COPPE/UFRJ

HYDRA: COMPONENTES PARA O PARALELISMO DE DADOS EM EXPERIMENTOS CIENTÍFICOS

Carlos Eduardo Barbosa

Dissertação de Mestrado 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 Mestre em Engenharia de Sistemas e Computação.

Orientadora: Marta Lima de Queirós Mattoso

Rio de Janeiro
Junho de 2010

HYDRA: COMPONENTES PARA O PARALELISMO DE DADOS EM EXPERIMENTOS CIENTÍFICOS

Carlos Eduardo Barbosa

DISSERTAÇÃO 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 MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

_____

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

_____

Prof. Alexandre de Assis Bento Lima, D.Sc.

_____

Prof. Fabio Andre Machado Porto, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

JUNHO DE 2010

Barbosa, Carlos Eduardo

Hydra: Componentes para o Paralelismo de Dados em Experimentos Científicos / Carlos Eduardo Barbosa. – Rio de Janeiro: UFRJ/COPPE, 2010.

VII, 64 p.: il.; 29,7 cm.

Orientadora: Marta Lima de Queirós Mattoso

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2010.

Referencias Bibliográficas: p. 54-60.

1. Many-Task Computing. 2. Workflows Científicos. 3. Paralelização de Atividades. I. Mattoso, Marta Lima de Queirós. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III Título.

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

HYDRA: COMPONENTES PARA O PARALELISMO DE DADOS EM EXPERIMENTOS CIENTÍFICOS

Carlos Eduardo Barbosa

Junho/2010

Orientadora: Marta Lima de Queirós Mattoso

Programa: Engenharia de Sistemas e Computação

Experimentos científicos lidam com uma grande quantidade de dados a ser processada por um encadeamento de atividades. *Workflows* científicos são utilizados como uma abstração para modelar esse encadeamento. Nesse contexto, essas atividades são candidatas a serem processadas de modo paralelo. Desta maneira, a execução do *workflow* científico pode se beneficiar do grande poder de processamento de máquinas paralelas. Com o intuito de diminuir a complexidade da especificação dessa paralelização, esta dissertação apresenta um conjunto de componentes num arcabouço, chamado Hydra. O objetivo do Hydra é facilitar a modelagem da paralelização de dados em atividades de workflows, integradas a um Sistema de Gerência de Workflows Científicos (SGWfC). Hydra foi avaliado e seus resultados experimentais mostram que ele permite a distribuição de atividades em computadores de alto desempenho, com ganho de desempenho próximo ao linear, de maneira transparente e sem nenhuma programação adicional.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

HYDRA: COMPONENTS FOR DATA PARALLELISM IN SCIENTIFIC EXPERIMENTS

Carlos Eduardo Barbosa

June/2010

Advisor: Marta Lima de Queirós Mattoso

Department: Computer Science Engineering

Scientific experiments deal with a large datasets to be processed and analyzed. Scientific workflows are used as processing model abstraction. In this context, these activities are candidates to be done as parallel tasks. Thus, the execution of scientific workflow can benefit from the large processing power of parallel machines and clusters. In order to reduce the complexity to specify this parallelization, this work presents a set of components in a framework, named Hydra. The Hydra's goal is to enable model parallel activities in workflows, integrated to Scientific Workflow Management Systems (SWfMS). Hydra was tested and its experimental results show that it allows the distribution of activities in high-performance computers, with almost linear acceleration, transparently and without any additional programming.

ÍNDICE

LISTAGEM DE FIGURAS

Capítulo 1 – Introdução

Experimentos científicos de grande escala são compostos por um conjunto de fluxos de dados gerados por uma cadeia de atividades que pode ser representada como um workflow. Cada experimento científico é concebido com o objetivo de confirmar ou refutar uma hipótese. Para confirmar ou refutar essa hipótese, o cientista explora variações de um workflow, fazendo ajustes de parâmetros, mudando um conjunto de dados ou mesmo atividades. Então, um workflow científico pode ser visto como um dos testes que são conduzidos em um experimento científico com o objetivo de avaliar uma situação controlada. O conjunto de testes representado por cada execução distinta do workflow define um experimento científico. Com isso, podemos dizer que um workflow científico pode ser visto como parte de um experimento científico. Sistemas de Gerência de Workflows Científicos (SGWfC) são hoje a principal ferramenta de apoio à gerência da execução de workflows.

Na execução de workflows científicos em grande escala, algumas atividades necessitam de soluções de alto desempenho para processar uma massa enorme de dados científicos. Várias atividades desses experimentos são baseadas em programas sequenciais legados, que processam os conjuntos de dados de maneira independente. Conforme o conjunto de dados se torna gigantesco, o processamento sequencial destas informações pode se tornar um gargalo. Além disso, no caso de código legado, programar sua paralelização dentro do contexto do workflow pode ser muito complexo, ou mesmo forçar que novos códigos sejam escritos.

Durante este processo de especificação de um workflow, diferentes estratégias de paralelização podem ser definidas e experimentadas. Estas variações devem ser controladas, tanto na especificação do workflow, quanto na sua execução correspondente. Quando o workflow é definido com interfaces de scripts ou de programação de baixo nível, o cientista pode se perder entre as diversas variações que estão sendo testadas para o mesmo experimento. Segundo (Ioan Raicu et al. 2008) existe uma necessidade de infraestruturas e plataformas genéricas no domínio científico para administração, planejamento, execução, acompanhamento e monitoramento de proveniência de workflows. Proveniência (Freire et al. 2008) é um recurso de armazenamento de um histórico que vincula a execução de um workflow com os dados por ele produzidos, o que permite ao cientista a verificação dos

resultados obtidos na execução, assim como recriar o experimento. O armazenamento da proveniência permite a rastreabilidade das execuções, podendo servir como uma auditoria, o que evita a reexecução do workflow em muitos casos (Marinho et al. 2009). A proveniência é uma característica fundamental de um SGWfC, uma vez que a análise dos resultados científicos, utilizando todo o fluxo de dados, só pode ser avaliada se os dados de proveniência tiverem sido coletados de uma forma estruturada e durante todas as fases do experimento. Existem dois tipos de proveniência: prospectiva e retrospectiva. A proveniência prospectiva é inserida pelo usuário em algum momento do ciclo de vida do experimento. A proveniência retrospectiva é coletada em algum momento do ciclo de vida. Os SGWfC atuais contêm mecanismos de coleta de proveniência (retrospectiva) de suas execuções.

O Open Provenance Model (OPM) é um modelo genérico de representação de artefatos, definido independente de tecnologia, e que tenta se tornar o modelo padrão para o armazenamento de execuções passadas de um workflow, isto é, de proveniência retrospectiva (Moreau et al. 2008). O OPM possui uma notação gráfica dividida em três entidades primárias. Um grafo OPM contém o relacionamento entre essas entidades primárias utilizando elementos de controle.

O ciclo de vida de um experimento científico deve ser coordenado como um todo. De acordo com Mattoso ET AL. (2008), o ciclo de vida do experimento possui três fases: composição, execução e análise.

Na fase de composição do experimento, diversos ensaios, com workflows abstratos, são concebidos, mapeados para uma especificação concreta e enviados para a fase de execução. Então seus resultados são analisados e comparados com os outros ensaios, basicamente utilizando dados de proveniência (Ioan Raicu et al. 2008). Segundo Freire ET AL. (2008) a proveniência deve ser coletada durante a composição (prospectiva) e durante a execução (retrospectiva) para que o experimento possa ser analisado e continuar seu ciclo de vida. Existem vários desafios para apoiar a composição, execução e análise de experimentos científicos de larga escala, em uma infraestrutura integrada. A composição de um workflow em larga escala é complexa não só devido à complexidade do workflow, mas, principalmente, ao número de ensaios que têm de ser gerenciados.

Existem poucas iniciativas para gerenciar as especificações abstratas de alto nível antes de compor workflows concretos e executáveis. Porto ET AL. (2008) propôs um Sistema de Gerenciamento de Modelos Científicos (SGMC) que provê ao cientista apoio para a concepção, criação, busca e execução de modelos científicos, e a gerência dos resultados das simulações e análises. O sistema se estrutura em quatro camadas. A camada de usuário fornece a interface aos cientistas para criar e editar os elementos do modelo, assim como solicitar serviços do sistema. A camada de gerência de metadados armazena os metadados do modelo científico e suporta os serviços de gerência de metadados. A camada de serviço provê suporte à avaliação da simulação e consulta aos resultados. A camada de gerenciamento de dados provê suporte à gerência de modelos científicos distribuídos e o encapsulamento de tipos dados complexos. Martinho ET AL. (2009) apresentam um processo de especificação de workflows conceituais com base na representação UML. Ogasawara, Paulino, ET AL. (2009) propuseram um modelo de linha de experimento, que é a evolução do conceito de linha de produto (Northrop 2002) para experimentos científicos. A ideia é que uma atividade possui uma analogia com um componente de reutilização de software, podendo ser substituído. Com isso, podemos definir a linha de experimento em um workflow e "instanciar" uma versão concreta dele, a partir de atividades que se comportam como componentes intercambiáveis. Cada atividade do workflow da linha de experimento pode receber atividades ou sub-workflows.

Para a fase de execução do experimento, existem inúmeros SGWfC. Eles estão sendo usados para orquestrar os fluxos de dados científicos, controlando toda a execução do workflow e coletando dados de proveniência durante a execução. Os SGWfC estão evoluindo no sentido de dar mais apoio ao desenho, à reutilização, ao versionamento e suporte à proveniência do workflow. No entanto, a fase de execução vem sendo tratada desconectada da fase de composição, com os SGWfC dando suporte ao workflow apenas em seu nível concreto. Um dos desafios da fase de execução em experimentos de grande escala é a combinação de características como interface gráfica para a construção do workflow concreto e a sua execução em ambientes de computação de alto desempenho (HPC). Atualmente, os SGWfC que são fortes na construção do workflow concreto e coleta de sua proveniência, como VisTrails (Callahan et al. 2006), Kepler (Altintas et al. 2004) e Taverna (Hull et al. 2006), carecem de suporte à HPC. Por outro lado, os SGWfC construídos especificamente

para computação distribuída (Jia Yu e Buyya 2005), tais como Pegasus (Deelman et al. 2007), Swift (Ioan Raicu et al. 2007) e Triana (Taylor et al. 2007) são fortes em suporte à HPC, mas não fornecem controles para suportar a construção do workflow com suporte à proveniência prospectiva correspondente. Por exemplo, em um SGWfC como Swift, cada variação de especificação representa uma nova codificação de workflow. Substituir a atividade de um pré-processamento de um workflow por outra atividade equivalente resulta em um novo workflow, diferente do original, e não uma variação dentro da evolução da construção de uma experiência científica. Controlar grande número de variações não é uma tarefa simples. Portanto, é um passo importante para a gerência de experimentos em larga escala a utilização de sistemas que combinam recursos na construção do workflow por meio de componentes visuais com recursos como controle de versão, sendo utilizados em um ambiente de HPC.

A fase de análise do experimento é complexa não só por causa do grande volume de dados de proveniência, mas também por causa da heterogeneidade da coleta de proveniência. A coleta de dados de proveniência no ambiente de alto desempenho não é simples, uma vez que o mecanismo de execução do workflow não é executado em ambientes remotos. O agendamento de tarefas no HPC é feito por um sistema específico, ao invés do mecanismo de workflow. Um SGWfC tradicional não está ciente da estratégia paralela utilizada na execução da atividade do workflow. Consequentemente, o controle da execução remota e a coleta de dados de proveniência das atividades paralelas são muito limitados do ponto de vista do SGWfC.

Considerando os desafios relativos a cada fase do experimento, esta dissertação foca a fase de execução, mantendo um relacionamento com a composição do workflow e a coleta de proveniência ao longo da execução paralela. O objetivo é fornecer facilidades de composição de workflows em SGWfC aliadas com o poder de processamento de ambientes de alto desempenho. Nesses experimentos científicos de grande escala, duas atividades comuns em workflows científicos se destacam: busca do parâmetro que dê o melhor resultado para o experimento; e o processamento conjunto de arquivos gigantescos, que podem demorar até semanas para serem processados. Ambos os tipos de atividades consomem um tempo de processamento grande, sendo candidatas a serem feitas de modo paralelo, podendo se beneficiar do grande poder de processamento de máquinas paralelas como HPC. Contudo, se um cientista tentar paralelizar sozinho esta atividade, ele precisará de conhecimentos

adicionais sobre computação paralela e com isso, a complexidade da modelagem do workflow pode aumentar significativamente. Mesmo que ele tenha essa habilidade, fazer a coleta de dados de proveniência no ambiente de HPC não é trivial.

A Computação de Torrente de Tarefas - *Many-Task Computing* (MTC) - é um paradigma muito atraente para paralelizar atividades de workflow em um ambiente de alto desempenho (I. Raicu, I.T. Foster, e Yong Zhao 2008). Contudo, a execução em paralelo de uma atividade pouco acoplada não deverá ser desconectada das atividades anteriores do workflow, executadas localmente. A abordagem Swift/Falcon consegue combinar o apoio à proveniência do workflow com MTC. No entanto, o Swift só trabalha com workflows programados em sua linguagem de especificação concreta para execução, baseada em scripts, e a proveniência prospectiva não é coletada. Vários SGWfC estão disponíveis, cada um com a sua própria estrutura de gerência de proveniência e ferramentas para a composição do workflow. O cientista deveria ser livre para escolher o SGWfC que melhor atende às necessidades da aplicação. E esta escolha não deve impedir a adoção de uma solução MTC para a execução de uma ou mais atividades de um workflow.

Neste contexto, o Hydra foi concebido como uma ponte entre SGWfC e ambientes de alto desempenho, fornecendo facilidades para a composição do workflow, assim como a coleta da proveniência de sua execução paralela no ambiente MTC, conforme a Figura 1. O Hydra contém um conjunto de componentes que podem ser incluídos na especificação do workflow de modo independente do SGWfC para controlar a paralelização de atividades como MTC. Além disso, esses componentes coletam dados de proveniência durante a execução remota do workflow paralelo. Com estes componentes, uma estratégia de paralelização MTC pode ser registrada, reutilizada e a proveniência pode ser coletada uniformemente.

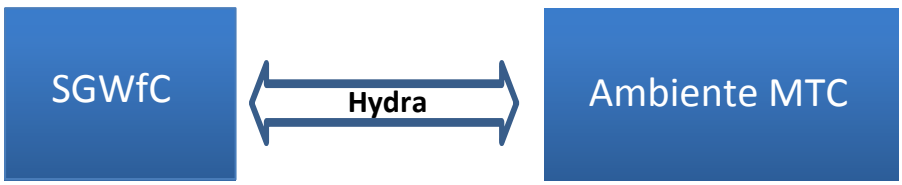

Figura 1. A conexão de um SGWfC a um ambiente MTC

Após a implementação do Hydra, o sistema foi validado em dois estudos de caso, com o objetivo de medir a sua eficiência na paralelização de atividades e verificar a sua capacidade

de paralelizar os casos propostos. O primeiro caso mostra que o Hydra possui a capacidade de ganho de desempenho em uma aplicação de mineração de dados. O segundo caso mostra a capacidade de coleta de dados de proveniência em uma aplicação de dinâmica de fluidos. Também foi possível avaliar a capacidade do Hydra em ser acoplado a um SGWfC (no caso o VisTrails).

Os estudos de avaliação do Hydra evidenciaram seu potencial de desempenho apresentando uma sobrecarga mínima na distribuição de atividades, agregação de dados e coleta da proveniência. O maior risco encontrado para a diminuição de desempenho do Hydra é a movimentação de dados de/para o ambiente distribuído, já que existem limitações de banda de entrada/saída. Uma vez que os conjuntos dados geralmente já estão disponíveis no ambiente distribuído, esta limitação geralmente pode ser contornada.

Os demais capítulos dessa dissertação estão organizados conforme a seguir. No Capítulo 2, são apresentados os conceitos básicos para o entendimento do Hydra, assim como é feito um aprofundamento sobre Workflows Científicos e paralelização de dados com Computação de Torrente de Tarefas. O Capítulo 3 é focado no Hydra, com a sua apresentação, a definição de sua arquitetura, a introdução de seus conceitos próprios, e a modelagem de seu repositório de proveniência. No final do capítulo, o Hydra é comparado a outras soluções existentes. No Capítulo 4, dois estudos de caso mostram que o Hydra é capaz de executar tarefas em ambiente MTC, com ganho de desempenho e coleta de dados de proveniência. No Capítulo 5, são apresentadas as conclusões sobre o Hydra, seus resultados, suas contribuições e o seu futuro.

Capítulo 2 – Workflows e Computação de Torrente de Tarefas

Este capítulo discute os conceitos básicos necessários para o entendimento do funcionamento do Hydra. Começando por uma explicação do que é uma atividade e uma tarefa, passando por workflows científicos, até as vantagens da utilização de técnicas de distribuição de processamento para Computação de Torrente de Tarefas. Com isso, pode-se notar as dificuldades em se criar workflows científicos que contenham atividades que são executadas em paralelo utilizando os SGWfC comuns, ou seja, que não foram especificamente criados para apoiar a execução de atividades paralelas.

2.1 Atividade e Tarefa

Nesta dissertação, definimos atividade como um comportamento ou uma ação para a qual possa ser visualizado um início e um fim; isto é, algo passível de execução. Uma atividade paralelizável é uma atividade que pode ter diversas instâncias simultâneas, cada qual consumindo um conjunto de dados de entrada diferente. Definimos como tarefa uma instância da atividade.

O Hydra é focado em atividades paralelizáveis, então, por simplicidade, nesta dissertação o termo atividade indica uma atividade paralelizável, a não ser que seja especificado o contrário.

2.2 Workflows

Um workflow é definido como um conjunto de atividades organizadas para realizar algum processo de negócio, incluindo a ordem de invocação das atividades e as condições em que as atividades devem ser iniciadas, assim como a sua sincronização e seu fluxo de dados (Georgakopoulos, Hornick, e Sheth 1995). Na definição primária de workflow, uma atividade pode ser executada por um ou mais sistemas de software, um ou mais seres humanos, ou uma combinação destes. As atividades humanas podem incluir interação forte com computadores (por exemplo, fornecendo comandos de entrada) ou interação fraca (por exemplo, usando computadores apenas para indicar o progresso da atividade).

Um workflow pode descrever as atividades de processos no nível conceitual necessário para a compreensão, avaliação e reformulação de todo o processo. Por outro lado, workflows podem capturar as atividades de processamento de informação em um nível que

descreve os requisitos do processo para a funcionalidade do sistema de informação. A distinção entre essas perspectivas de workflow nem sempre é feita, e por vezes o termo workflow é usado para descrever tanto a perspectiva de negócio, quanto a de sistemas de informação (Georgakopoulos, Hornick, e Sheth 1995).

2.3 Workflows Científicos

Com o amadurecimento da tecnologia de workflow, foi possível aplicar esse conhecimento no suporte a cientistas na criação de ensaios científicos e na análise de seus resultados. A grande diferença do workflow científico para o tradicional é a grande quantidade de dados para processamento, com longos períodos de execução do workflow. Com isso, os cientistas têm que evitar ao máximo as atividades humanas, para evitar perda de tempo na espera, pois, em workflows longos, a pessoa pode não estar disponível para dar continuidade a sua atividade no momento necessário.

Devido a essas características de desempenho, os sistemas de gerência de workflows científicos (SGWfC) precisam ter características diferentes dos sistemas de gerência de workflows comerciais. Como o seu foco é a *execução* de atividades científicas, os SGWfC devem ser robustos para o processamento científico. Uma das principais características de processamento científico é o processamento paralelo de atividades. O Hydra provê a oportunidade de implementar dois tipos de funcionalidades de paralelização com um pequeno esforço e com o mínimo de mudanças no workflow sequencial original. Para entendê-los, é necessário primeiro definir formalmente um workflow científico. Um workflow científico pode ser caracterizado como uma coleção de atividades organizadas para representar um modelo científico. De acordo com (Meyer et al. 2005) um workflow Wf pode ser representado pela quádrupla (A, Pt, I, O), onde:

- A é o conjunto de atividades $\{a_1, a_2, \ldots, a_n\}$ que compõem o workflow;
- Pt é o conjunto de parâmetros de entrada $\{pt_1, pt_2, \ldots, pt_r\}$ do workflow;
- I é o conjunto de dados de entrada $\{i_1, i_2, \ldots, i_m\}$ do workflow;
- O é o conjunto de conjuntos de saída de dados $\{O_1, O_2, \ldots, O_n\}$ de cada atividade do workflow, onde O_j representa o conjunto de dados de saída da atividade a_j.

Cada parâmetro em Pt, assim como cada dado de entrada em I pode assumir valores no domínio associado a ele. Assim, D_{pti} denota o domínio dos valores possíveis de pt_i, e D_{ij} denota do domínio de valores possíveis de dados de entrada i_j.

Existem vários tipos diferentes de paralelismo em workflows (Barga et al. 2008) e (Aalst et al. 2003). O Hydra foi especificado de modo a prover dois tipos: paralelismo de varredura de parâmetros (Walker e Guiang 2007) e (Samples et al. 2005) e paralelismo de dados (Meyer et al. 2007). Esses dois tipos de paralelismo podem representar uma barreira para o cientista controlar, coletar e registrar a proveniência do workflow no ambiente de computação paralela.

2.3.1 Paralelização de Dados em Workflows

O paralelismo de dados se caracteriza por executar a mesma tarefa em paralelo, consumindo dados diferentes. Implementa a ideia de acelerar a execução de atividades de workflows que processam um grande número de conjuntos de dados. Esses conjuntos de dados podem ser dados de entrada para todo o workflow, ou podem ser gerados dentro do próprio workflow para servir como entrada para a execução de uma única atividade, um sub-workflow, ou ainda podem ser parâmetros diferentes a serem explorados em inúmeras execuções. Ao invés de alimentar uma atividade com todo o conjunto de dados, o conjunto de dados é fragmentado. Então, cada um de seus fragmentos é utilizado por cada uma das instâncias da atividade, que será executada paralelamente. Uma vez que todos os fragmentos têm a execução agendada, podem existir diferentes semânticas de sincronização para prosseguir com a execução (Fabrício A.B. Silva, Carvalho, e Hruschka 2004).

Formalmente, conforme definimos em Ogasawara ET AL. (2009), o paralelismo de dados pode ser caracterizado como a execução simultânea de uma atividade de um workflow Wf, onde cada processo de execução de atividade a_i processa um subconjunto do conjunto de dados de entrada. Esse paralelismo pode ser conseguido replicando a atividade a_i em todos os nós do ambiente MTC e cada nó recebendo um subconjunto específico dos dados de entrada. Por exemplo, supondo que um workflow Wf tem como dados de entrada $I = \{i_1\}$. Assumindo que o domínio para os valores de entrada é $D_{i1} = \{1,3,5,7,9,12\}$, podem ser definidos fragmentos de $D_{i1,}$ utilizando uma função de fragmentação f. No exemplo, os fragmentos de D_{i1} resultantes de $f(D_{i1})$ são $D_{i1\text{-}f1} = \{1,3\}$, $D_{i1\text{-}f2} = \{5,7\}$, e $D_{i1\text{-}f3} = \{9,12\}$. Note que a função

de fragmentação precisa ser bijetiva. O conjunto de dados de saída FO_k gerado por cada nó k precisa ser agregado para produzir o resultado final (O_i), conforme apresentado na Figura 2.

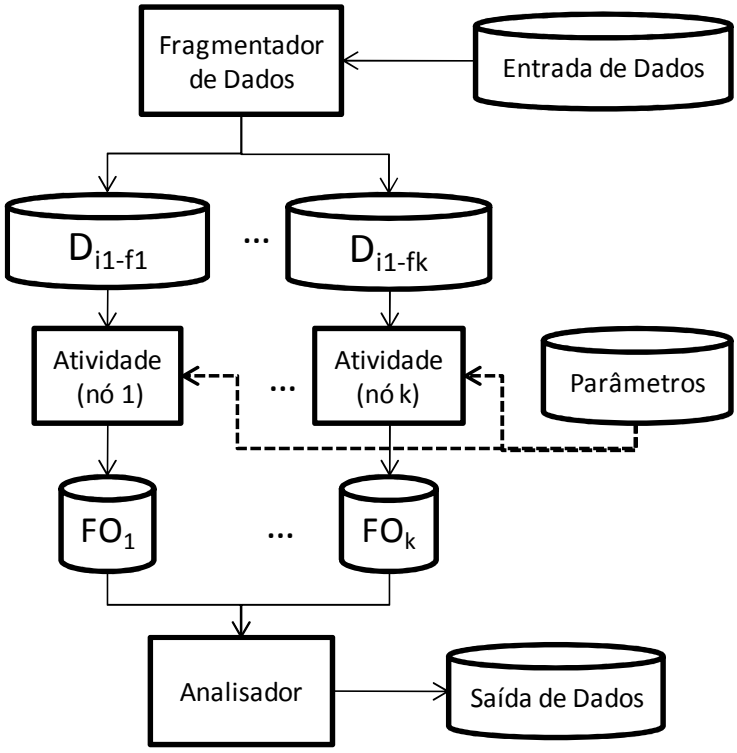

Figura 2. Paralelismo de Dados

2.3.2 Paralelização de Parâmetros em Workflows

O paralelismo de parâmetros se caracteriza em executar a mesma tarefa em paralelo, consumindo parâmetros diferentes. Frequentemente são chamados de varredura de parâmetros ou exploração de parâmetros. A varredura de parâmetros é importante porque os experimentos científicos costumam ser executados em várias configurações distintas, com o objetivo de descobrir o melhor parâmetro de entrada para se obter a saída desejada. Em workflows, a paralelização de parâmetros representa o encapsulamento da atividade que é executada com relação à configuração dos seus parâmetros de entrada, sendo cada configuração utilizada por uma das instâncias da atividade, que será executada paralelamente (Samples et al. 2005). Assim como na paralelização de dados, uma vez que todas as configurações de parâmetros têm a execução agendada, podem existir diferentes semânticas de sincronização para prosseguir com a execução.

Formalmente, conforme definimos em Ogasawara ET AL. (2009), o paralelismo de varredura de parâmetros é caracterizado pela execução simultânea de uma atividade de um

workflow Wf (ou todas as atividades de Wf), com cada execução utilizando valores diferentes para o parâmetro Pt. Por exemplo, supondo que um workflow Wf tem os parâmetros pt_1, pt_2 e pt_3; o domínio dos valores possíveis de pt_1, pt_2 e pt_3 são D_{pt1} = {x, x'}, D_{pt2} = {y, y'} D_{pt3} = {z, z'}. Uma instância de uma atividade a_i de Wf pode consumir os valores x, y, z para os parâmetros pt_1, pt_2 e pt_3, respectivamente, assim como outra instância pode consumir os valores x', y', z', entre outras combinações. Isto pode ser alcançado alocando diferentes valores de configuração para os parâmetros Pt para cada nó do ambiente MTC. Formalmente, essas configurações são formadas pegando os valores de parâmetro do domínio em certa ordem. Assim, a primeira configuração utiliza $D_{pt1}[1]$ (que denota a primeira configuração em D_{pt1}), $D_{pt2}[1]$, e $D_{pt3}[1]$. A segunda configuração utiliza $D_{pt1}[2]$, $D_{pt2}[2]$, e $D_{pt3}[2]$, e assim por diante. O conjunto de dados de entrada O_k gerado pela k-ésima execução precisa ser agregado, uma vez que ela se refere a diferentes execuções do mesmo experimento. A Figura 3 ilustra esse tipo de paralelismo, onde cada atividade é replicada em n nós. Neste caso, I é o mesmo em todos os nós.

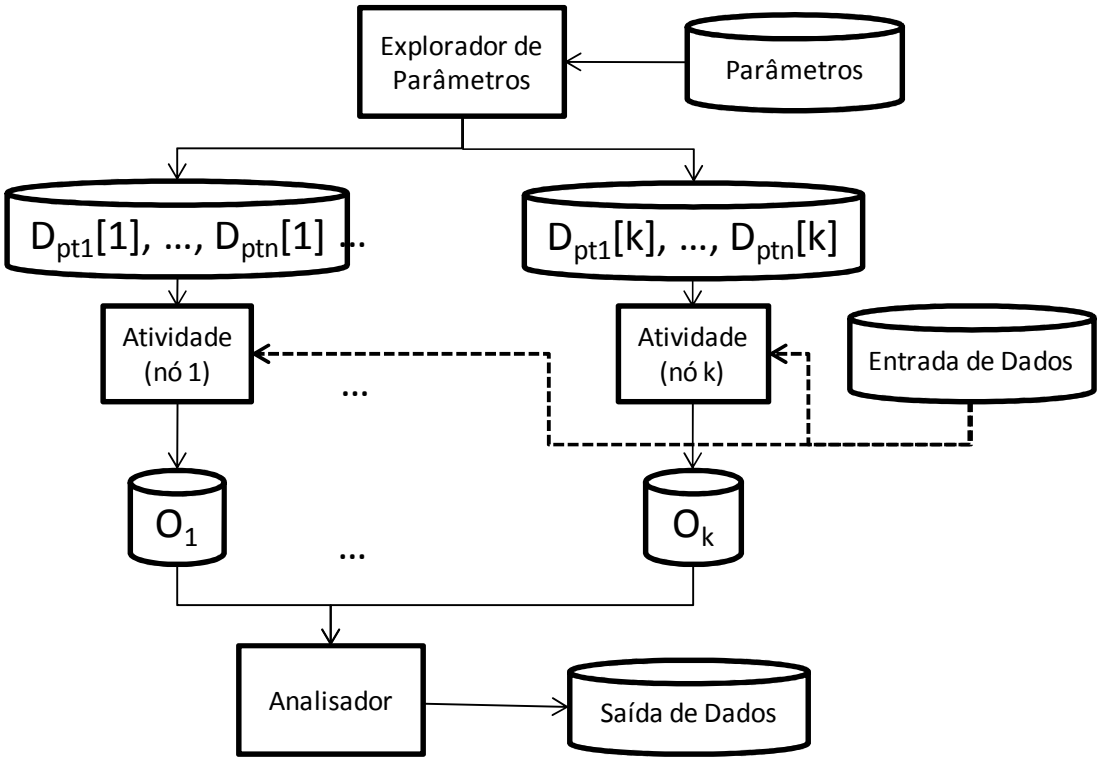

Figura 3. Paralelismo de Varredura de Parâmetros

2.4 Computação de Torrente de Tarefas

Computação de Torrente de Tarefas - *Many-Task Computing* (MTC) é definido como um conjunto gigantesco de tarefas, dependentes entre si ou não, que podem ser agendadas individualmente em muitos recursos computacionais, entre várias fronteiras administrativas, para atingir o objetivo maior da aplicação (I. Raicu, I.T. Foster, e Yong Zhao 2008). O diferencial do MTC para com outras terminologias é a sua escala. Uma abordagem MTC está focada em executar aplicações que podem ter milhões de tarefas e/ou processar petabytes de dados.

Algumas aplicações de Computação de Torrente de Tarefas são do tipo *Embarrassingly Parallel*, mas outras são extremamente complexas e com forte comunicação entre as tarefas e o sistema de arquivos. As tarefas em um MTC podem variar bastante: podem ser pequenas ou grandes, monoprocessadas ou multiprocessadas, focadas em dados ou focadas em cálculos. O conjunto de tarefas pode ser estático ou dinâmico, homogêneo ou heterogêneo, e fortemente dependente ou fracamente dependente entre si (I. Raicu, I.T. Foster, e Yong Zhao 2008).

Os exemplos mais comuns de aplicações MTC incluem simulações de Monte Carlo, buscas intensas, aplicações de manipulação de imagens e aplicação de algoritmos de mineração de dados. É importante lembrar que várias aplicações MTC são chamadas de aplicações de varredura de parâmetros (Fabrício A.B. Silva, Carvalho, e Hruschka 2004).

As aplicações MTC são boas candidatas para serem executadas em um HPC (I. Foster e Kesselman 2004), uma vez que elas apresentam características intrínsecas de escalabilidade. Em processamento paralelo, a escalabilidade pode ser definida como a capacidade de o sistema computacional aumentar a aceleração da execução na medida em que o número de processadores aumenta (Culler, Jaswinder Pal Singh, e Gupta 1999).

Muitas aplicações científicas são compostas por um conjunto de atividades sequenciais e independentes. Neste contexto, independente significa que não existem comunicações ou dependências entre as atividades. A entrada para uma atividade geralmente é um ou mais arquivos de entrada, e um arquivo pode ser entrada para mais de uma atividade. A saída de uma atividade geralmente é também um ou mais arquivos de saída, o que significa que cada atividade gera seu próprio conjunto de arquivos de saída (Sulistio e Buyya 2005).

Vários cenários diferentes podem se enquadrar como aplicações MTC. As Figuras 2 e 3 mostram cenários possíveis. Esses cenários de MTC podem ser classificados de acordo com a relação entre seus dados e o tempo de processamento. Como apresentado por Fujimoto e Hagihara (2006), um cenário típico para uma transferência de dados pequena é o SETI@Home, no qual uma tarefa tem 3,9 trilhões de operações de ponto flutuante, cujo processamento demora em torno de 10 horas em um Pentium II de 500MHz, envolve apenas 350KB para a carga de dados e 1KB para o retorno dos resultados. No lado oposto, existem aplicações com grande consumo de dados, que processam conjuntos de dados grandes e distribuídos, e são guiadas por fatores diferentes como a velocidade de acesso e transferência dos dados, assim como o seu tempo de processamento (Venugopal e Buyya 2005). Cada um desses cenários impõe diferentes limites para os mecanismos de agendamento.

Uma das características principais na execução de aplicações em ambientes distribuídos é o aumento da complexidade da distribuição no agendamento das tarefas. As aplicações MTC são especialmente adequadas a ambientes distribuídos porque elas podem ser executadas com recursos intermitentes, isto é, recursos sem garantias de disponibilidade ou confiabilidade. Neste cenário, bom desempenho e resultados confiáveis são conseguidos por escalonadores gulosos para ambientes distribuídos que confiam em replicação para evitar perdas de desempenho que são consequência de escolhas ruins de processador para uma tarefa em uma grade (Costa, Cirne, e Fireman 2005). Isto ocorre devido ao fato de que máquinas paralelas que compõem uma grade têm diferentes capacidades de computação. E isso inevitavelmente aumenta o *makespan*, que é o momento em que termina a execução da última tarefa, podendo também ser entendido como o tempo total de execução distribuída (Fujimoto e Hagihara 2006). Surpreendentemente, escalonadores gulosos não estão preparados para o uso de recursos do tipo espaço compartilhado. Isto ocorre porque o uso de recursos de espaço compartilhado envolve a submissão de uma requisição detalhada para o escalonador do recurso, especificando o número de processadores requisitados, e qual será o tempo de alocação desses processadores – informações que um escalonador guloso não está preparado para prover. Com isso, computadores de espaço compartilhado, o tipo mais poderoso de recurso computacional disponível, poderiam melhorar em muito o tempo de execução das aplicações MTC (Costa, Cirne, e Fireman 2005).

Capítulo 3 – O Hydra

A demanda por computação de alto desempenho tem aumentado em várias áreas de pesquisa. Experimentos científicos modelados como workflows científicos podem potencialmente se beneficiar do grande número de processadores, quando executados em um HPC, usando o paradigma Computação de Torrente de Tarefas.

Neste contexto, o Hydra é um sistema capaz de distribuir, controlar e monitorar a execução de atividades de um workflow científico em um ambiente MTC. O Hydra foi desenvolvido para prover um conjunto de componentes que liga o ambiente MTC e um SGWfC de maneira transparente, podendo ser instanciado sob demanda como parte de um workflow, preservando as outras atividades do workflow, que são executadas localmente. É possível classificar o Hydra como uma aplicação de Computação de Torrente de Tarefas, pois é capaz de efetuar execuções de atividades paralelas cujas tarefas sejam: estáticas, homogêneas e independentes entre si.

O Hydra provê uma abordagem sistemática para contemplar esses dois tipos de paralelismo com coleta de proveniência heterogênea distribuída, sendo desacoplado da máquina de execução do workflow, além de possuir outras características: (i) os componentes do Hydra agem de maneira transparente, isolando o cientista da complexidade do ambiente MTC; (ii) permite que os cientistas monitorem as aplicações legadas em recursos distribuídos através dos dados de proveniência; e (iii) provê uma interação pouco acoplada entre os SGWfC locais e o ambiente MTC distribuído.

O Hydra complementa os SGWfC que não são capazes de execuções paralelas, pois oferece controle da execução de uma atividade paralela em um workflow científico e a coleta da proveniência da execução paralela. Além disso, é possível reaproveitar os workflows científicos já criados, já que a paralelização de uma atividade com o Hydra afeta apenas a execução da atividade paralela, mantendo o restante do workflow intacto.

Quanto aos SGWfC que são capazes de execuções paralelas, o Hydra oferece coleta da proveniência da execução paralela com semântica, i. e., com informações sobre os dados consumidos e gerados, assim como sobre o contexto da execução paralela, associando a execução a uma atividade de um experimento.

É importante frisar que as aplicações focadas nesta dissertação não permitem a divisão arbitrária de computação entre os processos independentes, ou seja, aplicações com divisão de carga de trabalho não são consideradas. A quantidade de computação de cada tarefa é considerada fixa, não pode ser mudada e não é previamente conhecida. Este capítulo discute as principais características relacionadas ao desenvolvimento do Hydra.

Apesar de utilizar uma conexão segura para transferir os dados, existem tantas outras questões sobre segurança, que estão fora do escopo desta dissertação. Para informações sobre proveniência e questões de segurança, veja (Gadelha e M. Mattoso 2008) e (Hasan, Sion, e Winslett 2007).

3.1 Conceitos Utilizados

Um *workspace* (pasta de trabalho) é o diretório que contém os arquivos consumidos por uma determinada instância da atividade. Com isso, para cada tarefa do Hydra, será gerada uma pasta de trabalho. O *template* (pasta de modelo) é o diretório que contém os arquivos primários que são replicados para cada pasta de trabalho antes da execução da tarefa. No momento da replicação, os arquivos são instrumentados, ou seja, marcas textuais específicas no arquivo são procuradas e substituídas pelos respectivos parâmetros, específicos para cada instância da atividade.

A modelagem do Hydra utiliza a noção de cartuchos (Szyperski 1997). Um cartucho é uma unidade componente que pode ser substituída dinamicamente (Szyperski 1997).

3.2 Características Arquiteturais

A arquitetura do Hydra é dividida em duas camadas: Cliente e MTC. A camada Cliente fica na máquina local do cientista, e possui quatro componentes: Hydra Setup, Uploader, Dispatcher, Downloader. Esses componentes são responsáveis por configurar o experimento e a paralelização da atividade, enviar arquivos, disparar a execução paralela, e receber arquivos, respectivamente. A Camada MTC do Hydra possui apenas um componente: o Hydra MTC.

O Hydra utiliza o conceito de cartuchos, que é uma interface padrão para programas externos executarem agregações de resultados (cartucho de agregação) e/ou fragmentações de arquivos (cartucho de fragmentação). O objetivo desta interface é permitir uma grande

flexibilidade por parte do cientista na hora de executar seus experimentos, permitindo a seleção automática do resultado esperado, assim como a fragmentação automática dos seus arquivos de entrada.

Utilizando o Hydra, o cientista precisa ser capaz de desenvolver o seu workflow científico no SGWfC, e se precisar, de cartuchos específicos de agregação e fragmentação. Estes cartuchos têm a uma vantagem: só precisam ser construídos uma vez, para cada tipo de dado de entrada ou saída. Uma vez construído, um cartucho pode ser reutilizado, distribuído na Internet, etc.

A Figura 4 mostra o diagrama de componentes que inclui o workflow do usuário, os componentes clientes do Hydra, o componente MTC do Hydra, e os cartuchos de fragmentação e de agregação. Os componentes serão mostrados em detalhes na próxima subseção.

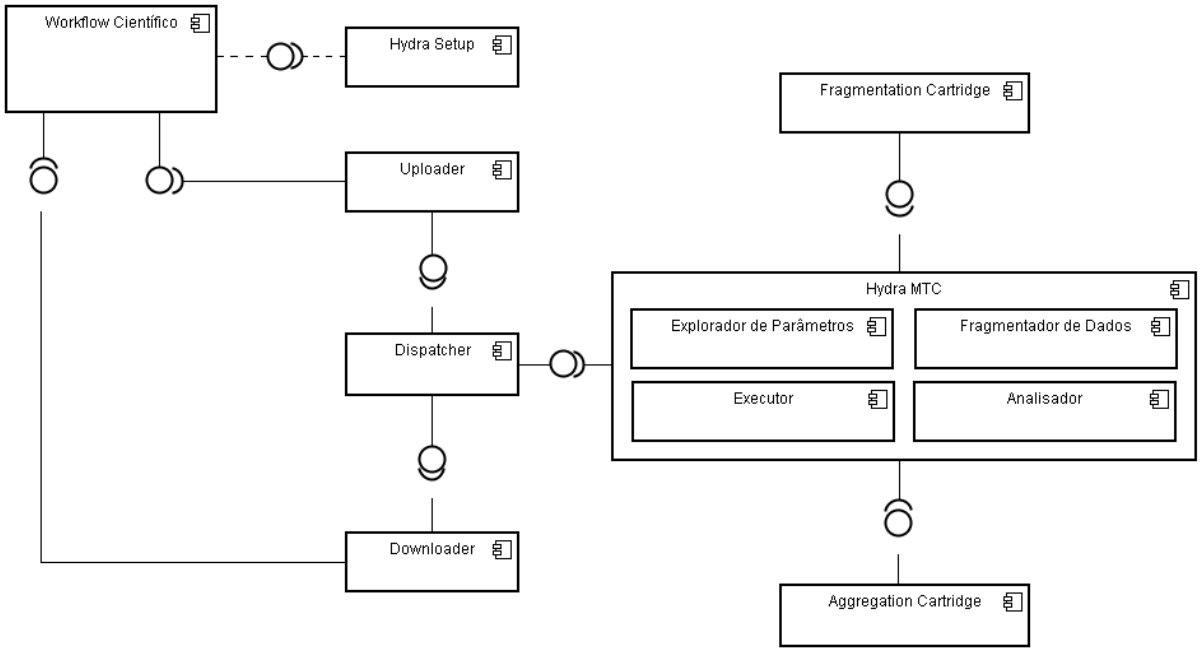

Figura 4. Diagrama de Componentes do Hydra

3.3 Arquitetura do Hydra

A Figura 5 mostra a arquitetura do Hydra em duas camadas. A primeira camada (do lado esquerdo da Figura 5) é formada por componentes locais que ficam instalados na máquina cliente do usuário. A segunda camada (lado direito) é formada pelos componentes que lidam com a distribuição no ambiente MTC. Os números ao lado das setas indicam a

sequência de execução dos componentes da arquitetura, e as letras denotam o fluxo de dados dos componentes da arquitetura. As subseções a seguir explicam os detalhes de cada camada.

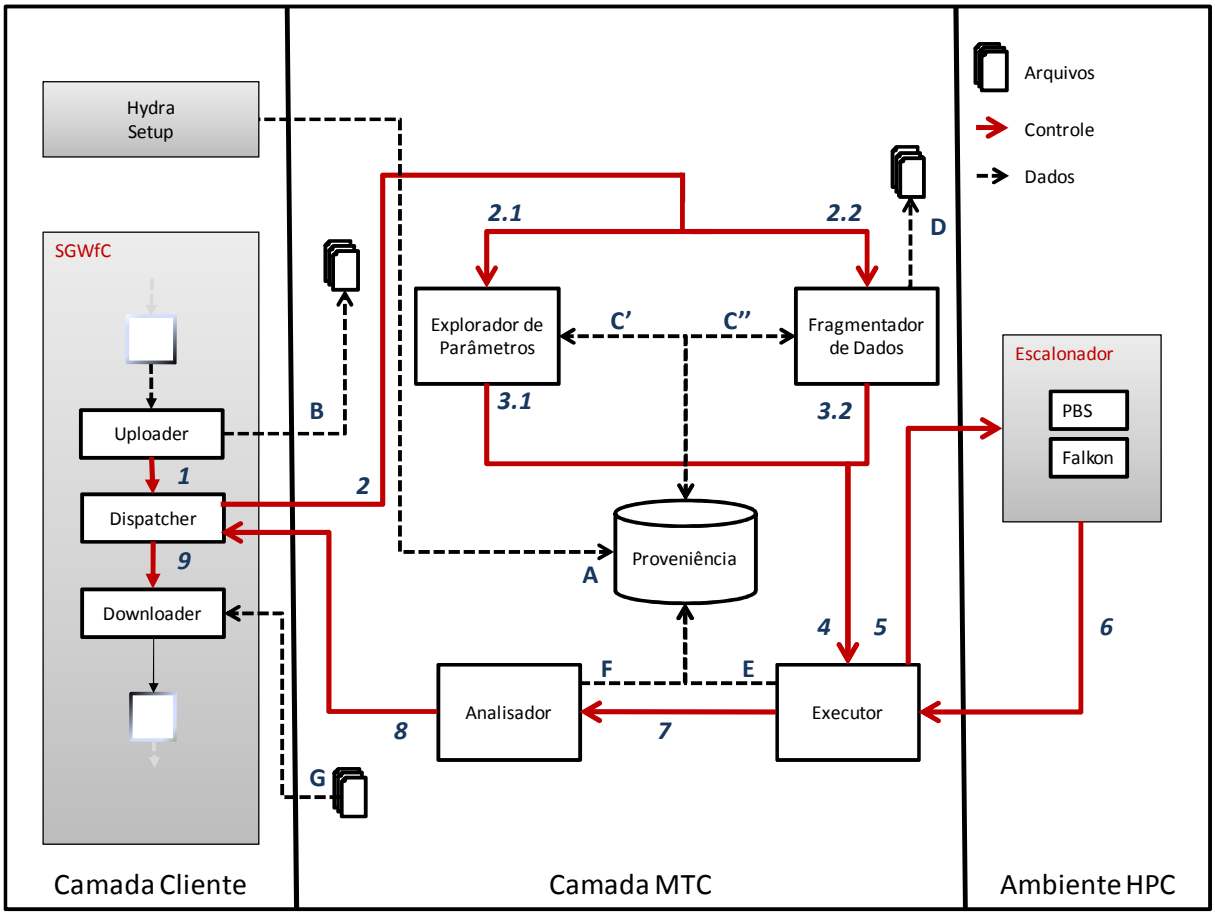

Figura 5. A Arquitetura do Hydra

3.3.1 Camada Cliente do Hydra

A ideia principal por trás dos clientes do Hydra é prover meios de paralelizar workflows científicos de maneira transparente e preservando suas características originais. Workflows científicos já podem ser modelados em seus ambientes de SGWfC locais (Deelman et al. 2009). Neste caso, os componentes da camada cliente do Hydra são inseridos no workflow científico para se comunicar com a camada MTC do Hydra, que executa a distribuição dos dados e sua paralelização. Na camada cliente, neste caso o SGWfC, o Hydra tem quatro componentes principais: *Hydra Setup*, *Uploader*, *Dispatcher* e *Downloader*. Esses quatro últimos componentes devem ser inseridos no workflow científico sequencial, já modelado pelo cientista, substituindo a atividade que precisa ser paralelizada. Se uma atividade do workflow for paralelizada, ela deve ser substituída pelas quatro atividades de infraestrutura para promover a sua paralelização.

O componente *Hydra Setup* é responsável entrada de dados de proveniência prospectiva, assim como pelas configurações gerais da execução paralela da camada MTC do Hydra. Este componente oferece ao cientista uma painel de controle, permitindo que ele defina o modelo de paralelização (dados ou varredura de parâmetros); os arquivos de dados que precisam ser transferidos ao ambiente MTC; os arquivos locais que precisam ser enviados à pasta de modelo de dados do ambiente MTC; e a gerência das aplicações distribuídas. Essas configurações são armazenadas no repositório de proveniência no ambiente MTC (seta A na Figura 5) e são utilizadas pela camada MTC do Hydra para configurar o ambiente MTC (seta C na Figura 5) e controlar o tipo de paralelismo durante a execução do workflow. Ele também fornece os dados coletados pelo monitoramento remoto do Hydra. Este componente é fundamental, uma vez que várias consultas de proveniência relacionadas a todo o experimento podem ser apenas respondidas quando os dados de proveniência são coletados. Estruturas de coleta de proveniência podem ser utilizadas para prover informações úteis no nível operacional para vários propósitos, como: análise e predição de desempenho, ajustes, detecção de falhas e agendamento (Freire et al. 2008).

O componente *Uploader* é o componente responsável pelo envio de conjunto de dados para o ambiente distribuído (seta B na Figura 5). Este componente pode ser configurado diretamente ou utilizar configurações registradas pelo *Hydra Setup* para obter esses conjuntos de dados e arquivos que precisam ser transferidos para o ambiente MTC.

Uma vez que os dados e aplicações já foram transferidos para o ambiente distribuído, o *Dispatcher* inicia o processo de execução da atividade distribuída no ambiente MTC (seta 2 na Figura 5). Adicionalmente, ele tem características de monitoramento que permitem ao cientista monitorar a situação das aplicações distribuídas que estão sendo executadas. A execução remota se torna transparente com o uso desse componente. Uma vez que a execução da aplicação termina, o *Dispatcher* retorna o controle ao SGWfC.

Finalmente, o componente *Downloader* é o componente responsável por trazer os dados de resultados conseguidos com a execução distribuída para o computador cliente (seta G na Figura 5). Este componente acessa as configurações feitas no *Hydra Setup*, sobre os diretórios utilizados localmente e remotamente para transferência de arquivos. Após a execução do *Downloader*, as outras atividades são executadas paralelamente, a não ser que alguma outra atividade esteja configurada para ser paralelizada.

3.3.2 Camada MTC do Hydra

A camada MTC do Hydra é formada pelo componente Hydra MTC, porém para o melhor entendimento sobre o seu funcionamento, ela foi representada diretamente pelos seus quatro subcomponentes: Explorador de Parâmetros, Fragmentador de Dados, Executor, e o Analisador.

Conforme pode ser observado na Figura 5, duas diferentes linhas de execução (2.1 e 2.2) podem ocorrer. Assim, os dois subcomponentes seguintes são responsáveis pela configuração das aplicações/workflows para a execução, de acordo com o tipo de paralelização.

O subcomponente Explorador de Parâmetros é responsável por manipular as combinações dos parâmetros recebidos pelo workflow cliente para uma atividade específica que está sendo paralelizada. Este subcomponente também lê dados do repositório de proveniência para criar pastas de trabalho isoladas para cada conjunto de parâmetros (seta C na Figura 5), criado a partir da replicação e instrumentação dos arquivos de modelo.

Por outro lado, o Fragmentador de Dados fragmenta os dados originais para ser distribuído para cada instância da atividade (tarefa) que está sendo paralelizada. O Fragmentador de Dados é dependente do problema, uma vez que precisa ser customizado para fragmentar diferentes tipos de dados de entrada, como a função Map na estratégia Map-Reduce (Dean e Ghemawat 2008). Esta função é definida por um cartucho específico, indicado pelo cientista. O Fragmentador de Dados delega a fragmentação dos arquivos ao cartucho indicado pelo cientista. Este subcomponente também é responsável pela criação da estrutura de diretórios que armazena os dados fragmentados. Ele lê dados do repositório de proveniência, que contém informações de proveniência prospectiva indicando como fragmentar dados que precisam ser colocados em cada pasta de trabalho, além dos arquivos de modelo, que precisam ser replicados e instrumentados em cada pasta de trabalho.

Com isso, é possível criar pastas de trabalho isoladas para cada fragmento de dado e/ou parâmetro utilizado. Cada pasta de trabalho é relacionada com uma execução simples da atividade que está sendo distribuída (se tornando assim uma tarefa), incluindo os arquivos de dados e os parâmetros específicos da execução específica. Uma vez que um dado conjunto de

dados é fragmentado em diferentes partes, cada parte é consumida por uma réplica da instância da atividade.

Alguns arquivos da pasta de modelo são instrumentados com *tags*, de maneira similar às *tags* do Java Server Pages (JSP) (Bergsten 2003), para diferenciar cada parâmetro ou fragmento de dados. Tanto o Explorador de Parâmetros quanto o Fragmentador de Dados podem resolver cada arquivo instrumentado em cada diretório replicado para apoiar a execução das tarefas distribuídas, utilizando o parâmetro ou fragmento de dados correto.

Após todo o ambiente ser configurado, o Executor é iniciado (seta 4 na Figura 5). Este componente é responsável por invocar escalonadores externos - como Torque (Bayucan, Henderson, e J. P. Jones 2000) ou Falkon (Ioan Raicu et al. 2007) - (seta 5 na Figura 5) para executar a atividade/workflow no ambiente distribuído. É importante enfatizar que o Hydra não está limitado a trabalhar com um escalonador específico. De fato, cartuchos podem ser desenvolvidos para permitir que o Hydra trabalhe com diferentes escalonadores externos. O problema de agendamento está fora do escopo da dissertação, já que, além de existirem várias metodologias, a arquitetura do Hydra é capaz de se expandir a quantos escalonadores forem possíveis. Alguns desses escalonadores são capazes de executar um SGWfC remoto. Por exemplo, imagine que uma atividade *X* no workflow (que contém os componentes do Hydra), é responsável por invocar um workflow de terceiros. Quando o Hydra se comunicar com os escalonadores externos, esses escalonadores vão invocar o SGWfC com diferentes parâmetros ou dados de entrada.

Após invocar o escalonador externo, o Executor continua a monitorar a execução paralela. Ele executa comandos de monitoração aos escalonadores do ambiente MTC para descobrir a situação da execução. Atualmente, o Hydra confia na tolerância a falhas do escalonador externo. Porém, essa questão deve ser melhorada para dar suporte à falha no escopo do workflow científico que está sendo executado, como feito por outras abordagens, como Pegasus/Condor (Deelman et al. 2007). Todos os dados e metadados coletados são armazenados no repositório de proveniência (seta E na Figura 5) na camada MTC do Hydra.

Depois do final da execução das instâncias paralelas, o subcomponente Analisador é iniciado (seta 7 na Figura 5). Este subcomponente é responsável pela combinação dos resultados finais (no caso da paralelização de dados) executando uma função de agregação

opcional para organizar os resultados finais e transferi-los para a camada cliente (SGWfC). Esta função é definida por um cartucho específico, indicado pelo cientista. Particularmente, no caso de fragmentação de dados, o cartucho que faz papel equivalente ao da função Reduce, na estratégia Map-Reduce. O subcomponente Analisador também armazena informações no repositório de proveniência, uma vez que os resultados finais são gerados por ele e eles precisam ser registrados para propósitos de proveniência (seta F na Figura 5).

Uma característica que os cartuchos de agregação devem ter é criar um arquivo chamado hydraoutput.txt, em cada pasta de trabalho do experimento executado. Os cartuchos de agregação utilizam esse arquivo como interface para a passagem de proveniência para o Hydra. O arquivo hydraoutput.txt tem um formato simples de nome da variável e valor, que são lidos e adicionados no banco de dados de proveniência.

3.4 Proveniência do Hydra

O repositório de proveniência do Hydra tem o objetivo de armazenar todos os dados e metadados coletados pela camada MTC do Hydra (proveniência retrospectiva), assim como os dados informados pelo cientista utilizando o Hydra Setup (proveniência prospectiva). Este repositório contém informações sobre o experimento, e os dados de suas execuções distribuídas. O repositório de proveniência foi projetado para permitir a ligação entre as proveniências prospectiva e retrospectiva, e que informações de proveniência possam ser coletadas durante o ciclo de vida do experimento. O Hydra armazena, em um banco de dados relacional, tanto a proveniência prospectiva do experimento científico quanto a parte distribuída da proveniência retrospectiva. Mesmo assim, devido ao fato de que a OPM não manipula a execução distribuída e a proveniência prospectiva, um desafio para Hydra será associar seu modelo de proveniência com o modelo OPM (Marinho et al. 2009), para resolver essas questões.

A coleta da proveniência paralela é importante, pois permite ao cientista realizar consultas de alto nível sobre os resultados da execução. Pode, por exemplo, ligar os arquivos gerados por cada execução com o conjunto de parâmetros consumido pela execução.

Observando a Figura 6, que apresenta a modelagem conceitual do repositório de proveniência do Hydra, é possível notar que foi feita uma separação dos dois tipos de proveniência existentes: prospectiva (preta) e retrospectiva (branca).

As classes da modelagem são: *ExperimentWorkflow*, *DistribuitedActivity*, *Parameter*, *TemplateFile*, *DataCartidge*, *AggregationCartidge*, *Aggregation*, *ExperimentFile*, *FileToFragment*, *FragmentatedFile*, *DistribuitedActivityExecution*, *OutputValue*, *Task*, *SweepedParameter* e *WorkspaceFile*.

A classe *ExperimentWorkflow* é responsável por armazenar as informações de proveniência prospectiva básica referente ao workflow do experimento.

A classe *DistribuitedActivity* é responsável por armazenar as informações de proveniência prospectiva referente a uma atividade do workflow a ser distribuída com a ajuda do Hydra. Além de armazenar o nome e a descrição da atividade, é onde informações de acesso SSH do ambiente MTC.

A classe *DistribuitedActivityExecution* é responsável por armazenar as informações de proveniência retrospectiva referente à execução de uma atividade do workflow no ambiente MTC. Armazena o número de processadores utilizados, a pasta de modelo, o horário de início e fim de execução, assim como o tempo de fila.

A classe *Task* é responsável por armazenar as informações de proveniência retrospectiva sobre a execução de uma instância da atividade que está sendo paralelizada. Armazena informações como o horário de início e fim, as saídas padrão e de erro, o *status* de saída do programa e até em qual processador do ambiente MTC foi executado.

A classe *Parameter* é responsável por armazenar as informações de proveniência prospectiva referente aos parâmetros que serão explorados no experimento, basicamente, o nome do parâmetro e tipo de dado esperado.

A classe *TemplateFile* é responsável por armazenar as informações de proveniência prospectiva referente aos arquivos que serão utilizados como modelo do experimento. Todos os arquivos cadastrados são copiados e instrumentados nos diversas pasta de trabalho durante a execução do experimento.

Figura 6. Conceito em alto nível do Repositório de Proveniência do Hydra

A classe *DataCartidge* é responsável por armazenar as informações de proveniência prospectiva referente aos cartuchos de programas de fragmentação de dados, especificamente, o seu nome e o caminho do programa.

A classe *AggregationCartidge* é responsável por armazenar as informações de proveniência prospectiva referente aos cartuchos de agregação de dados, especificamente, o seu nome, o seu caminho e o seu tipo de agregação.

A classe *Output* é responsável por armazenar as informações de proveniência prospectiva referente às saídas obtidas no experimento, basicamente, o nome da saída e tipo de dado esperado.

A classe *OutputValue* é responsável por armazenar as informações de proveniência retrospectiva sobre os valores das variáveis de saída do experimento. Esses valores são utilizados em *Aggregation* para diferenciar os resultados do experimento, isto é, escolher o(s) melhor(es) resultado(s).

A classe *FileToFragment* é responsável por armazenar as informações de proveniência retrospectiva sobre os arquivos que foram fragmentados e o número de fragmentos nos quais eles foram divididos.

A classe *FragmentatedFile* é responsável por armazenar as informações de proveniência retrospectiva referentes a cada arquivo de fragmento realizado pelo cartucho indicado em *DataCartidge* sobre o arquivo indicado em *FileToFragment*.

A classe *ExperimentFile* é responsável por armazenar as informações de proveniência retrospectiva sobre os arquivos gerados pelo Hydra MTC para experimento, seja pela fragmentação feita em *FileToFragment* quanto pela agregação feita em *Aggregation*. Os arquivos gerados naturalmente pelo experimento não participam desta classe, apenas os arquivos que foram fragmentados antes da execução ou agregados após a execução.

A classe *SweepedParameter* é responsável por armazenar as informações de proveniência retrospectiva do valor de um parâmetro utilizado em uma determinada execução da atividade definida em *Task*.

A classe *WorkspaceFile* é responsável por armazenar as informações de proveniência retrospectiva dos arquivos existentes (copiados da pasta de modelo mais os arquivos gerados pelo programa) na pasta de trabalho da execução da atividade definida em *Task*.

A modelagem conceitual do repositório de proveniência, exibida na Figura 6 foi implementada na modelagem de banco de dados mostrada na Figura 7.

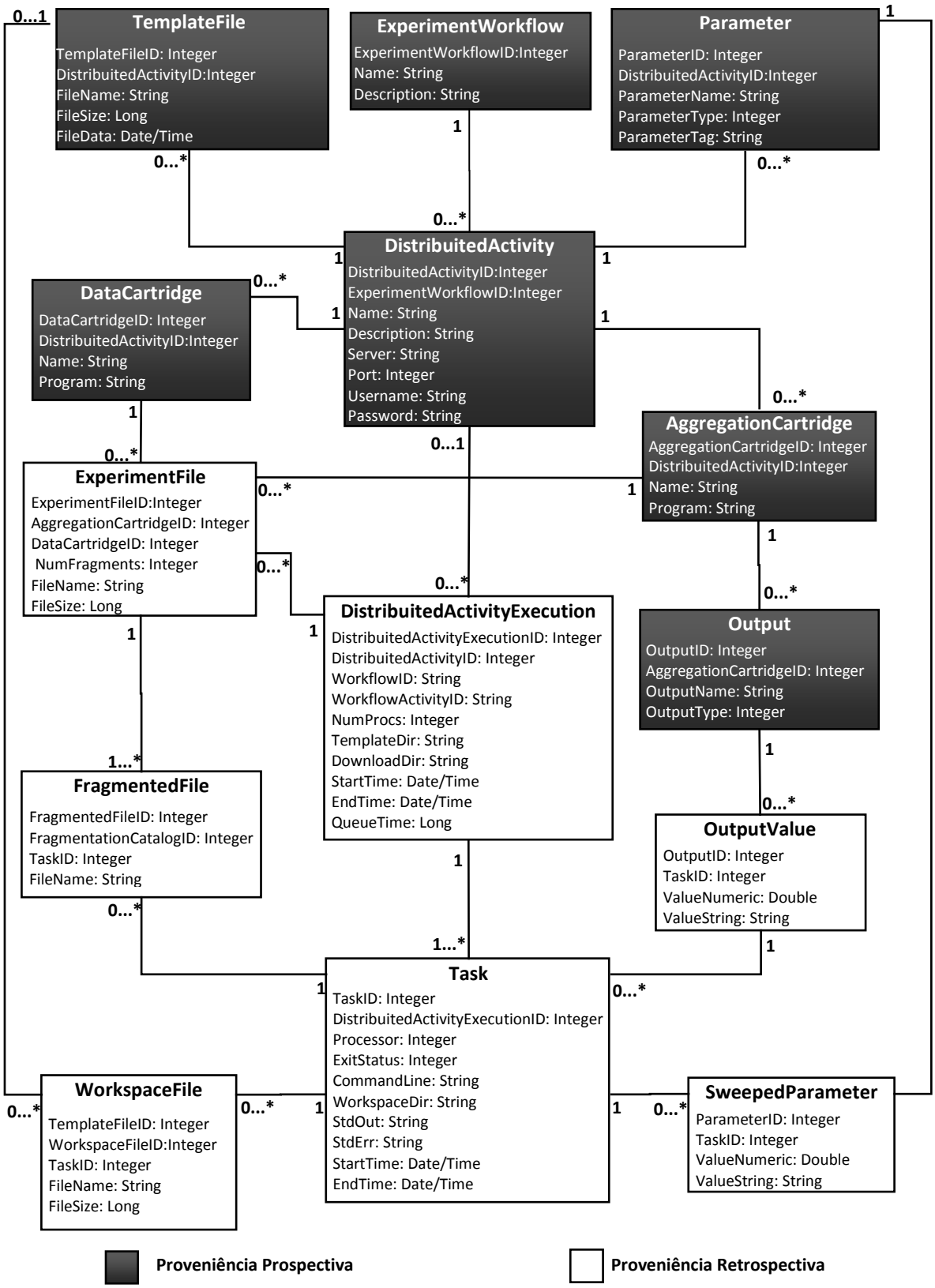

Figura 7. Projeto Físico do Repositório de Proveniência do Hydra

3.4.1 Consultas à Proveniência

Utilizando o esquema de banco de dados apresentado na Figura 7, é possível formar um grande número de consultas, para responder diversas perguntas sobre o experimento científico feito. Apesar de essas perguntas genéricas serem focadas na proveniência retrospectiva, elas não poderiam ser respondidas pelos SGWfC existentes. Este modelo dá flexibilidade ao cientista para escrever consultas personalizadas para a aplicação paralela de maneira fácil, ou mesmo permitir que outros sistemas utilizem dados capturados na proveniência para executar seu processamento. A seguir, apresentamos uma lista de consultas que podem ser feitas no repositório de proveniência sobre um determinado experimento. Essas consultas evidenciam o potencial da ferramenta, tanto na parte conceitual do relacionamento entre componentes do workflow executado, quanto na parte de consulta a dados de desempenho. Foram feitas consultas reais no Hydra e seus resultados, disponíveis no Anexo I desta dissertação.

"Quais são as pastas de trabalho das tarefas que tiveram o parâmetro X com valores maiores que Y e menores que Z?"

```
SELECT t.workspace FROM  Task t, SweepedParameter sp, Parameter p WHERE
p.ParameterID =  sp.ParameterID AND sp.TaskID = t.TaskID AND  p.ParameterName =
X AND  sp.NumericValue > Y AND  sp.NumericValue < Z;
```

"Qual é o tempo médio das tarefas que tiveram o parâmetro X com valores maiores que Y e menores que Z?"

```
SELECT  AVG(t.EndTime-t.StartTime)  FROM  Task t, SweepedParameter sp, Parameter
p WHERE  p.ParameterID =  sp.ParameterID AND  sp.TaskID = t.TaskID AND
p.ParameterName = X AND  sp.NumericValue > Y AND  sp.NumericValue < Z;
```

"Qual foram as tarefas que tiveram o parâmetro X com valores maiores que Y e menores que Z?"

```
SELECT  t.TaskID, t.CommandLine FROM Task t, SweepedParameter sp, Parameter p
WHERE p.ParameterID = sp.ParameterID AND sp.TaskID = t.TaskID AND  p.Name = X
AND  sp.NumericValue > Y AND sp.NumericValue < Z;
```

"Quais foram as saídas (stdout e stderr) de cada tarefa executada?"

```
SELECT StdOut, StdErr FROM Task WHERE DistribuitedActivityExecutionID=X;
```

"Quais foram os arquivos criados ou modificados durante uma tarefa?"

```
SELECT Filename FROM WorkspaceFile WHERE TaskID=X;
```

“As tarefas foram concluídas com sucesso?”

```
SELECT * FROM Task WHERE DistribuitedActivityExecutionID=X AND ExitStatus=0;
```

“Quantos processadores foram utilizados em uma atividade?”

```
SELECT NUMProcs FROM DistribuitedActivityExecution WHERE
DistribuitedActivityExecutionID = “X”;
```

“Qual foi o tempo médio de execução das tarefas da atividade?”

```
SELECT avg(EndTime-StartTime) FROM Task WHERE DistribuitedActivityExecutionID=X;
```

“Quais foram os parâmetros de cada tarefa executada?”

```
SELECT p.Name, p.Type, sp.NumericValue, sp.TextValue FROM Task t, Parameter p,
SweepedParameter sp WHERE p.ParameterID=sp.ParameterID AND sp.TaskID = t.TaskID
AND t.TaskID=X;
```

“Quais foram os tempos de execução de cada tarefa executada?”

```
SELECT TaskID, (EndTime-StartTime) FROM Task WHERE
DistribuitedActivityExecutionID=X;
```

“Qual foi o desvio padrão do tempo de execução das tarefas da atividade?”

```
SELECT STDEV(EndTime-StartTime) FROM Task WHERE
DistribuitedActivityExecutionID=X;
```

“Quantas execuções de atividades foram feitas em um determinado período?”

```
SELECT Count(1) FROM (SELECT DISTINCT DistribuitedActivityExecutionID FROM
DistribuitedActivityExecution WHERE StartTime > X AND StartTime < Y) as Execs;
```

3.5 Como Utilizar o Hydra

Para utilizar o Hydra, o cientista precisa modelar normalmente o workflow em um SGWfC. Então, é preciso configurar o experimento no Hydra Setup, fazendo o seu cadastro, informando as características do experimento científico, da atividade paralelizada, do acesso ao servidor HPC, e informando os conjuntos parâmetros, pasta de modelo, arquivos de modelo, os cartuchos de fragmentação de dados e agregação de resultados, e as saídas esperadas - o cadastro da proveniência prospectiva. Após essa configuração, o cientista precisa substituir o componente que será paralelizado pelos componentes do Hydra no workflow.

Os componentes do Hydra precisam ser conectados entre si, sendo necessário fazer uma pequena configuração, informando qual experimento ele representa no repositório de

proveniência. Durante a execução do experimento, o Hydra Setup pode ser utilizado para acompanhar a execução das tarefas via dados de proveniência, inclusive para saber quantas tarefas foram concluídas, calcular o tempo esperado para o final da execução, os arquivos gerados pelas tarefas que já terminaram, entre outras.

3.6 Hydra e as Soluções Existentes

As abordagens mais parecidas com o Hydra são o Swift/Falkon e o Nimrod/K. O Falkon provê uma maneira eficiente de despachar e executar diversas tarefas pequenas, de acordo com o paradigma MTC em grades (Ioan Raicu et al. 2007). O principal objetivo do Falkon é resolver problemas como longos tempos de fila e sistemas agendados por lotes, envio de tarefas em taxas baixas e escalabilidade ruim em sistemas de arquivos compartilhados. O Falkon tem sido utilizado com o Swift para prover MTC em workflows científicos. O Hydra também é focado em prover MTC para workflows, porém visa ser independente do SGWfC. Desse modo, o Falkon pode ser visto como complementar ao Hydra, uma vez que pode fazer o papel de escalonador para o Hydra. Quando comparamos o Swift/Falkon com o Hydra, a coleta de proveniência é focada no experimento, ao invés de ser focada nas instâncias isoladas do workflow, além de incluir proveniência da especificação abstrata do workflow e sua execução, incluindo o MTC. Assim, o Swift também pode se beneficiar com o Hydra.

Por outro lado, Nimrod/K é um complemento para o SGWfC Kepler. Nimrod/K inclui novos "diretores" e "atores" no Kepler para orquestrar a execução de uma varredura de parâmetros em HPC, grades e em nuvem. Nimrod/K também inclui um componente de otimização utilizado para otimizar o SGWfC distribuído. No entanto, Nimrod/K só funciona com Kepler e se limita a lidar com a varredura de parâmetros.

MyCluster (Walker e Guiang 2007) é um sistema que provê uma infraestrutura de *proxy* para a submissão de tarefas paralelas ao escalonador local (PBS, por exemplo) de um HPC com suporte para falhas transientes. Além disso, o MyCluster permite que tarefas acessem arquivos de maneira transparente do computador *host* através dos locais remotos. MyCluster também pode ser visto como complementar ao Hydra, uma vez que Mycluster suporta a execução de tarefas paralelas em cascata e poderia ser utilizado como um dos

componentes do Hydra. Ao contrário do MyCluster, o Hydra provê integração com SGWfC genérico, que inclui suporte à proveniência.

O projeto Dryad (Isard et al. 2007) provê uma infraestrutura avançada para permitir que um programador de computador utilize os recursos de um ambiente de HPC para executar paralelização de dados de programas sequenciais. O programador é capaz de usar milhares de máquinas, cada uma delas com múltiplos processadores ou núcleos, sem saber nada sobre programação concorrente. Embora o Dryad proporcione um ambiente poderoso para a MTC, carece de suporte a experimentos científicos, pois é um *framework* de programação. Com uma linguagem intuitiva, é simples para um cientista desenvolver aplicações paralelas utilizando o Dryad. No entanto, esta abordagem é desconectada de SGWfC, não provê varredura de parâmetros e não está focada com as questões de proveniência.

Outra abordagem que utiliza o paradigma MTC é apresentada em (Li Hui, Huashan Yu, e Li Xiaoming 2008). É um *framework* leve para executar várias tarefas independentes em grades. Ele foi aplicado como uma modificação do escalonador Gracie, da Universidade de Pequim, podendo ser aplicado a outros escalonadores. Ele agrupa dinamicamente as tarefas de diferentes granularidades e despacha simultaneamente os grupos para recursos computacionais distribuídos. Embora o trabalho em (Li Hui, Huashan Yu, e Li Xiaoming 2008) concentre-se em acelerar a execução intensa de tarefas independentes em ambientes de grade, provendo um agendamento/execução MTC transparente, ele não suporta fragmentação de dados, varredura de parâmetros e coleta de dados de proveniência.

Sawzal (Pike et al. 2005) é uma linguagem de programação e um *framework* para o suporte ao paradigma MTC. O Sawzal explora o paralelismo na análise de enormes conjuntos de dados. Embora apresente uma abordagem interessante para a paralelização de dados, esse *framework* não é acoplado a qualquer SGWfC. Além disso, Sawzal não provê varredura de parâmetros e não dá suporte a proveniência dos dados.

Com base nessas informações, foi construída uma tabela comparativa entre as abordagens. As abordagens possuem naturezas diferentes, algumas são frameworks de programação, outras afetam o escalonador, mas os principais concorrentes para o Hydra são um conjunto de componentes e um SGWfC paralelo. Esses são os concorrentes mais próximos para a execução de atividades científicas em larga escala. Os quesitos da

comparação são: o tipo de escalonador utilizado na execução, prover varredura de parâmetros, prover fragmentação de dados, permitir integração com SGWfC e por fim, o tipo de proveniência que pode ser colhida da execução paralela do experimento. A tabela comparativa é mostrada na Tabela 1.

Tabela 1. Tabela comparativa entre as abordagens.

Abordagem	Natureza	Escalonador	Varredura de Parâmetros	Fragmentação de Dados	Integração com SGWfC	Tipo de Proveniência
Hydra	Conjunto de Componentes	Independe	✓	✓	Qualquer	Prospectiva e Retrospectiva
Nimrod/K	Conjunto de Componentes	Utilizados no Kepler	✓		Kepler	Retrospectiva
Swift	SGWfC	Independe	✓		Não Aplicável	Retrospectiva
Sawzal	Framework	N/D		✓	N/D	N/D
Dryad	Framework	N/D		✓	N/D	N/D
MyCluster	Proxy de agendamento	Independe			N/D	N/D
Gracie[1]	Modificação no escalonador	Independe			N/D	N/D

[1] Neste caso, não se refere ao escalonador Gracie, mas sim a mudança feita nele, uma vez que a metodologia avaliada (Li Hui, Huashan Yu, e Li Xiaoming 2008) não possui um nome.

Capítulo 4 – Avaliação do paralelismo do Hydra

O Hydra foi avaliado em dois casos distintos: um workflow de mineração de dados, utilizando redes neurais, conforme descrito em (Barbosa et al. 2009); e um workflow de dinâmica de fluídos conforme descrito em (Ogasawara, Oliveira, et al. 2009), utilizando o EdgeCFD (Elias e Coutinho 2007) (Elias, Martins e Coutinho 2005). Esses dois casos foram escolhidos devido a grande diferença na aplicação fim; enquanto o primeiro é um aplicativo de predição de séries temporais, armazenadas em um banco de dados relacional, o segundo é uma aplicação científica mais tradicional, de processamento de um conjunto de arquivos de entrada, e que se divide em etapas de pré-processamento, processamento e pós-processamento. Ambos os casos são de varredura de parâmetros, uma vez que a implementação do mecanismo de fragmentação de dados ficou fora do escopo desta dissertação.

Os experimentos para estimar os ganhos em desempenho foram executados em uma máquina multiprocessada NUMA (SGI Altix ICE 8200), localizada no Laboratório de Alto Desempenho (NACAD) da COPPE/UFRJ. A máquina utiliza o Suse Linux Enterprise Server + SGI ProPack instalado, têm 64 CPU Quad Core Intel Xeon (com um total de 256 núcleos), com 512 GB de RAM e 32 TB de armazenagem, utilizando o SGI Infinite Storage NAS. Todos os computadores foram conectados através de uma rede Infinband. A execução paralela foi controlada pelo escalonador de tarefas PBS (Bayucan, Henderson, e J. P. Jones 2000). A infraestrutura paralela do Hydra orquestrou as múltiplas execuções paralelas do programa e foi desenvolvida utilizando MPJ (Carpenter et al. 2000).

Esses testes foram conduzidos sem qualquer tentativa de monitorar a carga da rede ou dos processadores antes ou durante a submissão. Os tempos relatados são os tempos médios observados. Não houve necessidade de alterar os programas sequenciais que as aplicações utilizam. Para evitar problemas de nomenclatura que podem acontecer com o uso das palavras processador e núcleo (núcleo de processador), no restante deste capítulo será utilizada a palavra núcleo para definir uma unidade computacional.

4.1 Caso 1: Mineração de Dados com Redes Neurais

Várias técnicas de computação podem ser usadas para a predição de uma série temporal, como redes neurais (Haykin 2008). Uma das vantagens das redes neurais é sua habilidade em identificar padrões que não são evidentes na série temporal. Porém, o desempenho de uma rede neural pode ser afetado por alguns parâmetros, como a estrutura da rede neural e a qualidade do processamento de dados. Esses parâmetros precisam ser explorados com o objetivo de se obter a rede neural ótima. Contudo, o teste manual de diferentes configurações de redes neurais para a seleção das redes mais eficientes é propenso a erros e consome tempo.

O objetivo do estudo de caso é utilizar o Hydra para sistematizar o processo de mineração de dados com a execução automática tanto a preparação dos dados quanto a exploração das diferentes estruturas de redes neurais para obter a rede neural ótima para a previsão.

As redes neurais usuais, como a *feedforward* (Haykin 2008), são utilizadas para reconhecimento de padrões estruturais. O processo de mineração de dados em séries temporais requer que o reconhecimento dos padrões das mudanças com o passar tempo, levando em conta na avaliação não apenas o valor atual, mas também seus predecessores. Então, dando um valor de entrada r_t que representa o valor atual, e seus n predecessores $r_{t-n},\ldots, r_{t-1}$, eles são tratados em uma memória de ordem n.

A Figura 8 mostra uma rede neural típica para uma série temporal (Haykin 2008). É uma rede neural do tipo *feedforward* com *back-propagation*, utilizando operadores *time lag* como entradas. Os valores das sinapses da rede são ajustados para minimizar o erro mínimo quadrático entre a saída da rede neural f(t), e o valor conhecido desejado w(t), que na série temporal é r_{t+1}.

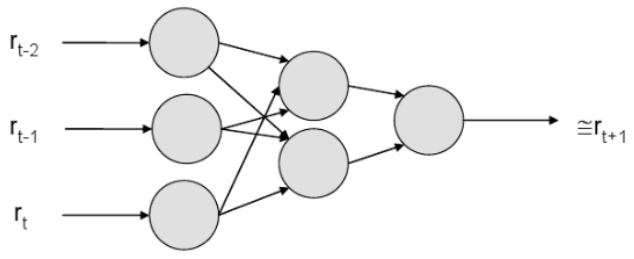

Figura 8. Uma Rede Neural Típica para Séries Temporais

O processo de mineração de dados (Han e Kamber 2006) típico, utilizando redes neurais, pode ser descrito em três passos:

- Configuração da rede neural (número de entradas, camadas escondidas, neurônios nas camadas escondidas);
- Análise estatística da série temporal;
- Preparação de um conjunto de dados para treinamento e validação cruzada.

Cada um desses passos pode ser feito de diferentes maneiras dependendo das características requeridas pelo problema específico que está sendo analisado (Eduardo Ogasawara, Leonardo Murta, et al. 2009).

4.1.1 Resultados Experimentais

O experimento feito segue a mesma paralelização do experimento executado em (Eduardo Ogasawara, Leonardo Murta, et al. 2009) este sem o auxílio dos componentes genéricos e fora do contexto de SGWfC. O workflow consiste de 540 redes neurais possíveis para serem treinadas para uma série temporal. Para cada série, a rede neural ótima deve ser encontrada. Para atingir este objetivo, a aplicação paralela treina todas as redes, calcula o erro médio quadrado de cada rede, e então une a identificação da rede neural com o seu erro médio quadrado nos resultados. O objetivo do experimento é encontrar a melhor configuração da rede neural.

O tempo de execução da aplicação de rede neural pode ser significativamente reduzido em um ambiente do HPC. A fonte de dados com 540 redes neurais foi utilizada como entrada para o workflow, que gera uma tarefa PBS, configurando parâmetros como número de núcleos, a linha de comando a ser executada e o tempo requisitado para a execução. O SGWfC transfere dos arquivos de dados, a aplicação paralela e a tarefa PBS para o HPC. Então a tarefa é executada, utilizando as configurações indicadas na tarefa PBS. A avaliação do treinamento é executada em apenas um núcleo do HPC.

Após todas as execuções do workflow, a média dos tempos de 3 execuções independentes, assim como e seu desvio padrão, para as execuções de 1 a 128 núcleos, para o experimento com 540 redes neurais e outro, simplificado, com 512 redes neurais, é apresentada na Tabela 2.

Tabela 2. Tempos de Execução do Experimento.

Número de Núcleos	Experimento com 540 Redes Neurais				Experimento com 512 Redes Neurais			
	Tempo do MTC Distribuído (min.)		Tempo Total do Workflow (min.)		Tempo do MTC Distribuído (min.)		Tempo Total do Workflow (min.)	
	μ	σ	μ	σ	μ	σ	μ	σ
1	524.91	13.97	527.44	14.02	462.65	0.61	464.89	0.53
2	264.62	1.31	267.19	1.28	237.36	1.47	239.91	1.92
4	137.18	2.71	139.80	2.81	123.27	2.52	125.46	2.53
8	69.16	1.34	71.68	1.30	61.67	0.93	63.83	0.93
16	36.07	0.21	38.59	0.30	33.08	0.19	35.28	0.17
32	19.65	0.18	22.22	0.19	18.02	0.05	20.19	0.05
64	10.77	0.09	13.24	0.11	9.72	0.06	12.05	0.07
128	6.79	0.32	9.29	0.28	5.91	0.01	8.15	0.05

Os resultados mostram que existe uma sobrecarga mínima na distribuição da aplicação para o cluster e na transferência dos dados da aplicação. Conforme o número de processadores aumenta, menor é o tempo de execução do workflow, conforme é mostrado na Figura 9.

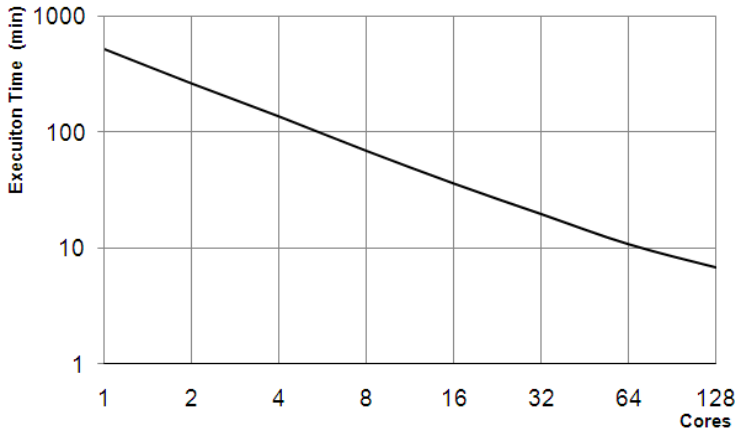

Figura 9. Tempo de Execução (log.) para 540 redes neurais

Similarmente, a aceleração e a eficiência (Gustafson 1988) do mesmo experimento são apresentadas nas Figuras 10 e 11, respectivamente. Essas figuras mostram a eficiência no uso dos recursos.

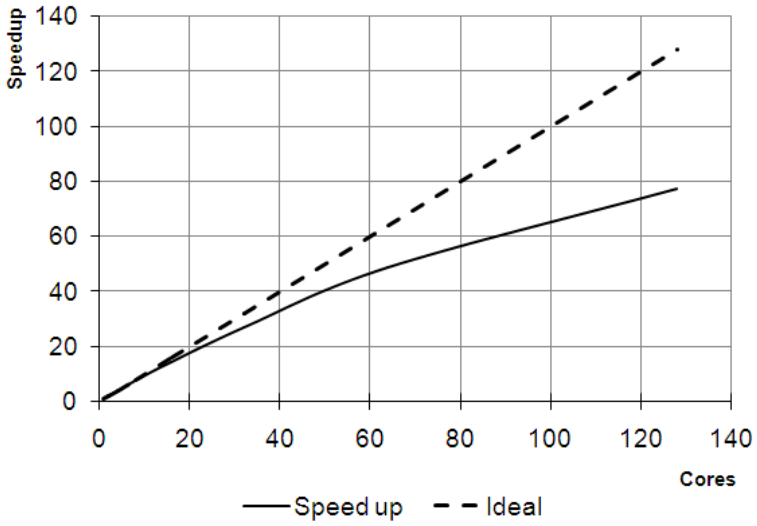

Figura 10. Aceleração do Hydra executando a aplicação com 540 redes neurais

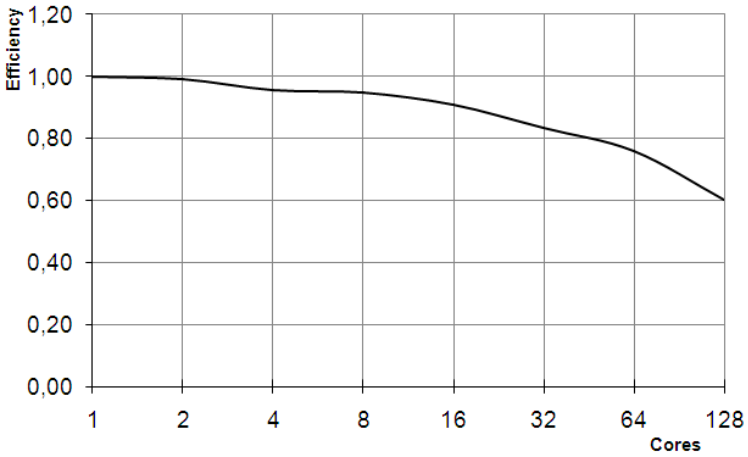

Figura 11. Eficiência do Hydra executando a aplicação com 540 redes neurais

Foi observada uma crescente diminuição no desempenho nas configurações a partir de 16 núcleos. Uma vez que o experimento foi realizado com 540 redes neurais, que não é múltiplo do número de núcleos utilizados, na última distribuição do MTC, o número de núcleos ociosos foi alto, gerando o desbalanceamento de carga. Por exemplo, quando treinamos as 540 redes neurais com 128 núcleos, existem cinco ciclos de distribuição. Os primeiros quatro ciclos de execução utilizaram os 128 núcleos disponíveis e treinaram 512 redes neurais. No último ciclo apenas 28 dos 128 núcleos são utilizados, deixando 100 núcleos ociosos no último ciclo de distribuição. Também é importante notar que o treinamento de uma rede neural é um processo estocástico, o que significa que cada rede neural pode ter um tempo diferente de treinamento.

Devido essas características de desbalanceamento, foi realizado um novo experimento, seguindo as mesmas características do primeiro, mas treinando apenas 512 redes neurais para uma determinada série temporal. O número de redes neurais foi escolhido por ser múltiplo do número de núcleos. A ideia principal é evitar núcleos ociosos enquanto executamos o experimento. A aceleração e a eficiência (Gustafson 1988) do mesmo experimento são apresentadas nas Figuras 12 e 13, respectivamente.

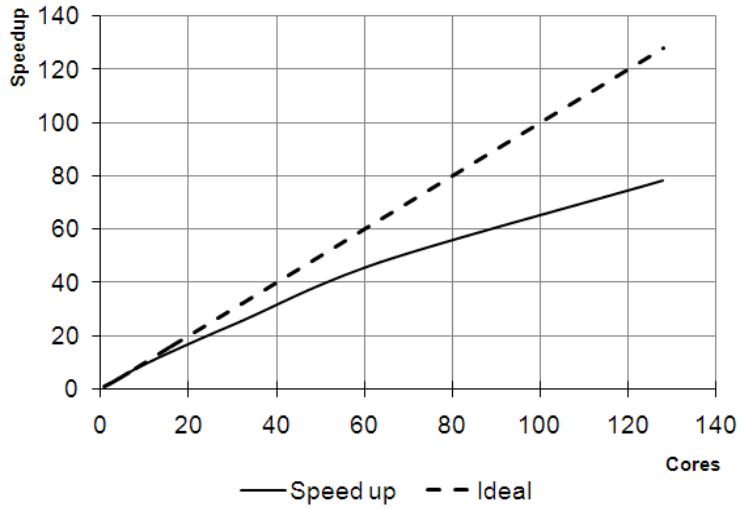

Figura 12. Aceleração do Hydra executando a aplicação com 512 redes neurais

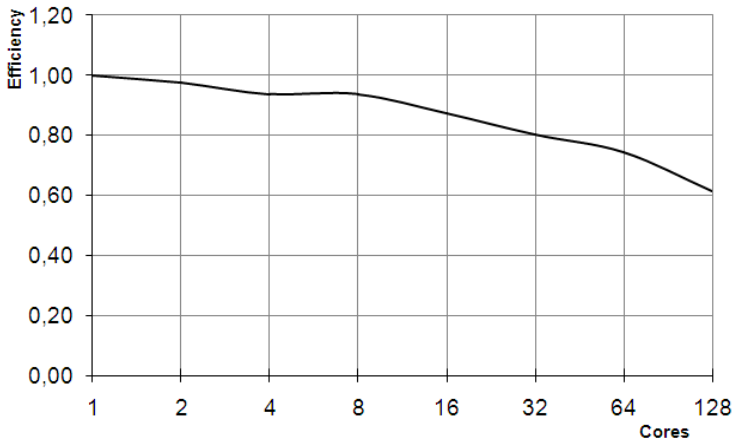

Figura 13. Eficiência do Hydra executando a aplicação com 512 redes neurais

Conforme apresentadas nas Figuras 12 e 13, a aceleração e a eficiência não foram melhoradas consideravelmente no novo experimento com 512 redes neurais treinadas. Assim, é possível concluir que os dados de entrada podem impactar consideravelmente no desempenho da aplicação, principalmente pelo fato que o tempo de treinamento de cada rede

neural varia, já que o treinamento é um processo estocástico. Este desvio de desempenho, chamado de *skew*, é relativamente comum em processamento paralelo.

Para resolver este problema, é necessário introduzir no Hydra algum tipo de balanceamento de carga adaptativo. Mas como o Hydra é um sistema genérico, seu desempenho tende a ser inferior ao de um sistema paralelo especializado, ainda mais considerando que o processamento paralelo exige uma sintonia fina para cada algoritmo de paralelização utilizado.

4.2 Caso 2: Dinâmica de Fluidos com EdgeCFD

Na área de Dinâmica de Fluidos Computacional - *Computational Fluid Dynamics* (CFD) – engenheiros e cientistas frequentemente resolvem o mesmo problema várias vezes, estudando as influências na característica do fluxo a partir de vários parâmetros, como viscosidade e o número de Reynolds. Tradicionalmente, CFD é uma das áreas científicas com maiores demandas computacionais, e está sempre utilizando o melhor das tecnologias de supercomputadores disponível (Bader 2008). Recentemente, houve um aumento no interesse em estudos de CFD complexos, que envolvem verificação, validação e quantificação incerteza a qual geralmente requer uma quantidade massiva de simulações e gera uma quantidade de dados gigantesca. Este cenário aumenta a complexidade na gerência de suas análises. Escalonadores simples, como em sistemas MTC padrão, não conseguem lidar com tanta complexidade. Workflows científicos são soluções promissoras para dar suporte a tais demandas exploratórias além de prover ambientes com menos propensão a erros. Particularmente, uma vez que a quantidade de dados gerados aumenta drasticamente, a proveniência passa a ter um papel fundamental neste cenário, e ela é suportada pelos workflows científicos.

O Hydra foi utilizado para buscar um parâmetro ótimo (Numero de Reynolds) em uma aplicação CFD. Os objetivos principais deste estudo de caso são prover suporte à coleta dos dados de proveniência; paralelizar a execução do workflow correspondente à exploração de números de Reynolds variando de 100 a 1000; e fornecer ao cientista uma maneira eficiente de visualizar os resultados.

Nesta seção será descrito um workflow de CFD, utilizando o EdgeCFD, uma ferramenta de resolução de fluxo incompressível com elementos finitos (Elias e Coutinho

2007) (Elias, Martins, e Coutinho 2005). Seguindo uma ferramenta de simulação tradicional de CAE (Computer Aided Engineering), o workflow EdgeCFD pode ser resumido em quatro passos.

No passo de modelagem, são criados modelos computacionais, descrevendo a geometria real, e discretizados em pedaços menores (elementos/células), em que equações que descrevem o fenômeno são resolvidas pelo método correspondente nos passos seguintes. Este passo geralmente requer intervenção humana.

No estágio de pré-processamento, parâmetros físicos e da solução, como condições iniciais e limites, tolerâncias no método da solução, tempo de simulação, propriedades do material, etc. são estabelecidos no modelo computacional. Todos esses parâmetros são candidatos em uma exploração por varredura de parâmetros e podem ser armazenados com informações de proveniência.

O passo de resolução é responsável por processar as equações que modelam o problema físico específico. Este passo geralmente é o mais demorado em um workflow de simulação.

Finalmente, no passo de pós-processamento, os resultados produzidos no passo anterior são interpretados com a ajuda de ferramentas de visualização que traduzem os números crus (geralmente em formato binário) em gráficos, imagens (por exemplo, a Figura 15) ou vídeos. No caso que está sendo testado, apenas o segundo, o terceiro e o quarto passos, que são os que consomem mais tempo e são automatizáveis, foram considerados.

O workflow abstrato (de alto nível) do workflow do EdgeCFD é mostrado na Figura 14. Ele foi modelado como um Diagrama de Atividade UML (Eduardo Ogasawara, Daniel Oliveira, Fernando Chirigati, Barbosa, Elias, Vanessa Braganholo, et al. 2009). Na Figura 14, as elipses denotam atividades, e retângulos denotam dados de entrada/saída. Note que o passo de resolução é representado pela atividade “EdgeCFDSolver e Aplicações de Varredura”. Esta atividade é uma atividade composta (com o estereótipo <<sub-workflow>>), uma vez que necessita de scripts de controle para a execução. Esses scripts foram inicialmente modelados como sub-atividade do passo de resolução. Existe também o estereótipo <<Varredura>>, que denota que a atividade pode realizar uma varredura de parâmetros em paralelo.

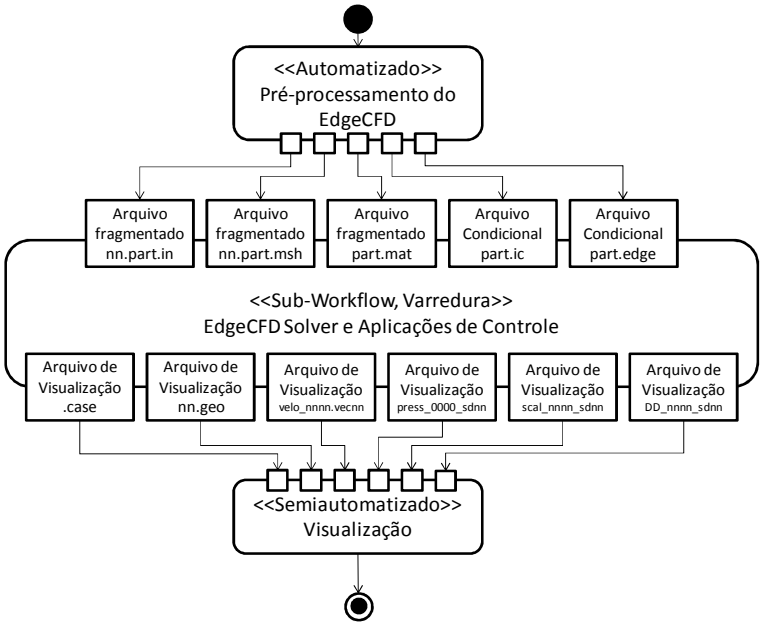

Figura 14. Workflow Científico do EdgeCFD

Um problema de escoamento sobre uma cavidade quadrada, um benchmark padrão para CFD, foi utilizado no estudo de caso. Neste problema, um fluído confinado em uma atividade cúbica, inicialmente em repouso, começa a escorrer após a retirada de uma “tampa”, fazendo com que se formem redemoinhos, como mostrados na Figura 15. É importante reforçar que a ferramenta EdgeCFD é um exemplo clássico de ferramenta MTC que pode ser executada de modo serial ou paralelo, utilizando vários núcleos que podem ser empregados para acelerar o processo de solução para cada número de Reynolds. Neste caso, contudo, o resultado mais importante é demonstrar a capacidade do Hydra de paralelizar automaticamente o workflow, agendando e provendo informações de proveniência em um caso de varredura de parâmetros, então, foram apenas consideradas configurações seriais das atividades do EdgeCFD.

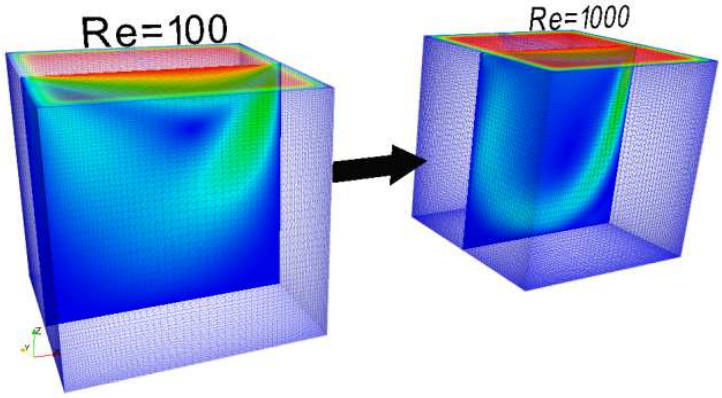

Figura 15. Resultados da Simulação para o primeiro e o último número de Reynolds

4.2.1 O Workflow Científico EdgeCFD

Para executar este estudo de caso, o workflow do EdgeCFD foi implementado utilizando o SGWfC VisTrails (Callahan et al. 2006). Esta implementação é definida como o workflow "concreto", para diferenciar do workflow abstrato que foi mostrado na Figura 14. A Figura 16 mostra os módulos do workflow no VisTrails.

Para o passo de visualização, foi utilizado o módulo *spreadsheet* do VisTrails. É por isso que o workflow concreto tem muito mais atividades do que o abstrato que foi apresentado na Figura 14. A atividade de visualização (terceira atividade na Figura 14), neste caso, é uma atividade complexa e requer muita conversão de dados. Então, a conversão foi implementada por todas as atividades que foram colocadas dentro da caixa pontilhada na Figura 16. Existem outras opções para a implementação do passo de visualização, como o Paraview (Paraview 2009) ou outro software de visualização. Contudo, isto não influencia o estudo de caso, uma vez que esta atividade não será paralelizada neste estudo.

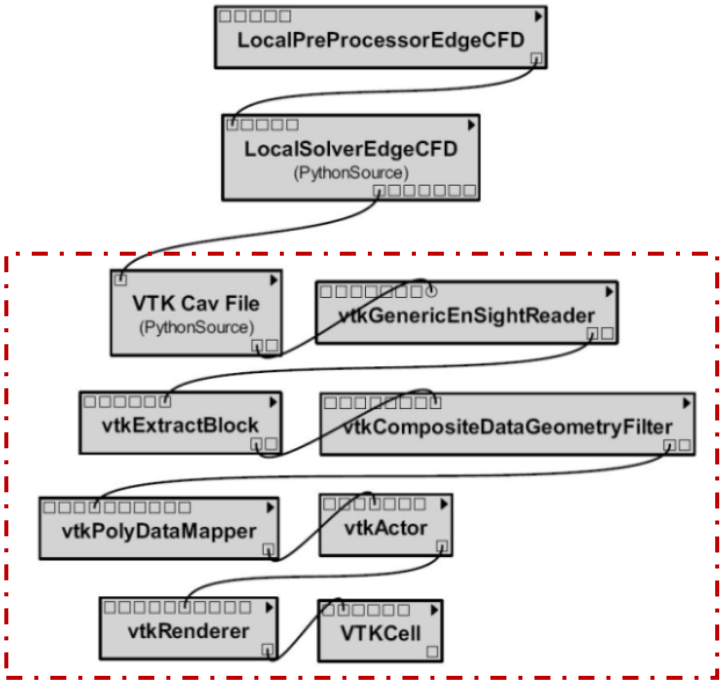

Figura 16. Workflow EdgeCFD no VisTrails

4.2.1.1 Workflow EdgeCFD no Hydra

Para modelar o workflow do EdgeCFD utilizando a arquitetura Hydra, temos que configurar os componentes da camada cliente do Hydra para a execução do VisTrails. A Figura 17 mostra o workflow EdgeCFD com os componentes do Hydra.

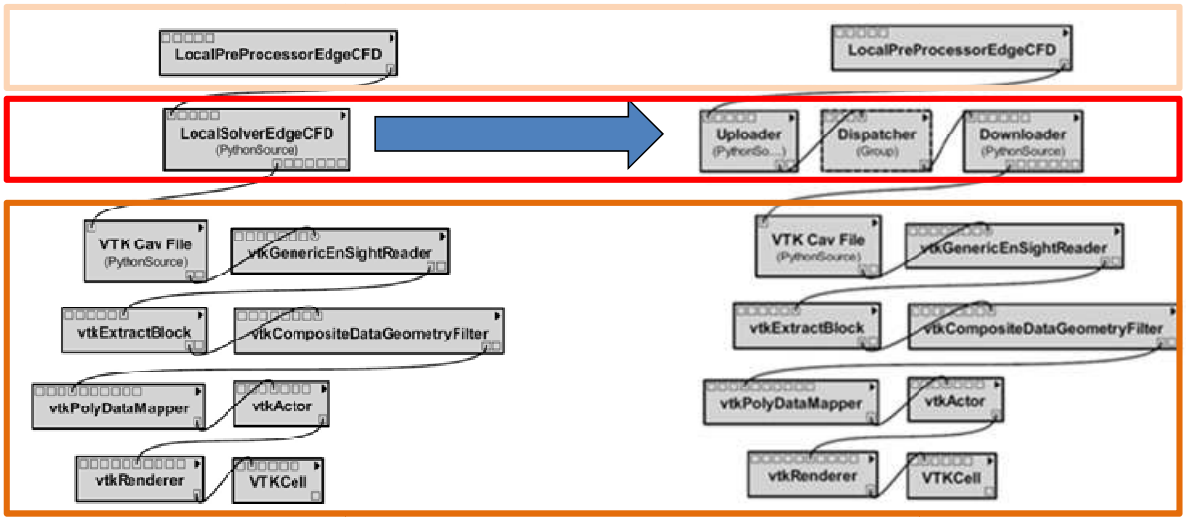

Figura 17. Workflow EdgeCFD sem (esq.) e com (dir.) os componentes do Hydra

Comparando o workflow da esquerda com da direita, podemos perceber que a atividade *LocalSolverEdgeCFD* foi substituída por três componentes do Hydra (*Uploader*, *Dispatcher*, e *Downloader*). As entradas e saídas das atividades foram preservadas, isto é, a entrada da atividade *LocalSolverEdgeCFD* se torna a entrada da atividade *Uploader* no workflow com os componentes do Hydra. Similarmente, a saída da atividade *LocalSolverEdgeCFD* corresponde à saída da atividade *Downloader*. Os componentes da execução paralela foram configurados diretamente no ambiente distribuído, utilizando o *Hydra Setup*, conforme apresentado na Figura 18.

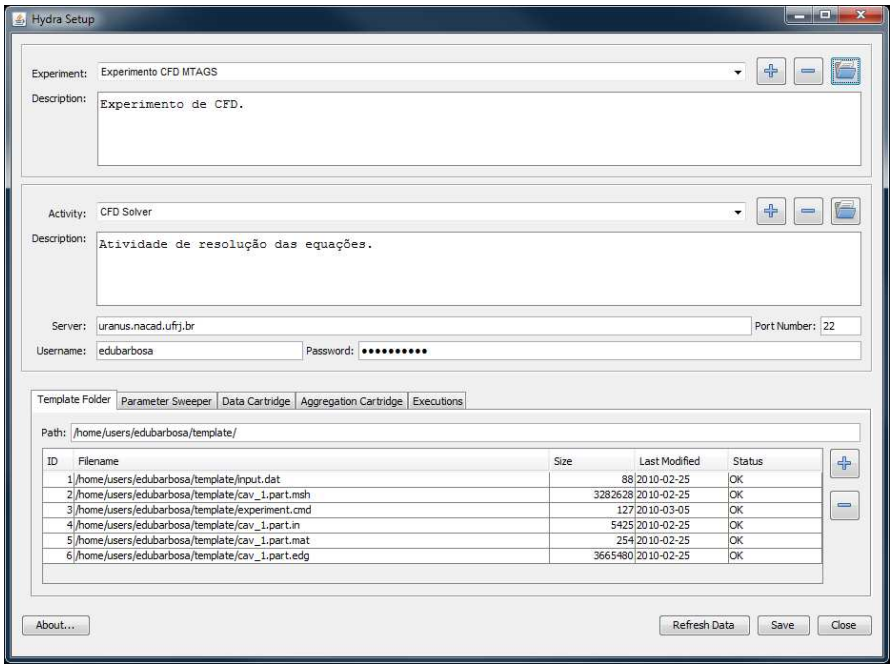

Figura 18. Configuração de distribuição do EdgeCFD

O experimento consiste em um workflow local com a distribuição da atividade de resolução (EdgeCFD). O motivo da distribuição é que este workflow exploratório usa uma mesma estrutura de workflow, porém varia um parâmetro específico em cada execução, neste caso o número de Reynolds. A Figura 19 mostra o arquivo instrumentado para a varredura de parâmetros cav_1.part.mat, utilizando *tags* similares a de JSP.

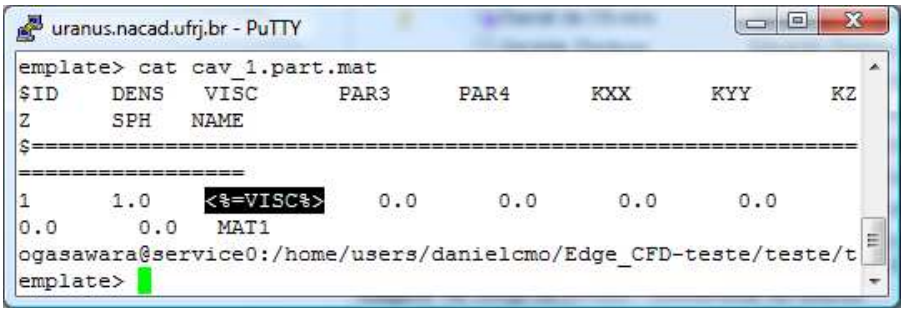

Figura 19. Arquivo instrumentado para a varredura de parâmetros

Este estudo de caso evidenciou as cinco principais contribuições do Hydra. Inicialmente, a especificação da varredura de parâmetros (Figura 14) indicou para o Hydra o potencial para (i) paralelismo de parâmetro implícito. O cientista é guiado a incluir os componentes necessários (ii) para modelar as atividades MTC conforme a Figura 16. Então, (iii) a submissão para o ambiente MTC é feita pelo *Dispatcher* (Figura 17) e (iv) pode ser configurada conforme mostrado na Figura 18. Finalmente, (v) a coleta de proveniência para a varredura de parâmetros mostra que, utilizando o Hydra, se torna possível registrar a proveniência prospectiva para a varredura de parâmetros do experimento exploratório. Adicionalmente, isto poderia ser uma atividade laboriosa e sujeita a erro humano, caso fosse feita manualmente pelos cientistas. Na infraestrutura do Hydra, a execução exploratória pode ser executada no ambiente MTC, considerando que uma varredura de parâmetros gera tantas configurações quanto necessárias para a execução da resolução.

4.2.1.2 Resultados Experimentais

O workflow EdgeCFD foi invocado pelo VisTrails de um desktop e distribuiu a atividade Solver, utilizando o paradigma MTC em um ambiente distribuído, utilizando a infraestrutura do Hydra para a varredura de parâmetros paralela. A execução do workflow do EdgeCFD utilizando o Hydra foi feita em um HPC.

Através deste experimento, foi possível observar que o Hydra provê uma abordagem sistemática para o suporte da exploração em varredura de parâmetros. Inicialmente, o *Hydra Setup* foi utilizado para definir a execução paralela para o workflow apresentado na Figura 18. Esta configuração do experimento foi armazenada no repositório de parâmetros. Este repositório de parâmetros representa a proveniência prospectiva da varredura de parâmetros paralela. Os arquivos de configuração ou parâmetros de linha de comando foram instrumentados, para cada programa, utilizando *tags* que são simples de entender, conforme mostrado na Figura 19, especificamente no texto selecionado. Então, com os arquivos instrumentados e com a definição do catálogo disponível, foi necessário apenas fazer pequenas mudanças no workflow científico para suportar o paralelismo desejado, conforme apresentado na Figura 17.

Após modificar o workflow, a execução é iniciada. As atividades de pré-processamento são executadas no VisTrails, e então os arquivos gerados requeridos pelo *EdgeCFDSolver* são transferidos para o ambiente MTC. Os componentes da camada cliente do Hydra invocam o Hydra MTC para paralelizar as diversas atividades de resolução. O Hydra MTC automaticamente manipula e cria pastas de trabalho isoladas que incluem todos os arquivos necessários para cada atividade distribuída executada no ambiente MTC. No caso de varredura de parâmetros, uma pasta de trabalho individual é criada para cada combinação de conjunto de parâmetros. O mecanismo de coleta de proveniência retrospectiva registra cada pasta de trabalho e a associa a um conjunto específico de parâmetros usados na exploração por varredura de parâmetros. Uma vez que todas as pastas de trabalho foram criadas, o Hydra MTC dispara a execução no escalonador configurado. Cada execução individual da atividade (tarefa) é monitorada. A proveniência retrospectiva da execução paralela do Hydra MTC é coletada. Se um erro ocorrer, ele é armazenado no repositório de proveniência do Hydra. A informação de cada tarefa da atividade também é armazenada no repositório de proveniência. Uma vez que a execução paralela termina, o controle retorna ao VisTrails.

Neste experimento, o subcomponente Analisador também foi avaliado. Um cartucho de agregação de vídeo foi implementado para ser invocado pelo subcomponente Analisador. Este cartucho gera um vídeo AVI que concatena todas as imagens produzidas por cada execução de cada configuração da varredura de parâmetros do workflow. Quando concatenadas, essas imagens criam um vídeo que mostra a imagem e os parâmetros utilizados

na sua geração. É uma maneira simples para os cientistas analisarem todos os resultados de uma exploração de parâmetros. A Figura 20 apresenta, como exemplo, um quadro do vídeo gerado pelo cartucho de agregação.

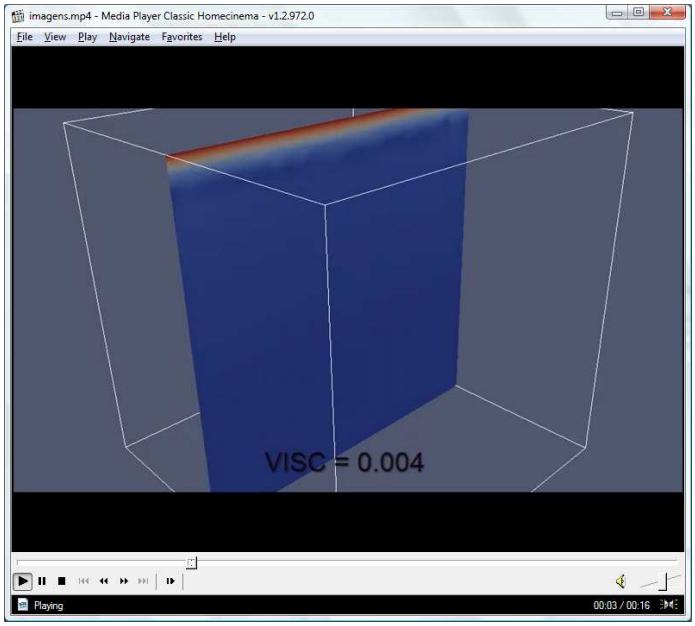

Figura 20. Vídeo gerado pelo subcomponente Analisador

A Tabela 3 mostra os resultados de uma execução do Hydra MTC obtida no repositório de proveniência da execução, utilizando 4 nós com 8 núcleos em cada, totalizando 32 núcleos de processamento. A execução da varredura dos números de Reynolds de 100 a 1000 se deu em 128 intervalos. Cada simulação foi executada até 1000 passos de tempo, o que é típico para o desenvolvimento das características de escoamento nesta faixa de parâmetros. Durante a execução distribuída, 128 parâmetros foram explorados, um para cada número de Reynolds. Esses parâmetros representam o número de tarefas da atividade paralelizada. Neste experimento, um arquivo que contém uma malha precisa ser transferido do SGWfC local para o ambiente HPC. É a entrada da atividade distribuída. Para transferir o arquivo, o tempo de envio do arquivo foi de 0,8 minutos. Após a execução da atividade distribuída, os resultados (arquivos CAV) em cada pasta de trabalho precisam ser copiados de volta, e o tempo de recebimento dos arquivos foi de 11,6 minutos. Essas transferências têm um tempo fixo e representa todos os dados transferidos entre o SGWfC (cliente) e o ambiente HPC (servidor). O tempo de total de execução é o tempo para executar todas as tarefas (1.683,6 minutos) incluindo todo o tempo de envio, preparação, execução e coleta de proveniência de todas as tarefas da atividade. Isso nos leva a uma aceleração de 31,

considerando os 32 núcleos disponíveis. A informação armazenada pela proveniência retrospectiva é importante para avaliar e reproduzir a simulação. Importantes dados de proveniência podem ser perdidos se uma metodologia controlada sistematicamente como o Hydra não for utilizada.

Tabela 3. Resultados de uma execução do Hydra MTC utilizando 32 núcleos, obtido pelo repositório de proveniência

Número de Atividades (Parâmetros Explorados)	128
Data/hora de Início	31/08/2009 - 01:40
Tempo de Envio	0,8 minutos
Tempo de Execução (Execução + Coleta)	1.671,2 minutos
Tempo de Recepção	11,6 minutos
Tempo Total de Execução	1.683,6 minutos
Aceleração	31,0
Número de Erros	0

Dos dados de proveniência retrospectiva obtidos do catálogo de execução, é possível construir o histograma de todos os tempos de execução de tarefas, apresentados na Figura 21.

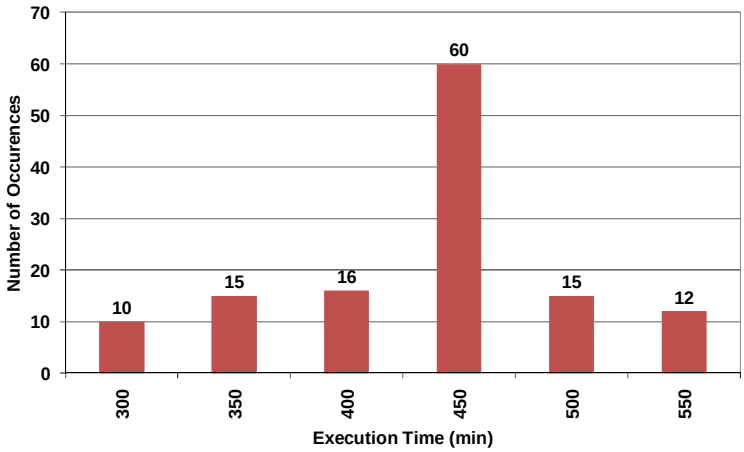

Figura 21. Distribuição do Tempo de Execução das Tarefas da Atividade

Os dados contidos no catálogo de execuções tornam possível calcular o tempo médio (408,3 minutos) e o desvio padrão (61,8 minutos) de cada tarefa. Também é possível estimar que o tempo equivalente para a execução sequencial de todas as tarefas seria de 52.256 minutos. Além disso, os cientistas podem usar o catálogo para verificar qual parâmetro gerou qual resultado desejado, entre outras consultas.

Esta informação de proveniência é fundamental para a análise do experimento científico. Baseados nos dados de proveniência, cientistas podem usar o esquema de proveniência para verificar qual parâmetro gerou o resultado desejado ou submeter consultas do tipo: “Quais são as pastas de trabalho das tarefas que foram associadas com o número de Reynolds entre 150 e 250?”. Esta consulta pode ser traduzida como a consulta SQL a ser feita no repositório de proveniência do Hydra, conforme a Figura 22:

```
SELECT t.workspace
FROM Task t, SweepedParameter sp,
   Parameter p, DistribuitedActivityExecution dae
WHERE p.ParameterID = sp.ParameterID AND sp.TaskID = t.TaskID
   AND dae.DistribuitedActivityExecutionID = t.DistribuitedActivityExecutionID
   AND dae.DistribuitedActivityExecutionID = 4
   AND p.Name = 'Reynolds Number'
   AND  sp.NumericValue > 200 AND sp.NumericValue < 210;
```

	workspace character varying
1	/home/users/edubarbosa/MTAGS/exp/VISC_111
2	/home/users/edubarbosa/MTAGS/exp/VISC_110
3	/home/users/edubarbosa/MTAGS/exp/VISC_109
4	/home/users/edubarbosa/MTAGS/exp/VISC_112
5	/home/users/edubarbosa/MTAGS/exp/VISC_108
6	/home/users/edubarbosa/MTAGS/exp/VISC_113
7	/home/users/edubarbosa/MTAGS/exp/VISC_114

Figura 22. Exemplo de consulta e seu resultado

Para avaliar a sobrecarga do Hydra utilizando diferentes números de nós, foi executado outro experimento utilizando apenas 5 passos em cada simulação e variando o número de Reynolds de 100 a 1000 utilizando 256 intervalos. O que leva a 256 pequenas tarefas. Essas tarefas não têm um significado físico, mas o objetivo é avaliar o Hydra em relação ao número de nós. O tempo de execução completo do workflow EdgeCFD foi significantemente reduzido conforme o número de nós envolvidos na atividade distribuída aumentou. A sobrecarga de transferir os dados do SGWfC local para o ambiente MTC e vice-versa foram aceitáveis quando comparamos com o grau de paralelismo obtido com a varredura de parâmetros nas atividades de resolução, e assim reduzindo o tempo total de execução.

O tempo de execução do Hydra MTC e de todo o processo de distribuição do workflow (*Uploader*, *Dispatcher* e *Downloader*) é apresentada na linha “Tempo Total do Workflow” da Figura 23, onde podemos observar a sobrecarga da transferência de dados. Desta maneira, a linha tracejada representa o tempo total de execução da atividade distribuída, considerando o tempo de envio e recebimento de dados do ambiente HPC. Por outro lado, a

linha “Hydra MTC” representa apenas o tempo de execução da atividade distribuída sem levar em conta o tempo de transferência. Conforme mencionado antes, o tempo de envio e recebimento é fixo, uma vez que os dados de entrada e saída são fixos.

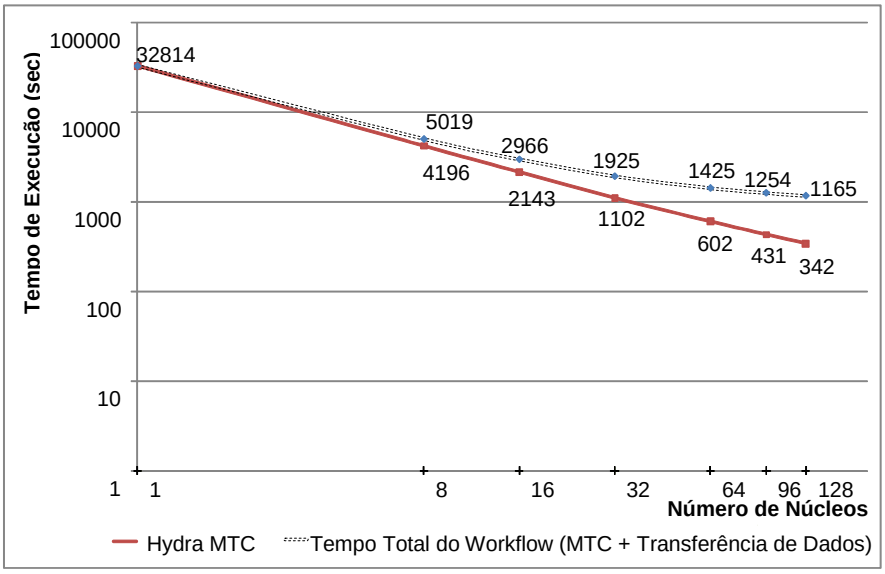

Figura 23. Tempo de Execução do Workflow

Podemos observar na Figura 23 que conforme aumenta do número de núcleos, o tempo total de execução das tarefas da atividade decresce como o esperado. Contudo, os tempos de transferências são os mesmos, aumentando o impacto do tempo de transferência no tempo total de execução, conforme o número de núcleos utilizados aumenta. Foi observado que os benefícios da paralelização são claros, mas dependendo dos tempos de envio dos dados de entrada e do recebimento dos dados de saída, adicionar mais núcleos para a execução não traz muitos benefícios, principalmente se o número de tarefas se torna muito próximo do número de núcleos. Frequentemente, o conjunto de dados de entrada será gerado por uma atividade anterior no MTC, então o tempo de transferência poderá ser evitado.

Os objetivos do estudo de caso foram atingidos, uma vez que foi provido suporte à coleta dos dados de proveniência prospectiva e retrospectiva; a execução do workflow correspondente à exploração de números de Reynolds foi paralelizada, em diversas configurações, e seus resultados foram comparados; e forneceu ao cientista uma maneira eficiente de visualizar os resultados, através de um vídeo que corresponde a com imagens sequenciais de cada configuração.

4.2.2 Análise de Sensibilidade em Simulações de Grande Turbulência

Esta seção explica em detalhes o problema da Quantificação de Incerteza - *Uncertainty Quantification* (UQ) - modelado pelo workflow científico de Simulações de Grande Turbulência - *Large Eddy Simulations* (LES). O LES é uma técnica popular para simular fluxos turbulentos, sendo utilizado em várias aplicações que tentam combinar a precisão com os custos computacionais.

O modelo Smagorinsky é o mais popular da viscosidade turbulenta no LES. O Hydra foi utilizado em uma análise de sensibilidade para selecionar o parâmetro modelo de Smagorinsky, sendo possível efetuar uma análise de sensibilidade do coeficiente de Smagorinsky (Lucor, Meyers, e Sagaut 2007). Com o Hydra, foi possível explorar esta simulação de grande potencial paralelo intrínseco e executar 64 tarefas seriais simultaneamente.

O caso foi utilizado para validar a utilização do cartucho de agregação com coleta de dados de saída (output). O Hydra foi utilizado para sistematizar a análise de sensibilidade para o parâmetro do modelo de Smagorinsky. Em uma varredura de parâmetros, a análise padrão do EdgeCFD foi executada em conjunto com um utilitário de extração de dados de energia cinética. Esta ferramenta gerou um arquivo adicional, chamado energy.txt, em cada pasta de trabalho. Um exemplo de arquivo energy.txt é mostrado na Figura 24. Os dados relevantes para a análise estão marcados.

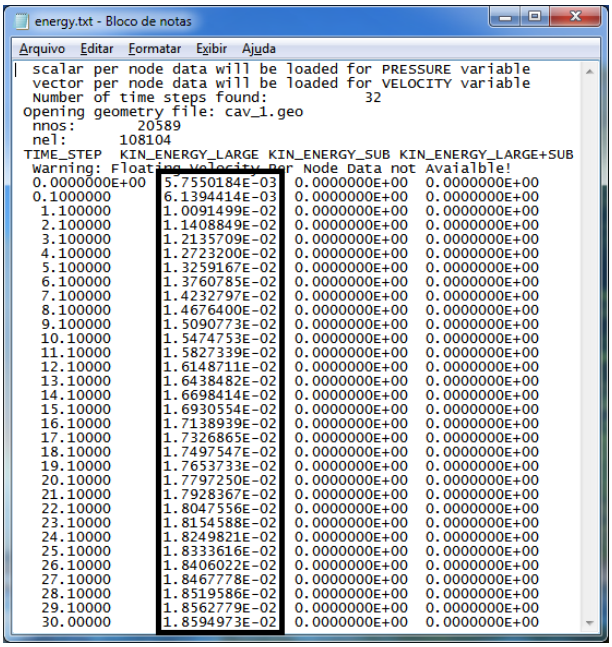

Figura 24. Exemplo de arquivo energy.txt

Após a execução paralela das tarefas, o Hydra utilizou o cartucho de agregação para gerar os arquivos hydraoutput.txt, em cada pasta de trabalho. Os arquivos hydraoutput.txt são utilizados para passar variáveis do cartucho de agregação para o Hydra MTC. No caso específico o hydraoutput.txt contém quatro variáveis: média da energia cinética, a variância da energia cinética, o número de entradas descartadas como fase transiente, e o número de entradas utilizadas na análise. Um exemplo de hydraoutput.txt é mostrado na Figura 25. Neste caso a fase transiente é de 30%, ou seja, os primeiros 30% do vetor de dados marcado na Figura 24 são descartados.

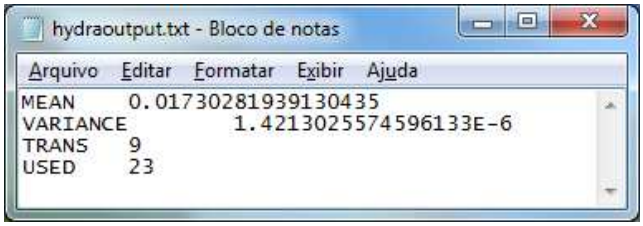

Figura 25. Exemplo de arquivo hydraoutput.txt

Terminada a fase de execução da agregação, o Hydra MTC abriu cada hydraoutput.txt gerado, e fez a leitura de seus dados, que foram incluídos no repositório de proveniência, ficando assim disponíveis para o cientista. Neste ponto, o arquivo hydraoutput.txt já cumpriu a sua função de ponte de dados entre o cartucho de agregação e a base de proveniência, então ele pode ser inclusive excluído. Em uma consulta personalizada ao repositório de proveniência, os dados do arquivo hydraoutput.txt do exemplo da Figura 25, podem ser vistos na Figura 26.

```
SELECT t.workspace, o.name,numericvalue as value
  FROM outputvalue ov, output o, task t
  WHERE ov.outputID = o.outputID
  AND t.taskID = ov.taskID
  AND t.workspace = '/home/users/edubarbosa/MTAGS/exp/task_1'
  AND t.distribuitedactivityexecutionID = 37;
```

	workspace character varying	name character varying	value double precision
1	/home/users/edubarbosa/MTAGS/exp/task_1	MEAN	0.0173028193913044
2	/home/users/edubarbosa/MTAGS/exp/task_1	VARIANCE	1.42130255745961e-006
3	/home/users/edubarbosa/MTAGS/exp/task_1	TRANS	9
4	/home/users/edubarbosa/MTAGS/exp/task_1	USED	23

Figura 26. Consulta e dados do arquivo hydraoutput.txt no Repositório de Proveniência

Com os dados armazenados no repositório de proveniência, o cientista pode fazer a análise de sensibilidade a partir de consultas ao banco de dados. O único senão é a

implementação do cartucho de agregação. Neste caso, o cartucho de agregação é um programa em Java com menos de 200 linhas. Para chegar perto do nível de automação de coleta de dados que o Hydra oferece, um cientista provavelmente escreveria *scripts* maiores que este programa. E não teria funcionalidades fundamentais para um experimento complexo, como o armazenamento de proveniência.

O Hydra foi capaz de prover uma varredura de parâmetros utilizando uma metodologia sistemática, uma vez que diminui as chances de erro durante a decomposição do workflow paralelo. Esta varredura corresponde à análise de sensibilidade de viscosidades turbulentas em LES. Os resultados da análise de sensibilidade em si estão fora do escopo desta dissertação. Os dados de proveniência relacionados à execução paralela foram coletados e disponibilizados para a análise do usuário. Também foi mostrado que a sensibilidade da solução LES para coeficiente de Smagorinsky incerto pode ser investigada com sucesso em um ambiente HPC utilizando SGWfC.

Capítulo 5 – Conclusões

Experimentos científicos lidam com uma grande quantidade de dados a ser processada e analisada. Esses experimentos são modelados como workflows científicos, que são uma abstração do processamento como um fluxo contínuo de atividades. Os workflows podem se beneficiar do grande número de processadores, quando executados em um HPC, usando Computação de Torrente de Tarefas, uma vez que algumas dessas atividades necessitam de alto desempenho e consomem um grande tempo de processamento.

O Hydra é formado por um conjunto de componentes, que são distribuídos entre o ambiente local e o ambiente MTC, para permitir a execução de atividades em paralelo. Foram implementados cinco componentes, dois de transferência de dados, dois para coordenar a execução paralela, e um para a configuração do sistema. O Hydra visa reduzir a complexidade envolvida na modelagem e gerência de atividades paralelizáveis em um workflow científico em larga escala, sistematizando a paralelização da atividade, sem a necessidade de programação adicional e, principalmente, fornecendo coleta de dados de proveniência, não apenas da execução realizada em paralelo, quanto do experimento como um todo.

O desenvolvimento do Hydra mostrou que é possível prover o paralelismo de dados a workflows de modo independente do SGWfC, além de coletar proveniência prospectiva e retrospectiva. Os resultados mostram um ganho de paralelismo com eficiência próxima de entre 70% e 80%, em média. O potencial de consultas aos dados de proveniência da execução paralela mostra que o cientista pôde realizar uma análise sobre os dados de agregação, em consultas de alto nível, conforme o exemplo dado na Figura 26.

A maior contribuição desta dissertação é o próprio Hydra, que procura automatizar parte significativa do trabalho do cientista na hora de modelar e executar atividades de um workflow científico em um ambiente HPC, reduzindo a probabilidade de erros e aumentando sua produtividade. O Hydra permitiu a coleta de proveniência retrospectiva e o cadastro de proveniência prospectiva, permitindo ao cientista mapear uma execução paralela de atividades com o experimento científico em si.

Os componentes do Hydra foram implementados e integrados ao SGWfC VisTrails. O Hydra foi avaliado em dois casos de natureza distintas: um workflow de mineração de dados, utilizando redes neurais; e um workflow de dinâmica de fluídos da mecânica computacional.

Foi observado que os benefícios da paralelização utilizando o Hydra são claros, mas o número de tarefas muito próximo do número de núcleos utilizados pode diminuir a eficiência da paralelização, dependendo da quantidade de dados movimentados. Além disso, o Hydra foi capaz de prover uma varredura de parâmetros utilizando uma metodologia sistemática, uma vez que diminui as chances de erro durante a decomposição do workflow paralelo. Os dados de proveniência relacionados à execução paralela foram coletados e disponibilizados para a análise do usuário.

Esses componentes permitem que os cientistas paralelizem atividades a partir de seu SGWfC favorito e o Hydra mantém a gerência da semântica da execução do workflow. As execuções do experimento mostraram uma ótima relação custo/benefício já que a sobrecarga imposta no desempenho é baixa. Outra vantagem do Hydra é a sua flexibilidade em ser utilizado a partir de SGWfC fortes na construção do workflow concreto e coleta de sua proveniência, como o VisTrails. Os componentes do Hydra foram adicionados como componentes do VisTrails e então utilizados normalmente dentro do workflow. A arquitetura do Hydra permite que essa incorporação de seus componentes possa ser feita por qualquer SGWfC semelhante.

Com o Hydra, no caso de uma falha na execução remota no HPC, os seus componentes são capazes de notificar o erro ao SGWfC, o que melhora o controle do workflow ao contrário de outras soluções onde o SGWfC não tem ideia do que está ocorrendo no HPC durante a execução remota e podem ficar esperando eternamente. Consequentemente, esse “isolamento” faz com que a coleta e registro dos dados de proveniência fiquem interrompidos com a transferência do controle da execução remota no HPC. A coleta da proveniência desde a definição do experimento até a execução paralela das atividades é um diferencial do Hydra com relação aos seus concorrentes, oferecendo consultas de alto nível para o cientista.

Os componentes do Hydra podem ser integrados SGWfC que permitem a chamada de componentes externos, permitindo que seja efetuada a coleta de sua proveniência. Um cientista pode paralelizar uma atividade de um workflow substituindo a atividade do workflow pelos componentes do Hydra.

Com a implementação do Hydra, foi possível executar varreduras de parâmetros paralelas, utilizando uma metodologia sistemática, diminuindo as chances de erro durante a decomposição do workflow e de maneira transparente ao SGWfC. Além disso, os dados de proveniência relacionados à execução paralela foram coletados e disponibilizados para a análise do usuário. O Hydra atingiu aceleração de desempenho próxima a linear, e cumpre a premissa de fazer a ponte entre o SGWfC e o ambiente MTC de maneira transparente.

Uma limitação do Hydra é que a aceleração da aplicação não é ótima, já que em aplicações paralelas, a sintonia fina do algoritmo é importante para seu desempenho. Atualmente sua aceleração fica em entre 70% e 80%. Como o objetivo do Hydra é fornecer uma solução genérica, esta limitação não chega a ser um grande problema. Cabe ao cientista ponderar se deve recorrer ao auxilio de um especialista em programação paralela para lhe ajudar, obtendo “sozinho” uma aceleração um pouco melhor que a execução com o Hydra ou utilizar o Hydra e obter outros benefícios, como coleta de proveniência.

O Hydra continuará sendo estendido e utilizado em simulações futuras de diversas áreas. Como existe uma parceria entre pesquisadores da COPPE com pesquisadores da Universidade de Utah, fica a possibilidade do Hydra amadurecer e, um dia, ser incorporado ao VisTrails.

No futuro, várias pesquisas do NACAD poderão usufruir do Hydra. Outras dissertações de mestrado podem utilizar o Hydra seja para executar suas aplicações, seja para estender a sua arquitetura ou mesmo melhorar a eficiência dos componentes. Como lacunas a serem preenchidas, está a implementação do mecanismo de fragmentação de dados, além de uma interface para a criação e cadastro de consultas personalizadas à proveniência, específicas para cada experimento. Também é importante gerar um pacote de fácil instalação do Hydra em uma máquina com o VisTrails. Um passo importante também será avaliar a orquestração de uma execução paralela utilizando outros SGWfC, como o Kepler.

Referências Bibliográficas

Aalst, W. M. P. Van Der, A. H. M. Ter Hofstede, B. Kiepuszewski, e A. P. Barros. 2003. Workflow Patterns. *Distrib. Parallel Databases* 14, n. 1: 5-51.

Altintas, I., C. Berkley, E. Jaeger, M. Jones, B. Ludascher, e S. Mock. 2004. Kepler: an extensible system for design and execution of scientific workflows. In *SSDBM*, 423-424. Greece. doi:10.1109/SSDM.2004.1311241.

Bader, D. A. 2008. *Petascale computing: algorithms and applications*. Chapman & Hall/CRC.

Barbosa, Carlos Eduardo, Eduardo Ogasawara, Daniel de Oliveira, e Marta Mattoso. 2009. Paralelização de Tarefas de Mineração de Dados Utilizando Workflows Científicos. In *V Workshop em Algoritmos e Aplicações de Mineração de Dados*. Fortaleza, Ceara, Brazil.

Barga, Roger S., Dan Fay, Dean Guo, Steven Newhouse, Yogesh Simmhan, e Alex Szalay. 2008. Efficient scheduling of scientific workflows in a high performance computing cluster. In *6th international workshop on Challenges of large applications in distributed environments*, 63-68. Boston, MA, USA: ACM. doi:10.1145/1383529.1383545. http://portal.acm.org/citation.cfm?id=1383529.1383545.

Bayucan, A., R. L. Henderson, e J. P. Jones. 2000. Portable Batch System Administration Guide. *Veridian System*.

Bergsten, Hans. 2003. *JavaServer pages*. O'Reilly Media, Inc., Dezembro 1.

Callahan, Steven P., Juliana Freire, Emanuele Santos, Carlos E. Scheidegger, Cláudio T. Silva, e Huy T. Vo. 2006. VisTrails: visualization meets data management. In *Proc. SIGMOD 2006*, 745-747. USA: ACM. doi:10.1145/1142473.1142574. http://portal.acm.org/citation.cfm?id=1142473.1142574.

Carpenter, Bryan, Vladimir Getov, Glenn Judd, Anthony Skjellum, e Geoffrey Fox. 2000. MPJ: MPI-like message passing for Java. *Concurrency: Practice and Experience* 12, n. 11: 1019-1038. doi:10.1002/1096-9128(200009)12:11<1019::AID-

CPE518>3.0.CO;2-G.

Costa, Lauro Beltrao, Walfredo Cirne, e Daniel Fireman. 2005. Converting Space Shared Resources into Intermittent Resources for use in Bag-of-Tasks Grids. In *Proceedings of the 17th International Symposium on Computer Architecture on High Performance Computing*, 243-250. IEEE Computer Society. http://portal.acm.org/citation.cfm?id=1112533.

Cruz, Sergio Manuel Serra da, M. Campos, e M. Mattoso. 2009. Towards a Taxonomy of Provenance in Scientific Workflow Management Systems. In *IEEE International Workshop on Scientific Workflows*. Los Angeles, California, United States.

Culler, David E., Jaswinder Pal Singh, e Anoop Gupta. 1999. *Parallel computer architecture: a hardware/software approach*. 2 ed.

Dean, Jeffrey, e Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, n. 1: 107-113. doi:10.1145/1327452.1327492.

Deelman, Ewa, Dennis Gannon, Matthew Shields, e Ian Taylor. 2009. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems* 25, n. 5: 528-540. doi:10.1016/j.future.2008.06.012.

Deelman, Ewa, Gaurang Mehta, Gurmeet Singh, Mei-Hui Su, e Karan Vahi. 2007. Pegasus: Mapping Large-Scale Workflows to Distributed Resources. In *Workflows for e-Science*, 376-394. Springer. http://dx.doi.org/10.1007/978-1-84628-757-2_23.

Elias, Renato N., e Alvaro L. G. A. Coutinho. 2007. Stabilized edge-based finite element simulation of free-surface flows. *International Journal for Numerical Methods in Fluids* 54, n. 6: 965-993. doi:10.1002/fld.1475.

Elias, Renato N., Marcos A.D. Martins, e Alvaro L.G.A. Coutinho. 2005. Parallel Edge-Based Inexact Newton Solution of Steady Incompressible 3D Navier-Stokes Equations. In *Euro-Par 2005 Parallel Processing*, 1237-1245. http://dx.doi.org/10.1007/11549468_135.

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

Freire, Juliana, David Koop, Emanuele Santos, e Cláudio T. Silva. 2008. Provenance for

Computational Tasks: A Survey. *Computing in Science and Engineering, v.10*, n. 3: 11-21.

Fujimoto, N., e K. Hagihara. 2006. A 2-Approximation Algorithm for Scheduling Independent Tasks onto a Uniform Parallel Machine and its Extension to a Computational Grid. In *IEEE International Conference on Cluster Computing*, 1-7. doi:10.1109/CLUSTR.2006.311905.

Gadelha, L.M.R., e M. Mattoso. 2008. Kairos: An Architecture for Securing Authorship and Temporal Information of Provenance Data in Grid-Enabled Workflow Management Systems. In *International Workshop on Scientific Workflows and Business Workflow Standards in e-Science (SWBES 2008)*, 597-602.

Georgakopoulos, Dimitrios, Mark Hornick, e Amit Sheth. 1995. An overview of workflow management. *Distributed and Parallel Databases - Special issue on software support for work flow management*, Abril, 2 edition. https://www.zotero.org/user/login.

Gustafson, John L. 1988. Reevaluating Amdahl's law. *Communications of the ACM* 31, n. 5: 532-533. doi:10.1145/42411.42415.

Han, Jiawei, e Micheline Kamber. 2006. *Data Mining: Concepts and Techniques*. Morgan Kaufmann.

Hasan, Ragib, Radu Sion, e Marianne Winslett. 2007. Introducing secure provenance: problems and challenges. In *Proceedings of the 2007 ACM workshop on Storage security and survivability*, 13-18. Alexandria, Virginia, USA: ACM. doi:10.1145/1314313.1314318. http://portal.acm.org/citation.cfm?id=1314313.1314318.

Haykin, Simon. 2008. *Neural Networks and Learning Machines*. 3 ed. Prentice Hall, Novembro 28.

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

Isard, Michael, Mihai Budiu, Yuan Yu, Andrew Birrell, e Dennis Fetterly. 2007. Dryad: distributed data-parallel programs from sequential building blocks. In *2nd ACM*

SIGOPS/EuroSys European Conference on Computer Systems, 72, 59. Lisbon, Portugal: ACM. http://dx.doi.org/10.1145/1272996.1273005.

Li Hui, Huashan Yu, e Li Xiaoming. 2008. A lightweight execution framework for massive independent tasks. In *MTAGS 08*, 1-9. doi:10.1109/MTAGS.2008.4777911.

Lucor, Didier, Johan Meyers, e Pierre Sagaut. 2007. Sensitivity Analysis of Large-Eddy Simulations to Subgrid-Scale-Model Parametric Uncertainty Using Polynomial Chaos. *Journal of Fluid Mechanics* 585, n. 1: 255-279. doi:10.1017/S0022112007006751.

Marinho, A., L. Murta, C. Werner, V. Braganholo, Sergio Manuel Serra da Cruz, e M. Mattoso. 2009. A Strategy for Provenance Gathering in Distributed Scientific Workflows. In *IEEE International Workshop on Scientific Workflows*. Los Angeles, California, United States.

Martinho, W., E. Ogasawara, D. Oliveira, F. Chirigati, I. Santos, G.H.T. Travassos, e M. Mattoso. 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.

Mattoso, M., C. Werner, G. Travassos, V. Braganholo, e L. Murta. 2008. Gerenciando Experimentos Científicos em Larga Escala. In *SEMISH - CSBC*. Belém, Pará - Brasil.

Meyer, Luiz, Doug Scheftner, Jens Vöckler, Marta Mattoso, Mike Wilde, e Ian Foster. 2007. An Opportunistic Algorithm for Scheduling Workflows on Grids. In *High Performance Computing for Computational Science - VECPAR 2006*, 1-12. http://dx.doi.org/10.1007/978-3-540-71351-7_1.

Meyer, Luiz A.V.C., Shaila C. Rössle, Paulo M. Bisch, e Marta Mattoso. 2005. Parallelism in Bioinformatics Workflows. In *High Performance Computing for Computational Science - VECPAR 2004*, 583-597. http://dx.doi.org/10.1007/11403937_44.

Moreau, Luc, Juliana Freire, Joe Futrelle, Robert McGrath, Jim Myers, e Patrick Paulson. 2008. The Open Provenance Model: An Overview. In *Provenance and Annotation of Data and Processes*, 323-326. http://dx.doi.org/10.1007/978-3-540-89965-5_31.

Northrop, L.M. 2002. SEI's software product line tenets. *IEEE Software* 19, n. 4: 32-40. doi:10.1109/MS.2002.1020285.

Ogasawara, Eduardo, Leonardo Murta, Geraldo Zimbrão, e Marta Mattoso. 2009. Neural networks cartridges for data mining on time series. In *IJCNN*, 2302-2309. Atlanta, USA. doi:10.1109/IJCNN.2009.5178615.

Ogasawara, Eduardo, Daniel Oliveira, Fernando Chirigati, Carlos Eduardo Barbosa, Renato Elias, Vanessa Braganholo, Alvaro Coutinho, e Marta Mattoso. 2009. Exploring many task computing in scientific workflows. In *MTAGS 09*, 1-10. Portland, Oregon: ACM. doi:10.1145/1646468.1646470. http://portal.acm.org/citation.cfm?id=1646468.1646470&coll=Portal&dl=ACM&type=series&idx=SERIES371&part=series&WantType=Proceedings&title=SC&CFID=://www.google.com/search?hl=en&CFTOKEN=www.google.com/search?hl=en.

Ogasawara, Eduardo, Carlos Paulino, Leonardo Murta, Cláudia Werner, e Marta Mattoso. 2009. Experiment Line: Software Reuse in Scientific Workflows. In *21th SSDBM*, 264–272. New Orleans, LA: Springer-Verlag.

Paraview. 2009. *Paraview*. http://www.paraview.org. Google Scholar.

Pike, Rob, Sean Dorward, Robert Griesemer, e Sean Quinlan. 2005. Interpreting the data: Parallel analysis with Sawzall. *Sci. Program.* 13, n. 4: 277-298.

Porto, Fabio, José Antônio Macedo, Javier Sanchez Tamargo, Yuanjian Wang Zufferey, Vânia P. Vidal, e Stefano Spaccapietra. 2008. Towards a Scientific Model Management System. In *Proceedings of the ER 2008 Workshops (CMLSA, ECDM, FP-UML, M2AS, RIGiM, SeCoGIS, WISM) on Advances in Conceptual Modeling: Challenges and Opportunities*, 55-65. Barcelona, Spain: Springer-Verlag. http://portal.acm.org/citation.cfm?id=1478002.

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

Raicu, Ioan, Zhao Zhang, Mike Wilde, Ian Foster, Pete Beckman, Kamil Iskra, e Ben Clifford. 2008. Toward loosely coupled programming on petascale systems. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing - Volume 00*, 1-12. Austin, Texas: IEEE Press. http://portal.acm.org/citation.cfm?id=1413393.

Raicu, Ioan, Yong Zhao, Catalin Dumitrescu, Ian Foster, e Mike Wilde. 2007. Falkon: a Fast and Light-weight tasK executiON framework. In *SC07*, 1-12. Reno, Nevada: ACM. doi:10.1145/1362622.1362680. http://portal.acm.org/citation.cfm?doid=1362622.1362680.

Samples, Michael E., Jason M. Daida, Matt Byom, e Matt Pizzimenti. 2005. Parameter sweeps for exploring GP parameters. In *2005 workshops on Genetic and evolutionary computation*, 212-219. Washington, D.C.: ACM. doi:10.1145/1102256.1102308. http://portal.acm.org/citation.cfm?id=1102308.

Silva, Fabrício A.B., Sílvia Carvalho, e Eduardo R. Hruschka. 2004. A Scheduling Algorithm for Running Bag-of-Tasks Data Mining Applications on the Grid. In *Euro-Par 2004 Parallel Processing*, 254-262. http://www.springerlink.com/content/p7lw2n96y7rvxp58.

Sulistio, Anthony, e Rajkumar Buyya. 2005. A Time Optimization Algorithm for Scheduling Bag-of-Task Applications in Auction-based Proportional Share Systems. In *Proceedings of the 17th International Symposium on Computer Architecture on High Performance Computing*, 235-242. IEEE Computer Society. http://portal.acm.org/citation.cfm?id=1112531.

Szyperski, Clemens. 1997. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Professional, Dezembro 19.

Taylor, Ian, Matthew Shields, Ian Wang, e Andrew Harrison. 2007. The Triana Workflow Environment: Architecture and Applications. In *Workflows for e-Science*, 320-339. Springer. http://dx.doi.org/10.1007/978-1-84628-757-2_20.

Venugopal, Srikumar, e Rajkumar Buyya. 2005. A Deadline and Budget Constrained Scheduling Algorithm for eScience Applications on Data Grids. In *Proceedings Of 6th International Conference On Algorithms And Architectures For Parallel Processing (Ica3pp-2005)*, 3719:60--72. doi:10.1.1.60.8734. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.60.8734.

Walker, Edward, e Chona Guiang. 2007. Challenges in executing large parameter sweep studies across widely distributed computing environments. In *Workshop on*

Challenges of large applications in distributed environments, 11-18. Monterey, California, USA: ACM. doi:10.1145/1273404.1273411. http://portal.acm.org/citation.cfm?id=1273404.1273411.

Yu, Jia, e Rajkumar Buyya. 2005. A Taxonomy of Workflow Management Systems for Grid Computing. *Journal of Grid Computing* 34, n. 3: 171-200. doi:10.1.1.59.8378.

Anexos

Anexo I: Consultas Reais à Proveniência do Hydra

Este anexo indica os resultados das consultas da seção 3.4.1.

“Quais são as pastas de trabalho das tarefas que tiverem o parâmetro 'Reynolds Number' com valores maiores que 263 e menores que 278?”

```
SELECT t.workspace FROM  DistribuitedActivityExecution dae, Task t,
SweepedParameter sp, Parameter p WHERE dae.DistribuitedActivityExecutionID =
t.DistribuitedActivityExecutionID AND p.ParameterID =  sp.ParameterID AND
sp.TaskID = t.TaskID AND dae.DistribuitedActivityExecutionID = 4 AND p.Name =
'Reynolds Number' AND CAST(sp.TextValue as numeric) > 0.0036 AND
CAST(sp.TextValue as numeric) < 0.0038;
```

```
|-------------------------------------------|
|workspace                                  |
|-------------------------------------------|
|/home/users/edubarbosa/MTAGS/exp/VISC_75   |
|/home/users/edubarbosa/MTAGS/exp/VISC_79   |
|/home/users/edubarbosa/MTAGS/exp/VISC_78   |
|/home/users/edubarbosa/MTAGS/exp/VISC_77   |
|/home/users/edubarbosa/MTAGS/exp/VISC_76   |
|/home/users/edubarbosa/MTAGS/exp/VISC_80   |
|-------------------------------------------|
```

“Qual é o tempo médio das tarefas que tiverem o parâmetro 'Reynolds Number' com valores maiores que 263 e menores que 278?”

```
SELECT  AVG(t.EndTime-t.StartTime) FROM  DistribuitedActivityExecution dae, Task
t, SweepedParameter sp, Parameter p WHERE dae.DistribuitedActivityExecutionID =
t.DistribuitedActivityExecutionID AND p.ParameterID =  sp.ParameterID AND
dae.DistribuitedActivityExecutionID = 4 AND sp.TaskID = t.TaskID AND p.Name =
'Reynolds Number' AND CAST(sp.TextValue as numeric) > 0.0036 AND
CAST(sp.TextValue as numeric) < 0.0038;
```

```
|-------------------------------------------|
|avg                                        |
|-------------------------------------------|
|01:33:00.161667                            |
|-------------------------------------------|
```

“Qual foram as tarefas que tiverem o parâmetro 'Reynolds Number' com valores maiores que 263 e menores que 278?”

```
SELECT  t.TaskID, t.CommandLine FROM DistribuitedActivityExecution dae, Task t,
SweepedParameter sp, Parameter p WHERE dae.DistribuitedActivityExecutionID =
t.DistribuitedActivityExecutionID AND p.ParameterID =  sp.ParameterID AND
sp.TaskID = t.TaskID AND dae.DistribuitedActivityExecutionID = 4 AND  p.Name
='Reynolds Number' AND CAST(sp.TextValue as numeric) > 0.0036 AND
CAST(sp.TextValue as numeric) < 0.0038;
```

```
|---------------------|----------------------|
|taskid               |commandline           |
|---------------------+----------------------|
|193                  |experiment.cmd        |
|195                  |experiment.cmd        |
|197                  |experiment.cmd        |
|196                  |experiment.cmd        |
|199                  |experiment.cmd        |
|200                  |experiment.cmd        |
|-----------------------------------------------|
```

"Quais foram as saídas (stdout e stderr) de cada tarefa executada?"

```
SELECT StdOut, StdErr FROM Task WHERE TaskID=2380;
```

```
|---------------------|----------------------|
|StdOut               |StdErr                |
|---------------------+----------------------|
|     +===============|                      |
|-----------------------------------------------|
```

"Quais foram os arquivos criados ou modificados durante uma tarefa?"

```
SELECT Filename FROM WorkspaceFile WHERE TaskID=2380;
```

```
|-------------------------------------------------------------|
|filename                                                     |
|-------------------------------------------------------------|
|/home/users/edubarbosa/MTAGS/exp/task_155/cav_1.part.edg     |
|/home/users/edubarbosa/MTAGS/exp/task_155/cav_1.part.in      |
|/home/users/edubarbosa/MTAGS/exp/task_155/cav_1.part.mat     |
|/home/users/edubarbosa/MTAGS/exp/task_155/cav_1.part.msh     |
|/home/users/edubarbosa/MTAGS/exp/task_155/experiment.cmd     |
|/home/users/edubarbosa/MTAGS/exp/task_155/input.dat          |
|/home/users/edubarbosa/MTAGS/exp/task_155/result_start.txt|
|-------------------------------------------------------------|
```

"As tarefas foram concluídas com sucesso?"

```
SELECT taskid FROM Task WHERE DistribuitedActivityExecutionID=4 AND
ExitStatus=0;
```

```
|-----------------------------------------|
|taskid                                   |
|-----------------------------------------|
|121                                      |
|122                                      |
|123                                      |
|128                                      |
|125                                      |
|127                                      |
|124                                      |
|-----------------------------------------|
```

"Quantos processadores foram utilizados em uma atividade?"

```
SELECT NUMProcs FROM DistribuitedActivityExecution WHERE
DistribuitedActivityExecutionID = 4;
```

```
|-----------------------------------------|
|numprocs                                 |
|-----------------------------------------|
|8                                        |
|-----------------------------------------|
```

“Qual foi o tempo médio de execução das tarefas da atividade?”

```
SELECT avg(EndTime-StartTime) FROM Task WHERE DistribuitedActivityExecutionID=4;
```

```
|-----------------------------------------|
|avg                                      |
|-----------------------------------------|
|01:22:19.254719                          |
|-----------------------------------------|
```

“Quais foram os parâmetros de cada tarefa executada?”

```
SELECT p.Name, p.Type, sp.NumericValue, sp.TextValue FROM Task t, Parameter p,
SweepedParameter sp WHERE p.ParameterID=sp.ParameterID AND sp.TaskID = t.TaskID
AND t.TaskID=2380;
```

```
|----------------|--------------|--------------|--------------|
|name            |type          |numericvalue  |textvalue     |
|----------------+--------------+--------------+--------------|
|Reynolds Number |2             |0.006435294117|              |
|-------------------------------------------------------------|
```

“Quais foram os tempos de execução de cada tarefa executada?”

```
SELECT TaskID, (EndTime-StartTime) as Time FROM Task WHERE
DistribuitedActivityExecutionID=X;
```

```
|--------------------|------------------------|
|TaskID              |Time                    |
|--------------------+------------------------|
|121                 |01:23:19.644            |
|122                 |01:25:06.349            |
|123                 |01:25:32.416            |
|128                 |01:26:23.173            |
|125                 |01:26:26.194            |
|127                 |01:26:34.702            |
|124                 |01:27:01.433            |
|-----------------------------------------|
```

“Qual foi o desvio padrão do tempo de execução das tarefas da atividade?”

```
SELECT (SELECT CAST(textcat(text(STDDEV(EXTRACT(EPOCH FROM (EndTime-
StartTime)))), text(' seconds')) as interval) FROM Task t WHERE
dae.DistribuitedActivityExecutionID = t.DistribuitedActivityExecutionID) as
AvgTaskExecutionTimeStdDev FROM DistribuitedActivityExecution dae WHERE
DistribuitedActivityExecutionID = 4;
```

```
|-----------------------------------------|
|avgtaskexecutiontimestddev                |
|-----------------------------------------|
|00:13:45.55449                           |
|-----------------------------------------|
```

“Quantas execuções de atividades foram feitas em um determinado período?”

```
SELECT Count(1) FROM (SELECT DISTINCT DistribuitedActivityExecutionID FROM
DistribuitedActivityExecution WHERE StartTime > '2010-03-01 12:00' AND StartTime
< '2010-04-01 12:00') as Execs;
```

```
|-------------------------------------------|
|count                                      |
|-------------------------------------------|
|10                                         |
|-------------------------------------------|
```