) \mid r \in OutputX(x_i) \wedge Dep(Act(x_j), Act(x_i))$. A Figura 18.b apresenta um exemplo de *workflow* com quatro atividades e nove ativações. No exemplo, as ativações x_7 and x_8 pertencem à atividade Y_3.

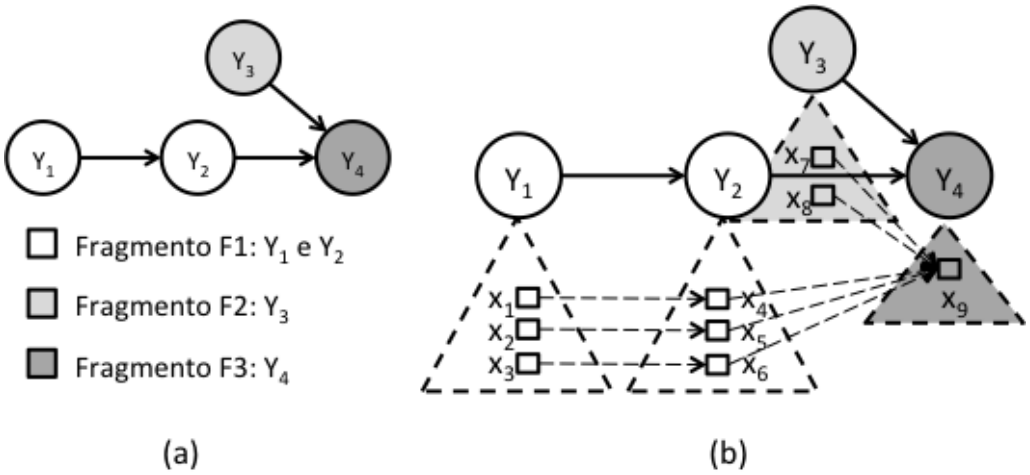

Figura 18 - Fragmento de *workflow* (a); Ativações para cada atividade (b)

Um escalonamento de *workflow* pode ser definido como uma sequência de ativações. Formalmente, dado um *workflow W* que inclui um conjunto de atividades $Y=\{Y_1, \ldots, Y_n\}$ e um conjunto $X=\{x_1, \ldots, x_k\}$ de ativações criado para uma execução de *workflow*, no qual $V_p(X)$ representa um escalonamento $<x_1, \ldots, x_k>$ para X, considere ord(x_i) como a posição de x_i na sequência. Tem-se que $x_i < x_j \leftrightarrow$ ord(x_i) < ord(x_j). $V_p(X)$ é dito um escalonamento válido para o *workflow* se, para qualquer par de ativações, ou elas são independentes ou $DepX(x_j, x_i) \rightarrow x_i < x_j$, *i.e.*, $V_p(X) \leftrightarrow \neg Dep(Act(x_j), Act(x_i)) \vee \neg(DepX(x_j, x_i) \wedge x_j < x_i), \forall x_i\, x_j \mid 1 \leq i, j \leq k, i \neq j$. Considerado o exemplo apresentado na Figura 18.b, a sequência $<x_1, x_4, x_7, x_2, x_5, x_8, x_3, x_6, x_9>$ é um escalonamento válido. Os escalonamentos possíveis $V(X)$ incluem todas as permutações de escalonamento válidos $V_p(X)$ para X.

Adicionalmente, considere o fragmento F para o *workflow W*, no qual X' é um conjunto de ativações em F. Um escalonamento estrito $V_p(X')$ para $X'=\{x_q, \ldots, x_m\}$ é um escalonamento válido se $\forall x_i \in X' \exists x_j \in X' \mid (DepX(x_i, x_j) \vee DepX(x_j, x_i))$. Finalmente, um escalonamento independente $V_p(X')$ para $X'=\{x_q, \ldots, x_m\}$ é um escalonamento válido se $\forall x_i, x_j \in X' \mid \neg(DepX(x_i, x_j) \vee DepX(x_j, x_i))$.

4.2.3 Algoritmo de execução de *workflows*

Nesta seção é apresentado o algoritmo para execução de *workflows*. Para tanto, inicialmente são introduzidas duas estratégias de fluxo de dados (*DS*) possíveis: *Primeira Tupla Primeiro* (do inglês, *FTF*), que particiona um conjunto de ativações *X* em conjuntos de escalonamentos estritos, e a *Primeira Atividade Primeiro* (do inglês, *FAF*), que particiona o conjunto de ativações em conjunto de escalonamentos independentes. A seguir, apresenta-se a associação de uma estratégia de fluxo de dados

a cada fragmento de *workflow*. Finalmente, para cada fragmento de *workflow* se apresenta também uma estratégia de despacho como uma forma de distribuir as ativações.

Conforme mencionado anteriormente, dado um *workflow W* com um conjunto de ativações $X=\{x_1, \ldots, x_k\}$, este *workflow* é executado de acordo com um escalonamento definido. A definição do escalonamento de ativações depende da estratégia de fluxo de dados atribuída a cada fragmento de *workflow*. Assim, dados um fragmento de *workflow* F_i e um fluxo de dados DS_i, uma função de mapeamento $DSF(F_i, DS_i)$ atribui um fluxo de dados a um fragmento de *workflow*. Neste contexto, seja um conjunto de ativações $X'=\{x_q, \ldots, x_m\}$ associado ao fragmento F_i, um fluxo de dados DS_i impõe uma ordenação parcial sobre as ativações de X'. Além disto, durante o escalonamento, um conjunto de instâncias de fragmento de ativações (*FAI*) é criado, estabelecendo uma partição para todas as ativações daquele fragmento. Cada *FAI* é uma sequência mínima de ativações que obedece à estratégia de fluxo de dados atribuída para aquele fragmento. Considerando o exemplo apresentado na Figura 18.b, assumindo-se $DSF(F_1, FTF)$, pode-se ter três *FAIs* de escalonamento estrito $\{< x_1, x_4 >, <x_2, x_5>, <x_3, x_6>\}$. Adicionalmente, assumindo-se $DSF(F_2, FAF)$, pode-se ter dois *FAIs* de escalonamento independente $\{< x_7 >, <x_8 >\}$.

Uma estratégia de despacho *disp* governa a distribuição de *FAIs* para os núcleos disponíveis ao longo dos diferentes nós computacionais. Existem dois tipos básicos de estratégia de despacho: estática e dinâmica. No caso de uma estratégia estática, todos os *FAIs* de um determinado fragmento de *workflow* são previamente alocados para os diferentes núcleos antes que uma das ativações de um fragmento comece a executar. Correspondentemente, em uma estratégia dinâmica, um subconjunto de *FAIs* de um determinado fragmento de *workflow* é direcionado para os núcleos como uma resposta à identificação de ociosidade dos mesmos.

Uma vez atribuídas as estratégias de fluxo de dados e despacho a um fragmento de *workflow*, uma estratégia de execução foi atribuída a cada fragmento. O processador de *workflow* gerencia os fragmentos, colocando os seus respectivos *FAIs* na sua fila de escalonamento de ativações. Os fragmentos podem ser classificados em quatro estados: *Disponível*, *Dependente*, *Executando* e *Finalizado*. Os fragmentos em estado *Dependente* são aqueles que possuem atividades dependentes de atividades pertencentes a outros fragmentos. Os fragmentos em estado *Disponível* são aqueles que possuem todas as suas dependências atendidas, mas não instanciaram ainda as suas ativações. Fragmentos em estado *Executando* são aqueles que já tiveram suas ativações instanciadas e elas estão sendo executadas. Os fragmentos passam ao estado *Finalizado* quando todas as suas ativações tiverem terminado.

Um processador de ativações está presente para cada núcleo disponível e tem como papel executar ativações. Ele recebe um conjunto de *FAIs* para ser

independentemente avaliado e notifica o processador de *workflows* sobre sua completude. Pode-se interpretar o processador de ativações como processos de *Map* (Afrati e Ullman 2010) em modelo de execuções de *MapReduce*.

A Figura 19 apresenta um algoritmo de execução de *workflows*. O algoritmo inicia ao receber um *workflow* e os processadores de ativações. Inicialmente, o algoritmo chama a função *optimizeWorkflow*(), responsável por dividir o *workflow* em fragmentos *fragSet* com as estratégias *DS* e *Disp* atribuídas a cada fragmento. Enquanto existirem fragmentos disponíveis para execução, um fragmento *Disponível* é trazido do *fragSet*. A partir do fragmento *frag*, o processador de *workflow* gera as ativações *FAISet*, alterando o estado do fragmento para *Executando*. Cada elemento do *FAISet* é despachado de acordo com as estratégias de despacho definidas para um determinado fragmento. Quando todos os elementos do *FAISet* tiverem sido completados, o *frag* é removido do *fragSet* e o fragmento muda de estado para *Finalizado*. Neste momento todas as dependências são reavaliadas para se alterar o estado dos fragmentos de *Dependente* para *Disponível*. O algoritmo de execução de *workflows* é apresentado na Figura 19.

*Workflow*Processor(*Workflow* W, ExecutionProcessesSet execProc)

```
1.  FragmentSet fragSet = optimize Workflow(W);
2.  while (hasElements(fragSet))
3.    frag = getReadyFragment(fragSet);
4.    FAISet faiSet = generateActivations(frag);
5.    DispatchStrategy dispStrat = getDispatchStrategy(frag);
6.    for each FAI f in faiSet
7.     dispatch(f, dispStrat, execProc);
8.    fragSet = removeCompleted(fragSet, frag);
```

Figura 19 - Algoritmo para execução de *workflows*

4.3 Otimização da execução paralela de *workflows*

A seção 4.1 apresenta as operações algébricas e mostra como os *workflows* científicos são especificados como expressões algébricas, enquanto que a seção 4.2 apresenta um modelo de execução para um *workflow* científico representado por expressões algébricas. Nesta seção apresenta-se a otimização de *workflows* para execução paralela ou, simplesmente, otimização de *workflows*. Um *workflow* expresso por operações algébricas pode ser transformado em um conjunto de *workflows* equivalentes por meio de transformações algébricas.

Cada um dos *workflows* equivalentes pode ser particionado em fragmentos de *workflows*, e cada um dos fragmentos pode ter uma estratégia de fluxo de dados e de despacho associada. Chama-se configuração de *workflow* um determinado *workflow* particionado em fragmentos com suas respectivas estratégias *DS* e *disp* atribuídas. Cabe salientar que todas as possíveis configurações produzem o mesmo resultado, mas com

um diferente custo computacional. Define-se como otimização de *workflow* o processo a partir do qual se exploram as possíveis configurações de *workflow* para se obter uma que minimize o custo computacional.

O número de possíveis configurações pode ser bem alto. No que tange à otimização, como é interessante identificar as transformações algébricas que podem influenciar a ordem de execução de atividades, pode-se assumir que existe um limite superior de n! *workflows* alternativos, no qual n é o número de atividades presentes no *workflow*. Adicionalmente, para cada *workflow* alternativo existem 2^n possíveis partições de *workflows*. Assim, como cada uma das partições pode assumir 4 possíveis combinações de estratégias de despacho e DS, tem-se, então, um limite superior total de 2^{n+2}n! possíveis configurações de *workflow* para explorar.

A otimização de *workflow* apresentada na Figura 20 é dividida em dois grandes passos: geração de espaço de busca usando transformações algébricas e estratégia de busca usando um modelo de custo. O processo de otimização de *workflow* remonta ao processo tradicional de otimização de consultas em banco de dados, que foca na identificação de um plano de execução de consulta que minimiza uma função de custo. O espaço de busca em banco de dados relacionais engloba planos de execução de consulta alternativos e seleciona o melhor plano de execução baseado em um modelo de custo. Selecionar o plano de execução de consulta ótimo é um problema *NP-hard* no número de relações (Ibaraki e Kameda 1984), o que, para consultas complexas, torna-se proibitivo pelo custo de otimização. Entretanto, a otimização de consultas tipicamente restringe o tamanho do espaço de busca que se pode considerar. Heurísticas são comumente aplicadas, como as que realizam as operações de projeção e seleção nas relações base de modo a minimizar o número de tuplas intermediárias antes das operações de junções. Adicionalmente, assume-se a aplicação de heurísticas que se concentram na otimização das árvores de junções, cujas operações podem ser tanto as de junções quanto as de produto cartesiano. Neste contexto, as permutações de uma árvore de junção comumente são as mais importantes para influenciar as consultas relacionais (Özsu e Valduriez 2011), particularmente focando na redução de tuplas das relações intermediárias.

Nesta perspectiva, a otimização de *workflows* proposta possui várias semelhanças em relação às tradicionais otimizações de consultas em banco de dados. O maior objetivo da otimização de *workflows* é reduzir o tamanho das relações intermediárias que servem como entrada para atividades que invocam programas. Neste contexto, as operações algébricas que executam expressões da álgebra relacional apresentam custo negligenciável frente às operações algébricas que executam programas. Assim, de modo análogo ao processamento de consultas, no qual a ordem das junções direciona a otimização, na abordagem algébrica para *workflows* a ordem de execução dos programas direciona a otimização de *workflows*.

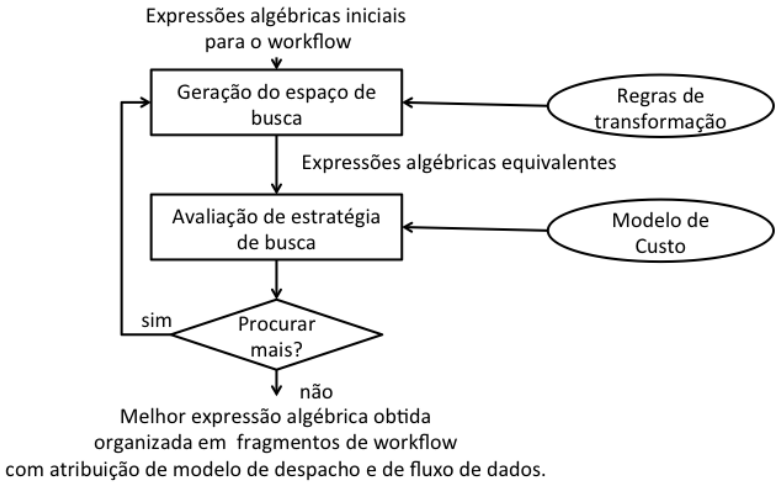

Figura 20 - Otimização de *workflow*

Cabe salientar que as dependências em *workflows* científicos impõem restrições às transformações algébricas que podem ser aplicadas durante o processo de otimização. Por exemplo, se uma atividade Y_i produz dados que são consumidos por atividades Y_j, a atividade Y_j não pode ser posicionada em uma ordem que anteceda a atividade Y_i. Em função destas restrições, a abordagem de otimização para *workflows* é inspirada na bem conhecida técnica de migração de predicado (Hellerstein e Stonebraker 1993). Sempre que possível, se antecipa a execução de atividades que reduzam a quantidade de tuplas nas relações intermediárias e, correspondentemente, se adiam as atividades que produzem mais dados.

No contexto do processo de otimização como um todo, na seção 4.3.1, apresenta-se uma heurística para reduzir os espaços de buscas. A redução permite que o foco do processo de otimização de consultas seja a aplicação de regras de transformação que possam mudar as ordens das atividades. Apresentam-se, também, as propriedades de relação e regras de transformação na seção 4.3.2. Finalmente, a seção 4.3.3 define as estratégias de busca e modelo de custo, bem como o algoritmo de otimização de *workflows* usado para identificar a configuração "ótima".

4.3.1 Heurística para reduzir o espaço de busca

Como mencionado anteriormente, há um total de $2^{n+2}n!$ possíveis configurações de *workflows* para se explorar, dos quais n! surge como limite superior para *workflows* alternativos, 2^n surge a partir do número de partições possíveis, e 2^2 pode ser obtido a partir de combinações de estratégias de fluxo de dados e de despacho. Ao se assumir o cenário de *workflows* executados em um *cluster* usando disco compartilhado, pode-se diminuir o espaço de busca por meio de um conjunto de heurísticas que sistematicamente definem partições de *workflows*, atribuindo a cada uma delas uma estratégia de fluxo de dados (*DS*) e de despacho (*disp*).

Para atingir este objetivo, inicialmente tem-se a necessidade da definição de atividade bloqueante. Uma atividade bloqueante é uma atividade que limita a possibilidade de execução, seja por necessidade de dados ou por necessidade de recurso computacional. No primeiro caso, ela exige que as suas relações de entrada sejam completamente produzidas antes que se possa iniciar a sua execução. No segundo caso, ela exige todos os núcleos de um nó para poder executar. Isto significa que este tipo de atividade não pode estar presente em fragmentos com estratégia de fluxo de dados *FTF*. Uma atividade bloqueante pode ocorrer devido a diferentes razões, tais como: (i) o tipo de operação regente, (ii) o tipo de programa a ser executado, e (iii) as questões de implementação referentes ao motor de execução. No primeiro caso, a operação algébrica pode impor este comportamento para a atividade. Uma operação *Reduce*, por exemplo, exige que a relação de entrada seja totalmente produzida antes de se começar a produção de tuplas a partir do agrupamento das tuplas de entrada. O segundo caso ocorre, por exemplo, na presença de atividade com restrição, *i.e.*, aquela que aloca todos os núcleos de um determinado nó para poder executar. Finalmente, no último caso, a maneira pela qual uma operação algébrica é implementada pelo motor de execução pode gerar um comportamento bloqueante para a atividade. Por exemplo, embora teoricamente *SRQuery* e *MRQuery* sejam apenas bloqueantes quando contenham alguma agregação em sua expressão de álgebra relacional, para fins de simplicidade eles são implementados como atividades bloqueantes na versão atual do Chiron.

Com base nos resultados experimentais obtidos em Ogasawara et al. (2011), é possível reduzir o espaço de busca por meio de um conjunto de heurísticas para a quebra do *workflow* em fragmentos de *workflow*. As atividades e dependências de dados estabelecem um grafo direcionado. Para simplificar, no que tange à heurística, pode-se ignorar a orientação do grafo. Inicialmente, identificam-se todas as atividades bloqueantes. Cada atividade bloqueante torna-se um fragmento de *workflow* independente. Cada um destes fragmentos é marcado com a estratégia *FAF* para fluxo de dados. Para a definição dos demais fragmentos, as atividades bloqueantes são retiradas do grafo. Neste caso, obtém-se uma floresta. Cada um dos componentes conexos do grafo torna-se um fragmento de *workflow*. Estes fragmentos são marcados com a estratégia *FTF* para fluxo de dados. Além disso, para cada fragmento de *workflow*, se a soma do tempo de execução médio para as atividades de um fragmento for maior que um limiar, este fragmento é marcado como despacho dinâmico; caso contrário, é marcado como despacho estático.

A Figura 21 apresenta um exemplo de *workflow*. Inicialmente, são identificadas todas as atividades bloqueantes, que são marcadas com a estratégia de fluxo de dados *FAF*. Estas atividades são excluídas do grafo, o que o transforma em uma floresta. Cada componente conexo desta floresta estabelece um novo fragmento, que é marcado com a estratégia de fluxo de dados *FTF*.

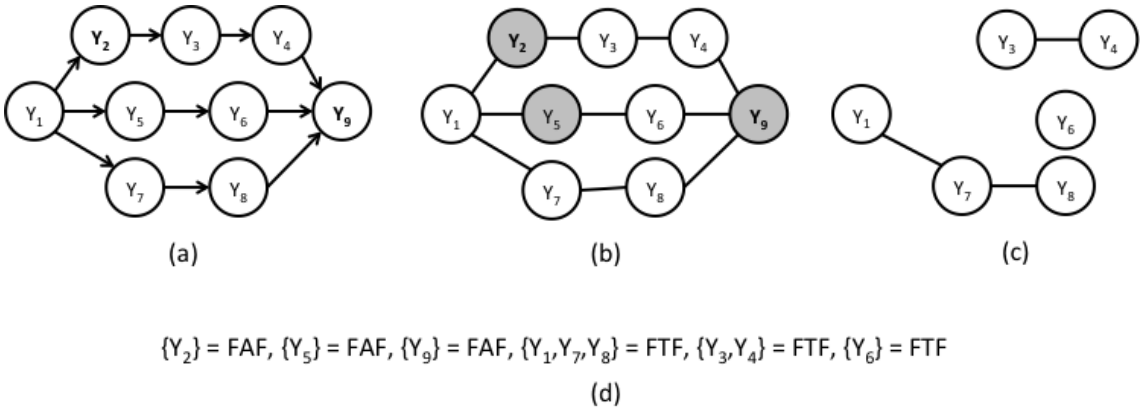

Figura 21 - *Workflow* Científico (a); Identificação das atividades bloqueantes (b); Identificação dos componentes conexos a partir de conjuntos disjuntos (c); Fragmentos de *workflows* com atribuição de fluxo de dados definida (d)

A partir destas heurísticas, o espaço de busca é reduzido a n! configurações de *workflows*, o que possibilita que os esforços sejam concentrados em se identificar a ordem de execução da atividade. A próxima subseção apresenta um conjunto de definições importantes para apoiar a otimização do *workflow* a partir deste espaço de busca reduzido.

4.3.2 Propriedades das relações e regras de transformações

Como mencionado na seção 4.1, os valores das tuplas na álgebra proposta são imutáveis. Eles são atribuídos como entrada geral para o *workflow* ou são produzidos pelas atividades. Este conceito é uma premissa para se apoiar a mudança de ordem entre as atividades de modo a permitir a otimização da execução do *workflow*. Entretanto, as relações consumidas e produzidas estabelecem restrições para a otimização. Com vistas a viabilizar as mudanças de ordem entre as atividades, os seguintes conceitos são apresentados nesta seção: dependência de dados ($\overset{dep}{\longleftarrow}$), preservação de chaves ($\overset{kp}{\leftarrow}$), atributos consumidos (ca) e atributos canônicos consumidos ($\overline{ca}$).

Inicialmente, define-se dependência de dados de R para S como $S \overset{dep}{\longleftarrow} R$ expressando a relação *S($\mathcal{S}$)* dependente da relação *R($\mathcal{R}$)* se existir uma expressão algébrica $S \leftarrow \alpha(R)$ que consuma R para produzir S. As dependências são também transitivas: $\{S \overset{dep}{\longleftarrow} R, T \overset{dep}{\longleftarrow} S\} \vDash T \overset{dep}{\longleftarrow} R$. A transitividade é importante, uma vez que é a partir dela que se torna possível descobrir se a ordem de uma atividade pode ser trocada por outra. Para provar a transitividade, pode-se assumir que (1) $S \overset{dep}{\longleftarrow} R$ e (2) $T \overset{dep}{\longleftarrow} S$ sejam válidas, *i.e.*, existem duas expressões algébricas nas quais (1) e (2) são válidas. A partir da simples composição destas expressões $T \leftarrow \beta(\alpha(R))$, pode-se verificar que $T \overset{dep}{\longleftarrow} R$ é uma expressão válida.

Adicionalmente, define-se preservação de chave entre as relações *R* e *S* como $S \overset{kp}{\leftarrow} R$ se a relação *S(𝒮)* possui dependência de dados com a relação *R(ℛ)* e se a chave de $\mathcal{S}$ contém a chave de $\mathcal{R}$. Formalmente, pode-se definir $(S \overset{kp}{\leftarrow} R) \leftrightarrow (S \overset{dep}{\longleftarrow} R) \wedge (\text{key}(S) \supseteq \text{key}(R))$. A preservação de chaves é também transitiva: $\{S \overset{kp}{\leftarrow} R,\ T \overset{kp}{\leftarrow} S\} \vDash T \overset{kp}{\leftarrow} R$.

Considerando-se uma atividade *Y* <$\{\mathcal{R}_i, \ldots, \mathcal{R}_j\}$, $\mathcal{S}$, *UC*> regida pela operação algébrica α, de modo que *Y* consome relações de entrada compatíveis com $\mathcal{R}_i, \ldots, \mathcal{R}_j$ e produz relações de saída compatíveis com $\mathcal{S}$, podem-se definir atributos consumidos *ca(Y, R)* como um conjunto de atributos a partir da relação de entrada *R*, tal que $R \in \{\mathcal{R}_i, \ldots, \mathcal{R}_j\}$, necessários e suficientes para que a instrumentação ocorra durante a ativação de *Y* (seção 4.2.1). Podem-se definir atributos canônicos consumidos para uma determinada relação de entrada *R* que ativa *Y* como sendo $\overline{ca}$*(Y,R) = key(R)* ∪ *ca(Y, R)*. De modo análogo, podem-se definir também os atributos canônicos não-consumidos por uma determinada relação de entrada *R* que ativa Y como sendo $\overline{nca}$*(Y,R) = atr(R)* – ($\overline{ca}$*(Y,R)* – *key(R)*). Em outras palavras, os atributos canônicos não-consumidos de uma relação *R* para uma atividade *Y* são caracterizados por todos os atributos que não são instrumentados pela ativação de *Y* juntamente com os atributos chave da relação *R*.

Ademais, define-se como $\bar{\alpha}$ a ativação das operações algébricas α usando os atributos canônicos para uma determinada relação de entrada *R(ℛ)* / $\mathcal{R} \in \{\mathcal{R}_i, \ldots, \mathcal{R}_j\}$. Formalmente, pode-se definir $\bar{\alpha}(Y, R) \equiv (\alpha(Y, \pi_{\overline{ca}(Y,R)}(R))$.

Usando-se as definições anteriores, três regras de transformação (decomposição de atividade, antecipação de atividade e procrastinação de atividade) podem ser definidas e usadas durante o processo de otimização de *workflows*, de modo a permitir a mudança de ordem entre as atividades, respeitando-se as restrições de dependência de dados. A regra de decomposição é uma regra básica que viabiliza a movimentação das atividades, tanto no contexto de antecipação quanto no contexto de procrastinação. A regra de antecipação visa a mover uma atividade na direção oposta ao fluxo de dados para que ela tenha a sua execução antecipada no *workflow*. Correspondentemente, a regra de procrastinação visa a movimentar a atividade no sentido do fluxo de dados, postergando a sua execução no *workflow*. Estas regras visam a diminuir as relações intermediárias que antecedem atividades computacionalmente custosas. Por meio destas regras, por exemplo, torna-se possível transformar o *workflow* graficamente expresso na Figura 22.a em uma versão equivalente apresentada na Figura 22.d.

Decomposição de atividade: Considerando uma expressão algébrica $T \leftarrow \alpha(Y, S)$, a regra de decomposição desmembra a produção de *T* em duas partes: (i) os dados que são produzidos pela atividade *Y* somados aos dados que pertencem aos

atributos canônicos consumidos por *Y*, *i.e.*, os dados no escopo de $\bar{\alpha}(Y,S)$, e (ii) os dados que pertencem diretamente à *S*, que não são instrumentados por *Y*. A decomposição de atividade pode ocorrer se e somente se $\alpha(Y,S)$ possuir preservação de chaves. Mais formalmente, dado $T \leftarrow \alpha(Y,S)$ com ($T \overset{kp}{\leftarrow} S$):

$$\alpha(Y,S) \equiv \bar{\alpha}(Y,S) * \pi_{\overline{nca}\,(Y,S)} S$$

A fundamentação teórica da regra de decomposição de atividade é apoiada pela fundamentação teórica da decomposição existente na teoria de normalização (Elmasri e Navathe 2006), possuindo propriedades similares, mais especificamente a preservação de atributos e a junção sem perdas (do inglês, *lossless join*) (Elmasri e Navathe 2006). A propriedade da preservação de atributo garante que cada atributo em *T* aparece em pelo menos uma das relações decompostas, enquanto a propriedade de junção sem perdas estabelece que a relação *T* é produzida a partir de uma junção natural entre as relações decompostas. Na decomposição de atividade, a preservação do atributo é observada, visto que cada atributo de *T* pode vir tanto diretamente dos atributos envolvidos nos dados consumidos/produzidos por $\bar{\alpha}(Y,S)$ quanto dos dados originais de S ($\pi_{\overline{nac}\,(Y,S)} S$), que não são consumidos por *Y*. Além disso, a propriedade de junção sem perdas é garantida a partir do pré-requisito de preservação de chave existente na regra de decomposição, que permite que a junção natural ocorra, mantendo-se os atributos preservados.

Exemplo: Considere uma expressão algébrica $T \leftarrow Filter(Y_2,S)$ com $\mathfrak{I}$(<u>id</u>, a, b, c), $\mathbb{S}$(<u>id</u>, a, b, c), $\overline{ca}(Y_2,S) = \{id, a\}$. Neste caso é possível aplicar a regra de decomposição e transformar $T \leftarrow Filter(Y_2,S)$ em $T \leftarrow \pi_{id,a,b,c}\ (\overline{Filter}(Y_2,S) * S)$. Adicionalmente, por conveniência, a expressão algébrica pode ser expressa por meio de uma relação intermediária *U* para a operação de $\overline{Filter}$, *i.e.*, $\{U \leftarrow \overline{Filter}(Y_2,S),\ T \leftarrow \pi_{id,a,b,c}\ (U * S)\} \equiv T \leftarrow Filter(Y_2,S)$. Este exemplo pode ser visualizado na transformação da Figura 22.a em Figura 22.b.

Antecipação de atividade: Considerando a expressão algébrica $\alpha(Y,S)$ e a relação *R* de modo que $S \overset{kp}{\leftarrow} R$, o objetivo desta regra é antecipar a execução de *Y* substituindo *R* no lugar de *S*. Esta regra pode ser aplicada apenas se os atributos consumidos de *Y* em relação à *S* estiverem contidos nos atributos da relação *R*, de modo que $\overline{ca}(Y,S) \subseteq atr(R)$. Mais formalmente, dado $\alpha(Y,S)$ com ($S \overset{kp}{\leftarrow} R$) e $\overline{ca}(Y, S) \subseteq atr(R)$:

$$\bar{\alpha}(Y,S) \equiv \bar{\alpha}(Y,R)$$

Exemplo: Considere uma expressão algébrica $U \leftarrow \overline{Filter}(Y_2,S)$ e uma relação R, de modo que $\mathbb{S}$(<u>id</u>, a, b, c), $\mathcal{U}$(<u>id</u>, a), $\mathcal{R}$(<u>id</u>, a, t). Tem-se que $S \overset{kp}{\leftarrow} R$, $\overline{ca}(Y_2,S)$ = *{id,a}* e $atr(R)$ = *{<u>id</u>, a, t}*. Neste caso, é possível usar a regra de antecipação de atividade e

transformar $U \leftarrow \overline{Filter}(Y_2, S)$ em $U \leftarrow \overline{Filter}(Y_2, R)$. Este exemplo pode ser visualizado na transformação da Figura 22.b em Figura 22.c.

Procrastinação de atividade: Considerando duas expressões algébricas $Q \leftarrow \alpha(Y_i, R)$, $U \leftarrow \beta(Y_j, R)$ e sabendo-se que $U \overset{kp}{\leftarrow} R$, a regra de procrastinação adia a execução de Y_i ao introduzir uma dependência de dados de *U* para *Q* por meio da execução de uma junção natural entre *U* e *R* como entrada para a atividade Y_i se e somente se existir uma relação *T* que dependa tanto de *U* quanto de *Q*, *i.e.*, $T \overset{dep}{\longleftarrow} U$ e $T \overset{dep}{\longleftarrow} Q$, e não existir uma relação *W*, de modo que $W \overset{dep}{\longleftarrow} Q$ e $\neg(T \overset{dep}{\longleftarrow} W)$, ou seja, todas as relações derivadas de *Q* levam a *T*, sem ir para outro fluxo independente. Mais formalmente, dado $Q \leftarrow \alpha(Y_i, R)$, $U \leftarrow \beta(Y_j, R)$ e $U \overset{kp}{\leftarrow} R$, se $\exists T | \ (T \overset{dep}{\longleftarrow} Q) \ \wedge \ (T \overset{dep}{\longleftarrow} U)$ e $\nexists W | \ (W \overset{dep}{\longleftarrow} Q) \ \wedge \neg(T \overset{dep}{\longleftarrow} W)$:

$$\bar{\alpha}(Y_i, R) \equiv \bar{\alpha}(Y_i, \pi_{atr\,(R)}(R * U))$$

Exemplo: Considere as expressões algébricas $Q \leftarrow \overline{Map}(Y_i, R)$, $U \leftarrow \overline{Filter}(Y_j, R)$ e $T \leftarrow Q * U$, de modo que $\mathcal{R}$(id, a, t), $\mathcal{Q}$(id, a, b, c), $\mathcal{U}$(id, a). Neste caso, é possível usar a procrastinação de atividade e transformar $Q \leftarrow Map(Y, R)$ em $Q \leftarrow Map(Y, \pi_{atr\,(R)}(R * U))$. Adicionalmente, por conveniência, a expressão algébrica pode ser desmembrada por meio de uma relação intermediária *V* para a operação de $\overline{Map}$, *i.e.*, $\{V \leftarrow \pi_{id,a,t}\,(U * R), Q \leftarrow \overline{Map}(Y_2, V)\} \equiv Q \leftarrow \overline{Map}(Y_2, U * R)$. Este exemplo pode ser visualizado na transformação da Figura 22.c em Figura 22.d.

Pode-se apresentar um exemplo completo considerando um *workflow* expresso como: $S \leftarrow Map(Y_1, R)$, $T \leftarrow Filter(Y_2, S)$ com os seguintes esquemas para as relações: $\mathcal{R}$(id, a, t), $\mathcal{S}$(id, a, b, c), $\mathcal{T}$(id, a, b, c), ca(Y_1,R) = {id, t} e ca(Y_2,S) = {id, a}. O *workflow* tem dependências de dados expressas na Figura 22.a, na qual a relação *S* é produzida a partir da operação *Map*, que consome a relação *R*. A relação *T* é produzida a partir da operação *Filter*, que consome *S*. Neste exemplo podem ser aplicadas três transformações algébricas (decomposição de atividade, antecipação de atividade e procrastinação de atividade), descritas a seguir:

(i) regra de decomposição em $S \leftarrow Map(Y_1, R)$ e $T \leftarrow Filter(Y_2, S)$, obtendo-se $Q \leftarrow \overline{Map}(Y_1, R)$, $S \leftarrow (Q * \pi_{id,a} R)$ e $U \leftarrow \overline{Filter}(Y_2, S)$, $T \leftarrow (U * \pi_{id,b,c} S)$. Esta transformação pode ser observada da Figura 22.a para a Figura 22.b;

(ii) antecipação de atividade em $U \leftarrow \overline{Filter}(Y_2, S)$, obtendo-se $U \leftarrow \overline{Filter}(Y_2, R)$. Esta transformação pode ser observada da Figura 22.b para a Figura 22.c;

(iii) procrastinação de atividade em $Q \leftarrow \overline{Map}(Y_1, R)$, obtendo-se: $V \leftarrow \pi_{id,a,t}(R * U)$, $Q \leftarrow \overline{Map}(Y_1, V)$. Esta transformação pode ser observada da Figura 22.c para a Figura 22.d;

Como pode ser visto na Figura 22.d, é possível produzir as mesmas saídas *S* e *T*; entretanto, ao se executar a operação *Filter* antes da operação *Map* pode-se obter um ganho de desempenho, uma vez que há um decréscimo do número de tuplas para executar a operação *Map*. Cabe salientar que as transformações adicionam algumas relações temporárias, que são manipuladas por meio de operações da álgebra relacional. Entretanto, estas operações são computacionalmente insignificantes quando comparadas à duração da invocação de programas envolvidos nas operações *Filter* e *Map*.

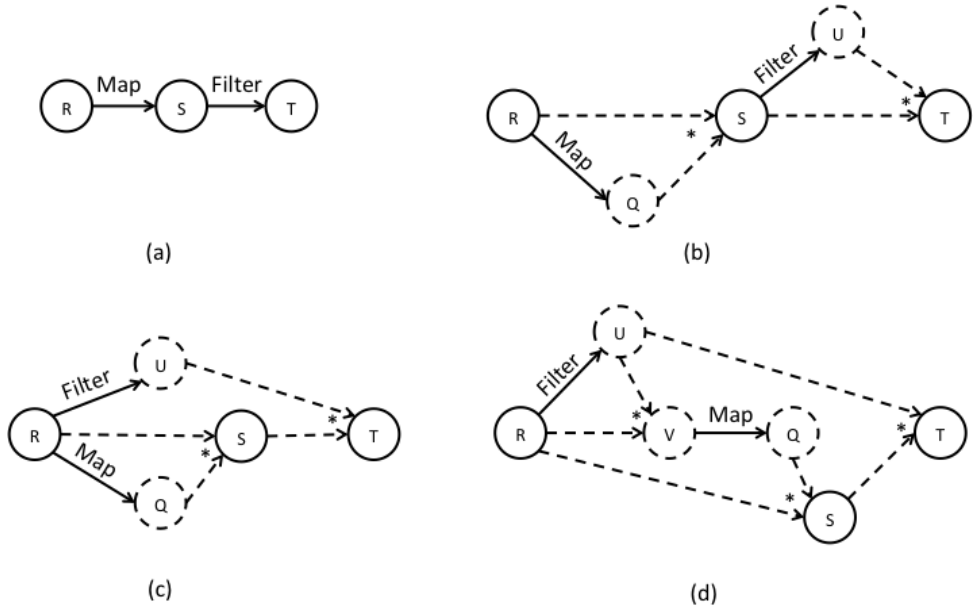

Figura 22 - Dependência de dados antes das transformações algébricas (a); Dependências de dados após as transformações algébricas (b)

As regras de transformação possibilitam as alterações na ordem de execução das atividades por meio da alteração nas dependências de seus dados. Durante as transformações, novas dependências de dados podem ser incluídas, desde que não violem as dependências de dados originalmente definidas nos *workflows*. Os conceitos de atributos de consumo e de preservação de chave permitem uma série de transformações algébricas que podem ser aplicadas a expressões algébricas que representam *workflows* científicos.

4.3.3 Algoritmo de otimização para execução paralela de *workflows*

O algoritmo proposto para otimização da execução paralela de *workflows* foi implementado explorando o espaço de busca e utilizando uma função de custo para encontrar uma configuração "ótima". O espaço de busca é definido pelo conjunto de configurações de *workflow* alternativas obtidas por meio da aplicação de regras de

transformação apresentadas na Seção 4.3.2. Em um primeiro momento, o número de alternativas de configurações que podem ser produzidas por meio da aplicação de regras de transformação é O(*N!*), no qual *N* representa o número de atividades. No entanto, as dependências entre as atividades limitam a possibilidade de mudar a ordem entre as atividades, uma vez que reduzem as regras de transformação que podem ser aplicadas. Basicamente, a ordem de execução de atividades dependentes só pode ser alterada quando for possível aplicar as regras de antecipação e procrastinação. Neste cenário, a preservação de chave é uma premissa fundamental para permitir que estas duas transformações possam ocorrer. A ausência desta premissa reduz o espaço de busca, limitando o número de configurações alternativas.

Estas configurações são equivalentes, à medida que elas produzem o mesmo resultado, mas diferem na ordem de execução das operações e, portanto, diferem no que tange ao desempenho. De modo a possibilitar uma comparação das configurações, apresenta-se um modelo de custo cuja função é prever o custo computacional de uma determinada configuração. Lembrando-se da premissa do uso de um *cluster* com disco compartilhado, não há quaisquer restrições em matéria de transferência de dados. A função de custo para o *workflow* inteiro é definida pela soma dos custos de execução de cada atividade. O custo de *workflow* (*Wc*) pode ser expresso pela soma do custo das *n* atividades (Yc_i) que compõem o *workflow*:

$$Wc = \sum_{i=1}^{n} Yc_i \quad (1)$$

Além disso, o custo da i-ésima atividade pode ser expresso como uma função do número estimado de ativações (a_i), o tempo de execução média da atividade ($\bar{t}_i$) e o número de núcleos disponíveis[1] (c_i) para que a atividade execute:

$$Yc_i = a_i \cdot \frac{\bar{t}_i}{c_i} \quad (2)$$

O número estimado de ativações é influenciado pelo número de tuplas das relações de entrada e pelo tipo de operação. As relações de entrada podem ser influenciadas pelos dados produzidos por uma atividade anterior. Para as equações (1) e (2), observa-se que, ao se diminuir o número de tuplas das relações de entrada para a execução da atividade, pode-se diminuir o custo total das atividades. A otimização de *workflows* científicos é conduzida por meio da definição de heurísticas que apoiem transformações algébricas que reduzam o número de tuplas nas relações intermediárias. É claro que as operações algébricas precisam ser analisadas de acordo com sua relação de dados consumidos/produzidos durante o processo de otimização. Desta forma, as operações como *Reduce*, *Filter* e *SRQuery*/*MRQuery* (quando diminuem o número de tuplas) precisam ser ordenadas o mais precocemente possível, enquanto que operações

[1] Assume-se que para fins de custos que os processadores do clusters são todos homogêneos.

como *SplitMap*, *SRQuery*/*MRQuery* (quando aumentam o número de tuplas) devem ser adiadas o máximo possível.

A Figura 23 apresenta um algoritmo que realiza uma estratégia de busca por configurações visando a otimizar a execução paralela do *workflow*. Cada configuração explorada é avaliada aplicando-se o modelo de custo. A estratégia de busca para a otimização de *workflow* é inspirada na técnica de migração de predicado usada em sistemas de bancos de dados relacionais (Hellerstein e Stonebraker 1993). O algoritmo apresentado na Figura 23 se concentra na identificação de fluxos de atividades que apresentem preservação de chave, separando-os em fragmentos (análogos a *streams* no algoritmo tradicional de migração de predicados), obtidos pela chamada à função *getKeyPreservationFragments* da linha 1 do algoritmo. Em cada fragmento, as atividades são decompostas (regras de decomposição) de modo a permitir a aplicação das demais regras de transformação (chamada a função *decomposeActivities* da linha 3 do algoritmo). Em cada fragmento ordenam-se as atividades por seus respectivos *ranks*. O *rank* de uma atividade Y_i é definido como:

$$\mathrm{rank}_i = s_i \cdot \frac{\bar{t}_i}{c_i} \qquad (3)$$

O *rank* se assemelha ao custo de uma atividade, na qual se substitui o número estimado de ativações pelo fator de seletividade (s_i) da atividade. O fator de seletividade de uma atividade é função do tipo de operação e das características da atividade. O fator de seletividade para uma determinada atividade Y_i pode ser definido como:

$$s_i = \frac{\bar{t}_i}{c_i} \qquad (3)$$

Dentro de cada fragmento, as atividades são ordenadas de modo crescente de *rank*. O objetivo é fazer com que as atividades que mais diminuam a quantidade de tuplas sejam priorizadas para execução o quanto antes no fragmento. Assim, na linha 4, faz-se a chamada à *sortByRank* para se obter a lista ordenada de modo crescente dos *ranks*. Para cada atividade *act* presente nesta lista ordenada, computam-se as atividades que antecedem a atividade *act* ordenadas de modo decrescente de distância, ou seja, quer-se realizar a troca de posições entre as atividades de maior *rank* por aquelas que estejam mais próximas do começo do fragmento. Esta lista decrescente *deps* é obtida pela chamada *sortByDep*. A cada atividade *y* que esteja na lista *deps* avalia-se se é possível realizar a troca de ordem entre as atividades e se a troca melhora o custo da execução do *workflow*. Todas as atividades do grupo são avaliadas duas a duas tentando-se procrastinar as atividades que aumentam o número de tuplas ou antecipar aquelas que diminuam o número de tuplas, desde que a função de custo compute a melhora na operação. Se houver melhora no custo, as atividades são trocadas e parte-se para buscar uma nova atividade *act* na lista *rankedAct*.

```
optimizeWorkflow(Workflow W)

1.  fragments = getKeyPreservationFragments(W);
2.  for each frag in fragments
3.    decomposeActivities(frag)
4.    rankedAct = sortByRank(frag)
5.    act = rankedAct.head
6.    while act ≠ null
7.      deps = sortByDep(act, frag)
8.      y = deps.head
9.      while y ≠ null
10.       cost = computeChange(act, y)
11.       if cost.improved
12.         reorder(act, y, cost, frag)
13.         y = null
14.       else
15.         y = deps.next()
16.     act = rankedAct.next()
17. assignStrategiesWorkflow(W)
```

Figura 23 - Algoritmo de otimização de execução do *workflow*

A complexidade do algoritmo é $O(n^2)$, no qual *n* representa o número de atividades do *workflow*. Cabe salientar que apesar de existirem três estruturas de varredura aninhadas (*for each*, *while* e *while*), o primeiro *for each* é complementar ao primeiro *while*, ou seja, o número de atividades processadas pelo primeiro *while* é dependente do número de atividades particionadas nos fragmentos. Assim, na prática, o pior caso ocorre quando se tem um único fragmento contendo todas as *n* atividades que compõem o *workflow*.

Uma vez obtido o *workflow* otimizado, que minimize o número de relações intermediárias, de acordo com as dependências de dados, é necessário definir os fragmentos do *workflow* com as suas respectivas estratégias de fluxo de dados e despacho atribuídas. Isto é feito na chamada à função *assignStrategies*(). Esta função é descrita no algoritmo de atribuição de estratégias apresentado na Figura 24. Os passos 1-5 seguem os preceitos descritos na seção 4.3.1. Inicialmente, marcam-se as atividades bloqueantes. A partir delas, se estabelecem os fragmentos. Posteriormente, são realizadas a atribuição dos fluxos de dados e o despacho para cada fragmento.

```
assignStrategies (Workflow W)

1.  markBlockingActivities();
2.  fragSet = computeWorkflowFragments();
3.  for each frag in fragSet
4.     computeDataflowStrategy(frag);
5.     computeDispatchStrategy(frag);
```

Figura 24 - Algoritmo de atribuição de estratégias do *workflow*

Capítulo 5 - Chiron: Motor de execução paralela de *workflows* científicos

No Capítulo 4 foi apresentada a abordagem algébrica para *workflows* científicos. Este capítulo apresenta o Chiron, que é um motor de execução paralela de *workflows* científicos. A motivação do desenvolvimento do Chiron foi decorrente do bom desempenho obtido pelo Hydra na distribuição de atividades de varredura de parâmetros e de fragmentação de dados na resolução de problemas reais (Coutinho et al. 2010, 2011, Guerra et al. 2009, Ogasawara et al. 2009b). A partir da análise de desempenho destas aplicações foi observada a necessidade de se paralelizar o *workflow* como um todo e de se identificar as oportunidades para otimização automática da distribuição das atividades em tempo de execução. Para contemplar estas demandas, primeiramente a abordagem algébrica foi concebida visando a um tratamento mais uniforme para a especificação do *workflow* e, posteriormente, foi realizada a implementação desta abordagem para execução paralela de *workflows* por meio do Chiron. Na subseção 5.1 é apresentada a arquitetura do Chiron. Os *workflows* processados pelo Chiron são definidos por meio de uma linguagem declarativa, representada em XML, que transforma a definição do *workflow* em expressões algébricas. Este processo é apresentado na subseção 5.2. A subseção 5.3 apresenta os detalhes referentes à otimização e execução de *workflows* no Chiron. Finalmente, a subseção 5.4 apresenta o modelo de proveniência do Chiron.

5.1 Arquitetura

O Chiron é um motor de execução de *workflows* projetado para executar em ambiente de PAD. Desta forma, a aplicação faz-se valer de uma arquitetura baseada em sistema de troca de mensagens, no qual processos do motor de execução são executados ao longo dos nós computacionais no ambiente PAD. A Figura 25 apresenta uma visão de alto nível da arquitetura do Chiron sob a perspectiva de processos.

Observa-se, pela arquitetura, que o Chiron é uma aplicação baseada em MPI (Gropp et al. 1999), ou seja, em cada nó computacional ele apresenta uma instância da aplicação identificada por meio de um identificador global. A Figura 25 assume a existência de *k* nós computacionais, nos quais as instâncias do Chiron são inicializadas para execução do *workflow*. No cenário avaliado para os testes, assumiu-se sempre a existência de disco compartilhado, ou seja, todos os nós são capazes de manipular quaisquer dados do experimento.

A execução paralela do *workflow* em um ambiente de PAD apresenta algumas dificuldades para a gerência de atividades e coleta de proveniência. Visando a um bom apoio à gerência de atividades e suas respectivas ativações, o Chiron foi inicialmente projetado e implementado para trabalhar com um banco de dados relacional. Cada uma

das instâncias pode executar ativações; entretanto, por fins de simplificação de gestão da execução das ativações, apenas a instância 0 (*nó cabeça*) pode acessar o banco de dados para ler/escrever dados de controle de execução e de proveniência.

Desta forma, o Chiron requer alguns programas adicionais para execução, a saber: o PostgreSQL (PostgreSQL 2009), o arcabouço MPJ (Carpenter et al. 2000, 2000, Shafi et al. 2009), a máquina Java e suas respectivas bibliotecas adicionais, como o HSQLDB (Simpson e Toussi 2002), os *drivers* JDBC do PostgreSQL e as bibliotecas do MPJ, todas adicionadas ao código fonte do Chiron. O Chiron encontra-se disponível para download em http://datluge.nacad.ufrj.br/chiron.

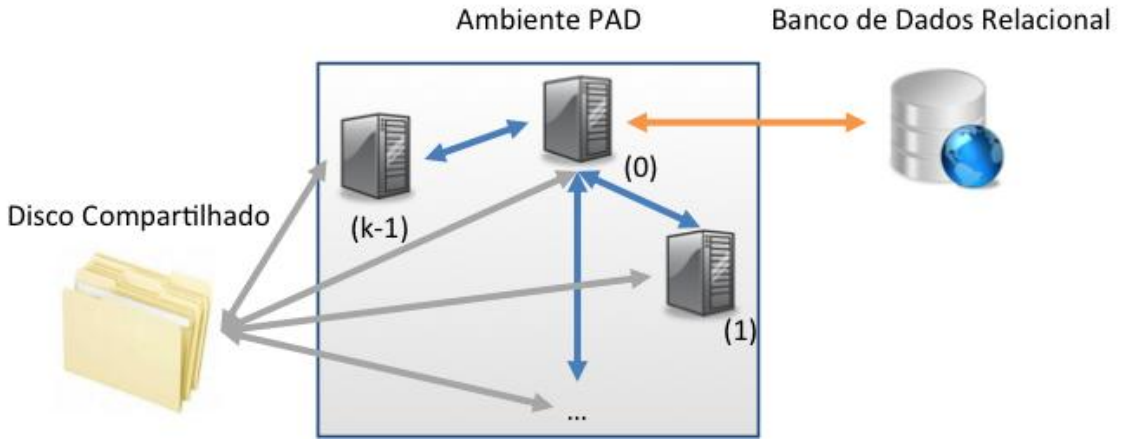

Figura 25 - Arquitetura do Chiron - Visão de Processos

O Chiron foi desenvolvido em Java e faz-se valer do MPJ, que adota como primitiva a conhecida técnica de paralelismo aninhado (Shafi et al. 2009), que mistura a distribuição entre processos por troca de mensagens em MPI e execução em vários núcleos por meio de linhas de execução (do inglês, *threads*). Assim, o paralelismo aninhado do Java com MPJ é bem adequado, uma vez que há uma tendência cada vez maior de que os nós computacionais possuam vários núcleos. A Figura 26 apresenta uma arquitetura do Chiron na visão das linhas de execução, na qual se tem *k* nós e *n* núcleos de processamento por nó. Os retângulos representam cada uma das instâncias do Chiron. Assim, cada nó tem uma instância do Chiron com uma linha de execução para distribuições de ativações. Apenas as linhas de execução de distribuição de ativações são usadas para realizar comunicações entre as instâncias e o processador de *workflows* existe tão somente no nó de identificador zero. Em cada instância, há várias linhas de execução para gerenciar a execução de ativações. Quando uma linha de execução responsável por processar ativações termina, ela sinaliza para a linha de execução de distribuição de ativações que há ativações concluídas para serem transmitidas ao nó cabeça, bem como realiza o pedido de novas ativações. A linha de execução de distribuição de ativações envia as ativações concluídas e pede novas ativações para o processador de *workflows*, que se encarrega de registrar a proveniência das ativações no banco de dados e de produzir novas ativações para responder ao

processador de *workflows*. Como algumas atividades são bloqueantes, algumas vezes o processador de *workflow* responde com mensagens de espera em vez de responder com novas ativações. O distribuidor de ativações volta a solicitar novas ativações posteriormente. Quando não houver mais a possibilidade de produzir novas ativações para serem processadas, o processador de *workflows* responde com uma mensagem de término de execução. O distribuidor de ativações, ao receber este tipo de mensagem, termina a execução daquela instância assim que todos os processadores de ativações terminarem as execuções em andamento. Toda a comunicação entre as diferentes instâncias é feita via consulta periódica (do inglês, *polling*) (Tanenbaum 2007) por meio das primitivas *send* e *receive* do MPI.

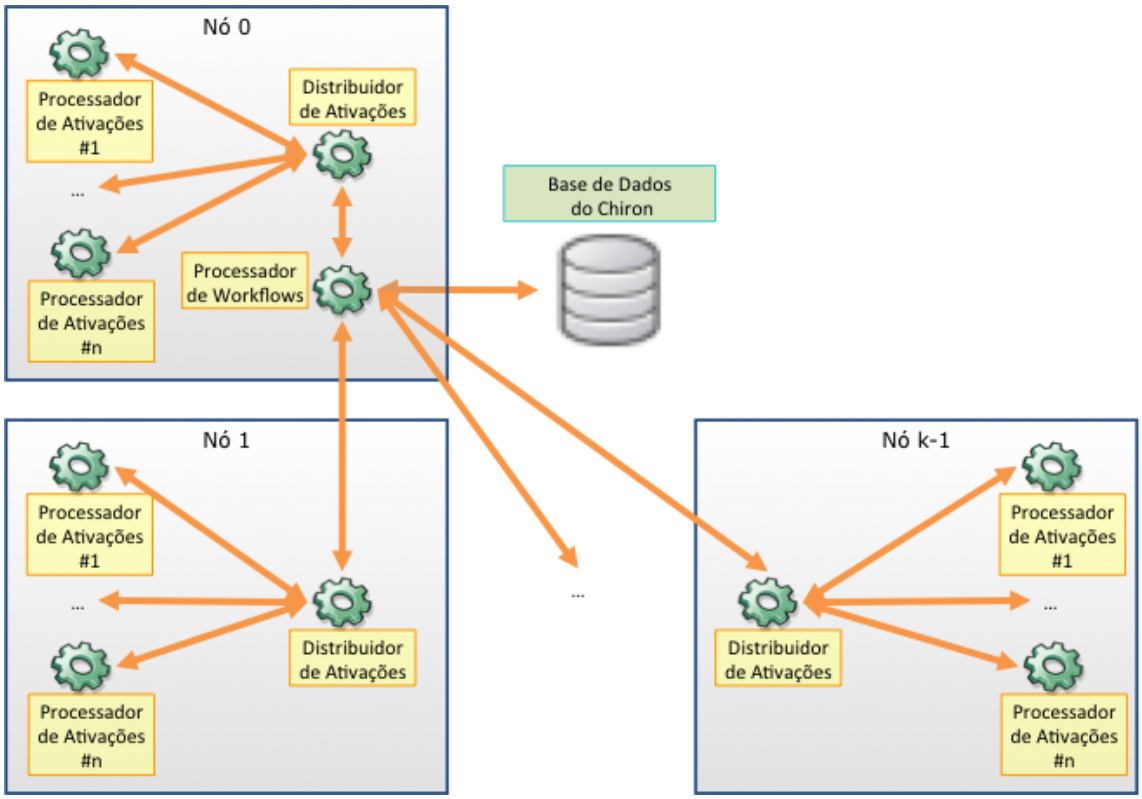

Figura 26 - Arquitetura do Chiron - Visão de Linhas de execução

5.2 Representação de *workflows*

O Chiron trabalha com um modelo algébrico para realizar a execução paralela de suas ativações. De acordo com o modelo algébrico implementado no Chiron, os dados de um *workflow* são representados como relações e todas as atividades do *workflow* são regidas por operações algébricas (*Map*, *SplitMap*, *Reduce*, *Filter*, *SRQuery* e *MRQuery*). O *workflow* é especificado por meio de um arquivo XML inspirado em XPDL (WfMC 2009). A especificação é feita desta forma, uma vez que se pretende que o *workflow* seja elaborado por meio de ferramentas de composição de *workflow*, como a GExpLine (Ogasawara et al. 2009c, Oliveira et al. 2010b).

A Figura 27 apresenta um exemplo de um *workflow* em XML com a especificação de *workflow* que encadeia duas atividades regidas por *SplitMap* e *Map*. Dentro do elemento raiz *Chiron*, o elemento *database* especifica a base de dados utilizada pelo Chiron para ler e gravar tuplas nas relações solicitadas pelas expressões algébricas, *i.e.*, as atividades do *workflow*. Logo abaixo, o elemento *Workflow* agrupa um conjunto de atividades. Cada atividade (*Activity*) é identificada por rótulos (atributo *tag*): *F* e *G*. A atividade *F* é uma operação algébrica do tipo *SplitMap*, que é ativada (atributo *activation*) pela aplicação Java *swb.jar*, que consome os atributos (atributo *Field*) *K* e *XF* de sua relação (*Relation*) de entrada. A atividade *F* possui uma relação de entrada *I_F* e uma relação de saída *O_F*. Os atributos *K*, *TG*, *XF* são consumidos e devem fazer parte da relação de saída. Já os atributos *I* e *XFF* são apenas produzidos pela atividade *F*.

```
<?xml version="1.0" encoding="UTF-8"?>
<Chiron>
 <database name="chiron-opt" server="146.164.31.200"
   port="5432" username="postgres" password="postgres"/>
 <Workflow tag="exp" description="exp" exectag="case1_OPT_8"
   expdir="/scratch/ogasawara/optimization/exp1/case1/1/exp">
 <Activity tag="F" description="F" type="SPLIT_MAP"
  activation="java -jar /optimization/SWB.jar
   -directory=%=DIREXP% -type=F -K=%=K% -F=%=XF%">
  <Relation reltype="Input" name="I_F" filename="I_F.txt"/>
  <Relation reltype="Output" name="O_F" filename="O_F.txt"/>
  <Field name="K" type="float" pk=”true”
   input="I_F" output="O_F"/>
  <Field name="TG" type="float" input="I_F" output="O_F"/>
  <Field name="XF" type="file" input="I_F" operation="COPY"/>
  <Field name="I" type="float" output="O_F"/>
  <Field name="XFF" type="file" output="O_F"/>
 </Activity>
 <Activity tag="G" description="G" type="MAP"
  activation="java -jar /optimization/SWB.jar
   -directory=%=DIREXP% -type=G -K=%=K% -T=%=TG%">
  <Relation reltype="Input" name="I_G" filename="I_G.txt"
   dependency="F"/>
  <Relation reltype="Output" name="O_G" filename="O_G.txt"/>
  <Field name="K" type="float" pk=”true”
   input="I_G" output="O_G"/>
  <Field name="I" type="float" pk=”true”
   input="I_G" output="O_G"/>
  <Field name="TG" type="float" input="I_G" output="O_G"/>
  <Field name="XFF" type="file" input="I_G" operation="COPY"/>
  <Field name="OTG" type="float" output="O_G"/>
 </Activity>
 </Workflow>
</Chiron>
```

Figura 27 - Especificação do *workflow* no Chiron que executa duas atividades

A atividade *G* possui uma relação de entrada *I_G* e uma relação de saída *O_G*. Note-se que a relação de entrada *I_G* tem dependência em relação à atividade *F*, ou

seja, a relação de saída *O_F* é atribuída à relação de entrada *O_G*. Os atributos *K*, *TG* são consumidos e devem fazer parte da relação de saída. Já o atributo *OTG* é produzido pela atividade *G*.

A Figura 28 apresenta o arquivo que representa a relação de entrada para a atividade *F*. O atributo *XF* é uma referência ao arquivo localizado no *ExpDir* (diretório do *workflow*) apresentado na Figura 27. Como se trata de uma entrada para uma operação *SplitMap*, cada linha na relação é transformada em uma ativação. O arquivo *XF* é dividido em várias partes. A semântica de divisão do arquivo é interna ao programa *swb.jar* invocado durante a ativação.

```
K;XF;TG
1;input1.txt;1684
2;input2.txt;7636
3;input3.txt;2741
4;input4.txt;3881
5;input5.txt;4821
6;input6.txt;2715
7;input7.txt;3211
8;input8.txt;1971
9;input9.txt;2497
10;input10.txt;6387
11;input11.txt;5391
12;input12.txt;2763
13;input13.txt;8370
14;input14.txt;5738
15;input15.txt;2071
16;input16.txt;5937
17;input17.txt;5953
18;input18.txt;3132
19;input19.txt;3021
20;input20.txt;2246
```

Figura 28 – Relação de Entrada para Atividade F

Note-se que, quando comparado às formas de se expressar *workflows* apresentadas na Figura 9, o arquivo de especificação do *workflow* para o Chiron tem uma característica declarativa, uma vez que se definem as atividades que fazem parte do *workflow* e se estabelece a relação de dependência entre a saída da atividade *F* e a relação de entrada da atividade *G*; entretanto, não se especifica nenhuma forma particular de execução.

5.3 Otimização e execução de *workflows*

O processo de otimização e execução de *workflows* é realizado a partir da entrada da especificação do *workflow* em XML. Resumidamente, o Chiron identifica as atividades, operações algébricas e relações de entrada na leitura do arquivo XML e realiza as otimizações algébricas visando a obter um *workflow* otimizado por meio da avaliação de sua função de custo. A partir do *workflow* "ótimo" é feita a decomposição do

workflow em fragmentos, que recebem uma atribuição de estratégias de fluxo de dados e de despacho.

Todos os fragmentos que contenham as relações de entrada já disponíveis são marcados no estado *Disponível*. Os fragmentos que não tenham produzidas as relações de entrada são marcados no estado *Dependente*. Na ausência de ativações para execução, os fragmentos em estado *Disponível* podem ser efetivados para o estado *Executando*, instanciando-se todas as ativações possíveis e deixando-as disponíveis para requisições por ativações feitas pelos distribuidores de ativações. Uma vez que todas as ativações de uma relação sejam concluídas, o fragmento é marcado no estado *Finalizado*, as relações produzidas no escopo daquele fragmento são persistidas, viabilizando-se a mudança de estado dos fragmentos que estão em estado *Dependente* para o estado *Disponível*. Quando não existirem mais fragmentos e ativações a executar, a execução do *workflow* é finalizada.

Apesar de tecnicamente as atividades regidas pelas operações *Reduce*, *SRQuery* e *MRQuery* (quando possuir comandos de agregação) e as atividades com restrições serem bloqueantes, a atual implementação do Chiron possui restrições adicionais, apresentadas na Tabela 4, por questões de simplificação de implementação.

Tabela 4 – Classificação das operações algébricas quanto ao tipo e à atividade

Operação	Tipo de Atividade
Map	Bloqueante ou Não-Bloqueante
SplitMap	Bloqueante
Reduce	Bloqueante
Filter	Bloqueante ou Não-Bloqueante
SRQuery	Bloqueante
MRQuery	Bloqueante

Pode-se retomar o exemplo apresentado na Figura 27 e na Figura 28 para se exemplificar o processo de otimização. A Figura 29.a apresenta o *workflow* imediatamente especificado pela Figura 27. A partir das transformações algébricas, o *workflow* passa a ser expresso pela Figura 29.b, na qual houve a inversão das atividades *G* e *F*. A atividade *G* ficou no fragmento F_1, que recebeu as estratégias *FTF* e dinâmica. A atividade *F* ficou no fragmento F_2, que recebeu as estratégias *FAF* e dinâmica.

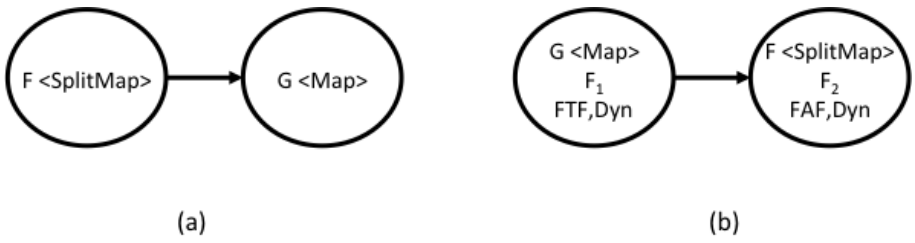

Figura 29 – *Workflow* inicialmente especificado (a); *Workflow* Otimizado (b)

5.4 Modelo de proveniência

O Chiron possui um mecanismo nativo de coleta de proveniência vinculado a um banco de dados relacional. Isto possibilita a gerência das ativações e o monitoramento em tempo de execução sobre o andamento da execução do *workflow*, apoiando os cientistas na realização de consultas que avaliam os dados de proveniência durante a execução do *workflow*. Desta forma, é possível saber quais atividades foram executadas ou estão executando e identificar se estão ocorrendo erros. Na base de dados do Chiron são armazenadas as ativações e os dados primitivos e referências a arquivos. Os arquivos propriamente ditos são mantidos na área de armazenamento da aplicação. O armazenamento destas informações estabelece o esquema de proveniência retrospectiva da execução do *workflow* em ambiente de PAD.

A Figura 30 apresenta o modelo de proveniência do Chiron. Em linhas gerais, os *workflows* são decompostos em atividades. As atividades podem possuir diversas relações de entrada e uma relação de saída. As relações possuem atributos. As atividades produzem diversas ativações. Cada ativação contém as tuplas correspondentes aos campos das relações que fazem parte do escopo da atividade. Finalmente, as atividades podem possuir arquivos de modelo que são usados para a execução das ativações. Da mesma forma, as ativações produzem e consomem arquivos no escopo de suas execuções. Os principais atributos das classes são apresentados a seguir.

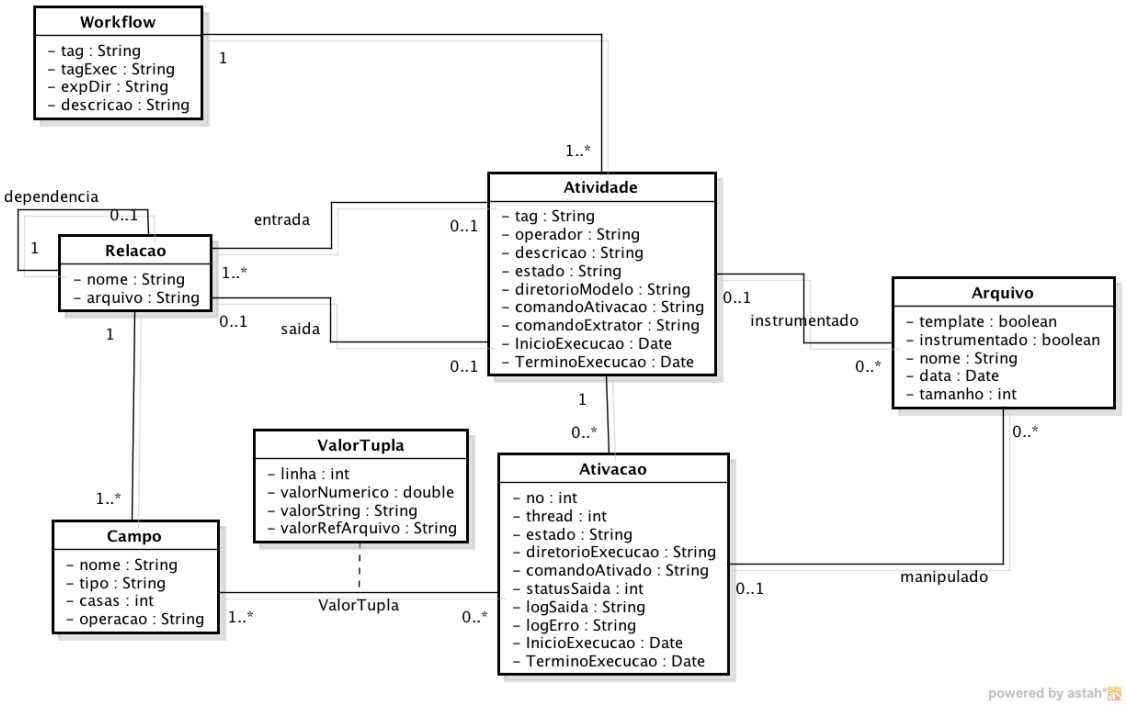

Figura 30 – Modelo de Proveniência Retrospectiva do Chiron

A classe *Workflow* possui os atributos *tag*, que servem para classificar um determinado tipo de *workflow*, e *tagExec*, que identifica unicamente um *workflow* executado. Quando um cientista solicita a execução de um *workflow* já existente, apenas as atividades não executadas são selecionadas para execução. O atributo *expDir* contém a indicação da localização dos dados no ambiente de disco compartilhado.

A classe *Atividade* possui um atributo *tag* que identifica a atividade no escopo de um determinado *workflow*. O atributo *operação* indica o nome da operação algébrica que rege a atividade. O estado representa os seguintes estados: *Enfileirada*, *Executando*, *Esperando* e *Executada*. O atributo *comandoAtivacao* é usado para instrumentação dos parâmetros a partir das tuplas. O atributo *comandoExtrator* é usado para executar o programa que extrai os dados do resultado da execução da ativação para transformá-los em tupla de saída. Os atributos *InicioExecucao*, *TerminoExecucao* contêm, respectivamente, a informação do início da execução da primeira ativação desta atividade e a informação do término da execução da última ativação desta atividade.

A classe *Ativacao* possui os atributos *no* e linha de execução que representam, respectivamente, o identificador do nó computacional no qual a ativação foi executada, bem como o número de identificação da linha de execução do processador de ativações. O atributo *diretorioExecucao* indica o local em que a ativação consumiu e produziu os seus dados. O atributo *comandoAtivado* apresenta o comando exato (pós instrumentação) utilizado na ativação, ou seja, com os valores das tuplas substituindo os valores parametrizados na definição do atributo *comandoAtivacao* de sua respectiva atividade. Os atributos *statusSaida*, *logSaida* e *logErro* são importantes para armazenamento de proveniência referente aos dados de execução da ativação. Os atributos *InicioExecucao* e *TerminoExecucao* contêm, respectivamente, a informação do início e término da execução desta ativação.

A classe *Relacao* contém atributos nome e arquivo, podendo também indicar dependência a outras relações, ou seja, uma relação de entrada pode depender de outra relação de saída. A classe *Campo* tem o nome e tipo dos atributos que fazem parte de uma relação. O atributo *casas* é usado para indicar o numero de casas decimais, quando o tipo for numérico. O atributo *operacao* é usado quando o tipo for *RefArquivo* para indicar a operação de movimentação dos arquivos, que pode ser *Manter*, *Mover*, *Copiar*, *Mover&Apagar*, *Copiar&Apagar.*

A classe *ValorTupla* indica o valor de uma tupla consumida ou produzida durante a execução de uma ativação. O atributo *linha* é usado para caracterizar uma tupla. Os atributos *valorNumerico*, *ValorString* e *ValorRefArquivo* são preenchidos de acordo com o tipo de atributo.

Finalmente, a classe *Arquivo* contém os arquivos de modelo que definem uma atividade, ou seja, arquivos que são copiados da atividade para cada ativação. Estes arquivos podem ser instrumentados, ou seja, ter suas *tags* substituídas por valores

instrumentados. A classe *Arquivo* também pode ser usada no contexto de uma ativação que serve para coletar as informações de todos os arquivos consumidos/produzidos durante a execução daquela ativação, ou seja, nome, data e tamanho.

Para exemplificar a base de dados do Chiron, a Figura 31 apresenta uma consulta sobre a base de proveniência. Esta consulta indica a duração em milissegundos de cada ativação para um determinado *workflow*.

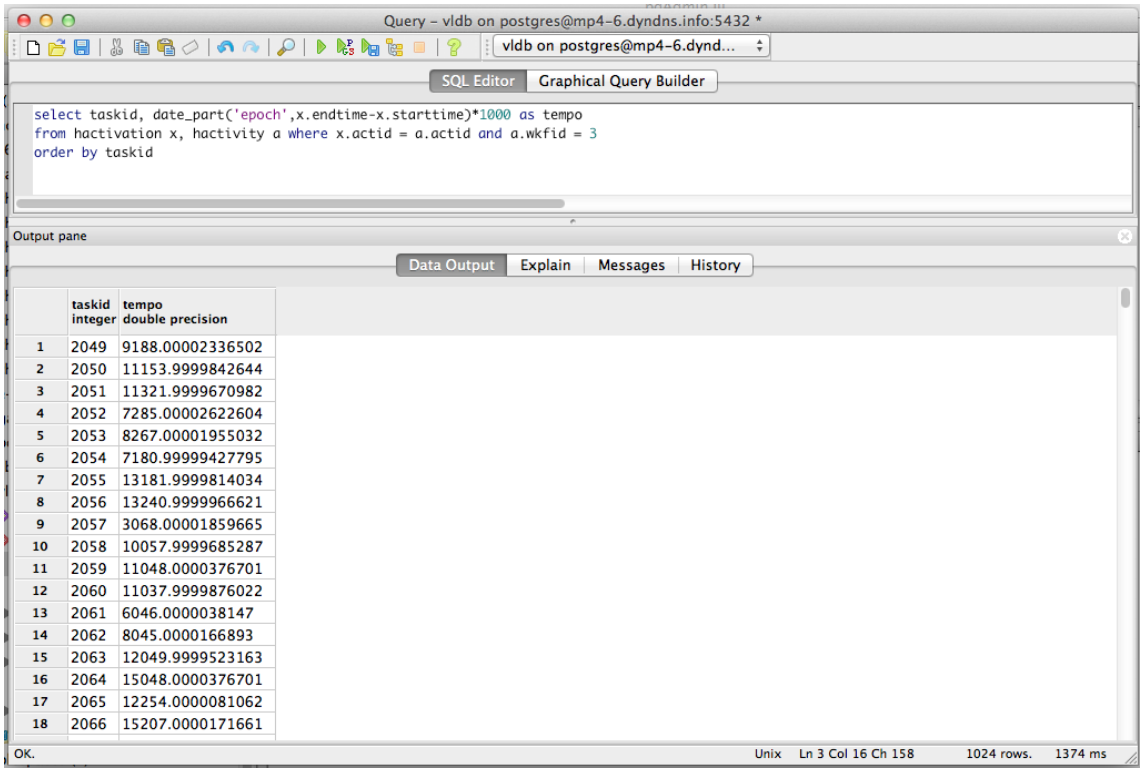

Figura 31 – Consulta sobre a base de proveniência do Chiron

Capítulo 6 - Avaliação experimental

Esta seção apresenta a avaliação experimental da abordagem algébrica. A seção 6.1 apresenta uma avaliação do algoritmo de atribuição de estratégias de despacho e de fluxo de dados. A seção 6.2 apresenta uma comparação do desempenho do Chiron frente às demais abordagens. A seção 6.3 apresenta a influência do algoritmo de otimização por meio da comparação de *workflows* com ou sem otimização algébrica. Finalmente, a seção 6.4 apresenta uma avaliação experimental usando *workflows* de análise de fadiga de *risers*.

6.1 Avaliação do algoritmo de atribuição de estratégias

A avaliação do algoritmo de atribuição de estratégias é feita por meio da comparação do desempenho da execução de *workflows* utilizando estratégias fixas com os *workflows* otimizados. As estratégias fixas representam a adoção, para os *workflows* como um todo, de uma das quatro combinações de estratégias de despacho e de fluxo de dados apresentadas: *S-FAF* (distribuição estática, *FAF*), *D-FAF* (distribuição dinâmica, *FAF*), *S-FTF* (distribuição estática, *FTF*) e *D-FTF* (distribuição dinâmica, *FTF*).

Na ausência de um benchmark para se medir o desempenho de *workflows* científicos, foram gerados *workflows* sintéticos inspirados em padrões de *workflow* (Aalst et al. 2003, Russell et al. 2006) para avaliar o algoritmo de atribuição de estratégias. Os cenários consideram apenas padrões de controle de *workflows* que são relevantes nos cenários de *workflows* científicos em larga escala presentes em muitos experimentos reais, como nos *workflows* do desafio de proveniência (do inglês, *Provenance Challenge*) (ProvChallenge 2010). Ademais, como se está assumindo uma arquitetura de disco compartilhado, os padrões de dados (Russell et al. 2005, 2004) para *workflow* não estão incluídos, uma vez que eles focam nas diferentes formas de se movimentar os dados nos *workflows*.

A avaliação dos *workflows* sintéticos utilizados para se medir as diferentes estratégias foi constituída por 176 execuções de *workflows*, totalizando 1.404 atividades com 667.648 ativações. A avaliação completa levou 46h52min de tempo de parede, correspondente a um tempo sequencial de 2,586h12min. Os experimentos foram realizados usando 8 nós com 8 núcleos em cada nó, totalizando 64 núcleos. Entretanto, nesta seção são apresentadas apenas as avaliações referentes a um subconjunto dos *workflows* sintéticos que tiveram comportamentos representativos. A avaliação completa usando todo o conjunto de *workflows* sintéticos pode ser observada em Ogasawara et al. (2011).

A Tabela 5 apresenta um conjunto de variáveis usadas durante as avaliações dos *workflows* sintéticos. As primeiras três variáveis influenciam a estrutura do grafo dos

workflows. As quatro últimas variáveis configuram o comportamento da execução: atividades e quantidade de dados.

Tabela 5 - Variáveis usadas na avaliação

Variável	Descrição
MD	Comprimento médio de sequências de atividades regidas pela operação *Map* no *workflow*.
FI	Maior grau de entrada para uma atividade no *workflow*.
FO	Maior grau de saída para uma atividade no *workflow*.
AC	Custo da atividade: Valor aleatório que segue uma distribuição *Gamma (*k, θ) (Larsen e Marx 2011) com forma θ = 1 e k = AC, que corresponde ao número de segundos médio para se executar a ativação.
IS	Tamanho da entrada: número de tuplas presentes na relação de entrada para o *workflow*.
SF	Fator de multiplicação: nas atividades regidas pelo *SplitMap*, o número de tuplas produzidas é definido pelo fator de multiplicação.
CA	Atividade com restrição: considerando um nó contendo *n* núcleos, uma atividade com restrição executada neste nó impede o uso de outros núcleos do nó enquanto estiver sendo executada.

6.1.1 *Workflows* de sequência de atividades

Conforme apresentado na Figura 32, o primeiro cenário de avaliação é constituído por uma sequência de operações *Map*, correspondendo ao uso repetido dos padrões de *workflow* (Aalst et al. 2003) de sequência (WCP-1). Desta forma, para *MD* = 1 tem-se um *workflow* composto por duas atividades regidas por *Map* em sequência, enquanto para *MD* = 5 tem-se um *workflow* composto por seis atividades regidas por *Map* em sequência. Como a existência de sequência de atividades é muito comum em diversos *workflows* científicos, neste primeiro cenário apresentam-se três análises.

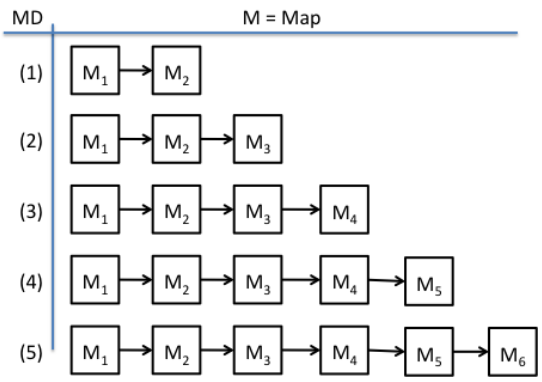

Figura 32 – *Workflows* variando-se o tamanho da sequência de *Map*

Na primeira análise é variado o parâmetro *MD* de 1 a 5 com *IS* fixo em 512 tuplas de entrada e *AC* = 10. Na ausência de atividades bloqueantes, o algoritmo de definição de estratégia cria um único fragmento com a estratégia *D-FTF*. Conforme pode ser observado na Figura 33, comparando-se as estratégias fixas com a estratégia otimizada, observa-se que a estratégia otimizada é idêntica à estratégia *D-FTF*, apresentando desempenho superior ao das demais.

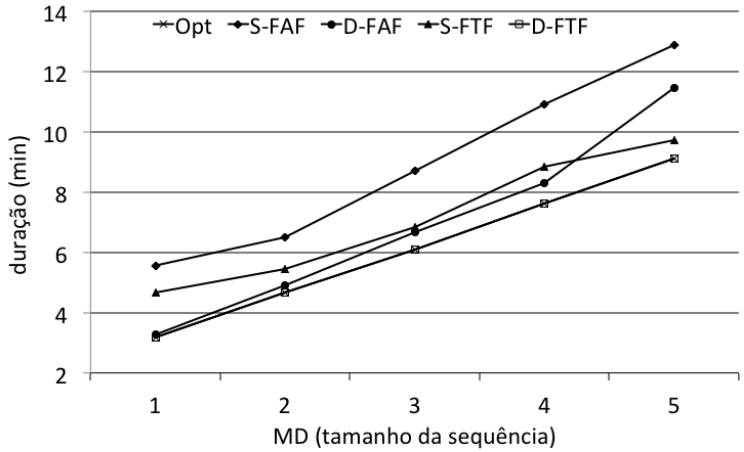

Figura 33 – Análise do tamanho de uma sequência de atividades

A segunda análise foi feita variando-se o *AC* de 5 a 25, considerando-se *IS* fixo igual a 512 tuplas de entrada e *MD* igual a 3, *i.e.*, *workflow* com 4 atividades em sequência. A Figura 34 apresenta o desempenho das abordagens variando-se o *AC*. A estratégia otimizada é feita a partir da definição de um único fragmento com a estratégia *D-FTF*. Note-se que mesmo com *AC* igual a 5, como o fragmento tem 4 atividades, o somatório do tempo médio das atividades é igual a 20, justificando a escolha da distribuição dinâmica. As estratégias *S-FTF* e *D-FAF* apresentam resultados semelhantes, mas o *D-FTF* apresentou o melhor desempenho, confirmando a boa escolha do algoritmo de atribuição de estratégia. A diferença de desempenho entre a melhor e a pior configuração foi de 27,6%.

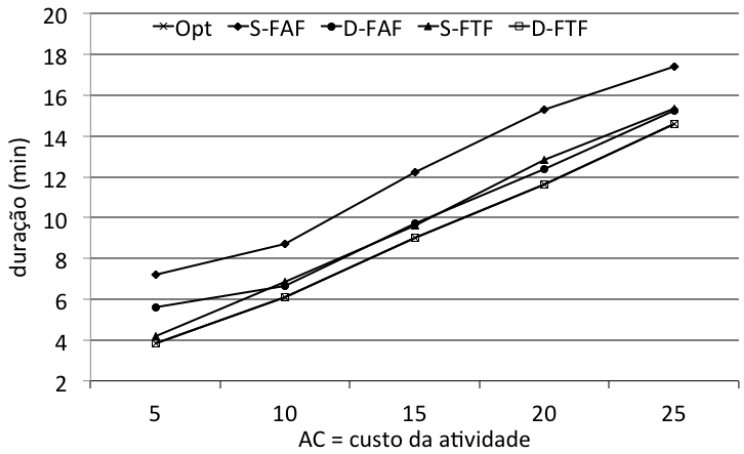

Figura 34 – Análise da variação de *AC* para *MD* = 3

A terceira análise avalia a influência da presença de uma atividade com restrição (*CA*). Conforme mencionado anteriormente, é muito comum o cenário de se invocar programas que são implementados de forma paralela e que precisam se fazer valer da quantidade máxima possível de núcleos de um nó. Devido a esta característica, uma ativação é dita pertencente a uma atividade com restrição se ela bloquear todos os núcleos, impossibilitando outras alocações para as demais ativações. Nesta análise, varia-se o número de atividades *CA* de 0 a 4, considerando-se *IS* fixo em 512, *MD* igual a 5 e *AC* igual a 5. A Figura 35 mostra que na presença de atividades com restrição no *workflow* (*CA*≥1), dentre as estratégias isoladas, as estratégias *FAF* apresentam o melhor desempenho. Entretanto, a versão otimizada apresenta o melhor desempenho quando comparada às estratégias isoladas, uma vez que o *workflow* é dividido em *1+CA* fragmentos. Os fragmentos contendo atividade com restrição devem ser executados usando *D-FAF*, enquanto que o fragmento adicional deve ser executado usando *D-FTF*. Uma *FAI* contendo ativação de atividades do tipo *CA* precisa esperar todas as demais *FAIs* em execução no nó terminarem para iniciar sua execução. Neste momento todos os núcleos são reservados para atender a esta *FAI*, bloqueando a execução dos demais processadores de ativação. A diferença entre a melhor e a pior estratégia pode chegar a 278%. Este resultado é importante, uma vez que em um *workflow* aparentemente simples tem-se casos concretos nos quais trechos diferentes do *workflow* devem apresentar estratégias diferenciadas.

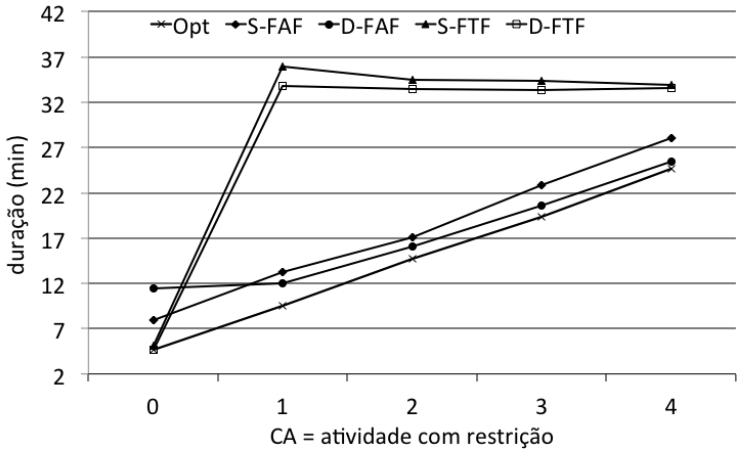

Figura 35 – Análise da variação de atividades do tipo *CA* em *workflows* com *MD* = 5

6.1.2 *Workflows* de sequência de atividades e atividade de sincronização

Este cenário avalia a influência da sincronização em *workflows*, sendo construído por uma combinação dos padrões de *workflow* de sequência (WCP-1) com o de sincronização (WCP-3). A avaliação usa sequências de tamanho *MD* e uma atividade de sincronização. No estudo faz-se uma avaliação do acréscimo do grau de entrada (*FI*) na atividade de sincronização. Para cada um dos ramos que levam à atividade de sincronização mantém-se sempre o *MD* fixo igual a 3. A Figura 36 apresenta os

diferentes *workflows* obtidos ao se variar o parâmetro *FI* de 1 a 5 e ao se manter fixo o *MD* igual a 3.

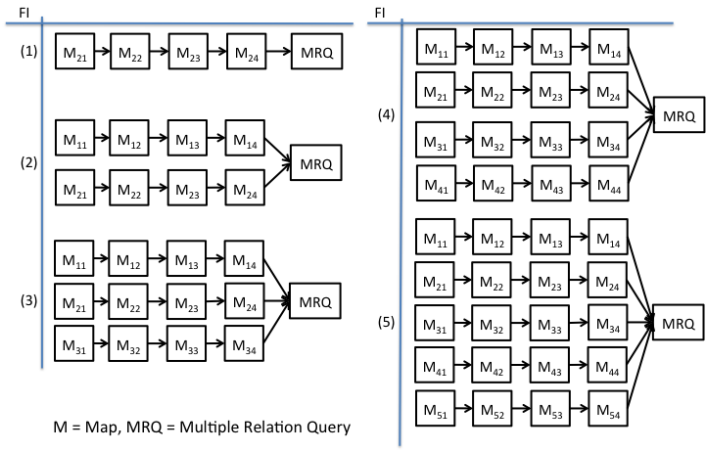

Figura 36 – *Workflows* variando o grau de entrada em atividades de sincronismo

A sincronização dos ramos de *Map* com *MD* igual a 3 é feita por meio do *MRQuery*. A avaliação varia o *FI*, considerando-se *AC* fixo em 10, *MD* igual a 3 e *IS* igual a 512 tuplas de entrada. Conforme apresentado na Figura 37, o algoritmo de otimização marca a atividade *MRQuery* como bloqueante (recebendo a estratégia *D-FAF*) e marca os ramos de sequências de *Map* como fragmentos do tipo *D-FTF*. Ao se aumentar o *FI*, o desempenho do *D-FTF* aumenta, se comparado às demais estratégias. Neste caso, observa-se que o desempenho da versão otimizada é idêntico ao do *D-FTF*.

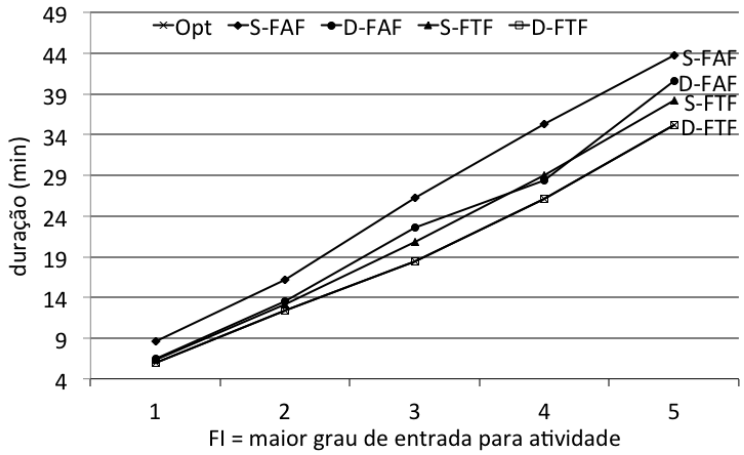

Figura 37 – Análise da variação do FI de 1 a 5

6.1.3 *Workflow* do tipo *MapReduce*

Este cenário avalia *workflows* com características de *MapReduce*. Este *workflow* é uma combinação dos padrões de *workflow* de divisão de linha de execução (WCP-42), sequência de atividades (WCP-1) e mescla de linha de execução (WCP-41). A Figura 38 apresenta o *workflow* que começa com uma atividade regida pela operação *SplitMap*, seguida por uma sequência de *Map* contendo quatro atividades e finaliza com um *Reduce*.

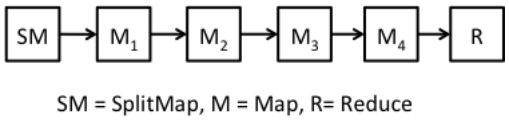

Figura 38 – *Workflow* de *MapReduce*

O experimento varia o número de tuplas de entrada (*IS*) com os seguintes valores: 128, 256, 384, 512, 640, com *AC*, *SF* e *MD* fixados, respectivamente, em 10, 3 e 3. Os resultados experimentais deste cenário são apresentados na Figura 39. O algoritmo de otimização produziu o mesmo desempenho do *D-FTF*. Cabe ressaltar que, nos cenários de *FTF*, quando a atividade só apoia *FAF*, é sempre separada em um fragmento isolado *FAF*. Neste caso, o *FTF* é configurado apenas nos casos possíveis e, por isso, o desempenho otimizado se iguala ao desempenho do *D-FTF*.

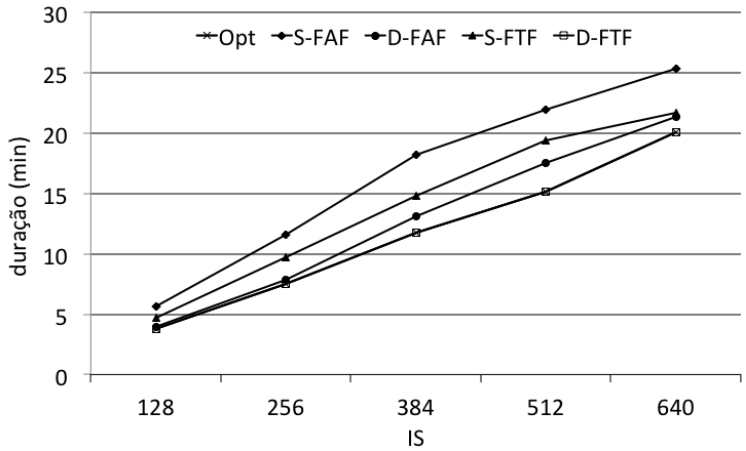

Figura 39 - Análise da variação do *IS* no *Workflow* de *MapReduce*

6.1.4 *Workflows* de divisão paralela com sequência de atividades e atividade de sincronização

Este cenário (Figura 40) é uma combinação de padrões de *workflows* de divisão paralela (WCP-2), um conjunto de sequências (WCP-1) e sincronização (WCP-3) em um *workflow* único. A saída do primeiro *Map* é repassada para cada um dos ramos. Cada um dos ramos contém uma atividade *SRQuery* seguida de uma sequência de *Map*. A saída dos ramos é sincronizada por meio de *MRQuery*.

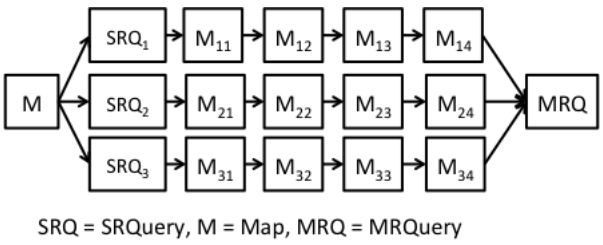

Figura 40 - *Workflow* combinado

O experimento (Figura 41) varia *AC* de 5 a 25 com *MD*, *FO*, *FI* e *IS* fixados, respectivamente, em 3, 3, 3 e 512. De acordo com o algoritmo de otimização, quando o

AC é baixo, a estratégia usada é estática, uma vez que a estratégia dinâmica gera uma sobrecarga de comunicação. A estratégia otimizada marca a primeira atividade como *S-FAF*, as atividades bloqueantes como *D-FAF* e as atividades e as sequências de *Map* de cada ramo como *D-FTF*.

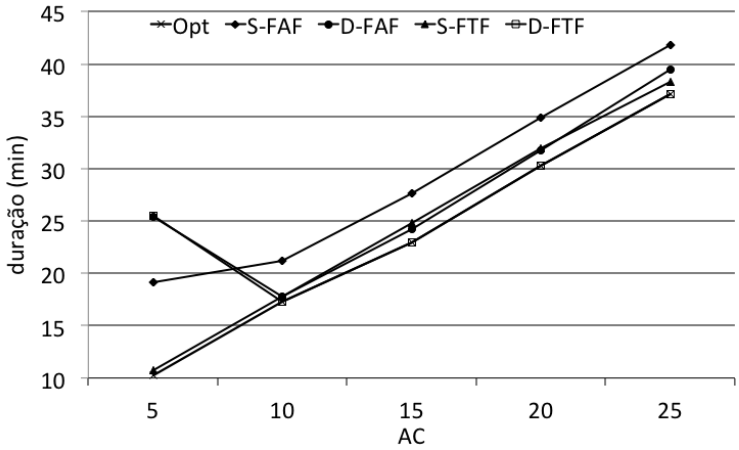

Figura 41 – Análise da variação do *AC* no *workflow* combinado

6.1.5 Considerações sobre algoritmo de otimização

Esta seção apresenta um resumo da análise de desempenho obtida a partir dos dados sintéticos, corroborando os critérios adotados pelo algoritmo de atribuição de estratégia. As sequências de *Map* ocorrem frequentemente em *workflows* de aplicações científicas. Neste cenário avaliado, como apresentado na Figura 32, o comportamento da estratégia *FAF* sofre devido ao sincronismo imposto a partir da execução guiada por atividade. A ocorrência de um desbalanceamento da execução provoca uma inatividade dos núcleos assim que eles terminam a execução de suas últimas ativações. O fenômeno é proporcionalmente agravado à medida que o *workflow* aumenta o seu tamanho e o período de ociosidade é acumulado ao longo das atividades do *workflow*, conforme apresentado na Figura 42. O desempenho obtido por meio do uso do despacho dinâmico diminui o efeito do sincronismo, uma vez que tem-se uma distribuição de execução nas quais alguns processos consomem maior quantidade de ativações, compensando as ativações de maior duração. Entretanto, o ganho obtido no uso de estratégias dinâmicas em *FAF* não é suficiente para garantir uma vitória sobre o *D-FTF*. Neste último caso, a consequência de um *FAI* não balanceado é o atraso na execução, cujo tempo, no pior dos casos, corresponde ao tempo de execução do último *FAI* executado. Assim, no pior caso, a última execução do *FAI* acaba ocorrendo em paralelo com outros *FAI*, o que faz o desempenho do *D-FTF* ser melhor do que na avaliação sincronizada presente em um *D-FAF*. A Figura 43 confirma a observação. Ela ilustra o comportamento de ociosidade implícito nos *workflows* da Figura 32. Na Figura 43 apresenta-se o percentual de ociosidade de núcleos durante a execução de cada uma das quatro combinações de estratégias apresentadas.

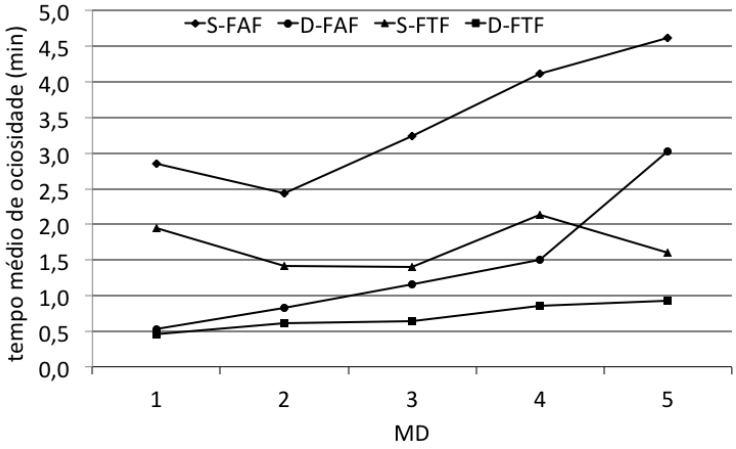

Figura 42 - Tempo médio durante a execução do *workflow* da Figura 33

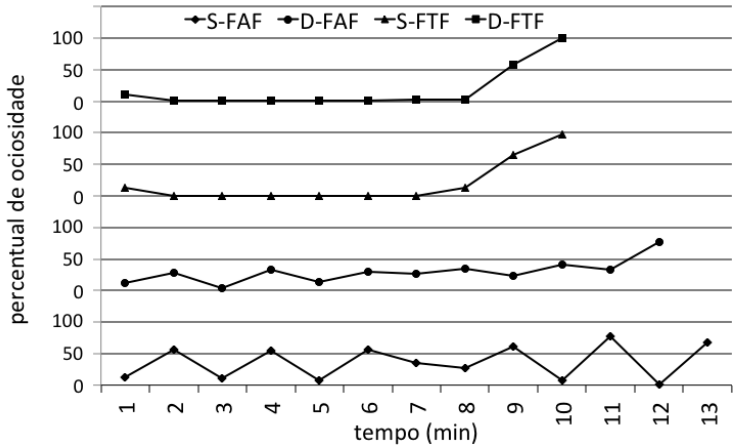

Figura 43 - Percentual de ociosidade durante a execução do *workflow* da Figura 33 com *WD* = 5 e *AC* = 10

Conforme mostrado na Figura 34 e em alguns outros casos, o desempenho relativo de *D-FAF* e *S-FTF* às vezes se alterna, de acordo com a variância do tempo de execução de ativações. Se a variância é baixa, o *S-FTF* tende a ser melhor do que o *D-FAF*; caso contrário, o *S-FTF* tende a ser pior. Um resultado interessante diz respeito à avaliação dos *workflows* com atividades com restrição (ver Figura 35). Na atual implementação do Chiron, quando um *FAI* apresenta ativações de atividades do tipo *CA*, todos os núcleos ficam reservados para aquele *FAI* durante toda a sua execução. Assim, durante a execução de ativações que não sejam de atividades *CA*, os demais *n-1* núcleos reservados naquele nó passam a ficar inativos. Observa-se que atividades do tipo *CA* trazem prejuízos a fragmentos com estratégia *FTF*, proporcionais ao tempo gasto pelas ativações restantes que compõem o *FAI*. Note-se, no entanto, que, se as ativações restantes forem significativamente menos dispendiosas do que a *CA*, ou se as ativações na *FAI* forem todas do tipo *CA* (tais como a execução de *workflow* com $CA \geq 4$ e $MD = 5$ apresentada na Figura 35), tanto o *FAF* quanto o *FTF* tendem a apresentar resultados semelhantes.

O cenário apresentado na Figura 36 explora *workflows* que tenham sincronismos. Observa-se pela Figura 37 que este cenário tem um comportamento semelhante a uma sequência de *Map*. Independentemente da estratégia de execução adotada, as atividades que processam expressões algébricas são executadas segundo a abordagem *FAF*. Assim, a comparação entre as estratégias de execução reduz-se ao caso de uma sequência de *Map*. Da mesma forma, o *workflow* expresso na Figura 40 mostra a presença de duas atividades que tenham tanto grau de entrada quanto de saída maiores que 1. Como esperado, o desempenho assemelha-se bastante ao dos resultados apresentados nos *workflows* de sequência de *Map*, conforme demonstrado na Figura 34. Um comportamento diferenciado aparece nos casos em que os custos das atividades são baixos (*CA* próximo a 5) com relação à estratégia de despacho. Nestas situações, as execuções estáticas superam as dinâmicas.

6.2 Avaliação do Chiron frente às demais abordagens

Esta seção apresenta uma comparação do Chiron frente a três diferentes abordagens para execução paralela de *workflows* científicos em ambiente de PAD: motor de execução sequencial com *MapReduce* (*MapReduce*), motor de execução sequencial com camada especializada (Hydra) e motor de execução paralela (MEP). Na comparação apresentada nesta seção, não se revela a identidade do SGWfC utilizado, uma vez que o objetivo é comparar a capacidade da proposta algébrica implementada no Chiron de processar *workflows* com as abordagens distintas, sem se prender a um sistema em particular.

Visando a uma comparação das abordagens baseada na perspectiva de consumo e produção de dados das atividades (Ogasawara et al. 2011) e na distribuição de fluxo de dados (Bharathi et al. 2008), nesta seção apresentam-se cinco *workflows* característicos. O *workflow* de sequenciamento (PL) (Figura 44.a) contém uma atividade que produz uma única saída de dados para cada dado de entrada consumido, isto é, a proporção é 1:1. No *workflow* de fragmentação (DF) (Figura 44.c), uma atividade produz um conjunto de dados de saída para cada entrada de dados, ou seja, a relação é de 1:n. Finalmente, o *workflow* de agregação (DA) (Figura 44.e) representa uma atividade que produz uma única saída de dados a partir de um conjunto de dados de entrada, ou seja, a proporção é de n:1.

Além disso, considerando-se as formas de distribuição de dados que ocorrem no fluxo de dados entre as atividades, apresentam-se mais dois *workflows* característicos: *workflow* de distribuição (DB) e *workflow* de junção (DJ). No *workflow* DB, os dados de saída de uma mesma atividade são usados como entrada para *n* atividades. Por exemplo, na Figura 44.b, a atividade C produz uma saída de dados que são copiados para diferentes tipos de atividades (D e E). No padrão DJ, cada entrada de dados vem de *n* tipos diferentes de atividades. Estes dados são unidos segundo algum critério para

produzir um único conjunto de dados de saída. Como ilustrado na Figura 44.e, dados de saída das atividades de H e I são unidos pela atividade J, que produz uma saída de dados única. Os *workflows* DB e DJ requerem controle e critérios para sua movimentação de dados. No *workflow* DB, as *n* seguintes atividades recebem uma cópia dos dados produzidos por C. Desta forma, estas atividades podem executar concorrentemente. O *workflow* DJ exige sincronismo, ou seja, a atividade só é executada quando todas as suas relações de entrada estiverem prontas para serem consumidas.

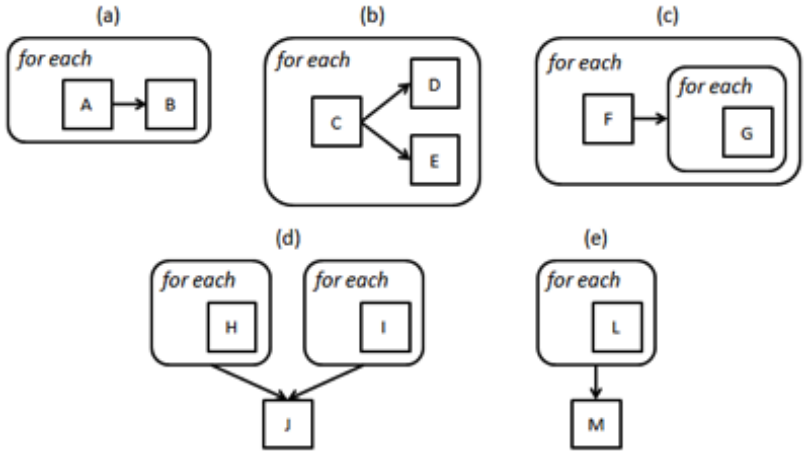

Figura 44 – *Workflow* de sequenciamento (a); *workflow* de fragmentação de dados (b); *workflow* de agregação (c); *workflow* de distribuição (d); *workflow* de junção (e)

Além da questão da relação consumo/produção entre dados, os *workflows* apresentados têm uma perspectiva de controle que rege como estes dados são transferidos entre as atividades. Estes controles apresentam uma correspondência aos padrões de *workflow* (Aalst et al. 2003). A Tabela 6 apresenta um resumo dos *workflows*, razão de consumo/produto de dados e sua correspondência com os padrões de *workflow*.

Tabela 6 – Detalhamento dos *workflows* utilizados para comparação

Workflow	**Razão de consumo/produto**	**Padrão de *Workflow*** (Aalst et al. 2003)
Workflow de sequência	1:1	WCP-1
Workflow de fragmentação	1:n	WCP-42
Workflow de agregação	n:1	WCP-41
Workflow de distribuição	1:n	WCP-2
Workflow de junção	n:1	WCP-3

Nesta validação experimental, comparam-se as quatro abordagens para PAD: *MapReduce*, Hydra, MEP e Chiron. Todos os *workflows* foram modelados usando-se os recursos fornecidos por cada abordagem. A descrição detalhada de como cada uma das atividades deve ser invocada, bem como a obtenção dos dados de entrada para o *workflow* podem ser obtidos no portal do SWB (2011). Na abordagem *MapReduce*, o

workflow foi modelado usando-se o *GUI* de um SGWfC e a programação das funções *Map* e *Reduce* foram feitas usando-se uma *API* bem conhecida de uma implementação para *MapReduce*. A execução de *workflow* é controlada por SGWfC, que coordena a transferência de dados e a invocação das funções *Map* e *Reduce*. Na abordagem Hydra, os *workflows* são modelados por um SGWfC e as atividades têm a sua execução em paralelo usando a camada Hydra, que é capaz de distribuir uma execução de uma atividade do *workflow* em um ambiente PAD. O MEP é um SGWfC bem conhecido que tem um motor de execução paralela e que é comumente usado em *clusters* e grades. Nesta abordagem, os *workflows* são modelados usando-se uma linguagem de programação específica, e as atividades são encapsuladas como uma aplicação invocada pelo MEP, que lhes permite executar *swb.jar* com dados de entrada. O conjunto de dados de entrada é armazenado em um arquivo como uma matriz, formato que é exigido pela sintaxe do MEP. No Chiron, os *workflows* são representados de modo declarativo (com base em um arquivo XML), convertendo-se em expressões algébricas, que são executadas em paralelo no ambiente de PAD.

Para se variar as propriedades dos *workflows* e a duração das atividades, foram definidos fatores globais denominados por: *ISF*, *DSF* e *ACF*. O *ISF* é o fator global responsável por definir o número de vezes que o *workflow* é executado usando dados de entrada diferentes, ou seja, a cardinalidade do conjunto de dados de entrada. A cardinalidade é definida como 2^{DSF+8}, para a $ISF \geq 1$. *DSF* é um inteiro que define o tamanho do arquivo consumido/produzido pelas atividades do *workflow*. Quando $DSF = 0$, então as atividades não têm uma entrada de arquivo de dados para consumir ou produzir. Caso contrário, quando $DSF \geq 1$, então o tamanho do arquivo é definido como 2^{DSF+8} KB. Finalmente, *ACF* é um valor inteiro positivo que caracteriza a complexidade das atividades. Basicamente, esta variável está relacionada ao tempo de execução (em segundos) das atividades. Este tempo de execução segue uma distribuição *Gama*, $\Gamma(\kappa, \theta)$, na qual $\kappa = 2^{2 \cdot ACF}$ e $\theta = 1$, para $ACF \geq 1$. Durante os experimentos utilizaram-se três possíveis combinações de valores iniciais *ACF*, *ISF* e *DSF* para comparar as quatro abordagens usando os cinco *workflows* adotados para comparação (*PL*, *DB*, *DF*, *DJ* e *DA*).

O *cluster* utilizado para realizar as medições possui 16 nós, contendo 8 núcleos cada nó. O *cluster* tem o sistema de filas (*PBS*) para controlar as submissões de tarefas. Devido à política do *PBS*, o *cluster* utilizado não permite conexão *secure Shell* (*SSH*) entre os nós. Como *SSH* é exigida pela implementação utilizada para *MapReduce*, não foi possível usar a abordagem de *MapReduce* em múltiplos nós. Além disso, a utilização do MEP associado ao *PBS* iria degradar o desempenho do MEP, o que levaria a uma comparação injusta entre as abordagens. Tanto o Hydra quanto o Chiron são projetados para trabalhar em *clusters* e levariam vantagem frente às demais abordagens no uso dos múltiplos nós. Assim, os experimentos foram realizados em um único nó com 8 núcleos visando a possibilitar uma comparação justa entre as abordagens.

A Figura 45 apresenta o resultado comparativo (expressados em minutos) das abordagens para a execução do *workflow* de sequência usando as diferentes combinações de *ACF*, *ISF* e *DSF*. A abordagem do *MapReduce* apresentou um desempenho bem inferior ao das demais abordagens. Observa-se um resultado muito semelhante ao do Chiron em relação ao MEP. O Hydra, apesar de próximo, teve um desempenho inferior quando comparado ao Chiron e MEP. Quando o *ACF* aumenta, a degradação do *MapReduce* diminui quando comparada à das demais abordagens. Isto está de acordo com a documentação do *MapReduce*, que diz não ser ela adequada para a execução de tarefas de curta duração.

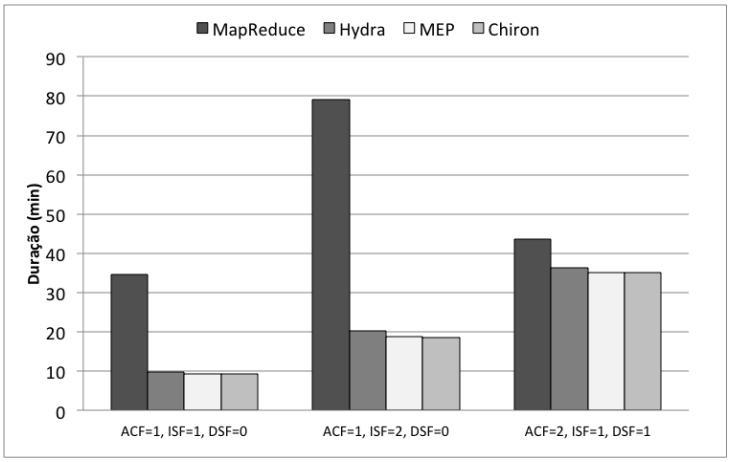

Figura 45 – Avaliação do *workflow* de sequência em diferentes combinações *ACF*, *ISF* e *DSF*

A Figura 46 apresenta o resultado comparativo das abordagens para *workflows* de fragmentação. Neste caso, a abordagem de *MapReduce* apresentou um desempenho menos distante em relação ao das demais abordagens. Neste cenário, a diferença de desempenho entre MEP e Hydra ficou mais clara. O MEP foi ligeiramente mais eficiente que o Chiron.

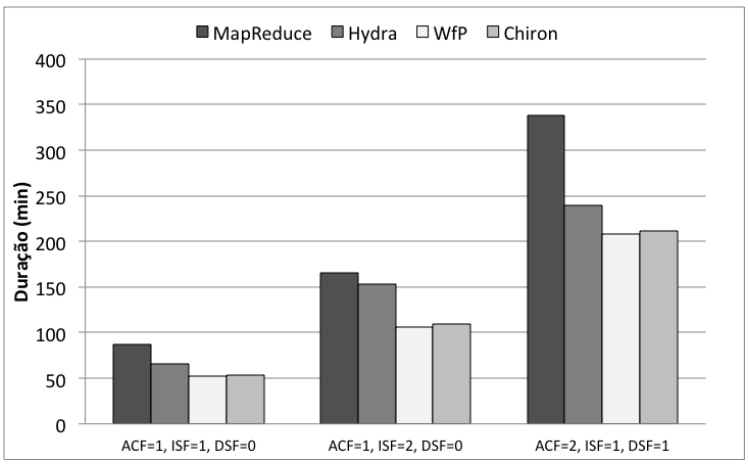

Figura 46 – Avaliação do *workflow* de fragmentação em diferentes combinações ACF, ISF e DSF

Visando a uma comparação mais geral entre as abordagens, a Figura 47 apresenta a média de *speedup* (Lawrence A. Crowl 1994) de cada abordagem

considerando-se os cinco *workflows* representativos. Note que o MEP e o Chiron apresentaram o melhor desempenho, sendo que o MEP foi ligeiramente melhor. Ademais, conforme o ACF aumenta, todas as abordagens melhoram o seu desempenho. Observa-se que o Hydra passa a se aproximar dos líderes e o *MapReduce* também melhora.

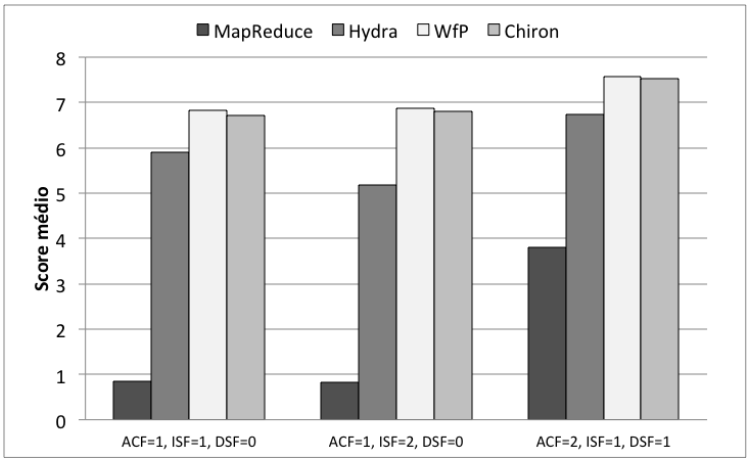

Figura 47 - Avaliação geral de desempenho em diferentes combinações ACF, ISF e DSF

Pode-se observar a partir dos resultados dos experimentos que a abordagem de *MapReduce* apresenta um desempenho bem abaixo ao das demais abordagens. Parte da justificativa se deve a sua sobrecarga para iniciar uma atividade. Se a atividade apresenta uma duração maior, a sobrecarga de execução de cada atividade tende a diminuir. O desempenho do Hydra também fica um degrau abaixo quando comparado ao MEP e ao Chiron. Particularmente este desempenho inferior está relacionado ao fato de a paralelização das atividades ser do tipo atividade por atividade, se assemelhando muito a um comportamento do tipo *FAF* apresentado no modelo de execução algébrico. O MEP apresentou, em linhas gerais, o melhor desempenho, mas carece de recursos oferecidos pelo Chiron, como uma base de proveniência *online* que possibilita ao usuário consultar o andamento de cada atividade, seu início, resultados produzidos e monitoramento de erros.

O aspecto positivo desta análise é que o Chiron teve um desempenho bem satisfatório, uma vez que a abordagem algébrica foi capaz de alcançar minimamente desempenhos semelhantes ao do MEP e com todo o beneficio de seu modelo de proveniência retrospectiva. Ademais, embora considerando que uma comparação de tempo de desenvolvimento de *workflows* exigiria um estudo rigoroso usando técnicas de Engenharia de Software Experimental (Juristo e Moreno 2001), o tempo para se desenvolver os *workflows* no MEP e no *MapReduce* foi de duas a três ordens de grandeza acima do tempo para desenvolvimento dos *workflows* no Chiron, o que mostra que o aspecto declarativo da especificação do Chiron traz vantagens para a elaboração dos *workflows* frente à imperatividade da especificação dos *workflows* no MEP e no *MapReduce*.

6.3 Avaliação das heurísticas de otimização

Para avaliar a influência das heurísticas de otimização baseadas em transformações algébricas, esta seção faz uso de um conjunto de quatro *workflows*, apresentados na Figura 48, que podem sofrer transformações algébricas.

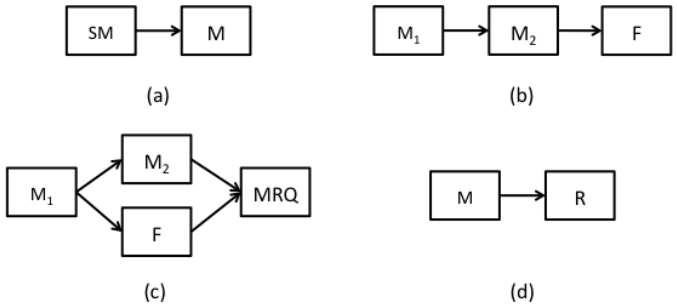

Figura 48 – *Workflows* utilizados para avaliação de transformações algébricas

No primeiro cenário, Figura 48.a, o *workflow* é expresso como: $S \leftarrow SplitMap(SM,\ R)$ e $T \leftarrow Map(M,\ S)$. O processo de otimização aplica as seguintes transformações algébricas: $T_p \leftarrow \overline{Map}(M,\ R)$, $S \leftarrow SplitMap(SM,\ T_p * R)$ e $T \leftarrow T_p * S$, fornecendo um *workflow* otimizado, no qual o *Map* é executado antes do *SplitMap*.

A Figura 49 apresenta a diferença de desempenho obtida ao se realizarem as transformações algébricas. A versão otimizada traz vantagens quando o fator de seletividade de número de tuplas de fragmentação é maior que 1. Este experimento foi executado em um nó com 8 núcleos, com número de tuplas inicial igual a 512 e custo médio da execução da atividade igual a 4 segundos.

No segundo cenário, Figura 48.b, o *workflow* é expresso como: $S \leftarrow Map(M_1,\ R)$, $T \leftarrow Map(M_2,\ S)$ e $U \leftarrow Filter(F,\ T)$. Este *workflow* possui duas possibilidades de otimização. Na primeira (Otimizado 1), o processo de otimização aplica as transformações algébricas: $S \leftarrow Map(M_1,\ R)$, $T_p \leftarrow \overline{Filter}(F,\ S)$, $T \leftarrow Map(M_2,\ S * T_p)$ e $U \leftarrow T_p * T$, fornecendo um *workflow* otimizado, no qual as atividades são executadas na seguinte ordem: *Map* M_1, *Filter* e *Map* M_2. Na segunda (Otimizado 2), o processo de otimização aplica as transformações algébricas: $T_p \leftarrow \overline{Filter}(F,\ R)$, $S \leftarrow Map(M_1,\ R * T_p)$, $T \leftarrow Map(M_2,\ S)$ e $U \leftarrow T_p * T$, fornecendo um *workflow* otimizado, no qual as atividades são executadas na seguinte ordem: *Filter*, *Map* M_1 e *Map* M_2.

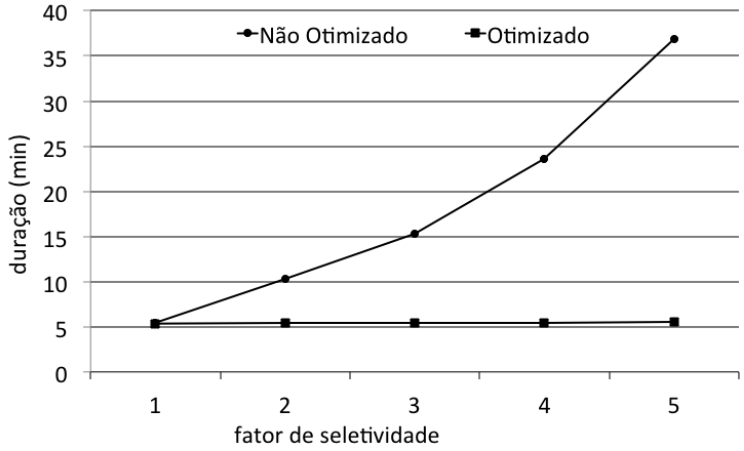

Figura 49 – Avaliação do fator de seletividade na otimização do *workflow* da Figura 48.a

A Figura 50 apresenta a diferença de desempenho obtida ao se realizarem as transformações algébricas. A versão otimizada traz mais vantagens quando o fator de seletividade decresce e deixa de ser vantajoso quando fica próximo a 1. Os benefícios da antecipação da operação *Filter* aumentam quando se consegue trazê-lo para o começo do *workflow* (otimizado 2). Entretanto, mesmo quando se deixa o *Filter* no meio do *workflow* (otimizado 1) os benefícios são significativos. Este experimento foi executado em um nó com 64 núcleos, com número de tuplas inicial igual a 512 e custo médio da execução da atividade igual a 32 segundos.

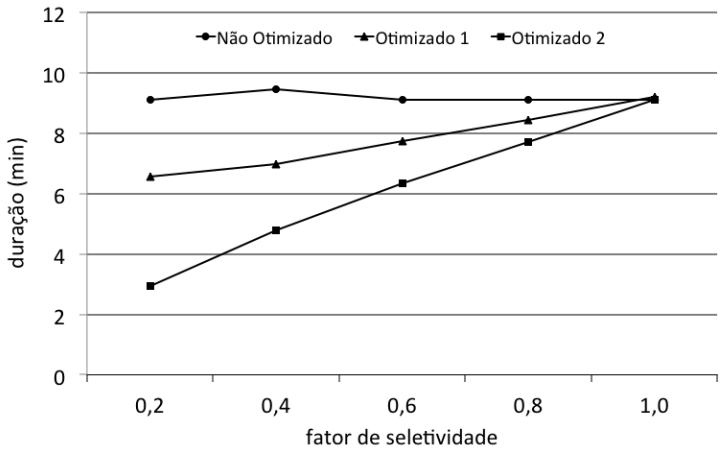

Figura 50 – Avaliação do fator de seletividade na otimização do *workflow* da Figura 48.b

No terceiro cenário, Figura 48.c, o *workflow* é expresso como: $S \leftarrow Map(M_1, R)$, $T \leftarrow Map(M_2, S)$, $U \leftarrow Filter(F, S)$ e $V \leftarrow MRQuery(U * T, \{U, T\})$. O processo de otimização aplica as seguintes transformações algébricas: $S \leftarrow Map(M_1, R)$, $U \leftarrow Filter(F, S)$, $T \leftarrow Map(M_2, S * U)$ e $V \leftarrow MRQuery(U * T, \{U, T\})$, fornecendo um *workflow* otimizado no qual o *Map* M_2 é procrastinado para depois do *Filter F*.

A Figura 51 apresenta a diferença de desempenho obtida ao se realizarem as transformações algébricas. A versão otimizada traz mais vantagens quando o fator de

seletividade decresce e deixa de ser vantajosa quando fica próximo a 1. Este experimento foi executado em um nó com 8 núcleos, com número de tuplas inicial igual a 1024 e custo médio da execução da atividade igual a 4 segundos.

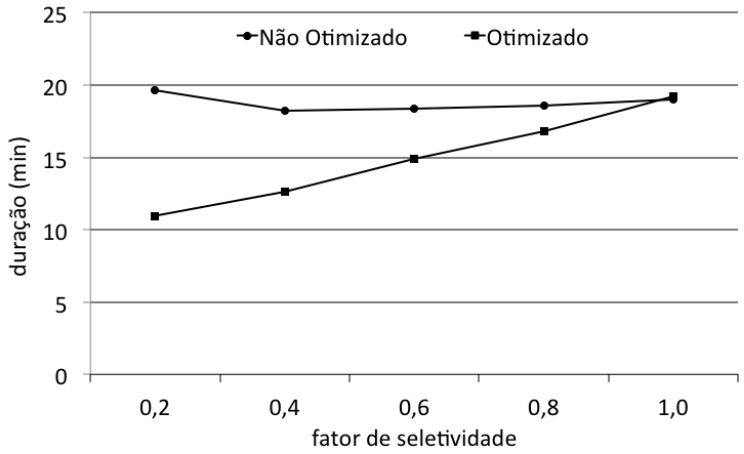

Figura 51 – Avaliação do fator de seletividade na otimização do *workflow* da Figura 48.c

No quarto cenário, Figura 48.d, o *workflow* é expresso como: $S \leftarrow Map(M, R)$ e $T \leftarrow Reduce(M, S)$. O processo de otimização aplica as seguintes transformações algébricas: $T_p \leftarrow \overline{Reduce}(M, R)$, $S \leftarrow Map(M, T_p * R)$ e $T \leftarrow T_p * S$, fornecendo um *workflow* otimizado, no qual o *Reduce* é executado antes do *Map*.

A Figura 52 apresenta a diferença de desempenho obtida ao se realizarem as transformações algébricas. A versão otimizada traz vantagens quando o fator de seletividade relacionado à agregação é baixo, próximo a zero. Este experimento foi executado em um nó com 64 núcleos, com número de tuplas inicial igual a 16.384 e custo médio da execução da atividade igual a 4 segundos.

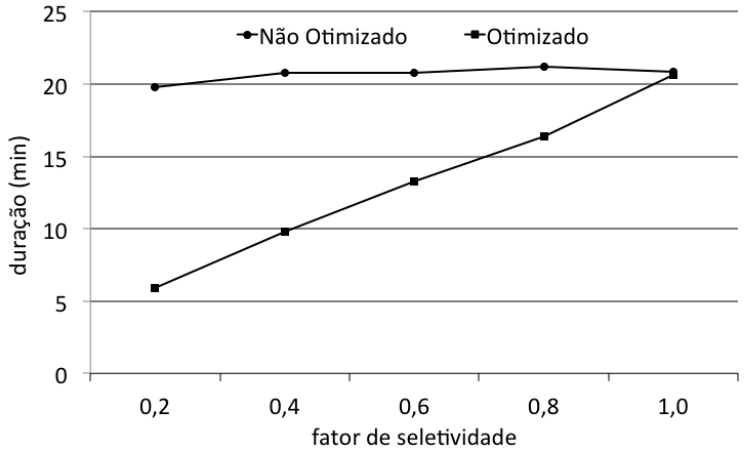

Figura 52 – Avaliação do fator de seletividade na otimização do *workflow* da Figura 48.d

Avaliando-se as otimizações algébricas, observa-se que os benefícios são significativos sempre que é possível aplicá-las. Quanto mais baixo é o fator de seletividade das atividades que podem reduzir o número de tuplas, maior o benefício.

Da mesma forma, sempre que é possível procrastinar as atividades que aumentam o número de tuplas, os benefícios podem ser significativos.

O processo de algoritmo de otimização possui uma complexidade quadrática $O(n^2)$, no qual *n* é o número de atividades do *workflow*. Imagine-se um fragmento de *workflow* a ser otimizado do qual se quer avaliar o impacto do algoritmo de otimização. Para tanto, é realizada uma avaliação do tempo de otimização de acordo com o aumento do número de atividades presentes em um fragmento, conforme observado na Figura 53. Pode-se observar a curva parabólica conforme se aumenta o número de atividades. Entretanto, a duração da execução do algoritmo de otimização com até 50 atividades em um único fragmento não consumiu tempo significativo.

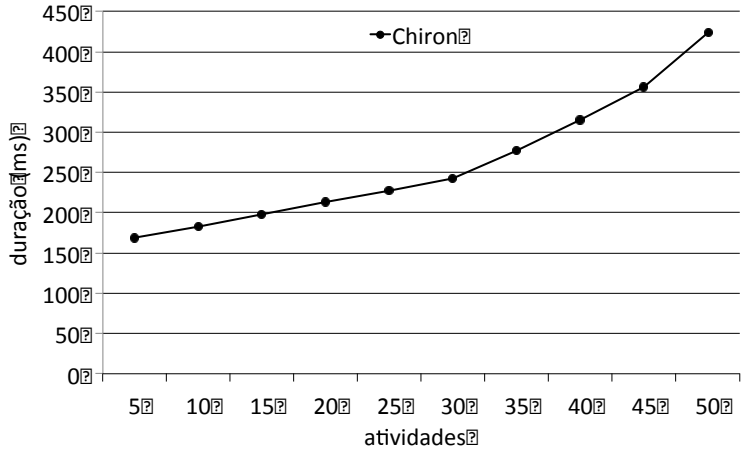

Figura 53 – Tempo necessário para otimizar um fragmento de workflow

6.4 *Workflow* de análise de fadiga de *risers*

O último estudo apresentado é referente a um *workflow* real. O *workflow* de RFA é apresentado em detalhes na Figura 1 e resumido na Figura 55.a. A atividade *ExtractRD* é regida por *SplitMap*. As atividades *PSRiser*, *SRiser*, *PDRiser*, *DRiser* são regidas por *Map*. As atividades *TAnalysis* e *CAnalysis* são regidas por *Filter*. A atividade *Match* é regida por um *MRQuery*. A atividade *CompressRD* é regida por um *Reduce*. Na Figura 55.a, as atividades marcadas apresentam preservação de chaves e compõem um fragmento para otimização algébrica. O algoritmo de otimização identifica que a atividade *CAnalysis* pode ser antecipada para ser executada imediatamente após a atividade *SRiser*. Finalmente, a atividade *PDRiser* pode ser procrastinada para ser executada posteriormente à atividade *CAnalysis*. O *workflow* equivalente otimizado é apresentado na Figura 55.b.

O *workflow* foi executado usando 358 estudos de casos de um único *riser* processados em um nó com 8 núcleos. A Figura 55 apresenta a melhoria de desempenho do *workflow* de RFA em função da otimização algébrica. De acordo com o critério de

aceitabilidade da análise de curvatura definido na atividade *CAnalysis*, os estudos de caso são filtrados. O aumento da filtragem diminui o número de execuções de pré-processamento de análise dinâmica (*PDRiser*) e análise dinâmica (*DRiser*). Se o percentual de filtragem for igual a 0, todas as tuplas da relação de entrada são passadas adiante e a otimização algébrica não tem vantagem prática. Conforme o percentual de filtragem aumenta, a duração da execução do *workflow* diminui. Uma filtragem de 20% oferece uma melhoria significativa de 6% na duração do *workflow* como um todo. Cabe salientar que a filtragem diminui a necessidade de execução de estudos de casos para as atividades *PDRiser* e *DRiser*. Entretanto, se houvesse mais atividades regidas por *Map* posteriores às atividades *PDRiser* e *DRiser*, o beneficio da antecipação da filtragem seria propagado e o desempenho melhoraria ainda mais.

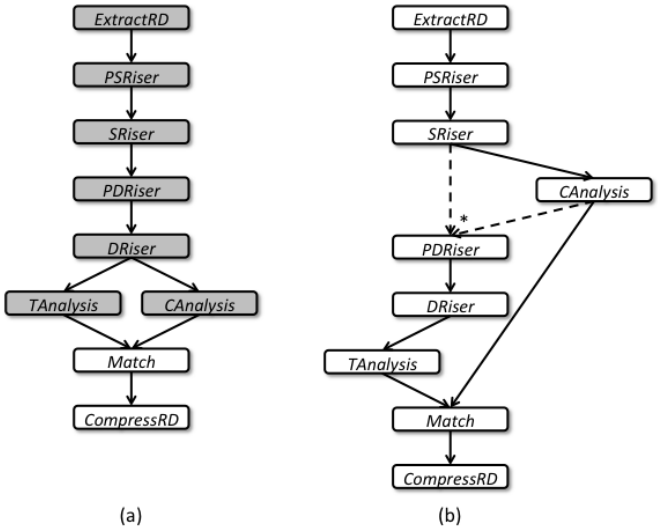

Figura 54 - Workflow RFA (a); Workflow RFA otimizado (b)

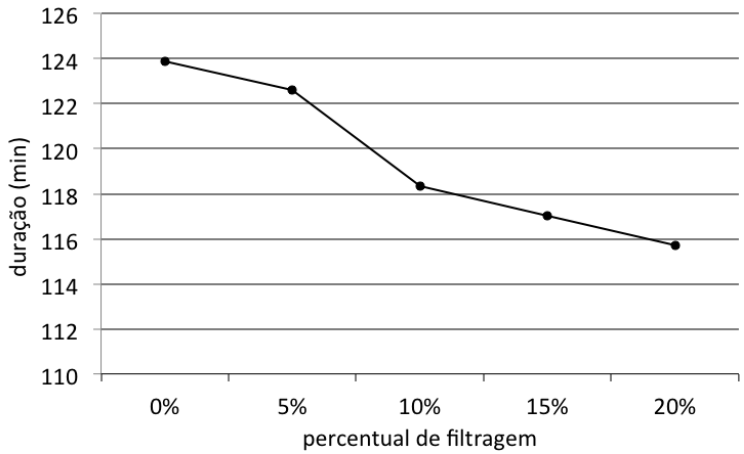

Figura 55 – Análise do workflow RFA variando-se o percentual de filtragem

6.5 Considerações sobre análise experimental

Os experimentos realizados foram divididos em quatro avaliações. A primeira apresenta uma avaliação do algoritmo de atribuição de estratégias de despacho e de fluxo de dados. Os experimentos mostram que não existe uma única estratégia isolada capaz de apresentar sempre o melhor desempenho, conforme esperado. Entretanto, foi possível confirmar que as heurísticas utilizadas pelo algoritmo de atribuição de estratégias foram capazes de produzir o melhor desempenho, quando comparadas ao uso de uma única abordagem isolada.

A segunda avaliação apresenta uma comparação do desempenho do Chiron frente às demais abordagens. Os experimentos utilizaram alguns padrões de *workflows* variando-se a duração média das atividades e o número de tuplas nas relações de entrada. O Chiron teve um desempenho bem satisfatório, uma vez que foi superior às abordagens baseadas em *MapReduce* e Hydra; além disso, teve resultados de desempenho semelhantes ao de um MEP. Esta pequena diferença de desempenho é compensada pelo melhor mecanismo de proveniência do Chiron, que viabiliza *workflow steering* e monitoramento dinâmico.

A terceira avaliação apresenta a influência do algoritmo de otimização por meio da comparação de *workflows* com ou sem otimização algébrica. O algoritmo de otimização foi capaz de produzir melhoras de desempenho significativas nos diferentes cenários. Ademais, a introdução do algoritmo de otimização não gerou uma sobrecarga no tempo da execução paralela do *workflow*.

Finalmente, a quarta avaliação usa o *workflow* real de análise de fadiga de *risers*. Neste caso real, pode-se perceber uma melhoria de desempenho ao se aumentar o percentual de filtragem. Foi possível observar que a otimização não trouxe nenhuma sobrecarga ao desempenho.

Todos os experimentos permitiram avaliar que a abordagem algébrica viabilizou a otimização da execução de modo transparente e foi capaz de apresentar uma melhora de até 278%. Estes resultados foram alcançados com a vantagem adicional de se ter uma proveniência retrospectiva possibilitando consultas à base de dados de proveniência durante a execução dos experimentos.

Capítulo 7 - Trabalhos relacionados

Existem poucas abordagens algébricas para *workflows* científicos, principalmente no que tange àquelas que apoiam a execução paralela das atividades. Entretanto, há diversas abordagens cujo objeto é a otimização da execução paralela de atividades em ambientes de larga escala. Para avaliar os diferentes trabalhos relacionados à otimização de execução paralela de *workflows* científicos, foi feita uma busca sistemática, datada de 22 de outubro de 2011, nas bases de dados do *Scopus* e *IEEE Xplore*, a partir das palavras-chave: "*workflow*" $\wedge$ "*optimization*" $\wedge$ ("*parallel execution*" $\vee$ "*distributed execution*").

A busca trouxe 446 artigos, sendo 206 do *Scopus* e 240 do *IEEE Xplore*. A partir da avaliação dos resumos dos 446 artigos, 57 foram selecionados para leitura completa por apresentarem propostas de execução paralela de *workflows*. Em paralelo a esta busca sistemática na área de *workflows* científicos foi feita uma pesquisa *ad hoc* visando às iniciativas algébricas ou declarativas na área de processos, que podem estar relacionadas a esta tese.

Alguns artigos apresentados possuem uma ampla variedade de temas, como apoio à varredura de parâmetros, apoio a coleções, escalonamento de atividades, adaptabilidade na execução e apoio a abstrações na representação (Bharathi et al. 2008, Callaghan et al. 2011, Deelman et al. 2009, Gil et al. 2007a, González-Vélez e Leyton 2010, Ludäscher et al. 2009). Assim, a partir dos artigos avaliados, pode-se classificar as abordagens que visam à otimização da execução de *workflows* em ambientes de PAD em: (i) abordagens orientadas a varredura de parâmetros (seção 7.1), (ii) abordagens orientadas a *MapReduce* (seção 7.2), (iii) abordagens orientadas a coleções aninhadas (seção 7.3), (iv) abordagens de escalonamento baseado em fluxo (seção 7.4), (v) abordagens de escalonamento baseado em recursos (seção 7.5), (vi) abordagens de escalonamento adaptativas (seção 7.6), (vii) abordagens declarativas (seção 7.7) e (viii) abordagens algébricas (seção 7.8). Finalmente, na seção 7.9 são apresentadas as considerações finais acerca dos trabalhos relacionados, comparando-os com a abordagem proposta nesta tese.

7.1 Abordagens orientadas a varredura de parâmetros

Algumas abordagens que otimizam a execução paralela de *workflows* se concentram no apoio à varredura de parâmetros. Essa varredura é muito comum em *workflows* científicos e correspondem a casos naturais de paralelismo, uma vez que cada caso pode ser processado separadamente, de forma independente. Abramson et al. (2011, 2010) apresentam o Nimrod como uma ferramenta capaz de facilitar a varredura de parâmetros e escalonamento das atividades em ambiente de PAD. Atividades de um determinado

tipo são executadas concorrentemente para cada valor presente no conjunto de parâmetros especificado para a varredura.

O Wings para o Pegasus (Gil et al. 2007b) usa *workflows* parametrizados que substituem trechos do *workflow* parametrizado por atividades concretas correspondentes a cada caso de varredura de parâmetros que se deseja processar. Assim, o *workflow* submetido já tem o número de atividades igual ao número de diferentes valores que se deseja explorar. As atividades são ligadas por meio de fluxo de controle. Os dados produzidos são consumidos pelas atividades subsequentes de modo implícito. O escalonamento das atividades no Pegasus fica a cargo do Condor (Couvares et al. 2007).

As abordagens orientadas a varredura de parâmetros possuem a limitação de paralelizar a varredura apenas quando todo o *workflow* está envolvido nessa varredura. Entretanto, existe um conjunto de problemas que requer executar varredura de parâmetros em algumas atividades do *workflow*. Nestes casos, as abordagens orientadas à varredura de parâmetros precisam ser utilizadas em conjunto com outras abordagens, tornando a elaboração do *workflow* mais laboriosa e dificultando as possibilidades de exploração de otimização do *workflow* como um todo.

7.2 Abordagens orientadas a *MapReduce*

Existe uma categoria de aplicações que pode ser paralelizada por meio de abordagem *MapReduce*. Algumas soluções (Matsunaga et al. 2008, Wang et al. 2009) costumam integrar SGWfC com implementações de *MapReduce* visando à paralelização de dados. Para isso, são criadas atividades de mapeamento, ou atividades *Map*, que são passíveis de serem paralelizadas. Ao final, uma função de agregação, denominada *Reduce*, é usada para permitir a integração dos resultados obtidos. Ambas as funções de mapeamento e de agregação devem ser definidas (programadas) pelos cientistas, pois elas dependem da natureza das aplicações que estão sendo paralelizadas. Porém, a distribuição de dados e a execução paralela correspondente no ambiente de PAD são dependentes do código e da implementação do *MapReduce* utilizada.

Alguns problemas, entretanto, não se encaixam diretamente como problemas modelados por *MapReduce*. As implementações de *MapReduce* fazem uma distribuição estática, baseada em chaves, dos dados. Esta distribuição pode retardar o tempo de execução de alguns programas científicos (Silva et al. 2011b). Além disto, as implementações de *MapReduce* retardam o disparo de tarefas, pois consomem um período de tempo para gerar as tarefas do tipo *Map* antes de iniciar a execução. Nas tarefas de curta duração (menos de um minuto), esta situação gera uma sobrecarga significativa na duração do *workflow*.

7.3 Abordagens orientadas a coleções aninhadas

O COMAD (McPhillips e Bowers 2005, McPhillips et al. 2009) é uma abordagem de modelagem e projeto orientado a coleções projetada para apoiar *workflows* que tenham em seu fluxo uma estrutura de coleções aninhadas. Neste tipo de estrutura, encontram-se, por exemplo, os dados semiestruturados presentes em um arquivo de XML. A partir de *tokens* de delimitação, desmembra-se o dado em fragmentos, como em uma linha de montagem. As atividades consomem fragmentos específicos, transformando-os. Os dados transformados substituem os fragmentos consumidos. Desta forma, as atividades têm um escopo sobre os dados semiestruturados. Esta noção de escopo possibilita a execução concorrente das atividades sobre estes dados. O grau de paralelismo é função do número de coleções aninhadas. Segundo McPhillips et al. (2009), o COMAD não é capaz de expressar alguns tipos de *workflows*, exigindo a ligação do COMAD a outros sistemas, como o Kepler. Esta integração com o Kepler produz uma quantidade grande de transferência de dados que não ocorre nos sistemas tradicionais.

O DFL designer (Sroka et al. 2010) é um outro trabalho que prioriza o processamento de coleções aninhadas. O DFL combina redes de Petri com cálculo relacional aninhado (NRC, do inglês *nested relational calculus*), produzindo uma linguagem para *workflow* científico baseada em coleções (COSW, do inglês *collection oriented scientific workflow*) que possibilita especificar manipulação de dados e aspectos de fluxo de controle conjuntamente. A partir do NRC se obtém um conjunto de operações e tipos de dados. Os aspectos de controle, como sincronismo, por exemplo, são apoiados pelas redes de Petri. A linguagem permite consumir dados em diferentes níveis hierárquicos.

As abordagens baseadas em coleções aninhadas exigem que os dados do *workflow* sejam organizados hierarquicamente. Segundo McPhillips et al. (2009), esta hierarquização não é natural em muitos problemas científicos, o que exige a mistura de modelos. Esta mistura pode produzir uma quantidade grande de transferência de dados que não ocorre nas abordagens tradicionais.

7.4 Abordagens de escalonamento baseado em fluxo

Existe um conjunto de abordagens que avalia a estrutura do fluxo de dados do *workflow* visando a definir estratégias para escalonamento da execução das atividades. Lemos e Casanova (2007, 2006) avaliam os *pipelines*, que são trechos do *workflow* cujo dado produzido por uma atividade é consumido por outra atividade subsequente no fluxo. O trabalho identifica os *pipelines* existentes nos *workflows* e apresenta uma estratégia gulosa para escalonar as atividades destes trechos. Eles observaram que a priorização de *pipeline* é adequada para alguns problemas, uma vez que possibilita ao cientista avaliar dados parciais já produzidos.

A interação de diferentes instâncias de *workflows* pode gerar problemas como conflito de nome de arquivos entre as instâncias concorrentes, que podem sobrescrever arquivos comuns (Wang e Lu 2011). Para estes cenários, Wang e Lu propõem o WaFS, que é um escalonador que avalia os arquivos produzidos ao longo do *workflow* visando à identificação de *pipelines* nos quais a sobrescrita não ocorre. Desta forma, o escalonador continua procurando *pipelines*, mas agora com critérios mais restritivos, ou seja, somente os *pipelines* que não sobrescrevam arquivos é que são avaliados.

7.5 Abordagens de escalonamento baseado em recursos

Existem diversas abordagens que procuram escalonar as atividades do *workflow* de acordo com os recursos presentes (Bharathi e Chervenak 2009, Bittencourt e Madeira 2008, Critchlow e Chin 2011, Garg et al. 2010, Ranaldo e Zimeo 2009). Estas abordagens são muito presentes em ambientes computacionais não homogêneos, como as grades. Nelas, entre outras restrições, há máquinas com poder computacional diferenciado, há limitações das máquinas que podem executar determinados programas e há preocupação em evitar transferências de dados desnecessárias.

Nestes tipos de abordagem é muito comum usar uma função de custo para avaliar os possíveis recursos para se executar o *workflow*. Em alguns casos, estes tipos de técnicas estão associadas a abordagens adaptativas, uma vez que, frequentemente, o estado dos recursos varia durante a execução dos programas.

7.6 Abordagens de escalonamento adaptativas

As abordagens adaptativas para execução de *workflows* podem estar associadas a diferentes conceitos. Normalmente, a adaptabilidade está associada ao escalonamento do *workflow*, ou seja, ao mapeamento de quais recursos devem ser alocados a quais atividades do *workflow*. A adaptabilidade também pode estar associada à gestão do espaço de parâmetros do *workflow*. Nestes dois casos, a adaptabilidade está ligada ao ajustamento dos recursos e valores dos parâmetros ao longo da execução do *workflow*.

Lee et al. (2011) apresentam uma abordagem adaptativa para a execução de *workflows* através de escalonamento adaptativo utilizando funções de utilidade e decomposição funcional. Desta forma, eles transformam o problema do escalonamento em um problema de otimização e mostram os ganhos de desempenho principalmente em ambientes de execução distribuída com recursos compartilhados, nos quais as medições de desempenho são incertas e podem variar ao longo do tempo. A camada SciCumulus (Oliveira et al. 2011c) também possui escalonamento adaptativo em nuvens. Como a computação em nuvem possibilita a obtenção e descarte de recursos sob demanda, e tais recursos possuem desempenho mensurável, a abordagem possibilita a execução adaptativa incluindo ou excluindo máquinas da execução conforme a necessidade do usuário. Além disso, dadas as características da atividade do *workflow*, o

SciCumulus é capaz de mapear tarefas mais custosas para recursos mais capazes enquanto executa atividades mais simples em recursos mais limitados.

A abordagem adaptativa apresentada por Dias et al. (2011) explora fatias do espaço de parâmetros de um experimento para melhorar o restante da execução. São adicionados pontos de controle ao *workflow* que monitoram métricas como erros associados, número de iterações e critérios de filtragem, que podem ser modificados ao longo da execução. Desta forma, baseado em resultados obtidos até certo ponto da execução, os cientistas podem fazer ajustes no *workflow* para melhorar a obtenção dos resultados.

Finalmente, Kumar et al. (2009) introduzem um arcabouço que integra um conjunto de componentes para *workflow* sobre o Wings (Gil et al. 2007b) que possibilita a distribuição de execução de atividades visando a otimizar cenários de varredura de parâmetros. Para tanto, o conceito de meta-atividade foi elaborado visando a agrupar um conjunto de atividades. As meta-atividades podem produzir e consumir dados, possibilitando o encadeamento com outras atividades ou meta-atividades. O agrupamento das atividades em meta-atividades é especificado pelos cientistas. Diferentemente do caso tradicional do Pegasus, no qual o número de varreduras de parâmetros especifica o número de atividades instanciadas, as meta-atividades propostas por Kumar et al. (2009) são instanciadas por núcleos e cada uma delas pode executar vários casos da varredura de parâmetros.

7.7 Abordagens declarativas

As abordagens declarativas tentam eliminar o aspecto procedural presente nas demais abordagens existentes por meio de uma linguagem de alto nível. Comumente as linguagens se assemelham ao SQL visando a apoiar a especificação do encadeamento dos *workflows*.

A abordagem BioFlow (Jamil e El-Hajj-Diab 2008) possibilita especificar funções definidas pelo usuário (UDF, do inglês *user defined function*) e incorpora primitivas de chamada a funções (*call*). A BioFlow também possibilita a criação de processos, que chamam estas funções, e a especificação de um fluxo de processos. Entretanto, o fluxo do processo, apesar de declarativo, apresenta-se de modo rígido. Por ser restrito ao domínio de bioinformática, os dados são modelados por meio de tabelas e o processamento é orientado a tuplas que operam basicamente dados primitivos.

Embora não seja diretamente voltado para apoiar *workflows* científicos, o Pig Latin (Olston et al. 2008) é uma linguagem declarativa para apoiar o processamento paralelo de dados. É uma abstração sobre o *MapReduce* que usa um estilo semelhante ao do SQL para especificação do processamento de dados.

Na área de processos, existem abordagens como a de Vrhovnik et al. (2007) que realizam processamento de dados e usam o SQL como elemento prioritário na especificação deste processamento. O trabalho de Vrhovnik et al. (2007) baseia-se na extensão do PGM, que possibilita associar o processamento de dados aos processos, possibilitando ligar a otimização de processos à otimização de banco de dados. A abordagem exige a reescrita de todos os *workflows* como UDF em banco de dados.

7.8 Abordagens algébricas

Existem algumas abordagens, como as apresentadas por Cantão et al. (2010), que são baseadas em álgebra de processo e modelam o *workflow* por meio de expressões algébricas. A otimização ocorre na avaliação das regras e na produção adequada dos resultados, mas não necessariamente na otimização da execução do *workflow*.

Outra abordagem algébrica baseada em matrizes é apresentada por Montagnat et al. (2009), que especifica *workflows* por meio de links de dependência entre atividades. As atividades consomem e produzem matrizes. Além disto, a abordagem também inclui algumas operações vetoriais, como produto escalar, produto cartesiano, entre outros, para possibilitar combinações de matrizes. As atividades consomem cada linha das matrizes de entrada produzindo linhas nas matrizes de saída e são disparadas assim que os dados tornam-se disponíveis para consumo. O uso de matrizes traz uma uniformidade importante, pois possibilita a paralelização transparente da execução.

A QEF (Oliveira et al. 2010c, Porto et al. 2007) é uma abordagem algébrica para especificar *workflows* científicos de visualização na qual cada atividade é uma operação algébrica capaz de consumir e produzir tuplas. O QEF faz-se valer de um processador de consultas expandido para contemplar tanto operações algébricas relacionadas à semântica do problema a ser resolvido como também algumas operações relacionadas às operações de controle. Cada operação (atividade) consome e produz o conteúdo de uma tupla. O processamento varia de acordo com a semântica desejada. Operações de controle também existem para apoiar a troca de dados e atividades de bloqueio, divisão e mescla.

7.9 Considerações finais sobre trabalhos relacionados

Existe um grande número de trabalhos observados que, de alguma forma, procuram otimizar a execução de *workflows* em paralelo. Estes trabalhos podem ser classificados em categorias como abordagem de orientação a tipos de problemas apoiados (varredura de parâmetros, *MapReduce*, coleções aninhadas), formas de escalonamento (fluxo de dados, recursos) e de representação (declarativas e algébricas).

Uma característica comum às abordagens orientadas a tipos de problemas consiste na dificuldade que elas apresentam em generalizar demais casos de

paralelismo. As abordagens de varredura de parâmetros carecem de primitivas para executar decomposições e agregações, enquanto que as de *MapReduce* têm dificuldade para representar, naturalmente, fluxos com controles e são essencialmente especificadas de modo imperativo. As abordagens baseadas em coleções aninhadas (COMAD e DFL designer) carecem de semântica nas relações de consumo e produção. Esta falta de informação diminui as possibilidades de otimização da execução do *workflow*, uma vez que dificulta a seleção dos *tokens* que devem ser processados prioritariamente por reduzir o número de *tokens* para posterior processamento. Ademais, nem todos os *workflows* se apresentam de modo hierarquizado, o que limita a aplicabilidade.

As abordagens relacionadas às diferentes formas de escalonamento são importantes e, ao mesmo tempo, complementares à abordagem algébrica proposta nesta tese. Elas podem direcionar as funções de custos ou as estratégias de escalonamento das ativações. Por exemplo, no caso das abordagens que avaliam os fluxos, elas indicam a importância de se decompor o *workflow* em partes para se definir uma estratégia de execução. Esta decisão corrobora a solução apresentada nesta tese de particionar o *workflow* em fragmentos. Cabe ressaltar que se pode ter que diferenciar os critérios de fragmentação de acordo com o ambiente computacional adotado.

No contexto de escalonamento, as abordagens adaptativas apresentadas são também complementares à abordagem algébrica apresentada nesta tese, pois oferecem melhorias no escalonamento das atividades do *workflow* enquanto a álgebra busca uniformizar e otimizar o fluxo de dados modelado. As abordagens de escalonamento adaptativas estão na etapa seguinte à otimização algébrica, distribuindo e executando as atividades do *workflow* já otimizado algebricamente nos recursos computacionais distribuídos.

As abordagens declarativas estão mais focadas em processamento de dados e carecem de uma visão mais focada nos aspectos científicos. As abordagens declarativas são de fato declarativas na especificação das atividades, mas não na especificação dos fluxos. Os fluxos possuem comumente uma especificação rígida, limitando as possibilidades de otimização entre as atividades. Ou seja, aparentemente, o modelo de fluxo de dados destas abordagens é FAF e não deixa a possibilidade de se alterar a ordem da execução das atividades especificadas.

As abordagens algébricas relacionadas foram desenvolvidas para apoiar processamento de matrizes ou processos. Elas não trazem uma uniformidade de dados necessária para atender grande parte dos modelos científicos. No caso das abordagens matriciais, como as atividades consomem apenas uma linha da matriz por vez, falta expressividade para representar uma relação de consumo e produção diferenciada, ficando tais abordagens dependentes de operações mais limitadas, como as de produto escalar e vetorial.

A abordagem algébrica mais próxima à apresentada nesta tese é a do QEF (Oliveira et al. 2010c, Porto et al. 2007), que tem um foco centrado em dados, fazendo-se valer de tuplas como unidade mínima de consumo e produção pelas atividades. Entretanto, falta expressividade para representar uma relação de consumo e produção diferenciada, ou seja, enquanto o grão mínimo no caso do QEF é uma tupla, o grão mínimo na abordagem apresentada nesta tese é uma ativação, que pode consumir e produzir uma ou mais tuplas, sendo capaz de expressar cenários como *MapReduce* de modo natural. Cabe ressaltar que o QEF é capaz de expressar *workflows* que possuem ciclos, ou seja, que contenham grafos direcionados cíclicos, por meio da operação *orbit*. Esta tese não apoia este tipo de cenário, focando no atendimento aos problemas que podem ser expressados por meio de grafos direcionados acíclicos.

A partir da busca por trabalhos relacionados, pode-se observar a inexistência direta de abordagens algébricas para *workflows* científicos que propiciem uma uniformização dos dados consumidos e produzidos pelas atividades como relações. Além disto, diversas abordagens relacionadas, por não serem focadas em apoiar *workflows* científicos, deixam a proveniência a cargo do cientista. Em contrapartida, esta tese concebeu as operações algébricas, atividades e ativações visando a uma coleta de proveniência. Desta forma, a implementação do Chiron sobre o arcabouço algébrico apresentou uma proveniência que, inclusive, é capaz de apoiar monitoramento durante a execução do *workflow*.

Capítulo 8 - Conclusões

Esta tese apresenta uma abordagem algébrica para *workflows* científicos de larga escala. A abordagem, inspirada no já consolidado modelo relacional, apresenta uma álgebra para *workflows* científicos na qual os dados são uniformemente representados por meio de relações e as atividades de *workflows* científicos são regidas por operações que estabelecem uma semântica de consumo e produção destes dados. A abordagem, portanto, possibilita um tratamento homogêneo para os dados e as atividades dos *workflows*.

A representação das atividades que são regidas por operações algébricas serve de base para a definição do modelo de execução de *workflows*. Este modelo é apoiado pelo conceito de ativação de atividade, semelhante à ativação em banco de dados (Bouganim et al. 1996), que é um elemento fundamental para distribuição transparente das atividades em ambientes de PAD. No modelo de execução, os *workflows* podem ser divididos em fragmentos, ou seja, em sequências de atividades que possuem relação de dependência. Nestes fragmentos se especificam estratégias de fluxo de dados e de despacho que, juntas, estabelecem o escalonamento de ativações para execução paralela.

Dado um conjunto de ativações, a definição de estratégia de fluxo de dados e de despacho em cada fragmento, embora produza o mesmo resultado, leva a um desempenho diferente. Pode-se interpretar que cada combinação das estratégias de fluxo de dados e de despacho corresponde a uma forma de especificar *workflows* científicos usando as abordagens tradicionais (como ocorre no caso das linguagens imperativas de *script* ou implementação *MapReduce*). Esta especificação tradicional limita a possibilidade de otimização da execução, que fica toda a cargo do escalonador. Por meio de experimentos é possível avaliar estas diferentes combinações de estratégias e observar que não existe uma estratégia geral adequada para todos os cenários, mas é possível classificar os cenários nos quais uma estratégia é mais adequada que a outra.

Neste contexto, a abordagem algébrica, por ter um aspecto declarativo, possibilita a otimização da atribuição destas estratégias por meio de heurísticas, criadas a partir dos resultados experimentais. Nesta tese foi desenvolvido um algoritmo que, fazendo-se valer das heurísticas, otimiza a distribuição de estratégias de fluxo de dados e de despacho para *workflows*. O algoritmo proposto foi avaliado em diversos experimentos, produzindo os melhores resultados.

A abordagem também contempla um conjunto de transformações algébricas que, quando aplicáveis, possibilita trocar a ordem de execução das atividades no *workflow*. A mudança de ordem das atividades impacta nos números de tuplas das relações intermediárias. Estes números têm influência direta no desempenho da execução do *workflow*, uma vez que eles definem o número de ativações. Nesta tese foi desenvolvido

um algoritmo de otimização da execução do *workflow* inspirado no algoritmo de migração de predicado (Hellerstein e Stonebraker 1993). O algoritmo avalia as possíveis transformações buscando antecipar a execução de atividades que reduzem o número de tuplas e postergar as atividades que aumentem o número de tuplas de relações intermediárias. Os experimentos realizados mostram os benefícios que podem ser alcançados a partir das transformações algébricas.

Para que todos os experimentos fossem elaborados e avaliados, desenvolveu-se o Chiron, que é um motor de execução de *workflows* projetado para ambiente de PAD. O Chiron trabalha com o modelo algébrico para realizar a execução paralela de suas atividades. De acordo com o modelo algébrico implementado no Chiron, os dados de um *workflow* são representados como relações e todas as atividades do *workflow* são regidas por operações algébricas e desmembradas em ativações como elemento atômico para paralelização. O Chiron foi utilizado em diversos experimentos de *workflows* reais (Guerra et al. 2011, Ogasawara et al. 2011), mostrando-se um motor de execução capaz de apoiar a execução paralela de experimentos científicos. Neste contexto, a proveniência é importante para viabilizar a reprodutibilidade dos experimentos, que, no caso do Chiron, é armazenada no grão fino (via ativações) em um banco de dados relacional. Usando-se o Chiron é possível monitorar o status do experimento e avaliar os resultados disponíveis durante a execução. Este tipo de recurso é um diferencial, uma vez que a maioria das abordagens paralelas para execução de *workflows* científicos não apoia a coleta de proveniência em ambientes distribuídos, e aquelas que apoiam não viabilizam consultas durante a execução.

A tese apresenta contribuições no ciclo de vida do experimento científico nas suas diferentes fases (Mattoso et al. 2010, 2009): composição, execução e análise. Na etapa de composição foram feitos estudos buscando introduzir maior nível de abstração na especificação de *workflows* visando à geração de *workflows* prontos para execução. Estes estudos foram baseados em técnicas de reutilização de software (Ogasawara et al. 2008, 2009c, Oliveira et al. 2010b), algumas vezes apoiadas por ontologias (Oliveira et al. 2011b, 2009b) e por técnicas de versionamento (Costa et al. 2009, Ogasawara et al. 2009d, Silva et al. 2010b, 2011a). Nestes estudos foi observado que há uma impedância entre a especificação dos *workflows* e a sua preparação para execução nos ambientes de PAD, o que serviu como motivação para um estudo mais aprofundado da abordagem algébrica como elemento viabilizador da ligação entre a composição e a execução.

Na etapa de execução, que acabou sendo o foco da tese, inicialmente foram realizados estudos de execução paralela de tarefas de mineração de dados de modo *ad hoc* (Barbosa et al. 2009, Ogasawara et al. 2009a), que serviram de base para a elaboração de uma camada para apoio à varredura de parâmetros e paralelismo de dados (Coutinho et al. 2010, 2011, Guerra et al. 2009, Ogasawara et al. 2009b). Esta camada foi usada para avaliar diferentes tipos de experimentos (Ocaña et al. 2011a, 2011b) e

servir de base para estudos de diferentes técnicas de escalonamento (Dias et al. 2010a, 2010b, 2010c, Ogasawara et al. 2010). Estes estudos viabilizaram a proposta algébrica (Ogasawara et al. 2011), que já pôde ser avaliada em alguns cenários (Dias et al. 2011, Guerra et al. 2011).

Finalmente, na etapa de análise foram feitos estudos acerca dos desafios inerentes à coleta de dados de proveniência em ambientes heterogêneos e à necessidade de interligação de dados provenientes tanto da composição quanto da execução (Marinho et al. 2011, 2010a, 2010b). A partir destes estudos foi concebida a abordagem algébrica e desenvolvido o Chiron.

Resumidamente, a abordagem algébrica apresentada nesta tese apoia os experimentos científicos de larga escala por apresentar as seguintes características: (i) uma representação uniforme que traz semântica às atividades por meio das operações algébricas; (ii) um modelo de execução que possibilita a distribuição e paralelização de atividades de modo transparente; (iii) um conjunto de regras de transformações algébricas e algoritmos que visam à otimização do tempo de execução dos *workflows* científicos; (iv) um motor de execução de *workflows* (Chiron) com apoio à proveniência retrospectiva; e, finalmente, (v) uma avaliação extensiva da abordagem algébrica baseada nos experimentos realizados com o Chiron.

A partir desta tese, pode-se focar em duas frentes gerais para trabalhos futuros. A primeira frente é a disponibilização de ferramentas e modelos com maior nível de abstração que possibilitem trabalhar com *workflows* abstratos e que derivem *workflows* especificados para o Chiron. Neste tipo de cenário, conceitos como linhas de experimentos (Ogasawara et al. 2008) e ferramentas como GExpLine (Oliveira et al. 2010b) servem de insumo para pesquisas. Como segunda frente, pode-se realizar pesquisa em controle de paralelismo e *workflow steering* para aplicações científicas modeladas como problemas de otimização (Dias et al. 2011). Este tipo de aplicação produz um conjunto de dados que precisa ser reavaliado, gerando novas entradas para o *workflow*. Em outras palavras, este tipo de aplicação recai nos cenários de *workflows* modelados como grafos direcionados cíclicos. Neste tipo de cenário tem-se a necessidade de criar novas operações e novas técnicas de otimização para apoiar os ciclos nos grafos.

Referências bibliográficas

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

Aalst, W. van der, Hofstede, A., Kiepuszewski, B., Barros, A., (2003), "Workflow patterns", *Distributed and Parallel Databases*, v. 14, n. 1, p. 5-51.

Abramson, D., Bethwaite, B., Enticott, C., Garic, S., Peachey, T., (2009a), "Parameter Space Exploration Using Scientific Workflows", *Computational Science – ICCS 2009*, , chapter 5544, Springer Berlin / Heidelberg, p. 104-113.

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

Abramson, D., Bethwaite, B., Enticott, C., Garic, S., Peachey, T., Michailova, A., Amirriazi, S., (2010), "Embedding optimization in computational science workflows", *Journal of Computational Science*, v. 1, n. 1 (maio.), p. 41-47.

Abramson, D., Bethwaite, B., Enticott, C., Garic, S., Peachey, T., Michailova, A., Amirriazi, S., Chitters, R., (2009b), "Robust workflows for science and engineering". In: *2nd Workshop on Many-Task Computing on Grids and Supercomputers*, p. 1-9, Portland, Oregon, USA.

Abramson, D., Enticott, C., Altinas, I., (2008), "Nimrod/K: towards massively parallel dynamic grid workflows". In: *Proc. of International Conference for High Performance Computing, Networking, Storage and Analysis*, p. 1-11, Austin, Texas, USA.

Afrati, F. N., Ullman, J. D., (2010), "Optimizing joins in a map-reduce environment". In: *EDBT*, p. 99–110, New York, NY, USA.

Alt, M., Hoheisel, A., Pohl, H.-W., Gorlatch, S., (2006), "A Grid Workflow Language Using High-Level Petri Nets", In: Wyrzykowski, R., Dongarra, J., Meyer, N., Waśniewski, J. [orgs.] (eds), *Parallel Processing and Applied Mathematics*, Berlin, Heidelberg: Springer Berlin Heidelberg, p. 715-722.

Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., Mock, S., (2004), "Kepler: an extensible system for design and execution of scientific workflows". In: *Scientific and Statistical Database Management*, p. 423-424, Greece.

Asetti, I. Q., Correa, F. N., Jacob, B. P., (2003), "Hybrid and Coupled Analysis Methodologies for the Design of Floating Production Systems". In: *XXIV Congresso Ibero Latino-Americano sobre Métodos Computacionais para Engenharia*, p. 1-8, Ouro Preto, MG, Brazil.

Baeten, J. C. M., (2005), "A brief history of process algebra", *Theor. Comput. Sci.*, v. 335, n. 2-3, p. 131-146.

Baeten, J. C. M., Basten, T., Reniers, M. A., (2009), *Process Algebra: Equational Theories of Communicating Processes*. 1 ed. Cambridge University Press.

Barbosa, C. E., Ogasawara, E., Oliveira, D. de, Mattoso, M., (2009), "Paralelização de Tarefas de Mineração de Dados Utilizando Workflows Científicos". In: *V*

Workshop em Algoritmos e Aplicações de Mineração de Dados, p. 91-96, Fortaleza, Ceara, Brazil.

Barga, R. S., Fay, D., Guo, D., Newhouse, S., Simmhan, Y., Szalay, A., (2008), "Efficient scheduling of scientific workflows in a high performance computing cluster". In: *6th international workshop on Challenges of large applications in distributed environments*, p. 63-68, Boston, MA, USA.

Barga, R., Gannon, D., (2007), "Scientific versus Business Workflows", *Workflows for e-Science*, Springer, p. 9-16.

Barker, A., van Hemert, J., (2008), "Scientific Workflow: A Survey and Research Directions", *Parallel Processing and Applied Mathematics*, , p. 746-753.

Bayucan, A., Henderson, R. L., Jones, J. P., (2000), "Portable Batch System Administration Guide"

Bharathi, S., Chervenak, A., (2009), "Scheduling data-intensive workflows on storage constrained resources". In: *4th Workshop on Workflows in Support of Large-Scale Science*, p. 3:1–3:10, New York, NY, USA.

Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.-H., Vahi, K., (2008), "Characterization of scientific workflows". In: *Workflows in Support of Large-Scale Science*, p. 1 -10, Austin, Texas, USA.

Bittencourt, L. F., Madeira, E. R. M., (2008), "A performance-oriented adaptive scheduler for dependent tasks on grids", *Concurr. Comput. : Pract. Exper.*, v. 20, n. 9 (jun.), p. 1029–1049.

Bose, R., Frew, J., (2005), "Lineage retrieval for scientific data processing: a survey", *ACM Computing Surveys*, v. 37, n. 1, p. 1-28.

Bouganim, L., Florescu, D., Valduriez, P., (1996), "Dynamic load balancing in hierarchical parallel database systems". In: *Proc. of VLDB*, p. 436-447

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

Bowers, S., Mcphillips, T., Riddle, S., Anand, M. K., Ludäscher, B., (2008b), "Kepler/pPOD: Scientific Workflow and Provenance Support for Assembling the Tree of Life". In: *Second International Provenance and Annotation Workshop*, p. 70-77, Salt Lake City, UT, USA.

Braghetto, K. R., (2006), *Padrões de Fluxos de Processos em Banco de Dados Relacionais*, Universidade de São Paulo

Brandao, S. N., Silva, W. N., Silva, L. A. ., Fagundes, V., de Mello, C. E. ., Zimbrao, G., de Souza, J. M., (2009), "Analysis and visualization of the geographical distribution of atlantic forest bromeliads species". In: *IEEE Symposium on Computational Intelligence and Data Mining*, p. 375-380, Nashville, TN, USA.

Callaghan, S., Maechling, P., Small, P., Milner, K., Juve, G., Jordan, T. H., Deelman, E., Mehta, G., Vahi, K., Gunter, D., Beattie, K., Brooks, C., (2011), "Metrics for heterogeneous scientific workflows: A case study of an earthquake science application", *International Journal of High Performance Computing Applications*, v. 25, n. 3, p. 274 -285.

Callahan, S. P., Freire, J., Santos, E., Scheidegger, C. E., Silva, C. T., Vo, H. T., (2006), "VisTrails: visualization meets data management". In: *SIGMOD*, p. 745-747, Chicago, Illinois, USA.

Cantão, M. E., de Araújo, L. V., Lemos, E. G. M., Ferreira, J. E., (2010), "Algebraic approach to optimal clone selection applied in metagenomic projects". In: *International Symposium on Biocomputing*, p. 36:1–36:6, New York, NY, USA.

Carpenter, B., Getov, V., Judd, G., Skjellum, A., Fox, G., (2000), "MPJ: MPI-like message passing for Java", *Concurrency: Practice and Experience*, v. 12, n. 11, p. 1019-1038.

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

Casanova, M. A., Lemos, M., (2007), "Workflow Parallelization by Data Partition and Pipelining". *3rd VLDB Workshop on Data Management in Grids*, Viena.

Cavalcanti, M. C., Targino, R., Baião, F., Rössle, S. C., Bisch, P. M., Pires, P. F., Campos, M. L. M., Mattoso, M., (2005), "Managing structural genomic workflows using web services", *Data & Knowledge Engineering*, v. 53, n. 1, p. 45-74.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C., (2009), *Introduction to Algorithms*. third edition ed. The MIT Press.

Costa, B., Ogasawara, E., Murta, L., Mattoso, M., (2009), "Uma Estratégia de Versionamento de Workflows Científicos em Granularidade Fina". In: *III e-Science*, p. 49-56, Fortaleza, Ceara, Brazil.

Coutinho, F., Ogasawara, E., de Oliveira, D., Braganholo, V., Lima, A. A. B., Dávila, A. M. R., Mattoso, M., (2010), "Data parallelism in bioinformatics workflows using Hydra". In: *19th ACM International Symposium on High Performance Distributed Computing*, p. 507–515, New York, NY, USA.

Coutinho, F., Ogasawara, E., Oliveira, D., Braganholo, V., Lima, A. A. B., Dávila, A. M. R., Mattoso, M., (2011), "Many task computing for orthologous genes identification in protozoan genomes using Hydra", *Concurrency and Computation: Practice and Experience*, v. 23, n. 17 (dez.), p. 2326-2337.

Couvares, P., Kosar, T., Roy, A., Weber, J., Wenger, K., (2007), "Workflow Management in Condor", *Workflows for e-Science*, Springer, p. 357-375.

Critchlow, T., Chin, G., (2011), "Supercomputing and Scientific Workflows Gaps and Requirements". In: *IEEE World Congress on Services (SERVICES)*, p. 208-211, Washington, DC, USA.

Dantas, C. M. S., de Siqueira, M. Q., Ellwanger, G. B., Torres, A. L. F. L., Mourelle, M. M., (2004), "A Frequency Domain Approach for Random Fatigue Analysis of Steel Catenary Risers at Brazil's Deep Waters", *ASME Conference Proceedings*, v. 2004, n. 37432 (jan.), p. 199-209.

Dantas, M., (2005), *Computação Distribuída de Alto Desempenho: Redes, Clusters e Grids Computacionais*. 1 ed. Rio de Janeiro, Axcel Books.

Davidson, S. B., Freire, J., (2008), "Provenance and scientific workflows: challenges and opportunities". In: *ACM SIGMOD international conference on Management of data*, p. 1345-1350, Vancouver, Canada.

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.

Dean, J., Ghemawat, S., (2008), "MapReduce: simplified data processing on large clusters", *Commun. ACM*, v. 51, n. 1, p. 107-113.

Deelman, E., Gannon, D., Shields, M., 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., Vahi, K., (2007), "Pegasus: Mapping Large-Scale Workflows to Distributed Resources", *Workflows for e-Science*, Springer, p. 376-394.

Dias, J., Ogasawara, E., Mattoso, M., (2010a), "Paralelismo de dados científicos em workflows usando técnicas P2P". In: *IX Workshop de Teses e Dissertações em Banco de Dados*, p. 85-91, Belo Horizonte, Minas Gerais, Brazil.

Dias, J., Ogasawara, E., de Oliveira, D., Pacitti, E., Mattoso, M., (2010b), "Improving Many-Task computing in scientific workflows using P2P techniques". In: *IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS)*, p. 1-10, New Orleans, LA, USA.

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

Dias, J., Rodrigues, C., Ogasawara, E., Oliveira, D., Braganholo, V., Pacitti, E., Mattoso, M., (2010c), "SciMulator: Um Ambiente de Simulação de Workflows Científicos em Redes P2P". In: *Workshop P2P 2010*, p. 45-56, Gramado, Rio Grande do Sul - Brazil.

Digiampietri, L. A., Perez-Alcazar, J. de J., Medeiros, C. B., (2007), "An ontology-based framework for bioinformatics workflows", *Int. J. Bioinformatics Res. Appl.*, v. 3, n. 3 (set.), p. 268–285.

Disco, (2011), *Disco Project Webpage*, http://discoproject.org/.

Elmasri, R., Navathe, S. B., (2006), *Fundamentals of Database Systems*. 5 ed. Addison Wesley.

Emmerich, W., Butchart, B., Chen, L., Wassermann, B., Price, S. L., (2005), "Grid Service Orchestration using the Business Process Execution Language (BPEL)", *Journal of Grid Computing*, p. 283-304.

Fahringer, T., Prodan, R., Rubing Duan, Nerieri, F., Podlipnig, S., Jun Qin, Siddiqui, M., Hong-Linh Truong, Villazon, A., Wieczorek, M., (2005a), "ASKALON: a Grid application development and computing environment". In: *6th IEEE/ACM International Workshop on Grid Computing*, p. 122-131, Seattle, Washington, USA.

Fahringer, T., Qin, J., Hainzer, S., (2005b), "Specification of grid workflow applications with AGWL: an Abstract Grid Workflow Language". In: *5th IEEE International Symposium on Cluster Computing and the Grid*, p. 676–685, Washington, DC, USA.

Fileto, R., Liu, L., Pu, C., Assad, E. D., Medeiros, C. B., (2003), "POESIA: An ontological workflow approach for composing Web services in agriculture", *The VLDB Journal*, v. 12, n. 4 (nov.), p. 352–367.

Foster, I., Kesselman, C., (2004), *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann.

Fowler, M., (2003), *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3 ed. Addison-Wesley Professional.

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

Garg, S. K., Buyya, R., Siegel, H. J., (2010), "Time and cost trade-off management for scheduling parallel applications on Utility Grids", *Future Generation Computer Systems*, v. 26, n. 8 (out.), p. 1344-1355.

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

Gil, Y., Ratnakar, V., Deelman, E., Mehta, G., Kim, J., (2007b), "Wings for Pegasus: Creating Large-Scale Scientific Applications Using Semantic Representations of Computational Workflows". In: *The National Conference On Artificial Intelligence*, p. 1767-1774, Vancouver, BC, Canada.

Goble, C., Wroe, C., Stevens, R., (2003), "The myGrid project: services, architecture and demonstrator". In: *Proc. of the UK e-Science All Hands Meeting*, p. 595-602, Nottingham, UK.

González-Vélez, H., Leyton, M., (2010), "A survey of algorithmic skeleton frameworks: high-level structured parallel programming enablers", *Softw. Pract. Exper.*, v. 40, n. 12 (nov.), p. 1135–1160.

Governato, F., Brook, C., Mayer, L., Brooks, A., Rhee, G., Wadsley, J., Jonsson, P., Willman, B., Stinson, G., Quinn, T., Madau, P., (2010), "Bulgeless dwarf galaxies and dark matter cores from supernova-driven outflows", *Nature*, v. 463, n. 7278 (jan.), p. 203-206.

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

Greenwood, M., Goble, C., Stevens, R., Zhao, J., Addis, M., Marvin, D., Moreau, L., Oinn, T., (2003), "Provenance of e-Science Experiments - Experience from Bioinformatics", *UK OST e-Science second All Hands Meeting*, v. 4, p. 223-226.

Gropp, W., Lusk, E. L., Skjellum, A., (1999), *Using MPI - 2nd Edition: Portable Parallel Programming with the Message Passing Interface*. second edition ed. The MIT Press.

Guan, Z., Hernandez, F., Bangalore, P., Gray, J., Skjellum, A., Velusamy, V., Liu, Y., (2006), "Grid- Flow: a Grid- enabled scientific workflow system with a

Petri- net- based interface", *Concurrency and Computation: Practice and Experience*, v. 18, n. 10 (ago.), p. 1115-1140.

Guerra, G., Rochinha, F., Elias, R., Coutinho, A., Braganholo, V., Oliveira, D. de, Ogasawara, E., Chirigati, F., Mattoso, M., (2009), "Scientific Workflow Management System Applied to Uncertainty Quantification in Large Eddy Simulation". In: *Congresso Ibero Americano de Métodos Computacionais em Engenharia*, p. 1-13, Búzios, Rio de Janeiro, Brazil.

Guerra, G., Rochinha, F., Elias, R., Oliveira, D., Ogasawara, E., Dias, J., Mattoso, M., Coutinho, A. L. G. A., (2011), "Uncertainty Quantification in Computational Predictive Models for Fluid Dynamics Using Workflow Management Engine", *International Journal for Uncertainty Quantification, accepted*

Hadoop, (2010), *Apache Hadoop Web page*, http://hadoop.apache.org/.

Hellerstein, J. M., Stonebraker, M., (1993), "Predicate migration: optimizing queries with expensive predicates". In: *ACM SIGMOD Record*, p. 267–276, New York, NY, USA.

Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, B., Good, J., (2008), "On the use of cloud computing for scientific workflows". In: *IEEE Fourth International Conference on eScience (eScience 2008), Indianapolis, USA*, p. 7–12

Hoheisel, A., Alt, M., (2007), "Petri Nets", *Workflows for e-Science*, Springer, p. 190-207.

Hollingsworth, D., (1995), "Workflow Management Coalition: The Workflow Reference Model", *The Workflow Management Coalition*

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

Ibaraki, T., Kameda, T., (1984), "On the optimal nesting order for computing N-relational joins", *ACM Trans. Database Syst.*, v. 9, n. 3 (set.), p. 482–502.

Jamil, H., El-Hajj-Diab, B., (2008), "BioFlow: A Web-Based Declarative Workflow Language for Life Sciences". In: *IEEE Congress on Services*, p. 453-460, Honolulu, HI, USA.

Jarrard, R. D., (2001), *Scientific Methods*. Online book, Url.: http://emotionalcompetency.com/sci/booktoc.html.

Juristo, N., Moreno, A. M., (2001), *Basics of Software Engineering Experimentation*. 1 ed. Springer.

Kumar, V. S., Sadayappan, P., Mehta, G., Vahi, K., Deelman, E., Ratnakar, V., Kim, J., Gil, Y., Hall, M., Kurc, T., Saltz, J., (2009), "An integrated framework for performance-based optimization of scientific workflows". In: *18th ACM international symposium on High performance distributed computing*, p. 177–186, New York, NY, USA.

Larsen, R. J., Marx, M. L., (2011), *Introduction to Mathematical Statistics and Its Applications, An*. 5 ed. Prentice Hall.

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

Lawrence A. Crowl, (1994), *How to Measure, Present, and Compare Parallel Performance*, text, http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/mags/pd/&toc=comp/mags/pd/1994/01/p1toc.xml&DOI=10.1109/88.281869.

Lee, K., Paton, N. W., Sakellariou, R., Deelman, E., Fernandes, A. A. A., Mehta, G., (2008), "Adaptive Workflow Processing and Execution in Pegasus". In: *3rd International Conference on Grid and Pervasive Computing*, p. 99-106, Kunming, China.

Lee, K., Paton, N. W., Sakellariou, R., Fernandes, A. A. A., (2011), "Utility functions for adaptively executing concurrent workflows", *Concurrency and Computation: Practice and Experience*, v. 23, n. 6 (abr.), p. 646-666.

Lemos, M., Casanova, M. A., (2006), "On the Complexity of Process Pipeline Scheduling". In: *SBBD*, p. 57-71, Florianópolis, SC, Brazil.

Lemos, M., Casanova, M. A., Seibel, L. F. B., Macedo, J. A. F., Miranda, A. B., (2004), "Ontology-Driven Workflow Management for Biosequence Processing Systems", In: Galindo, F., Takizawa, M., Traunmüller, R. [orgs.] (eds), *Database and Expert Systems Applications*, Berlin, Heidelberg: Springer Berlin Heidelberg, p. 781-790.

Livny, I., Ioannidis, Y., Livny, M., Haber, E., Miller, R., Tsatalos, O., Wiener, J., (1994), "Desktop Experiment Management", *IEEE Data Engineering Bulletin*, v. 16, p. 1-5.

Ludäscher, B., Weske, M., Mcphillips, T., Bowers, S., (2009), "Scientific Workflows: Business as Usual?". In: *7th International Conference on Business Process Management*, p. 31–47, Berlin, Heidelberg.

Marinho, A., Murta, L., Werner, C., Braganholo, V., Cruz, S. M. S. da, Mattoso, M., (2009), "A Strategy for Provenance Gathering in Distributed Scientific Workflows". In: *IEEE International Workshop on Scientific Workflows*, p. 344-347, Los Angeles, California, United States.

Marinho, A., Murta, L., Werner, C., Braganholo, V., Cruz, S. M. S. da, Ogasawara, E., Mattoso, M., (2011), "ProvManager: a provenance management system for scientific workflows", *Concurrency and Computation: Practice and Experience*, v. (online)

Marinho, A., Murta, L., Werner, C., Braganholo, V., Cruz, S. M. S., Ogasawara, E., Mattoso, M., (2010a), "Managing Provenance in Scientific Workflows with ProvManager". In: *International Workshop on Challenges in e-Science - SBAC*, p. 17-24, Petrópolis, RJ - Brazil.

Marinho, A., Murta, L., Werner, C., Braganholo, V., Ogasawara, E., Cruz, S. M. S., Mattoso, M., (2010b), "Integrating Provenance Data from Distributed Workflow Systems with ProvManager", In: McGuinness, D. L., Michaelis, J. R., Moreau, L. [orgs.] (eds), *Provenance and Annotation of Data and Processes*, Berlin, Heidelberg: Springer Berlin Heidelberg, p. 286-288.

Martinho, W., Ogasawara, E., Oliveira, D., Chirigati, F., Santos, I., Travassos, G. H. T., Mattoso, M., (2009), "A Conception Process for Abstract Workflows: An Example on Deep Water Oil Exploitation Domain". In: *5th IEEE International Conference on e-Science*, Oxford, UK.

Matsunaga, A., Tsugawa, M., Fortes, J., (2008), "CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications", *IEEE eScience 2008*, p. 229, 222.

Mattos, A., Silva, F., Ruberg, N., Cruz, M., (2008), "Gerência de Workflows Científicos: Uma Análise Crítica no Contexto da Bioinformática", *COPPE/UFRJ*, n. Relatório técnico

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

Mattoso, M., Werner, C., Travassos, G., Braganholo, V., Murta, L., (2008), "Gerenciando Experimentos Científicos em Larga Escala". In: *SEMISH - CSBC*, p. 121-135, Belém, Pará - Brasil.

Mattoso, M., Werner, C., Travassos, G., Braganholo, V., Murta, L., Ogasawara, E., Oliveira, F., Martinho, W., (2009), "Desafios no Apoio à Composição de Experimentos Científicos em Larga Escala". In: *SEMISH - CSBC*, p. 307-321, Bento Gonçalves, Rio Grande do Sul, Brazil.

McGough, A. S., Lee, W., Cohen, J., Katsiri, E., Darlington, J., (2007), "ICENI", *Workflows for e-Science*, Springer, p. 395-415.

McPhillips, T. M., Bowers, S., (2005), "An approach for pipelining nested collections in scientific workflows", *ACM SIGMOD Record*, v. 34, n. 3 (set.), p. 12.

McPhillips, T., Bowers, S., Zinn, D., Ludäscher, B., (2009), "Scientific workflow design for mere mortals", *Future Generation Computer Systems*, v. 25, n. 5, p. 541-551.

Medeiros, C., Vossen, G., Weske, M., (1995), "WASA: A workflow-based architecture to support scientific database applications", *Database and Expert Systems Applications*, , p. 574-583.

Meyer, L., Scheftner, D., Vöckler, J., Mattoso, M., Wilde, M., Foster, I., (2007), "An Opportunistic Algorithm for Scheduling Workflows on Grids", *VECPAR 2006*, 1 ed, p. 1-12.

Montagnat, J., Isnard, B., Glatard, T., Maheshwari, K., Fornarino, M. B., (2009), "A data-driven workflow language for grids based on array programming principles". In: *4th Workshop on Workflows in Support of Large-Scale Science*, p. 7:1–7:10, New York, NY, USA.

Ocaña, K. A. C. S., Oliveira, D., Dias, J., Ogasawara, E., Mattoso, M., (2011a), "Optimizing Phylogenetic Analysis Using SciHmm Cloud-based Scientific Workflow". In: *2011 IEEE Seventh International Conference on e-Science (e-Science)*, p. 190-197, Stockholm, Sweden.

Ocaña, K. A. C. S., Oliveira, D., Ogasawara, E., Dávila, A. M. R., Lima, A. A. B., 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. [orgs.] (eds), *Advances in Bioinformatics and Computational Biology*, Berlin, Heidelberg: Springer Berlin Heidelberg, p. 66-70.

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

Ogasawara, E., Dias, J., Oliveira, D., Rodrigues, C., Pivotto, C., Antas, R., Braganholo, V., Valduriez, P., Mattoso, M., (2010), "A P2P approach to many tasks computing for scientific workflows". In: *9th international conference on high performance computing for computational science*, p. 327–339, Berlin, Heidelberg.

Ogasawara, E., Murta, L., Werner, C., Mattoso, M., (2008), "Linhas de Experimento: Reutilização e Gerência de Configuração em Workflows Científicos". In: *2 Workshop E-Science*, p. 31-40, Campinas, São Paulo, Brazil.

Ogasawara, E., Murta, L., Zimbrao, G., Mattoso, M., (2009a), "Neural networks cartridges for data mining on time series". In: *International Joint Conference on Neural Networks (IJCNN)*, p. 2302-2309, Atlanta, GA, USA.

Ogasawara, E., Oliveira, D., Chirigati, F., Barbosa, C. E., Elias, R., Braganholo, V., Coutinho, A., Mattoso, M., (2009b), "Exploring many task computing in scientific workflows". In: *MTAGS*, p. 1-10, Portland, Oregon, USA.

Ogasawara, E., Paulino, C., Murta, L., Werner, C., Mattoso, M., (2009c), "Experiment Line: Software Reuse in Scientific Workflows", In: Winslett, M. [org.] (eds), *Scientific and Statistical Database Management*, Berlin, Heidelberg: Springer Berlin Heidelberg, p. 264-272.

Ogasawara, E., Rangel, P., Murta, L., Werner, C., Mattoso, M., (2009d), "Comparison and versioning of scientific workflows". In: *ICSE Workshop on Comparison and Versioning of Software Models*, p. 25–30, Vancouver, BC, Canada.

Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M. R., Wipat, A., (2004), "Taverna: a tool for the composition and enactment of bioinformatics workflows", *Bioinformatics*, v. 20, p. 3045-3054.

Oinn, T., Li, P., Kell, D. B., Goble, C., Goderis, A., Greenwood, M., Hull, D., Stevens, R., Turi, D., Zhao, J., (2007), "Taverna/myGrid: Aligning a Workflow System with the Life Sciences Community", *Workflows for e-Science*, Springer, p. 300-319.

Oliveira, D., Cunha, L., Tomaz, L., Pereira, V., Mattoso, M., (2009a), "Using Ontologies to Support Deep Water Oil Exploration Scientific Workflows". In: *IEEE International Workshop on Scientific Workflows*, p. 364-367, Los Angeles, California, United States.

Oliveira, D., Ocana, K., Ogasawara, E., Dias, J., Baiao, F., Mattoso, M., (2011a), "A Performance Evaluation of X-Ray Crystallography Scientific Workflow Using SciCumulus". In: *IEEE International Conference on Cloud Computing (CLOUD)*, p. 708-715, Washington, D.C., USA.

Oliveira, D., Ogasawara, E., Baiao, F., Mattoso, M., (2011b), "Adding Ontologies to Scientific Workflow Composition". In: *XXVI SBBD*, p. 1-8, Florianópolis, SC, Brazil.

Oliveira, D., Ogasawara, E., Baião, F., Mattoso, M., (2010a), "SciCumulus: A Lightweight Cloud Middleware to Explore Many Task Computing Paradigm in

Scientific Workflows". In: *3rd International Conference on Cloud Computing*, p. 378–385, Washington, DC, USA.

Oliveira, D., Ogasawara, E., Chirigati, F., Silva, V., Murta, L., Werner, C., Mattoso, M., (2009b), "Uma Abordagem Semântica para Linhas de Experimentos Científicos Usando Ontologias". In: *III e-Science*, p. 17-24, Fortaleza, Ceara, Brazil.

Oliveira, D., Ogasawara, E., Ocana, K., Baiao, F., Mattoso, M., (2011c), "An Adaptive Parallel Execution Strategy for Cloud-based Scientific Workflows", *Concurrency and Computation: Practice and Experience*, v. (online)

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

Oliveira, D., Porto, F., Giraldi, G., Schulze, B., 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*, p. 6:1–6:6, New York, NY, USA.

Oliveira, F., Murta, L., Werner, C., Mattoso, M., (2008), "Using Provenance to Improve Workflow Design". In: *IPAW*, p. 136 - 143, Salt Lake City, UT, USA.

Olston, C., Reed, B., Srivastava, U., Kumar, R., Tomkins, A., (2008), "Pig latin: a not-so-foreign language for data processing". In: *Proc. of SIGMOD*, p. 1099-1110, Vancouver, Canada.

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

Parker, S. G., Johnson, C. R., (1995), "SCIRun: a scientific programming environment for computational steering". In: *ACM/IEEE conference on Supercomputing (CDROM)*, p. 52-71, San Diego, California, United States.

Pereira, W. M., Travassos, G. H., (2010), "Towards the conception of scientific workflows for in silico experiments in software engineering". In: *CM-IEEE International Symposium on Empirical Software Engineering and Measurement*, p. 34-38, Bolzano-Bozen, Italy.

Pinheiro, W. A., Vivacqua, A. S., Barros, R., Mattos, A. S., Cianni, N. M., Monteiro, P. C. L., Martino, R. N., Marques, V., Xexéo, G., Souza, J. M., (2007), "Dynamic Workflow Management for P2P Environments Using Agents", In: Shi, Y., Albada, G. D., Dongarra, J., Sloot, P. M. A. [orgs.] (eds), *Computational Science – ICCS 2007*, Berlin, Heidelberg: Springer Berlin Heidelberg, p. 253-256.

Porto, F., Tajmouati, O., Da Silva, V. F. V., Schulze, B., Ayres, F. V. M., (2007), "QEF - Supporting Complex Query Applications". In: *7th IEEE International Symposium on Cluster Computing and the Grid*, p. 846–851, Washington, DC, USA.

PostgreSQL, (2009), *PostgreSQL*, http://www.postgresql.org.

ProvChallenge, (2010), *Provenance Challenge Wiki*, http://twiki.ipaw.info/bin/view/Challenge/WebHome.

Qin, J., Fahringer, T., Pllana, S., (2007), "UML based Grid Workflow Modeling under ASKALON", *Distributed and Parallel Systems*, , p. 191-200.

Raicu, I., Foster, I. T., Yong Zhao, (2008), "Many-task computing for grids and supercomputers". In: *Workshop on Many-Task Computing on Grids and Supercomputers*, p. 1-11, Austin, Texas, USA.

Ranaldo, N., Zimeo, E., (2009), "Time and Cost-Driven Scheduling of Data Parallel Tasks in Grid Workflows", *IEEE Systems Journal*, v. 3, n. 1 (mar.), p. 104-120.

Rodrigues, M. V., Correa, F. N., Jacob, B. P., (2007), "Implicit domain decomposition methods for coupled analysis of offshore platforms", *Communications in Numerical Methods in Engineering*, v. 23, n. 6, p. 599-621.

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

Russell, N., ter Hofstede, A., Edmond, D., van der Aalst, W., (2005), "Workflow Data Patterns: Identification, Representation and Tool Support", *Conceptual Modeling – ER 2005*, , chapter 3716, Springer Berlin / Heidelberg, p. 353-368.

Russell, N., Hofstede, T., Edmond, D., van der Aalst, W., (2004), *Workflow Data Patterns* Disponível em: http://tmitwww.tm.tue.nl/staff/wvdaalst/Publications/p220.PDF.

Russell, N., Ter Hofstede, A. H. ., van der Aalst, W. M. ., Mulyar, N., (2006), "Workflow control-flow patterns: A revised view", *BPM Center Report BPM-06-22, BPMcenter. org*, p. 06–22.

Samples, M. E., Daida, J. M., Byom, M., Pizzimenti, M., (2005), "Parameter sweeps for exploring GP parameters". In: *2005 workshops on Genetic and evolutionary computation*, p. 212-219, Washington, D.C., USA.

Scheidegger, C. E., Vo, H. T., Koop, D., Freire, J., Silva, C. T., (2008), "Querying and re-using workflows with VsTrails". In: *ACM SIGMOD international conference on Management of data*, p. 1251-1254, Vancouver, Canada.

Shafi, A., Carpenter, B., Baker, M., (2009), "Nested parallelism for multi-core HPC systems using Java", *Journal of Parallel and Distributed Computing*, v. 69, n. 6 (jun.), p. 532-545.

Shields, M., (2007), "Control- Versus Data-Driven Workflows", *Workflows for e-Science*, Springer, p. 167-173.

Silberschatz, A., Korth, H., Sudarshan, S., (2010), *Database System Concepts*. 6 ed. McGraw-Hill Science/Engineering/Math.

Silva, E., Ogasawara, E., Oliveira, D., Benevides, M., Mattoso, M., (2010a), "Especificação Formal e Verificação de Workflows Científicos". In: *IV e-Science*, Belo Horizonte, Minas Gerais, Brazil.

Silva, V., Chirigati, F., Maia, K., Ogasawara, E., Oliveira, D., Braganholo, V., Murta, L., Mattoso, M., (2010b), "SimiFlow: Uma Arquitetura para Agrupamento de Workflows por Similaridade". In: *IV e-Science*, p. 1-8, Belo Horizonte, Minas Gerais, Brazil.

Silva, V., Chirigati, F., Maia, K., Ogasawara, E., Oliveira, D., Braganholo, V., Murta, L., Mattoso, M., (2011a), "Similarity-based Workflow Clustering", *Journal of Computational Interdisciplinary Science*, v. 2, n. 1, p. 23-35.

Silva, V., Chirigati, F., Ogasawara, E., Dias, J., Oliveira, D., Porto, F., Valduriez, P., Mattoso, M., (2011b), "Uma avaliação da Distribuição de Atividades Estática e Dinâmica em Ambientes Paralelos usando o Hydra". In: *V e-Science*, p. 1-8, Natal, Rio Grande do Norte, Brazil.

Simpson, B., Toussi, F., (2002). Hsqldb User Guide. Disponível em: http://www-lia.deis.unibo.it/Courses/TecnologieWeb0708/materiale/laboratorio/guide/hsqldb/guide.html. Acesso em: 8 set 2011.

Smanchat, S., Indrawan, M., Ling, S., Enticott, C., Abramson, D., (2009), "Scheduling Multiple Parameter Sweep Workflow Instances on the Grid". In: *5th IEEE International Conference on e-Science*, p. 300-306, Oxford, UK.

Soanes, C., Stevenson, A., (2003), *Oxford Dictionary of English*. 2 ed. Oxford University Press.

Sonmez, O., Yigitbasi, N., Abrishami, S., Iosup, A., Epema, D., (2010), "Performance Analysis of Dynamic Workflow Scheduling in Multicluster Grids". In: *10th IEEE International Symposium on High Performance Distributed Computing*, p. 49-60, Chicago, Illinois, USA.

Sroka, J., Włodarczyk, P., Krupa, Ł., Hidders, J., (2010), "DFL designer: collection-oriented scientific workflows with Petri nets and nested relational calculus". In: *1st International Workshop on Workflow Approaches to New Data-centric Science*, p. 5:1–5:6, New York, NY, USA.

Staples, G., (2006), "TORQUE resource manager". In: *ACM/IEEE conference on Supercomputing*, p. 8, Tampa, Florida, USA.

SWB, (2011), *SWB - Homepage*, http://datluge.nacad.ufrj.br/swb.

Tanenbaum, A. S., (2007), *Modern Operating Systems*. 3 ed. Prentice Hall.

Taylor, I. J., Deelman, E., Gannon, D. B., Shields, M., (2007a), *Workflows for e-Science: Scientific Workflows for Grids*. 1 ed. Springer.

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

Taylor, I., Shields, M., Wang, I., Philp, R., (2003), "Distributed P2P Computing within Triana: A Galaxy Visualization Test Case". In: *17th International Symposium on Parallel and Distributed Processing*, p. 16.1, Nice, France.

Thain, D., Tannenbaum, T., Livny, M., (2002), "Condor and the Grid", *Grid Computing: Making the Global Infrastructure a Reality. John Wiley & Sons Inc*

Travassos, G. H., 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*, p. 117-130, Rome, Italy.

Vaquero, L. M., Rodero-Merino, L., Caceres, J., Lindner, M., (2009), "A break in the clouds: towards a cloud definition", *SIGCOMM Comput. Commun. Rev.*, v. 39, n. 1, p. 50-55.

Vieira, I. N., Lima, B. S. L. P., Jacob, B. P., (2008), "Optimization of Steel Catenary Risers for Offshore Oil Production Using Artificial Immune System", In:

Bentley, P. J., Lee, D., Jung, S. [orgs.] (eds), *Artificial Immune Systems*, Berlin, Heidelberg: Springer Berlin Heidelberg, p. 254-265.

Vrhovnik, M., Schwarz, H., Suhre, O., Mitschang, B., Markl, V., Maier, A., Kraft, T., (2007), "An approach to optimize data processing in business processes". In: *VLDB*, p. 615-626, Vienna, Austria.

Walker, E., Guiang, C., (2007), "Challenges in executing large parameter sweep studies across widely distributed computing environments". In: *Workshop on Challenges of large applications in distributed environments*, p. 11-18, Monterey, California, USA.

Wang, J., Crawl, D., 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*, p. 1-8, Portland, Oregon.

Wang, Y., Lu, P., (2011), "Dataflow detection and applications to workflow scheduling", *Concurr. Comput. : Pract. Exper.*, v. 23, n. 11 (ago.), p. 1261–1283.

WfMC, I., (2009), *Binding, WfMC Standards*, WFMC-TC-1023, http://www. wfmc. org, 2000.

Wieczorek, M., Prodan, R., Fahringer, T., (2005), "Scheduling of scientific workflows in the ASKALON grid environment", *SIGMOD Rec.*, v. 34, n. 3, p. 56-62.

Wilde, M., Foster, I., Iskra, K., Beckman, P., Zhang, Z., Espinosa, A., Hategan, M., Clifford, B., Raicu, I., (2009), "Parallel Scripting for Applications at the Petascale and Beyond", *Computer*, v. 42, n. 11, p. 50-60.

Yu, J., Buyya, R., (2005), "A Taxonomy of Workflow Management Systems for Grid Computing", *Journal of Grid Computing*, v. 34, n. 3-4, p. 171-200.

Yu, J., Buyya, R., Tham, C. K., (2005), "Cost-Based Scheduling of Scientific Workflow Application on Utility Grids". In: *International Conference on e-Science and Grid Computing*, p. 140-147, Melbourne, Victoria, Australia.

Yunhong Gu, Grossman, R., (2008), "Exploring data parallelism and locality in wide area networks". In: *Workshop on Many-Task Computing on Grids and Supercomputers*, p. 1-10, Austin, TX, USA.

Zhao, Y., Dobson, J., Foster, I., Moreau, L., Wilde, M., (2005), "A notation and system for expressing and executing cleanly typed workflows on messy scientific data", *ACM SIGMOD Record*, v. 34, n. 3, p. 37-43.

Zhao, Y., Hategan, M., Clifford, B., Foster, I., von Laszewski, G., Nefedova, V., Raicu, I., Stef-Praun, T., Wilde, M., (2007), "Swift: Fast, Reliable, Loosely Coupled Parallel Computation". In: *3rd IEEE World Congress on Services*, p. 206, 199, Salt Lake City, USA.