PARALELISMO DE DADOS CIENTÍFICOS UTILIZANDO TÉCNICAS P2P

Jonas Furtado Dias

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

Fevereiro de 2011

PARALELISMO DE DADOS CIENTÍFICOS UTILIZANDO TÉCNICAS P2P

Jonas Furtado Dias

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. Philippe Olivier Alexandre Navaux, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

FEVEREIRO DE 2011

Dias, Jonas Furtado

Paralelismo de Dados Científicos Utilizando Técnicas P2P/ Jonas Furtado Dias. – Rio de Janeiro: UFRJ/COPPE, 2011.

IX, 83 p.: il.; 29,7 cm.

Orientadora: Marta Lima de Queirós Mattoso

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2011.

Referencias Bibliográficas: p. 75-83.

1. Paralelismo de dados. 2. Workflows Científicos. 3. Peer-to-Peer. I. Mattoso, Marta Lima de Queirós. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III Título.

À minha família.

AGRADECIMENTOS

O agradecimento deveria ser discurso de toda conquista, pois nada se conquista sozinho. Relembrar minha trajetória nestes dois anos é reviver muitos momentos de dedicação e trabalho. Porém, o mais gratificante é rever na memória tantos ao meu lado me apoiando, me ensinando e me ajudando. O incentivo, o aprendizado e a ajuda são fundamentais para o crescimento, pois crescer é doloroso, difícil e trabalhoso. Mas é por isso que nessa vida temos família, professores e amigos.

Agradeço aos amigos pela ajuda. Ajuda no trabalho e na diversão conjunta, porque ajudar é dividir. Agradeço aos amigos Carlos, Pedro e Rafael pelas parcerias produtivas e bem humoradas nos trabalhos das disciplinas do mestrado. Também agradeço os amigos Eduardo Ogasawara e Daniel de Oliveira por me ajudarem em tantos momentos durante o mestrado. Agradeço aos amigos de colégio que, até hoje, compartilham comigo muitos momentos felizes. Em especial, agradeço a uma amizade que se tornou muito mais do que uma amizade. Carol, obrigado por toda ajuda, amor e compreensão.

Agradeço aos professores pelo aprendizado. Conhecimento é o alimento de todo crescimento humano. Sendo assim, agradeço aos professores Geraldo Xexéo, Jano Souza, Geraldo Zimbrão, Valmir Barbosa e Guilherme Travassos. Agradeço também ao professor Álvaro Coutinho, ao Albino Aveleda e à Myrian Costa. Um agradecimento ainda maior à professora Marta Mattoso, que me orientou com muita responsabilidade, dedicação e atenção. Também agradeço aos professores Philippe Navaux e Alexandre Assis pela participação na banca desta dissertação.

Seria pouco agradecer à família pelo incentivo. Família também é professora, família é amiga. Por isso, agradeço à família por tudo, pelo incentivo que me deu, por tudo que me ensinou e pela ajuda incansável a qualquer hora. Aos meus pais agradeço por todo esforço que fizeram pela minha educação, por toda a dedicação e amor infinito que sinto deles e por eles. À minha mãe agradeço o exemplo de seriedade, dedicação, equilíbrio e por toda a liberdade que me deu, mesmo que isso lhe custasse um coração aflito. Ao meu pai agradeço a sua genialidade, o exemplo de capacidade e o estímulo na construção de ideias. À minha irmã também agradeço especialmente pelo carinho, cuidado e por todos os momentos divertidos que vivemos juntos nestes anos que moramos juntos. Meus tios, minhas tias, primos e primas... Agradeço a todos por todo o amor.

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

PARALELISMO DE DADOS CIENTÍFICOS UTILIZANDO TÉCNICAS P2P

Jonas Furtado Dias

Fevereiro/2011

Orientadora: Marta Lima de Queirós Mattoso

Programa: Engenharia de Sistemas e Computação

A complexidade e o tempo de processamento dos modelos de simulação computacional em experimentos científicos trouxe desafios na condução destes experimentos. Os workflows científicos vêm sendo adotados para a ciência em larga-escala. A utilização intensa e os grandes volumes de dados nestes workflows demandam por paralelismo. Entretanto, paralelizar um workflow requer ferramentas específicas e domínio de programação para executá-lo paralelamente em clusters. Buscando tornar a paralelização de workflows mais transparente para o cientista, esta dissertação propõe a abordagem Heracles. O Heracles proporciona um mecanismo de tolerância a falhas e gerência dinâmica de recursos utilizando técnicas P2P. O intuito do Heracles é executar atividades em paralelo sem que o cientista precise estipular o número de nós da execução bem como redistribuir automaticamente as tarefas em caso de falhas no ambiente computacional. Desta forma, o cientista só precisa definir um prazo para a atividade. O Heracles foi avaliado por meio de simulação e neste ambiente mostrou ser capaz de cumprir os prazos definidos para a execução de atividades e de se recuperar de falhas eficientemente. Portanto, pode ser interessante incorporar a abordagem Heracles em escalonadores reais para realizar avaliações mais profundas.

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

SCIENTIFIC DATA PARALLELISM USING P2P TECHNIQUES

Jonas Furtado Dias

February/2011

Advisor: Marta Lima de Queirós Mattoso

Department: Computer Science Engineering

The complexity and the processing time of computational simulation models of scientific experiments bring challenges on the conduction of these experiments. Scientific workflows have being adopted on large-scale science. The intense utilization of great volumes of data on these workflows demands parallelism. However, parallelize a workflow requires specific tools and programming skills to run it in parallel in clusters. Trying to turn workflow parallelization more transparent to the scientist, this dissertation proposes the Heracles approach. Heracles proposes a fault tolerant and dynamic resource management mechanism using P2P techniques. The purpose of Heracles is to execute activities in parallel without asking the scientists to specify the number of nodes involved in the execution and to automatically reschedule failed tasks. This way, the scientists only need to define the deadline for the activity. Heracles was evaluated through simulation and showed that it is capable of fulfilling the deadlines of the activities and to recover from failures efficiently. Thus, it may be interesting to integrate Heracles approach on real schedulers to realize deeper evaluations.

ÍNDICE

LISTAGEM DE FIGURAS

Capítulo 1 – Introdução

A experimentação científica é um mecanismo importante para o avanço da ciência. É por meio de experimentos que um cientista pode avaliar hipóteses com o objetivo de propor uma tese. O ciclo de vida do experimento científico (Mattoso et al. 2010) pode ser dividido em três fases: composição, execução e análise. Estas fases traduzem a lógica do processo em que o cientista planeja o experimento compondo os artefatos a serem estudados, executa o estudo e analisa os resultados. Se os resultados não forem conclusivos, o processo é realimentado dando início a uma nova iteração do ciclo.

Dentre os diversos tipos de experimentos, os *in silico* (Stevens et al. 2007) vêm adquirindo grande expressão no meio científico. Tais experimentos consistem na composição de modelos matemáticos e computacionais que representem uma abstração da realidade. O objeto de estudo é analisado com os resultados da execução de simulações sobre este modelo em intervalos de tempos discretos. Um único experimento pode envolver a execução encadeada ou iterativa de diversas atividades que executam diferentes simulações. Muitas dessas atividades podem ser relacionadas ou dependentes entre si. Além disso, pode-se ter atividades de tomada de decisão entre uma atividade de simulação e outra. Sendo assim, a modelagem e controle de um único experimento demanda grande esforço. Executar tais atividades respeitando os relacionamentos e dependências, além do controle de transferências de dados e resultados obtidos, exige um alto nível de gerência. Portanto, tais experimentos são apoiados por sistemas de workflows científicos (Brown et al. 2007).

Um workflow científico é um encadeamento de atividades que podem consumir dados de entrada e produzir dados de saída em um experimento. Uma atividade é uma ação que, em geral, representa a execução de um programa, serviço, ou script de computador. Os dados de entrada e saída podem ser grupos de parâmetros ou arquivos para leitura e escrita. A saída de uma atividade pode ser consumida por atividades seguintes até o término do workflow. O fluxo intenso de dados entre atividades é bem característico em workflows científicos. Normalmente, workflows científicos são modelados e gerenciados por sistemas de gerência de workflows científicos (SGWfC), que são softwares preparados para a representação gráfica de workflows e gerência de sua execução.

Do ponto de vista do experimento científico, muitas variáveis independentes de um experimento podem ser testadas assumindo diferentes valores e gerando diversas combinações. Consequentemente, o número de execuções da simulação de um modelo pode ser muito grande. Isso significa que o workflow daquele experimento ou algumas atividades deste workflow podem ser executadas repetidas vezes. Desta forma, o número total de execuções pode chegar a centenas de milhares de execuções ou até mesmo milhões de execuções para um único experimento. Um exemplo é a análise de comportamento estrutural de uma plataforma de petróleo *offshore* sob inúmeras combinações de condições ambientais (Mattoso et al. 2010). Tal demanda, além do crescente refinamento e complexidade dos modelos sendo simulados exige grande poder computacional para que estes sejam processados e os resultados, então, obtidos. A gerência e controle manual sobre a execução do experimento como um todo é inviável.

A intensidade computacional envolvendo as diversas atividades dos workflows dos experimentos motiva o paralelismo de dados nas execuções para realizar tais experimentos em tempo hábil. Em muitos casos, esta paralelização em workflows científicos pode ocorrer por meio de fragmentação dos dados de entrada (Coutinho et al. 2010) ou varredura de parâmetros (Ogasawara et al. 2009a). Ambientes de computação de alto desempenho como clusters e nuvens são indicados para a execução paralela de atividades de workflows. O crescente poder computacional dos supercomputadores potencializa o desenvolvimento de experimentos científicos em larga escala (Deelman et al. 2009).

A taxa do crescimento de processadores por cluster vem aumentando anualmente como mostram as estatísticas do portal top500.org (TOP500 2010), que registra as características dos maiores clusters, ou supercomputadores, do mundo anualmente. Segundo as estatísticas do portal, reproduzidas na Tabela 1, na lista dos 500 maiores supercomputadores em novembro de 2010, mais de 87,00% possuem mais que 4096 núcleos de processamento. Além disso, 1,80%, ou seja, nove destes supercomputadores possuem mais 128000 núcleos de processamento (*cores*). Embora esse crescimento represente um avanço no meio científico e industrial, ele gera problemas de gerência e controle do elevado número de processadores dos supercomputadores, que podem falhar. A falha recorrente de processadores nestas grandes máquinas é uma ameaça no desenvolvimento de trabalhos utilizando os recursos de tais supercomputadores.

Tabela 1: Participação de supercomputadores por número de processadores (TOP500 2010)

Numero de Processadores	Contagem	Participação (%)
1025-2048	2	0,40 %
2049-4096	60	12,00 %
4k-8k	291	58,20 %
8k-16k	96	19,20 %
16k-32k	20	4,00 %
32k-64k	17	3,40 %
64k-128k	5	1,00 %
>128k	9	1,80 %
Total	500	100,00 %

Tradicionalmente, experimentos custosos são executados de maneira distribuída em supercomputadores para terem resultados obtidos mais rapidamente ou em tempo hábil. Alguns experimentos também requerem que sua execução seja feita em um dado espaço de tempo, pois, após aquele limite, os resultados não têm mais utilidade. Um exemplo são as simulações de modelos climáticos para previsões meteorológicas, onde não faz sentido prever um evento climático após o seu acontecimento. Em função dessa necessidade pela execução veloz, supercomputadores tornaram-se elementos fundamentais na fase de execução do ciclo de vida de experimentos em larga escala. Muitas aplicações podem ser executadas rapidamente com o apoio de supercomputadores. Entretanto, os escalonadores na computação de alto desempenho não exploram alguns mecanismos como o controle detalhado sobre a execução do experimento científico. Além disso, eles não tiram proveito de informações vindas da fase de composição para produzir os dados para análise de maneira mais eficiente. Portanto, para proporcionar um controle melhor na execução do experimento, pode ser interessante utilizar ferramentas adicionais que usem os supercomputadores apenas como infraestrutura de execução.

Executar um experimento em um cluster envolve a paralelização ou distribuição das atividades do experimento nos recursos do cluster. É a paralelização que trará ganhos no desempenho, reduzindo o tempo total da execução. Uma abordagem tradicional de paralelização é utilizar uma interface de troca de mensagens entre processos sendo executados simultaneamente, como a *Message Passing Interface* (MPI). A MPI traz grandes vantagens na computação de alto desempenho, porém, para um código sequencial legado e complexo,

adicionar a lógica MPI pode ser muito custoso e propenso a erros. Um código MPI para ser executado por mais de mil processos é ainda mais complexo de ser programado, em função dos requisitos de eficiência necessários. Além disso, envolver muitos processos em uma única execução MPI pode ser arriscado, visto que estes processos estão sujeitos a falhas, o que pode comprometer a execução como um todo, já que a MPI não suporta a gerência dinâmica de recursos de forma transparente. Programar o modelo dinâmico de criação de processos MPI é difícil e custoso para os cientistas (Cera et al. 2006). Ademais, a modificação do código para funcionar com a MPI pode inserir erros no programa.

Uma alternativa à MPI seria a utilização de escalonadores voltados à computação em grade. Estes escalonadores, tal como o Condor (Couvares et al. 2007), podem receber inúmeras tarefas para processar, as quais são escalonadas conforme os recursos computacionais ficam disponíveis. O maior problema desta abordagem é o tempo gasto nas filas de espera para a execução de tarefas já que cada tarefa é vista como um elemento a ser designado a um determinado recurso. Tal granularidade gera uma sobrecarga já que o sistema consome tempo realizando os procedimentos de inicialização e término de cada tarefa em cada recurso. Segundo Raicu *et al.* (2007), o tempo para disparar uma tarefa nesses sistemas pode ultrapassar os 30 segundos, o que seria equivalente ao máximo de duas tarefas por minuto. Além disso, escalonadores tradicionais suportam políticas para as filas que podem impedir a submissão de muitas tarefas por um único usuário. A política do escalonador, pode, por exemplo, permitir que cada usuário submeta apenas 4 tarefas (ou *jobs*, na visão do escalonador) por vez.

Em função das dificuldades com a computação de alto desempenho utilizando, por exemplo, a MPI, e a computação de alta vazão como nos casos das grades computacionais, idealizou-se um novo modelo de computação voltado para cenários envolvendo a execução de muitas tarefas heterogêneas sobre múltiplos processadores possivelmente heterogêneos. Este novo paradigma de computação, conhecido como *Many Task Computing* (MTC) (Raicu et al. 2008), ou computação de muitas tarefas, parece ser adequado para o processamento de experimentos que gerem uma grande quantidade de execuções de atividades de workflow. O MTC enfatiza o uso de grandes quantidades de recursos computacionais para processar grandes quantidades de tarefas computacionais. Entretanto, para obter mais eficiência, novas propostas de submissão, execução e controle de tarefas em ambientes de alto desempenho

ainda precisam ser estudadas para atender aos requisitos do MTC e do controle do experimento e seus workflows científicos, tal como a gerência de dados de proveniência distribuída.

Ainda que o paradigma MTC seja novo e promissor, ainda existem muitas questões importantes em aberto para encontrar as estratégias adequadas para diferentes tipos de escalonamento e execução. As soluções de MTC atuais (Zhao et al. 2007, Raicu et al. 2007, Blythe et al. 2005, Smanchat et al. 2009) exploram a paralelização de workflows, mas são, de certa forma, orientadas à computação em grade e não exploram algumas características de clusters atuais. Por exemplo, como lidar com o comportamento dinâmico dos nós em um cluster? Esta questão é importante, pois o número de núcleos de processamento por cluster está crescendo e, mesmo com melhores equipamentos, conforme o número de componentes eletrônicos cresce, maiores são as chances de falhas (Dias e Aveleda 2010, Oliner et al. 2005).

Para a gerência tradicional de recursos de clusters, se um determinado conjunto de recursos está alocado para processar uma determinada requisição, se qualquer elemento do conjunto de recursos falhar, a requisição será abortada e caracterizada como falha. Já as grades computacionais seguem o modelo conhecido como *bag of task*s (Silva et al. 2004), ou sacola de tarefas. Neste modelo, cada falha é interpretada de maneira singular, sem afetar a execução das outras tarefas da sacola. De qualquer maneira, nenhuma das duas abordagens resolve o problema da gerência de falhas na execução. Após o término da execução, fica a cargo do cientista verificar quais tarefas falharam e precisam ser submetidas novamente. Na prática, quando o número de tarefas é muito grande, como ocorre na maioria dos experimentos científicos, torna-se inviável realizar tal verificação manual e o cientista acaba por ressubmeter o conjunto completo de tarefas para executar novamente. Esse problema da gerência de falhas gera desperdício de recurso computacional e atrasos na execução de experimentos, podendo comprometer o objetivo dos mesmos.

Sonmez *et al.* (2010) analisam o desempenho de políticas de escalonamento tradicionais em grades computacionais de múltiplos clusters e descobrem que não existe uma única política de escalonamento para workflows com bom desempenho nos cenários investigados por eles. Eles também perceberam que os nós de serviço dos clusters (*head-nodes*) ficam sobrecarregados com frequência durante a execução de workflows. Esta é uma

razão que motiva novos estudos envolvendo a execução de workflows científicos utilizando MTC em clusters de grande porte.

O grande número de nós, o grau acentuado de dinamismo e a possível heterogeneidade do *hardware* dos supercomputadores, faz com que consideremos tal cenário como semelhante ao que ocorre na computação *peer-to-peer* (P2P). Sendo assim, acreditamos que a utilização de técnicas P2P em clusters seja útil para a gerência de atividades que demandem MTC, para o controle de falhas e heterogeneidade da rede. Segundo Pacitti et al. (2007), técnicas P2P também se mostram úteis em grades computacionais de larga escala. Entretanto, ainda faltam estudos que avaliem a utilização de técnicas P2P para o processamento de tarefas de workflows científicos em grandes supercomputadores.

O objetivo desta dissertação é propor a abordagem Heracles, que envolve a utilização de técnicas P2P para explorar a gerência do paralelismo de dados em atividades de workflow científico que demandem MTC em grandes supercomputadores, tais como os clusters que possuam mais de 5000 processadores. Dentre as técnicas P2P, será explorado (1) um mecanismo de gerência dinâmica de recursos, (2) a hierarquização de recursos da execução e (3) um mecanismo de tolerância a falhas. As técnicas mencionadas não são exploradas atualmente em gerentes tradicionais de recursos de clusters. O objetivo da abordagem Heracles é tornar o processo de execução de um experimento científico mais transparente em ambientes distribuídos.

O mecanismo de tolerância a falhas envolve o reescalonamento das tarefas que eram processadas pelo nó que falhou. Tal reescalonamento implica a possível alocação de novos recursos, retransmissão de dados e reinicialização da tarefa. O processo de alocação de novos recursos está no escopo da gerência dinâmica de recursos. Este também é útil para acelerar ou frear o processamento de uma atividade alocando mais ou menos nós para processar tarefas. Métricas como ganho de desempenho (*speed up*) parcial e eficiência parcial podem ser monitoradas para ajustar o número de recursos alocados em função do tempo desejado para o processamento de uma atividade. Dessa forma, um cientista não precisa definir quantos nós ele deseja alocar para processar uma determinada atividade, mas somente definir o tempo no qual ele precisa dos resultados daquela execução do experimento. Sendo assim, a execução paralela da atividade do workflow fica mais transparente para o cientista.

Nesta dissertação é proposta uma organização virtual hierárquica dos nós do cluster para o processamento de tarefas. Essa organização é importante para a gerência do processamento de atividades que envolvam centenas ou milhares de nós. A hierarquização de recursos auxilia nos mecanismos de tolerância a falhas e gerência dinâmica de recursos facilitando a entrada e saída de nós no processamento da atividade. Na proposta hierárquica, um nó de controle pode dividir a rede de recursos que ele controla em grupos. Em seguida, ele elege um nó de cada grupo para se tornar o líder. O líder de um grupo assume as operações de controle daquele grupo como, por exemplo, a distribuição de tarefas internamente. Se alguma requisição de controle não puder ser resolvida localmente, ela é encaminhada para o líder superior na hierarquia. O topo da hierarquia é o nó do cluster que iniciou a execução da atividade. Uma consequência da hierarquização de recursos no processamento de atividades é o balanceamento da carga, pois as operações de controle de tarefas ficam distribuídas dentre os recursos reservados do cluster. O balanceamento de carga parece interessante para evitar eventuais sobrecargas nos servidores de serviço do cluster, como foi observado por Sonmez et al. (2010).

Embora as técnicas propostas mostrem-se interessantes na teoria, é importante avaliar se elas realmente causam melhoria no processamento paralelo de atividades que demandem MTC. Essa avaliação pode ser feita em um supercomputador de grande porte. Entretanto, construir ou utilizar tal infraestrutura real de alto desempenho para realizar estudos iniciais de viabilidade de uso de técnicas P2P é muito caro e inviável para a grande maioria dos pesquisadores que não têm acesso a grandes supercomputadores (Almeida et al. 2008). Como alternativa, podem ser feitos estudos de simulação, onde se pode avaliar o conjunto de técnicas P2P mencionadas em clusters simulados. Na simulação, diversos estereótipos de atividades de workflows científicos podem ser considerados, tais como atividades que demandem fragmentação de dados ou varredura de parâmetros.

Para avaliar nossa proposta, construímos um ambiente de simulação chamado HeracleSim. O HeracleSim é uma adaptação do SciMulator (Dias et al. 2010c). O SciMulator oferece flexibilidade na construção de diferentes organizações de rede com diferentes níveis de controle, desde o centralizado até o totalmente descentralizado. Em estudos preliminares, utilizamos o SciMulator para avaliar uma solução de paralelismo em workflows científicos e vimos que a arquitetura P2P híbrida hierárquica é a mais indicada (Ogasawara et al. 2010).

Assim como o SciMulator, o HeracleSim mostrou-se ágil na simulação de cenários custosos e robusto no consumo de memória. Os resultados obtidos com o HeracleSim foram confrontados com resultados reais utilizando o sistema Hydra (Ogasawara et al. 2009a) em um cluster de 128 cores. O simulador foi capaz de reproduzir o mesmo comportamento de desempenho. Além disso, dados reais da execução foram usados para calibrar os parâmetros do simulador. Os resultados obtidos permitiram a avaliação do desempenho da paralelização de atividades de workflows científicos com uma distribuição simples de tarefas. A análise dos resultados sugere que a estratégia P2P para execução de tarefas é promissora e indica um conjunto de possíveis melhorias na estratégia de distribuição.

A modelagem de atividades de workflows seguindo o paradigma MTC no simulador permite a decomposição das atividades em diversas tarefas que podem ser escalonadas nos nós da rede do cluster simulado de acordo com algumas estratégias. A simulação foi calibrada utilizando dados de experimentos reais (Ogasawara et al. 2009a). Dessa forma, comparamos as estratégias tradicionais de execução de tarefas com a abordagem Heracles, que envolve a utilização de técnicas P2P. Para tal, foram realizados estudos simulando um cluster de 7168 núcleos, que equivale ao maior supercomputador do Núcleo de Atendimento em Computação de Alto Desempenho (NACAD) da UFRJ. Foram considerados diferentes tipos de atividades, variando o número de tarefas gerado por elas e o tempo de execução destas tarefas. A característica de dinamismo da rede foi levada em consideração. Os resultados permitiram avaliar o quanto as técnicas P2P podem auxiliar na gerência da execução de atividades em clusters de grande porte. O Heracles mostrou ser eficiente na recuperação de falhas e capaz de atender prazos estipulados para atividades de maneira satisfatória, mesmo com prazos bem curtos.

O bom desempenho das técnicas P2P na execução distribuída de atividades de workflows motivou a concepção da abordagem Heracles (Dias et al. 2010a, 2010b). A abordagem Heracles pode ser implementada em escalonadores MTC de workflow com o intuito de tornar a execução de experimentos mais transparente aos cientistas. A abordagem também cumpre os requisitos de transparência e coleta de proveniência distribuída utilizando os mecanismos de gerência dinâmica de recursos, hierarquização dos recursos e tolerância a falhas. Uma meta da proposta Heracles é ser integrada ao Hydra (Ogasawara et al. 2009a), um *middleware* de apoio ao escalonamento de tarefas de workflows científicos em clusters.

O restante desta dissertação está organizado da seguinte maneira: No Capítulo 2, discutem-se os experimentos científicos em larga escala e as abordagens existentes que dão apoio à execução destes experimentos. Nele, apresentaremos conceitos de ciclo de vida do experimento, workflows científicos, conceitos de processamento paralelo, e ferramentas existentes que apoiem a execução de experimentos científicos. No Capítulo 3, discutimos conceitos de sistemas *peer-to-peer*, apresentando a classificação destes sistemas, uma avaliação das abordagens e trabalhos relacionados que utilizem técnicas P2P na computação científica. No Capítulo 4 é apresentado o Heracles, explicando-se a arquitetura proposta, tal como detalhes da abordagem. No capitulo 5 é descrito o simulador do Heracles, o HeracleSim. No Capítulo 6, é apresentada uma prova de conceito envolvendo a simulação da abordagem Heracles comparada com uma abordagem tradicional de execução de experimentos. No Capítulo 7, discutimos as conclusões da dissertação e propomos trabalhos futuros.

Capítulo 2 – Experimentos Científicos e sua Execução Paralela

Este capítulo aborda o processo de experimentação científica em larga escala apresentando o ciclo de vida do experimento e os detalhes de suas etapas. Ele também discute as questões envolvidas na execução paralela de experimentos científicos utilizando workflows científicos. Experimentos em larga escala são custosos e envolvem o processamento de muitas atividades e cálculos. Por tal motivo, normalmente a execução destes experimentos é feita de maneira paralela em recursos computacionais distribuídos, tais como clusters, grades computacionais ou nuvem. Sendo assim, no decorrer deste capítulo também são apresentadas abordagens existentes que dão apoio à execução destes experimentos em ambientes distribuídos. As questões apresentadas neste capítulo conduzem à motivação do Heracles, em virtude da necessidade por melhorias nas abordagens de execução de atividades em supercomputadores de maneira mais transparente para o cientista. Os conceitos necessários para o entendimento da arquitetura do Heracles também serão apresentados no contexto dos experimentos científicos.

Existem diversos tipos de experimentos científicos. Os mais tradicionais são os *in vivo*, e os *in vitro*. No primeiro, os estudos são conduzidos com os objetos de estudo em seu próprio ambiente natural. Já no segundo, os objetos de estudo estão em um ambiente controlado. Os avanços nos sistemas computacionais vêm apoiando novas formas de experimentos baseados em computadores, chamados de *in virtuo* e *in silico* (Travassos e Barros 2003). Experimentos *in virtuo* são caracterizados pelos objetos de estudo participando interativamente do experimento em um ambiente simulado pela infraestrutura computacional, enquanto nos experimentos *in silico*, tanto os objetos de estudo quanto a infraestrutura computacional são simulados por computadores. O foco desta dissertação são os experimentos *in silico*. Estes experimentos são computacionalmente custosos, pois envolvem a utilização massiva de dados. São cenários onde, normalmente, é interessante a paralelização desses dados e da própria execução do experimento.

Experimentos *in silico* são comuns em diversas áreas de pesquisa, tais como a meteorologia (Gannon et al. 2007), bioinformática (Coutinho et al. 2010, Romano et al. 2007, Greenwood et al. 2003) e exploração de petróleo (Oliveira et al. 2009, Carvalho 2009). Tais experimentos são considerados como de larga escala pelo aspecto exploratório de diversos

fatores e conjunto de dados. Normalmente, tais experimentos consomem muitos recursos computacionais e, se não forem executados de maneira distribuída, podem demorar dias para executar em máquinas mono processadoras. A complexidade destes experimentos torna difícil o controle sobre eles de maneira manual. Os princípios de um experimento científico exigem que estes sejam sistemáticos e bem documentados para garantir sua reprodutibilidade. Portanto, o controle e a visão evolutiva que o cientista precisa ter do seu experimento é muito importante para o sucesso deste. Isto significa que o cientista precisa controlar a evolução do seu experimento e ter a visão do que está produzindo cada um dos resultados e impactando nas análises para, eventualmente, modificar parte do modelo do experimento para novas execuções. Mattoso *et. al.* (2010) propõe a definição deste processo na forma de um ciclo de vida de um experimento em larga escala.

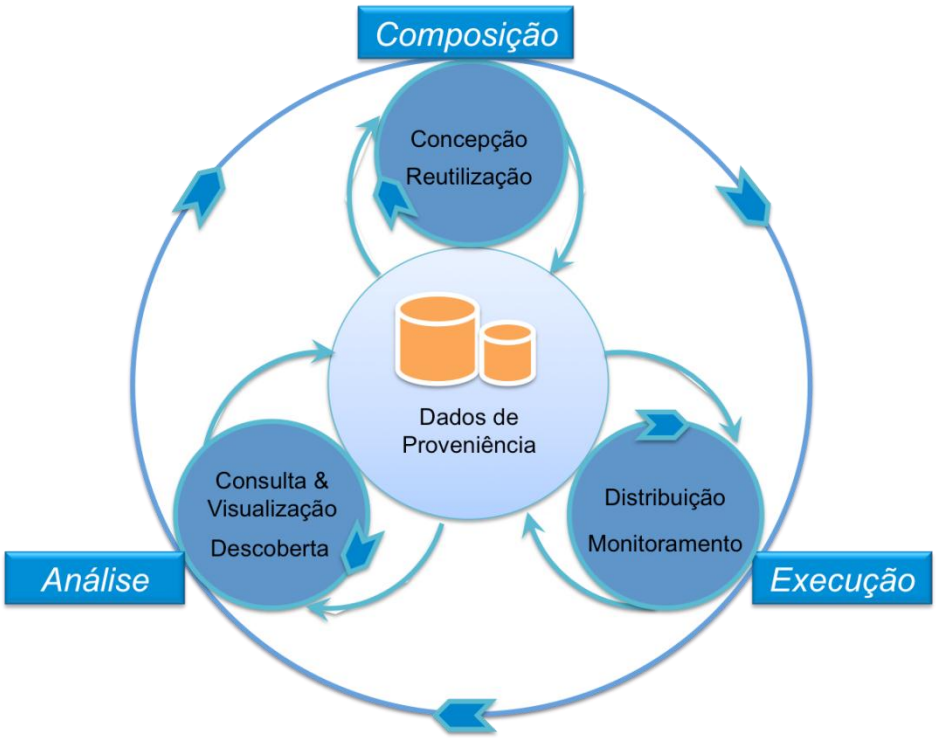

Figura 1: Ciclo de vida do experimento científico segundo Mattoso et.al. (2010)

A seção 2.1 explica com maiores detalhes o ciclo de vida do experimento científico adotado nesta dissertação. Como ele é dividido em três fases, a seção 2.1.1 detalha a fase de composição, a seção 2.1.2 detalha a fase de execução e a 2.1.3 detalha a fase de análise. Como o foco desta dissertação é a fase de execução, a seção 2.2 retoma o tema dando enfoque à execução paralela de workflows científicos. Dentro deste tema, discute-se o escalonamento e

a paralelização de workflows na seção 2.2.1 e os desafios na paralelização de workflows na seção 2.2.2.

2.1 Ciclo de Vida de um Experimento Científico

O ciclo de vida de um experimento científico em larga escala pode ser dividido em três grandes fases: composição, execução e análise (Mattoso et al. 2010). Na composição, o cientista define os procedimentos do experimento em um nível abstrato, modelando o processo executado para verificar determinada hipótese. Durante a fase de execução o cientista configura o processo experimental definindo os aplicativos que serão executados, a ordem na qual serão executados e como eles devem ser configurados para executar. Na fase de análise, os resultados obtidos na fase de execução e a forma com que eles foram atingidos são analisados com o fim de avaliar a hipótese estudada. O resultado da análise pode ser conclusivo ou servir para realimentar o ciclo de vida gerando uma nova iteração do experimento. A Figura 1 apresenta um esquema gráfico do ciclo de vida segundo Mattoso et. al. (2010). Nas seções seguintes, será explicada em mais detalhes cada fase do ciclo de vida do experimento científico.

2.1.1 Composição

A composição é a fase onde o experimento é estruturado e configurado, onde fica estabelecida a ordem lógica entre atividades do experimento. Atividade é um conceito abstrato que define uma ação no contexto do experimento, que pode ser executada tanto pelo cientista quanto por um computador, por exemplo. Sendo assim, uma atividade pode possuir diferentes tipos de dados de entrada, ou parâmetros, para gerar diferentes tipos de dados de saída. A definição das atividades e dos tipos de entrada e saída de cada atividade é feita na composição do experimento. A ordem lógica que define a sequência na qual as atividades precisam ser executadas constitui um workflow científico daquele experimento. Tal workflow traduz o processo do experimento assim como um workflow é capaz de traduzir um processo de negócio, no meio empresarial. A estrutura do workflow facilita a visualização consistente e unificada do processo além de torná-lo mais simples de manter e modificar. A Figura 2 apresenta um exemplo de workflow científico. Na figura, cada retângulo cinza representa uma atividade. As duas primeiras atividades representam atividades de pré-processamento do experimento, enquanto a terceira é o núcleo a ser executado e a quarta é o pós-processamento.

Um novo workflow pode ser criado do zero ou nascer da extensão de outro workflow. Frequentemente, cientistas utilizam partes de workflows utilizados anteriormente ou estendem workflows de outros experimentos para dar apoio ao novo experimento. Sob esse ponto de vista, os workflows dos experimentos podem ser vistos como componentes reutilizáveis. A fase de composição, portanto, pode ser dividida em duas subfases: concepção e reutilização.

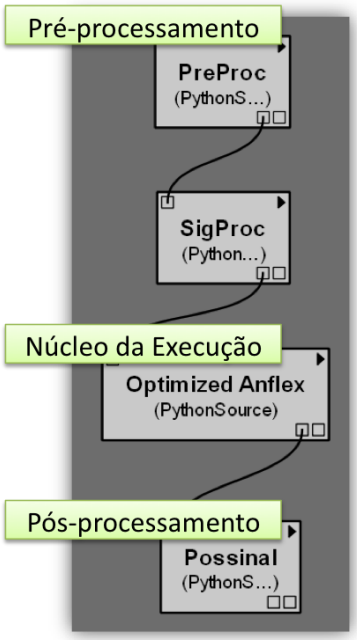

Figura 2: Exemplo de workflow científico

A concepção do experimento representa a modelagem do que seria o workflow de negócio do experimento, que representa o protocolo do experimento científico, i.e., o procedimento adotado para executá-lo. A partir deste workflow mais conceitual, pode-se obter os workflows abstratos, que agregam a informação de quais procedimentos ou programas de computador irão executar cada atividade. E com o workflow abstrato, pode-se ter o workflow concreto, que é um workflow pronto para execução e que agrega a informação de quais recursos computacionais (máquinas) executam qual atividade. A Figura 2 é um exemplo de workflow abstrato. Porém, as anotações de pré-processamento, núcleo da execução e pós-processamento são informações inerentes ao workflow conceitual do experimento.

Outro aspecto da fase de composição é a reutilização. A reutilização tem papel importante no ciclo de vida do experimento, pois permite o crescimento de um novo experimento ou workflow baseado em outros experimentos ou workflows já executados. De

acordo com os resultados de uma análise em um experimento, pode ser desejável modificar o workflow inicial ou realizar pequenas variações que agreguem algum diferencial no novo experimento. O intuito da reutilização é obter um novo workflow a partir de outros workflows já modelados. Uma opção é obter workflows abstratos e concretos a partir de um workflow conceitual se este estiver, por exemplo, na forma de uma linha de experimento (Ogasawara et al. 2009b). A linha de experimento, inspirada em técnicas de linhas de produtos de software (Bosch 2000, Northrop 2002), permite transformar o workflow conceitual inicial do experimento em um ativo reutilizável que pode gerar workflows concretos. De uma mesma linha de experimento podem ser derivados diferentes workflows, pois as atividades da linha possuem variabilidade, ou seja, podem ser implementadas por diferentes procedimentos, e opcionalidade, que permite que uma atividade ou mesmo trechos do workflow sejam ou não executados.

2.1.2 Execução

A partir dos workflows concretos obtidos ao final da fase de composição, pode-se dar início à fase de execução. A execução é responsável por executar o workflow concreto em uma máquina de workflow. A execução do workflow envolve a execução de cada atividade do workflow respeitando os relacionamentos de dependências entre elas. Uma atividade '*A'* é dependente de outra '*B'* se algum conjunto de dados de entrada de '*A'* for produzido por '*B'*. As relações de dependência garantem que o workflow será executado respeitando uma determinada ordem de atividades. O objetivo da execução do workflow é produzir os resultados que serão utilizados na fase seguinte de análise. A infraestrutura utilizada para executar o workflow deve ser capaz de executar as atividades, fazer o registro do histórico da execução (*log*) e realizar o monitoramento.

Algumas atividades de workflows científicos ou o próprio workflow pode ser muito custoso para ser executadas de maneira sequencial. Uma única atividade deste workflow pode gerar um grande número de instâncias envolvendo diferentes combinações de parâmetros. Cada uma dessas instâncias pode ser vista como uma tarefa do workflow. Sendo assim, é muito comum que workflows científicos ou parte deles sejam executados em recursos computacionais distribuídos, tais como clusters, grades ou nuvens. Sendo executado de maneira distribuída, o monitoramento da execução destes workflows é ainda mais importante, embora também mais complexo. É necessário guardar dados de qual trecho da atividade foi

executado por qual recurso e como decorreu a execução. Por tal motivo, a fase de execução pode ser subdividida em duas subfases: a distribuição e o monitoramento.

A distribuição da execução do workflow é inerente à máquina de execução do workflow e as máquinas de execução estão associadas a um determinado sistema de gerência de workflows científicos (SGWfC). Um SGWfC provê a construção de workflows científicos, interpreta o workflow descrito em uma determinada linguagem executando-o em uma máquina de execução (*engine*) de workflow. Existem muitos tipos de SGWfC com diferentes abordagens para modelar ou descrever os workflows e com métodos diferentes para sua execução.

Muitos SGWfC, tais como o VisTrails (Callahan et al. 2006) ou o Kepler (Altintas et al. 2004) são voltados para o desenvolvimento e execução local do workflow. Eles oferecem uma interface gráfica para modelagem visual dos workflows e oferecem componentes para criar atividades de diversos tipos como, por exemplo, atividades que invoquem um programa local ou atividades que invoquem um serviço web. Esses SGWfC possuem originalmente máquinas de execução de workflow científico locais, *i.e.*, eles irão processar cada atividade a partir da máquina do cientista. Estes sistemas permitiam invocar uma atividade remotamente, tal como um serviço web ou uma execução em um cluster ou grade, porém, a infraestrutura remota de execução fica desacoplada do controle interno do SGWfC e precisa estar previamente configurada para ser invocada pelo gerente. O foco da execução deste tipo de SGWfC é a execução local e o registro semântico dos workflows executados.

Outros SGWfC são focados na distribuição da execução de workflows tais como o DAGMan (Team 2005), Pegasus (Deelman et al. 2007) e o Swift (Zhao et al. 2007). Estes sistemas focam na execução distribuída, mas dão pouco apoio ao registro semântico a respeito das atividades do workflow e à facilidade de uso através de uma interface gráfica. Estes SGWfC utilizam, normalmente, linguagens de scripts e arquivos de configuração para descrever os workflows científicos, diferente dos SGWfC locais, que oferecem modelagem gráfica. Os gerentes de execução distribuída também costumam depender de uma infraestrutura orientada a computação em grade para submeter suas atividades distribuídas de maneira mais eficiente. Embora não seja uma restrição, tais sistemas tiram maior proveito se a infraestrutura de submissão utilizar um escalonador como o Condor (Couvares et al. 2007) ou o Falkon (Raicu et al. 2007). O Condor é um escalonador voltado a computação em grade. Já

o Falkon é um escalonador mais abrangente, porém ele necessita (Falkon Project 2010) da instalação do *middleware* de grade GT4 (Spooner et al. 2005) com uma autoridade certificadora (Welch et al. 2003). Essas restrições podem ser complicadores em cenários onde as máquinas para processamento utilizem um regime voltado à computação de alto desempenho com escalonadores tradicionais para clusters, tal como o Torque (Staples 2006), e não uma infraestrutura de grade.

Com o objetivo de unir os benefícios dos SGWfC locais com a execução distribuída em recursos computacionais de alto desempenho, existem soluções tais como o Hydra (Ogasawara et al. 2009a). O Hydra é um *middleware* que, a partir de um SGWfC local tal como o VisTrails, faz a distribuição de uma atividade do workflow em um recurso distribuído como, por exemplo, um cluster. O Hydra paraleliza uma atividade seguindo estereótipos de paralelismo de dados, tais como fragmentação de dados de entrada ou varredura de parâmetros. A fragmentação de dados envolve a divisão de um arquivo de entrada maior em arquivos menores que serão processados individualmente de maneira paralela. Cada tarefa gerada consumirá um fragmento do arquivo original e produzirá um resultado. Já a varredura de parâmetros se refere à execução de um dado programa utilizando diferentes combinações de conjuntos de parâmetros. Cada tarefa irá executar o programa com um conjunto de parâmetros diferentes e irá gerar um resultado. No final, os resultados de cada tarefa serão agregados em um resultado único da atividade.

Sendo assim, a partir da atividade descrita e configurada pelo cientista no SGWfC com a definição do conjunto de dados de entrada, o Hydra é capaz de gerar um conjunto de tarefas para processar. A descrição de uma atividade paralela em um SGWfC envolve a descrição de uma atividade simples de workflow, que seria o aplicativo ou serviço que será executado pela atividade, suas entradas e saídas esperadas. Além disso, é necessário informar o conjunto de parâmetros, ou seja, os parâmetros que serão consumidos por cada execução. Se a execução envolver uma fragmentação de dados, o método de fragmentação também precisa ser informado. Cada tipo de arquivo pode sofrer uma fragmentação diferente (a fragmentação de um arquivo binário é diferente da fragmentação de um XML, por exemplo), portanto, o método de fragmentação precisa ser fornecido pelo cientista. Com a descrição da atividade paralela, o Hydra é capaz de gerar as tarefas e executá-las em um cluster realizando o registro de proveniência distribuída.

Assim como o Falkon (Raicu et al. 2007), o Hydra também distingue o processo de aquisição de recursos do processamento de tarefas. Isto significa que o Hydra requisita um dado número de recursos computacionais ao cluster e após adquirir o direito de utilizar estes recursos, distribui o processamento das tarefas dentre os núcleos de processamento disponíveis nos recursos. Essa abordagem é vantajosa, pois diminui a sobrecarga gerada se as tarefas fossem individualmente submetidas ao escalonador do cluster.

A etapa de monitoramento do ciclo de execução do experimento permeia o registro de tudo que foi executado e tudo o que está sendo executado no momento. Tempo de execução, qual recurso executou qual tarefa, resultados obtidos, erros e exceções ocorridos na execução são exemplos de informações do experimento importantes para a fase de análise. A informação de quais tarefas executaram conforme o esperado e quais falharam também é essencial para prover qualidade de serviço. Entretanto, poucos sistemas utilizam esses dados para reescalonar tarefas automaticamente. Embora a etapa de monitoramento seja a mais simples de ser explicada, ela é de grande importância.

Quando uma atividade do workflow é executada de maneira distribuída, pode não ser trivial tratar o monitoramento distribuído das tarefas. Deve-se levar em consideração aonde esses dados serão gravados e possíveis problemas de concorrência para escritas em disco ou bancos de dados. O número de tarefas sendo executadas simultaneamente é extremamente alto, da ordem de milhares podendo chegar a centenas de milhares, e se o registro do monitoramento for feito individualmente por cada tarefa, pode trazer sobrecarga na gerência de recursos compartilhados. Sendo assim, é necessário utilizar algoritmos distribuídos eficientes, um mecanismo hierárquico ou com papéis definidos na rede de recursos de execução para proporcionar acesso eficiente aos recursos de armazenamento.

2.1.3 Análise

A fase de análise do experimento científico está associada aos resultados obtidos na execução e à chamada base de proveniência (Stevens et al. 2007) do experimento. A proveniência prospectiva do experimento registra as decisões e parâmetros definidos na fase de composição enquanto a proveniência retrospectiva é o registro das decisões ocorridas e resultados obtidos durante a execução do experimento. Tal informação coletada é vital para o processo sistemático experimental e fundamental para a fase de análise. A construção da base

de proveniência permeia as fases de composição e execução e é fundamental para garantir a reprodutibilidade do experimento, pois guarda uma forma de histórico do que foi feito e como foi feito durante todo o ciclo de vida do experimento.

Após a execução do experimento científico, o cientista analisa os resultados obtidos para verificar se estes são satisfatórios e suficientes para confirmar ou refutar sua hipótese. Além dos resultados obtidos diretamente da execução, o cientista também pode buscar correlações dos resultados com decisões tomadas na fase de composição e relações com os dados de proveniência armazenados. Todos esses procedimentos ocorrem na fase de análise do experimento científico.

A análise pode refutar a hipótese do experimento em questão. Nesse caso, o cientista poderá reformular a hipótese e executar um novo experimento reiterando o ciclo de vida do experimento. Mesmo se a análise confirmar a hipótese do experimento, o cientista não afirma com total acurácia que aquela hipótese é verdadeira, pois o experimento pode estar incorreto ou não ter levado todos os fatores necessários em consideração. Sendo assim, embora os resultados sejam relevantes, tal experimento precisa ser passível de reprodução, de forma que outros cientistas possam executá-lo novamente, possivelmente realizando os ajustes desejados e então obter novos resultados que corroborem com a confirmação da hipótese ou então refutem o que antes havia sido confirmado.

O processo de análise envolve um conjunto de consultas à base de proveniência e aos resultados com possíveis visualizações gráficas para facilitar o entendimento do conjunto de dados obtidos. Sendo assim, a fase de análise pode ser dividida em duas subfases: Consulta & Visualização e Descoberta.

O processo de consulta e visualização no ciclo de vida do experimento está relacionado ao estudo dos resultados obtidos na execução do experimento e aos dados de proveniência com o objetivo de atingir a conclusão do estudo. Ou seja, é nesse ponto que se chegará à conclusão a respeito da hipótese do experimento. Existem diversos métodos para consulta dos resultados e dados de proveniência do experimento tais como as consultas preliminares, as consultas de análise e as consultas investigativas (*follow-up queries*) (Ioannidis e Wong 1987 apud Mattoso et al. 2010). Através das consultas já pode ser possível chegar às conclusões do experimento, porém, em função do grande volume de dados gerados

nos experimentos em larga escala, é comum a utilização de técnicas de visualização científica para realizar melhor análise do conjunto de dados e suas características. Através de gráficos, mapas, imagens e vídeos, o cientista pode verificar comportamentos, tendências e características em geral dos resultados como um todo.

A descoberta dentro do ciclo de vida do experimento está fortemente associada à base de proveniência. É um processo que irá buscar padrões e características navegando nos dados e seus relacionamentos na base de proveniência. Exemplos de consultas com o intuito de descoberta acerca da proveniência são (Mattoso et al. 2010): "Quantos workflows concretos distintos utilizaram a versão X de uma aplicação legada em um determinado experimento?", "Quantos processadores foram utilizados na execução paralela do workflow Z?", "Quantos workflows foram utilizados no experimento Y?".

Um problema que envolve a fase de descoberta são as diferentes propostas de armazenamento de dados de proveniência com as diferentes formas de linguagem de busca. Recentemente, a comunidade vem se esforçando para fazer do modelo aberto de proveniência (*Open Provenance Model* ou OPM) (Moreau et al. 2008) um padrão, porém poucas soluções adotam o modelo nativamente. Alguns SGWfC permitem importar e exportar os seus dados de proveniência na forma do OPM. Entretanto, o intercâmbio de dados entre SGWfC diferentes através da exportação/importação de dados no formato OPM ainda não funciona como o esperado. Também não se chegou a um consenso sobre a melhor linguagem de busca sobre dados de proveniência, até porque a linguagem depende, em geral, da forma na qual os dados estão armazenados. Por exemplo, se base de dados for relacional, pode-se usar uma linguagem como a SQL. Porém, se a base estiver em XML, alguma linguagem como a XQuery (Chamberlin 2003) precisa ser utilizada.

2.2 Execução paralela de workflows científicos

Dentro do ciclo de vida do experimento científico, as estratégias de paralelismo de workflows estão mais ligadas à fase de execução. O escalonamento de um workflow significa que uma atividade de um workflow foi agendada para executar em um recurso externo tal como um cluster ou grade. A paralelização do workflow significa que a execução de uma ou mais atividades do workflow é feita em paralelo para acelerar o processo. Entretanto, mesmo quando uma atividade é escalonada para executar em um recurso computacional externo, o

controle sobre a execução do workflow é detido pelo SGWfC. Existem diferentes abordagem para o paralelismo ou execução remota de atividades de workflows, de acordo com a política do SGWfC. As subseções seguintes discutem essa e outras questões envolvendo a paralelização de workflows científicos em mais detalhes.

2.2.1 Escalonamento e Paralelização de Workflows

Experimentos científicos *in silico* são frequentemente modelados como workflows científicos e gerenciados por SGWfC. Cada SGWfC tem suas características particulares, notação e linguagem. Eles focam em diferentes recursos tais como visualização científica, proveniência ou execução paralela. Quando um workflow é modelado em um determinado SGWfC, o workflow fica dependente de questões inerentes àquela tecnologia do SGWfC. Além disso, a representação em um SGWfC em particular pode não deixar claro o conhecimento por trás do workflow modelado. Durante um experimento, se o cientista precisar executar alguma atividade com alto desempenho, ele poderá ter que remodelar seu workflow em um novo SGWfC e provavelmente modificar o código fonte da atividade para obter o nível desejado de paralelismo e melhora de desempenho. Entretanto, remodelar um workflow requer esforço e também pode gerar erros. Os cientistas podem não estar familiarizados com linguagens de programação e métodos de paralelização. Por tais motivos, acreditamos que a execução paralela de workflows deve ser feita de maneira implícita. Uma abordagem implícita tenta fazer a execução paralela do workflow de maneira mais transparente para o cientista sem alterar as aplicações. Isto significa que a abordagem deve ser não invasiva, de forma que não altere o código fonte da aplicação. Isto é muito importante já que muitas aplicações científicas têm código complexo e legado (Chan e Abramson 2008), que é muito custoso de ser modificado. A aplicação também pode ser proprietária, o que significa que o cientista não tem acesso ao código fonte. Porém, mesmo sem acessar o código, é ainda possível executar essas aplicações em paralelo utilizando, por exemplo, o modelo *bag of tasks* (Cirne et al. 2004). O modelo *bag of tasks* diz respeito à execução de muitas tarefas, geralmente desacopladas, em recursos distribuídos. Entretanto, a execução dessas tarefas não preza o alto desempenho e sim a alta vazão da execução. Um paradigma novo de computação voltado a execução de muitas tarefas, denominado *Many-Task Computing*, ou computação de muitas tarefas, engloba o modelo *bag of tasks* e preza o alto desempenho na execução. Dessa forma, tal modelo parece interessante para o processamento paralelos de workflows.

O paradigma *Many-Task Computing* (Raicu et al. 2008) ou MTC é estabelecido entre a computação de alto desempenho (*High Performance Computing* ou HPC) e a computação de alta vazão (*High Throughput Computing* ou HTC). De maneira simplificada, o MTC costuma ter um volume de tarefas semelhante ou maior que o HTC, mas as suas tarefas demandam um desempenho semelhante às tarefas HPC. Por tal motivo, o MTC é considerado um paradigma intermediário e oferece flexibilidade na execução paralela. A flexibilidade está relacionada ao fato do cientista não precisar, por exemplo, programar sua atividade usando uma interface voltada ao HPC tal como o MPI (Gropp 2001) ou OpenMP (Chandra 2001) e ao mesmo tempo não precisar utilizar uma infraestrutura como uma grade, que, geralmente, está associada a um processo burocrático rigoroso. O paralelismo no MTC está normalmente associado à execução de um mesmo procedimento com diferentes conjuntos de dados ou parâmetros. Sendo assim ele está fortemente associado ao paralelismo de dados. Portanto, a granularidade do paralelismo da execução MTC é externa às tarefas e não interna. Vale ressaltar que uma tarefa MTC também pode ser internamente paralela, utilizando MPI, por exemplo. Nesse caso, ter-se-iam dois níveis de paralelismo, o paralelismo interno da tarefa e o paralelismo externo no processamento simultâneo de múltiplas tarefas. O acoplamento entre tarefas MTC é normalmente baixo e a troca de informações entre tarefas é, em geral, feita por escrita e leitura de arquivos. O MTC também preza a execução dessas muitas tarefas em ambientes computacionais de alto desempenho, porém utilizando um escalonador de tarefas intermediário, que obtenha os recursos através do gerente de recursos nativo, mas gerencie a execução de suas tarefas separadamente.

Tanto o Swift/Falkon quanto o Hydra, são soluções orientadas ao MTC. Outros trabalhos também vêm explorando as características do MTC em suas soluções. O Kestrel (Stout et al. 2009), por exemplo, é um arcabouço para execução de aplicativos MTC que utiliza o protocolo XMPP para melhorar o sistema de tolerância a falhas em ambientes de processamento distribuído. Outra proposta é o Sphere (Yunhong Gu e Grossman 2008), um sistema de computação em nuvem sobre uma rede de longas distâncias (WAN) para processamento de aplicativos que demandem paralelismo de dados. Ambas as propostas consideram heterogeneidade da rede, o balanceamento de carga e a tolerância a falhas, contudo não assumem que o aplicativo distribuído é parte de um workflow científico, o que reduz o grau de controle sobre um experimento sendo executado como um todo.

O framework GXP (Dun et al. 2010) oferece o processamento paralelo de workflows MTC através de um sistema de *shell* paralelo (GXP parallel shell), um sistema de arquivo distribuído (GMount) e uma máquina de workflow (GMake) que utiliza arquivos *make* para descrever workflows científicos. Embora ofereça uma maneira simples de executar workflows em ambientes distribuídos, o GXP não proporciona uma modelagem visual do workflow e não oferece um sistema adequado de coleta de proveniência, o que pode comprometer o ciclo de vida do experimento.

A abordagem posposta nesta dissertação traz transparência à execução paralela de atividades de workflow, mas o controle sobre as execuções do workflow permanece com os SGWfC. Cientistas podem permanecer utilizando seus SGWfC prediletos para escalonar seus workflows em clusters através de um escalonador MTC tal como o Falkon ou o Hydra. O escalonador pode usar a abordagem Heracles para lidar com a execução da atividade em paralelo trazendo transparência, balanceamento de carga e tolerância às falhas na execução.

Durante o ciclo de vida do experimento científico, o mesmo experimento modelado como workflows científicos pode ser executado diversas vezes explorando diferentes conjuntos de parâmetros ou dados de entrada. É geralmente possível rodar múltiplas combinações de parâmetros simultaneamente, em paralelo, designando instâncias da atividade do workflow para cada máquina. Este cenário é conhecido como varredura de parâmetros (Smanchat et al. 2009, Walker e Guiang 2007). A fragmentação de dados também é usada em workflows científicos para aumentar o desempenho da execução de experimentos, já que a entrada de dados de determinadas aplicações pode ser fragmentada em pedaços menores que serão processados mais rapidamente pela aplicação. Portanto, processar fragmentos em paralelo leva a uma execução global mais veloz. Cada instância dessas atividades que processam um diferente grupo de parâmetros ou fragmento de dados pode ser vista como uma tarefa. Tanto a varredura de parâmetros quanto a fragmentação de dados podem ser vistas como abordagens de paralelismo de dados científicos.

Outra abordagem popular é o modelo de programação MapReduce. Este modelo é explorado no escalonamento de workflows como um caso particular de paralelismo de dados. Basicamente, um grande conjunto de dados é repartido em pedaços menores que são mapeados para serem processados nos nós de processamento. Estes pedaços são direcionados para a função de fragmentação (Splitter) ou função Map como pares de (chave, valor). Depois

que a função Map é executada, os valores intermediários para uma determinada chave de saída são agregados em uma lista por uma função de agregação (denominada Reduce). A função Reduce combina os valores intermediários em um ou mais resultados finais relacionados a uma determinada chave de saída.

Alguns trabalhos relacionados na literatura propõem soluções gerais para gerência da execução de workflows em grandes clusters. Um exemplo é o GlideinWMS (Bradley et al. 2010). Ele é um sistema de gerência de workflow (SGW) de uso geral que funciona sobre o Condor com código adicional específico do GlideinWMS. Outra abordagem é o Corral (Juve et al. 2010), um sistema que executa um conjunto de aplicações baseadas em workflows reais voltados a problemas da astronomia, ciência dos terremotos e genética. A reserva de recursos do Corral antes da execução do workflow reduziu o tempo de execução de uma aplicação de astronomia. Embora essas abordagens proponham soluções para a gerência de workflows em ambientes distribuídos, eles ainda não proporcionam uma abordagem transparente para submissão, execução e coleta de dados de proveniência de atividades de workflow. Eles também não seguem o paradigma MTC para prover paralelismo não invasivo. Sendo assim, baseados nos estudos discutidos, acredita-se que ainda há desafios em aberto para melhorar o paralelismo na execução de workflows científicos.

2.2.2 Desafios na paralelização de workflows

Durante a configuração do experimento, quando os cientistas configuram como as instâncias do workflow serão executadas, especificar fatores técnicos que definem como a infraestrutura de processamento deve lidar com a execução das atividades pode gerar erros na execução. A execução paralela das atividades de workflow deve permanecer transparente para os cientistas assim como o processamento de consultas em sistemas de bancos de dados distribuídos (Ozsu e Valduriez 1999).

Geralmente, em um ambiente paralelo, o cientista precisa especificar o número de processadores que deseja utilizar para processar suas tarefas. Entretanto, é mais transparente para o cientista se ele puder somente especificar o tempo limite para obter os resultados. Esta abordagem se adequa ao modelo de computação utilitária aonde os usuários pagam apenas por aquilo que utilizam (Yu et al. 2005). Os cientistas especificariam somente o que eles querem respondido e em quanto tempo. O SGWfC submeteria a atividade para o escalonador que

alocaria os recursos sob demanda e gerenciaria eventuais falhas automaticamente. Portanto, para atender as restrições de tempo definidas pelo cientista, o escalonador deve ser capaz de gerenciar os recursos disponíveis dinamicamente. Uma abordagem inicial seria colocar um módulo especial nos nós de serviço (ou *head node*) do cluster para controlar o escalonamento e execução das tarefas de workflows dinamicamente. Entretanto, trabalhos anteriores (Sonmez et al. 2010) notaram que os nós de serviço dos clusters ficam sobrecarregados em função do grande número de tarefas de workflows e transferência de arquivos que eles tem que gerenciar em um cenário de escalonamento dinâmico. Sendo assim, melhores estratégias para prover a execução paralela de workflows científicos transparentes e confiáveis ainda são necessárias.

O foco desta dissertação é a fase de execução do experimento científico, envolvendo a distribuição e monitoramento. Entretanto, decisões tomadas na fase de composição podem influenciar no processo de execução como, por exemplo, a escolha da abordagem de paralelização. Além disso, os resultados da execução e os dados de proveniência armazenados são essenciais para a fase de análise. O objetivo da proposta Heracles é trazer transparência na fase de execução utilizando os dados da composição para escolher a abordagem de paralelização e prover os resultados para a análise com qualidade de serviço. Além disso, o Heracles preza uma computação paralela eficiente, buscando estipular a quantidade de recursos para cada atividade de acordo com sua necessidade.

Capítulo 3 – Conceitos de Rede Peer-to-Peer

Como parte do objetivo desta dissertação é analisar o potencial de técnicas P2P para aprimorar a paralelização de atividades de workflows científicos que demandem a computação de muitas tarefas (MTC) em grandes clusters, apresentamos conceitos e técnicas de sistemas P2P (Oram 2001). Sendo assim, apresentamos uma discussão sobre as tecnologias P2P existentes para avaliar o que é vantajoso para ser utilizado em ambientes de larga escala como clusters. Também é necessário entender os mecanismos que as redes P2P utilizam para tratar os requisitos que sejam importantes no contexto de workflows científicos, tais como: transparência, balanceamento de carga e qualidade de serviço. Neste capítulo são discutidos os conceitos de classificação dos diferentes tipos de redes P2P, mostra-se um estudo qualitativo a respeito das características de diferentes abordagens P2P e são levantadas as principais técnicas e conceitos necessários relevantes ao cenário de execução de workflows científicos.

3.1 Tipos de sistemas P2P

Sistemas P2P podem ser classificadas quanto ao grau de centralização ou a estrutura de sua rede (Lua et al. 2005). Sendo assim, um sistema P2P pode ser classificado como centralizado ou descentralizado. Porém, a rede P2P também pode ser classificada como estruturadas ou não estruturada. Sistemas P2P centralizados possuem um ou mais servidores (também chamados de *super-peer*s (Lua et al. 2005)) responsáveis por armazenar informações sobre os nós ativos na rede e responder às requisições tais como a entrada de novos nós da rede. Em sistemas P2P de compartilhamento de dados, os servidores centrais também centralizam serviços de indexação e busca e recuperação de arquivos presentes na rede P2P. A Figura 3 exemplifica um esquema de rede P2P centralizada. No exemplo, há dois servidores que podem se comunicar para intercâmbio de dados (linha pontilhada). Os *peers* registram-se nos servidores (linha contínua) e acessam os serviços da rede também através deles. Por exemplo, ao buscar por um arquivo, os *peers* direcionam as buscas ao servidor que retorna os resultados baseados no índice que ele guarda dos arquivos presentes na rede. Ao obter o resultado da busca, um *peer* pode estabelecer uma conexão direta com outro *peer* para troca de dados (linha com setas).

Nos sistemas descentralizados não há *super-peers*, de maneira que as consultas são propagadas diretamente entre os nós, utilizando dados de vizinhança. Enquanto os sistemas centralizados facilitam o acesso aos serviços da rede, os sistemas descentralizados oferecem maior balanceamento de carga e ausência de pontos únicos de falha, presentes nos sistemas centralizados.

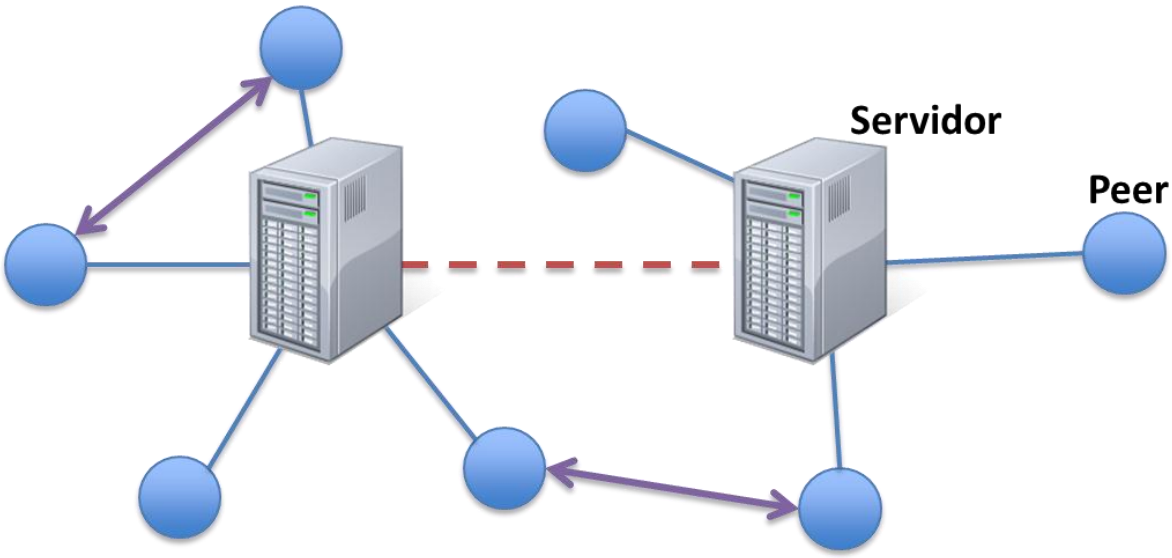

Figura 3: Esquema exemplificando uma rede P2P centralizada.

Com relação à estrutura das redes P2P, as redes não estruturadas são caracterizadas por um processo de construção aleatório. Assim, as consultas são realizadas através do método de inundação, sendo propagadas até que encontrem um *peer* satisfatório ou ultrapassem o limite máximo de saltos na rede. A Figura 4 apresenta um exemplo de rede P2P descentralizada e desestruturada. Nela, os *peers* são autônomos e guardam dados da vizinhança (linha contínua). O acesso aos serviços da rede, tais como as buscas e indexação de arquivos são feitas através de algoritmos de inundação que fluem por rotas definidas pelas informações de vizinhança. O intercâmbio de dados entre *peers*, entretanto, pode ocorrer diretamente, seja entre vizinhos ou entre *peers* que foram atingidos pelo algoritmo de inundação (linha com seta).

As redes estruturadas já possuem uma topologia bem definida, construída através de algoritmos determinístico que otimizam o processo de busca e recuperação de recursos. Um exemplo de redes estruturadas são as baseadas em tabelas *hash* distribuídas ou DHT (Buchmann e Bohm 2004), utilizadas para prover a organização da rede e mapear as chaves e

os endereços IP dos *peers* em um mesmo espaço de identificadores. Exemplos de sistemas P2P estruturados que utilizam DHT são os sistemas Chord (Stoica et al. 2003), Pastry (Rowstron e Druschel 2001) e Tapestry (Zhao et al. 2001).

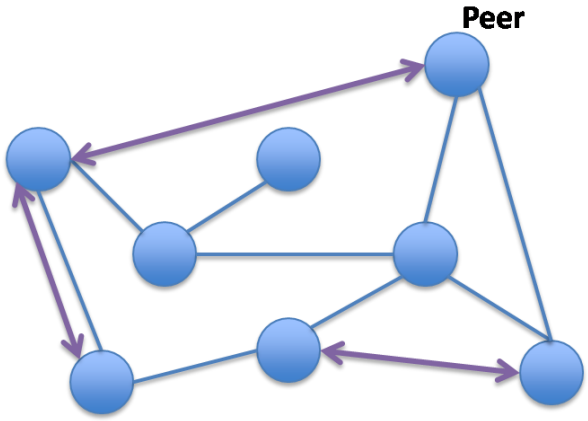

Figura 4: Exemplo de uma rede P2P descentralizada

Com o objetivo de unir características das diferentes abordagens P2P surgiram os sistemas P2P híbridos. Os sistemas híbridos buscam conjugar características de diferentes abordagens com o intuito de balancear algumas características. Os sistemas FastTrack (Liang et al. 2006), por exemplo, utilizam uma rede descentralizada e não estruturada. Porém, alguns dos *peers* são eleitos *super-peers* e formam, entre si, uma rede estruturada. Tais *super-peers* armazenam metadados da rede e facilitam os mecanismos de busca e indexação. Embora os *super-peers* não sejam essenciais para o funcionamento da rede FastTrack, eles tornam o sistema muito mais escalável. Aplicações conhecidas como o Kazaa (Kazaa 2011) utilizam o FastTrack. Outra abordagem híbrida conhecida é o protocolo BitTorrent (Lua et al. 2005) para transferências de arquivos. Ele utiliza uma infraestrutura centralizada, porém oferece um sistema de descoberta de novos *peers* através de uma estrutura DHT baseada na abordagem descentralizada Kademlia (Meshkova et al. 2008).

As abordagens P2P hierárquicas (Ganesan et al. 2004, Garcés-Erice et al. 2003, Martinez-Yelmo et al. 2008), por exemplo, são soluções híbridas com o objetivo de reduzir a complexidade e custo de manutenção de sistemas P2P descentralizados de larga-escala. Aplicando o conceito de "dividir para conquistar", esta abordagem cria grupos de *peers* dentro da rede P2P. Dentro de cada grupo, os *peers* formam uma rede descentralizada. Entretanto, alguns *peers* do grupo podem estabelecer comunicação com *peers* de outro grupo de forma

que funcionem como um *super-peer* daquele grupo. Se uma dada requisição não puder ser resolvida internamente em um grupo, o *peer* responsável pela comunicação com outros grupos encaminha a requisição para ser resolvida externamente. Como a criação dos grupos é feita de maneira hierarquizada, um grupo é criado na hierarquia mais baixa e os *peers* responsáveis pela comunicação externa se conectam aos *peers* de uma camada hierárquica acima. Ao final da construção da rede, fica estabelecida uma estrutura hierárquica onde funcionam sistemas P2P descentralizados e possivelmente desestruturados no interior dos grupos com uma rede estruturada funcionando entre os grupos através dos elementos *super-peer* de cada grupo. Vale ressaltar que os *super-peers* da rede hierárquica não centralizam os serviços da rede como os *super-peers* das redes P2P centralizadas, eles apenas funcionam como mecanismos de intercâmbio de dados entre grupos (Garcés-Erice et al. 2003) e não são responsáveis por serviços de busca e indexação. A Figura 5 apresenta um exemplo de rede P2P hierárquica. Os círculos grandes azuis representam os grupos formados pelos *peers* da rede. Os círculos pequenos azuis claro são *peers* comuns e se comunicam com os outros *peers* daquele grupo em uma rede descentralizada interna. Os *peers* laranja representam os líderes dos grupos, que formam uma rede estruturada hierárquica com os outros líderes da rede.

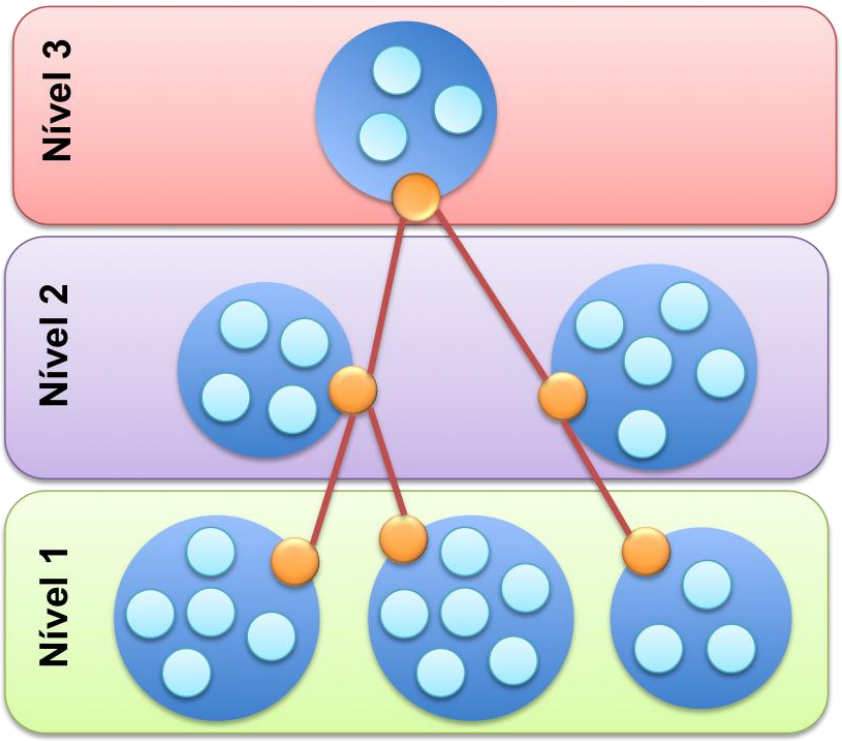

Figura 5: Exemplo de uma rede P2P hierárquica.

3.3 Avaliação das soluções P2P

Existem vários aspectos que devem ser considerados ao optar por uma abordagem P2P capaz de apoiar a execução de atividades de workflow de maneira distribuída e eficiente. Uma única atividade de workflow pode gerar milhares de tarefas que serão distribuídas por meio da rede P2P, seja ela uma rede P2P real ou sob a rede de um cluster. Portanto, o fator balanceamento de carga é muito importante para manter a rede estável sem a sobrecarga de qualquer um dos nós do cluster. A escalabilidade também é muito importante já que um cluster de larga escala tem milhares de nós e cada processador deste cluster será visto como um *peer* da rede. Sistemas P2P devem ser tolerantes a falhas, porém os eventos de entradas e saídas de nós da rede são previstos e conhecidos como eventos de falha, ou *churn* (Wu et al. 2008). Sendo assim, o impacto que um evento de falha causa na rede P2P no cenário de pior caso também deve ser considerado. O custo de manutenção da topologia P2P também é um aspecto importante pois afeta a troca de mensagens entre os nós, o que pode gerar uma sobrecarga desnecessária em um cluster, por exemplo. A Tabela 2 apresenta uma análise (Dias et al. 2010a) considerando os diferentes tipos de abordagens P2P já discutidas e os seguintes fatores: (i) balanceamento de carga, (ii) escalabilidade, (iii) impacto da falha e (iv) custo de manutenção. Esta analise busca refletir o conteúdo apresentado em diferentes trabalhos que discutem as abordagens P2P (Garcés-Erice et al. 2003, Lua et al. 2005, Valduriez e Pacitti 2005, Xie et al. 2008). Os sinais de positivo e negativo indicam se a característica naquele contexto é boa ou ruim, respectivamente. A ausência do sinal indica imparcialidade.

Tabela 2: Análise das abordagens P2P existentes

Abordagem / Fator	**Centralizada**	**Descentralizada**	**Hierárquica**
Balanceamento de carga	Baixo (-)	Alto (+)	Moderado
Escalabilidade	Baixa (-)	Moderada	Alta (+)
Impacto da falha	Alto (-)	Baixo (+)	Moderado
Custo de manutenção	Baixo (+)	Alto (-)	Moderado

A abordagem P2P centralizada não tem balanceamento de carga já que os *super-peers* centralizam os serviços da rede. A abordagem descentralizada distribui todo o controle da rede pelos nós, provendo um alto balanceamento de carga. A abordagem hierárquica centraliza parte dos serviços nos nós de comunicação entre grupos, provendo um balanceamento de carga moderado, já que existe uma leve sobrecarga do *super-peer* do grupo. A abordagem centralizada depende das capacidades dos nós centrais, portanto, não é muito escalável. A abordagem descentralizada tem uma escalabilidade moderada já que é complexo manter uma rede grande com um controle completamente distribuído. Já a abordagem hierárquica escala melhor já que atribui parte do controle aos nós *super-peer* dos grupos.

Com relação aos riscos ou impactos das falhas, em uma rede centralizada, se uma falha ocorre em um nó central (ou *super-peer*), toda a rede falha. Em uma rede descentralizada, a falha é pontual, já que os nós são completamente independentes. Em uma rede P2P hierárquica, se uma falha ocorrer em um *super-peer* do grupo, o grupo como um todo falha. O custo de manutenção de uma rede centralizada é baixo, já que toda a informação está contida nos nós centrais e pode ser atualizada diretamente neles. Na abordagem descentralizada, porém, a informação é distribuída pelos nós e as atualizações envolvem algoritmos de inundação. Na abordagem hierárquica, os *super-peers* dos grupos mantêm alguma informação sobre o seu grupo e a forma de se conectar a outros grupos. Processos de inundação para atualizar dados ocorrem somente no interior dos grupos.

Considerando estes aspectos, acredita-se que a abordagem P2P hierárquica seja a solução mais adequada para a execução de workflows científicos em redes de larga escala como clusters, pois oferece a melhor relação de custo benefício dentre os fatores analisados. Entretanto, ainda é necessária a realização de estudos que mostrem os benefícios trazidos pelos sistemas P2P no processamento de tarefas de workflow em clusters.

3.2 P2P na computação científica

Após o sucesso das redes de compartilhamento de arquivos, a tecnologia P2P permaneceu como uma alternativa às soluções tradicionais cliente-servidor. Ao mesmo tempo, Rodrigues e Druschel (2010) mostram que os sistemas P2P potencializam a inovação e podem ser combinados com soluções centralizadas tradicionais para criar soluções altamente escaláveis e seguras. Além disso, proporcionam uma solução barata para serviços

experimentais inovadores. Os autores ressaltam que a maior força do P2P é o fato dele ser independente de uma infraestrutura dedicada com controle centralizado. Porém, isto também é uma fraqueza, já que poder gerar desafios na área técnica, comercial e jurídica. O crescente compartilhamento ilegal de arquivos de música através de aplicativos como o Kazaa (Kazaa 2011) e o eMule (eMule 2010) é um exemplo clássico dos desafios jurídicos gerados pela tecnologia P2P. Do ponto de vista técnico, incorporar tecnologias P2P em soluções de armazenamento de arquivos e processamento de tarefas pode trazer desafios na área de segurança, confidencialidade e tarifação.

Na computação científica, diversos trabalhos na área de computação distribuída vêm explorando conceitos e técnicas P2P para buscar maior eficiência e escalabilidade. Ames et al. (2006) utilizam protocolos P2P para construção de um grid MultiCluster de alto desempenho. Briquet et al. (2007) apresentam um modelo de escalonamento de grandes volumes de tarefas com intensa utilização de dados em grades P2P utilizando o protocolo BitTorrent para realizar a transferência de dados. Transferências de grandes volumes de dados através deste protocolo mostram-se mais eficientes do que as soluções tradicionais. O SwinDeW (Jun Yan et al. 2006) é um sistema de gerência de workflows descentralizado baseado em técnicas P2P. Nele, tanto os dados utilizados pelo workflow modelado quanto o controle sobre a execução são distribuídos de forma que as funções do workflow são obtidas através da comunicação e coordenação direta entre os *peers* (ou recursos da rede) relevantes. Com esta abordagem, gargalos de desempenho presentes nas soluções centralizadas tendem a ser eliminados, assim como maior resistência a falhas e melhor escalabilidade. Vale ressaltar que soluções totalmente descentralizadas como o SwinDeW podem trazer dificuldades na gerência e monitoramento da execução das atividades distribuídas do workflow.

Ranjan et al. (2008) utilizam um espaço de coordenação P2P dentro de uma grade computacional para coordenar o escalonamento de aplicações de workflow. O trabalho propõe um algoritmo de escalonamento cooperativo e descentralizado para workflows. As funcionalidades chave do sistema tal como descoberta de recursos e coordenação do escalonamento são delegados ao espaço de coordenação P2P. A factibilidade da abordagem é provada com um estudo de simulação. Recentemente, iniciativas como SciMule (Ogasawara et al. 2010) simulam a paralelização de atividades de workflows científicos em redes P2P. O

SciMule possui uma arquitetura voltada a execução de tarefas e recuperação de falhas usando um rede P2P híbrida.

Os estudos voltados à aplicação de técnicas P2P no processamento de atividades de workflow são voltados para ambientes em grade ou redes P2P. Entretanto, muitos cenários da computação científica utilizam ambientes de processamento do tipo cluster. Embora seja possível instalar um *middleware* de grade em um cluster, isso pode ser um atrapalho na política local do laboratório ou centro de pesquisa. Outro fator importante é o registro da proveniência distribuída que traz uma série de desafios envolvendo o armazenamento de grandes volumes de dados de maneira concorrente. Ainda faltam estudos que apliquem os conceitos e técnicas P2P em clusters grandes para avaliar se uma abordagem com controle descentralizado traz benefícios no processamento de quantidades massivas de tarefas em clusters. Desafios relacionados ao balanceamento de carga, tolerância a falhas, registro de proveniência distribuída e configuração de atividades paralelas de workflows são candidatos a serem tratados por abordagens P2P.

Capítulo 4 – Processamento Distribuído com o Heracles

Baseado nos estudos apresentados anteriormente, analisando seus avanços e limitações, esta dissertação propõe uma nova abordagem denominada Heracles. O nome Heracles foi escolhido baseado na mitologia grega. Segundo a mitologia, um dos doze trabalhos de Heracles foi vencer a Hidra de Lerna, uma serpente com corpo de dragão e nove cabeças. Durante a batalha com a Hidra, ao cortar uma de suas cabeças, mais cabeças surgiam do ferimento. Como o Heracles foi idealizado no contexto do desenvolvimento do middleware Hydra (Ogasawara et al. 2009a), achamos a associação interessante como analogia à gerência dinâmica de recursos. O objetivo do Heracles é aprimorar a paralelização de atividades de workflows científicos que demandem a computação de muitas tarefas (MTC) em grandes clusters. Para atender este objetivo, o Heracles utiliza um conjunto de técnicas e conceitos P2P para prover a gerência dinâmica de recursos, balanceamento de carga e qualidade de serviço. A abordagem Heracles foi planejada para dar apoio a grandes sistemas de computação em clusters que normalmente têm procedimentos difíceis para configuração de tarefas a serem submetidas (configuração de *jobs*), controle centralizado e falhas de hardware frequentes. O intuito do Heracles é tornar o processo de execução paralela de workflows científicos em clusters mais transparente para os cientistas.

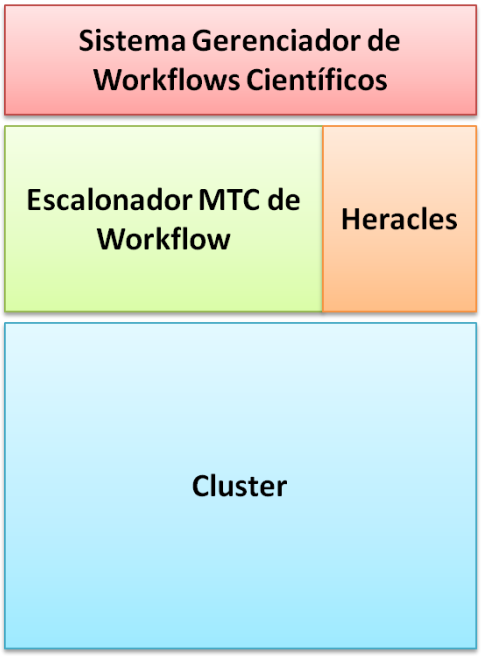

Figura 6: Visualização conceitual da estrutura do Heracles

A abordagem Heracles foi concebida para ser implementada dentro de um escalonador MTC de workflows (tal como o Falkon (Zhao et al. 2007) ou o Hydra (Ogasawara et al. 2009a)) para controlar a paralelização de atividades de workflow. A integração com o SGWfC é uma tarefa do escalonador. No caso do Falkon, a integração é feita diretamente com o Swift (Zhao et al. 2007), enquanto o Hydra já oferece a integração com outros SGWfC tradicionais tais como o VisTrails (Callahan et al. 2006) e o Kepler (Altintas et al. 2004). A Figura 6 apresenta uma visualização conceitual da estrutura do Heracles.

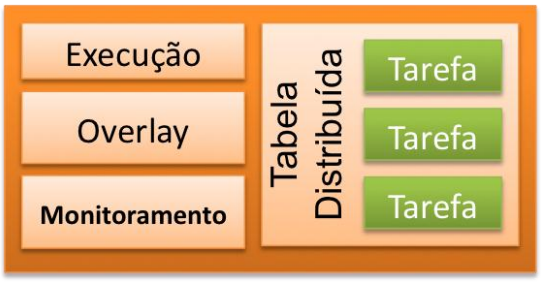

Figura 7: Esquema de um processo Heracles

O SGWfC submete as atividades de workflow para serem executadas no cluster utilizando o escalonador MTC. As diferentes instâncias das atividades de workflow com diferentes dados de entrada ou parâmetros podem ser encapsuladas como tarefas. Assumindo que os escalonadores MTC de workflows estão usando o Heracles, estas tarefas não são diretamente escalonadas para executar no cluster. Ao invés disso, o Heracles registra os metadados de cada tarefa em uma tabela distribuída (Akbarinia et al. 2007) compartilhada por um grupo de processos Heracles. Um processo Heracles é o objeto real que é escalonado pelo gerente de recursos do cluster para ser executado em paralelo em cada nó do cluster. A Figura 7 apresenta um esquema conceitual do processo Heracles. Esses processos são disparados como outros tarefas ou *jobs* do cluster e, depois de iniciadas, os processos usam técnicas P2P para controlar a execução das tarefas do workflow e coletar os dados de proveniência necessários requisitados pelo escalonador MTC. O escalonador, pode, então, encaminhar os dados de proveniência para o SGWfC. Como os processos do Heracles são autônomos, eles têm componentes para lidar com a rede de comunicação virtual que eles estabelecem entre si (Overlay), executar tarefas (Execução) e coletar dados de proveniência durante a execução das tarefas (Monitoramento). A rede virtual está associada à rede P2P hierárquica que os

processos do Heracles estabelecem entre eles para comunicação e troca de dados. A rede física permanece como a rede de alta velocidade do cluster.

A vantagem de implementar o Heracles dentro de um escalonador MTC é que o Heracles não precisa se preocupar com mecanismos importantes tais como *data staging*, ou seja, transferência de dados para o ambiente distribuído e depois transferência dos resultados coletados. *Data staging* também é papel da solução MTC para workflows (tais com o Swift/Falkon e o Hydra).

As seções seguintes deste capítulo descrevem as estratégias utilizadas pelo Heracles. O objetivo inicial do Heracles é trazer transparência na configuração e execução do experimento. Portanto, a seção 4.1 descreve as estratégias adotadas para prover a gerência dinâmico de recursos. Na seção 4.2, discute-se a questão da hierarquização de recursos. A seção 4.3 apresenta o modelo de qualidade de serviço do Heracles apresentando a estratégia de tolerância a falhas. Apesar de os conceitos de gerência dinâmica de recursos, hierarquização e qualidade serviço estarem sendo apresentados de modo separado, cabe notar que as estratégias descritas são dependentes entre si com um objetivo de apoiar a execução paralela de workflows científicos tornando-a mais transparente para o cientista.

4.1 Gerência dinâmica de recursos

Um dos objetivos do Heracles é aprimorar a maneira que os cientistas configuram suas atividades para execução em paralelo. Ao invés de configurar o número de nós do cluster que serão envolvidos no processamento da atividade paralela, os cientistas podem especificar somente o prazo para aquele experimento ser executado. Esta abordagem é mais transparente, pois os cientistas não precisam ter a consciência do número ideal de processadores necessários para executar um experimento em um prazo determinado. Fazendo uma alusão às redes de transferência de dados P2P, tal como o eMule (eMule 2010), quando um usuário transfere um arquivo, ele não precisa informar de quantos nós ou de quais nós ele deseja baixar o arquivo. O sistema P2P descobre os nós da rede que detém aquele arquivo (ou parte daquele arquivo) e inicia os processos de transferência em paralelo. Ao final de um determinado tempo, a transferência do arquivo está completa. Sendo assim, acreditamos que a execução de atividades do experimento deva ser similar, de forma que o cientista precise apenas manifestar que deseja executar aquela tarefa em paralelo e que precisa dos resultados

em um determinado prazo. A partir desses dados, o SGWfC e o escalonador de execução paralela fariam o restante.

Diferente de uma rede P2P de compartilhamento de arquivos, todos os nós de um cluster têm o potencial para executar qualquer tarefa de um workflow. Entretanto, a seleção de nós pode levar em conta princípios de localidade (Dick et al. 2009) para minimizar o custo de transferências de arquivo entre nós, por exemplo. Os nós também podem ser designados para processar tarefas de acordo com o acoplamento entre elas de forma que tarefas mais acopladas sejam executadas por um mesmo grupo de processadores, com o objetivo de minimizar transferências de dados e tempo de troca de mensagens. Entretanto, os critérios de seleção de nós na rede do cluster não estão no escopo desta dissertação. Por ora, a proposta de gerência dinâmica do Heracles é direcionada ao cálculo do número de nós necessários para executar determinada atividade no prazo especificado.

Ao inicializar a atividade paralela, o Heracles define um número inicial de processadores para processar as tarefas daquela atividade. Estes processadores iniciais são requisitados ao gerente de recursos do cluster e compõem a base de recursos inicial para execução. Esta configuração inicial é baseada no prazo estabelecido pelo cientista que configurou a atividade, nos recursos disponíveis do cluster e informações obtidas em execuções passadas (dados da base de proveniência (Freire et al. 2008)). A base de recursos pode crescer ou diminuir ao longo do tempo da execução dependendo da restrição de tempo. Para expandir a base de recursos, o Heracles requisita novos processadores ao gerente de recursos do cluster. A requisição aguarda na fila do gerente até obter o número de processadores desejado. As decisões a respeito dessas expansões ou reduções são baseadas em duas métricas: A eficiência parcial E'_p para o conjunto de processadores p e o número de tarefas completas por unidade de tempo C'_t. Essas métricas podem ser calculadas utilizando os dados coletados e armazenados na tabela distribuída dos processos do Heracles ou na base de proveniência externa (um banco de dados relacional, por exemplo). A utilização da tabela distribuída pode ser mais eficiente, pois ela está disponível na memória durante a execução. Porém, se os dados da tabela não forem suficientes, o Heracles pode consultar a base de proveniência.

Para ilustrar o funcionamento da utilização das métricas para o cálculo de expansões do Heracles, consideremos o seguinte exemplo: Um cientista configura uma atividade de

workflow que gera *k* tarefas e tem um prazo de execução de *w* horas. Baseado nos dados de proveniência, o Heracles decide iniciar o processo no cluster utilizando um número *p* de núcleos de processamento. Após um número arbitrário *n* de tarefas terminadas, o Heracles atualiza o número de tarefas completas por unidade de tempo C'_t. Para decidir se é necessário expandir ou reduzir a base de recursos, ele mede a eficiência parcial E'_p com a seguinte fórmula:

$$E'_p = \frac{\sum_{i=1}^{n} T_i}{pT_p} \quad (1)$$

Na equação 1, T_i é o tempo gasto para processar a tarefa *i* e T_p é o tempo total decorrido da execução. A fórmula tradicional da eficiência busca avaliar o quão melhor foi a execução ocorrida com *p* processadores quando comparada com a execução feita com apenas 1 processador. A equação 2 apresenta a fórmula tradicional da eficiência.

$$E_p = \frac{T_1}{pT_p} \quad (2)$$

Na fórmula da eficiência parcial da equação 1 utilizada no Heracles, o somatório do tempo gasto para processar cada tarefa *i* ($\sum_{i=1}^{n} T_i$) representa a estimativa do tempo gasto se todas as tarefas *i* fossem executadas sequencialmente por somente um processador. Como as tarefas já estão sendo executadas em paralelo, pode-se usar o tempo decorrido T_p como o tempo gasto pelos *p* processadores.

Calculada a eficiência parcial (E'_p), o Heracles pode estimar um número ideal de processadores (p_{new}) para terminar as *n* tarefas que faltam ser executadas dentro prazo (*w*) especificado. A equação 3 apresenta a fórmula utilizada pelo Heracles para o cálculo de p_{new}. A multiplicação do prazo (*w*) pela taxa de tarefas completas por unidade de tempo (C'_t) indica quantas tarefas podem ser executadas no prazo total seguindo aquela taxa. A subtração do número total de tarefas (*k*) pelo número já processas (*n*) indica quantas tarefas ainda precisam ser executadas. Dessa maneira, a divisão de (*k* - *n*) por ($w.C'_t$) indica quanto do total possível de execução falta para ser executado. Se faltar muito para ser executado, a tendência de p_{new} é crescer. Ao mesmo tempo, se a eficiência parcial for baixa, a tendência de p_{new} também é crescer.

$$p_{new} = \frac{p}{E'_p}\left(\frac{k-n}{w.C'_t}\right) \quad (3)$$

Para ilustrar a estratégia descrita, considere w = 72 horas, p = 32 núcleos de processamento e k = 10.000 tarefas. Após 160 tarefas terem terminado em aproximadamente três horas, Heracles mede que $C'_t = 89.6$ tarefas por hora. A eficiência parcial calculada é de $E'_p = 0.85$. Portanto o novo tamanho da base de processamento p_{new} será de 57 núcleos de processamento. Este valor pode ser aproximado considerando a infraestrutura do cluster. Por exemplo, se cada nó de computação do cluster tiver oito núcleos de processamento, p_{new} pode ser aproximado para 64 processadores já que 64 é múltiplo de 8. A Figura 8 apresenta a evolução de C'_t e p ao longo de tempo utilizando este exemplo arbitrário. Idealmente a curva do número de tarefas completas por unidade de tempo (C'_t) indica o comportamento da curva de núcleos de processamento (p). Através destas curvas, é possível observar o comportamento do Heracles na gerência dinâmico de recursos.

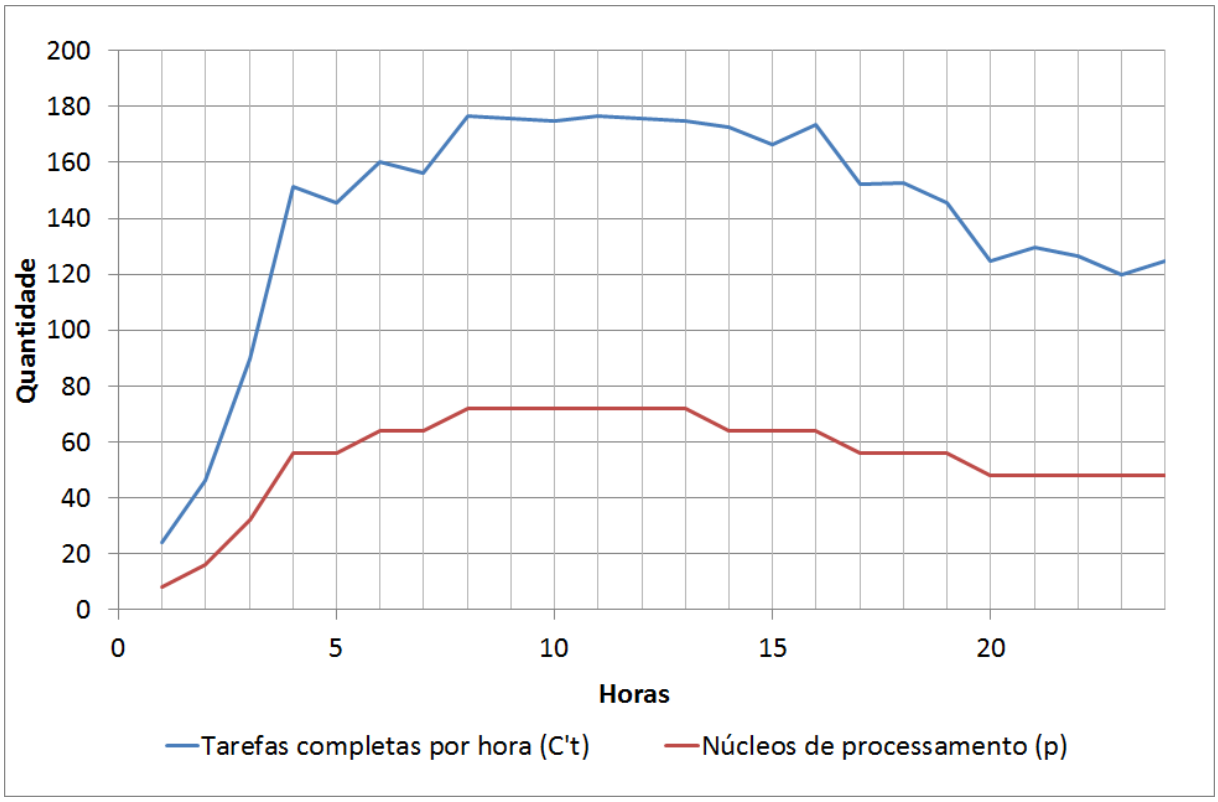

Figura 8: Simulação da variação de C'_t e p ao longo do tempo em horas

O Heracles inicia a execução com uma base de recursos pequena e depois a expande para atingir um determinado pico na taxa de tarefas por hora. O Heracles também precisa

considerar a quantidade de recursos disponíveis no cluster já que pode não haver processadores disponíveis para alocar todos os p_{new} processadores. Ainda assim, baseado no valor de p_{new}, o Heracles segue uma tendência e, quando possível, requisita (ou libera) núcleos de processamento. A Figura 8 mostra a tendência inicial do Heracles, no exemplo, em obter mais núcleos para o processamento das tarefas. Entretanto, durante a execução, ele tende por uma execução mais eficiente como mostra a Figura 9. Esse comportamento é interessante mas ainda é necessário avaliar o funcionamento da gerência dinâmica de recursos em uma prova de conceito mais extensa.

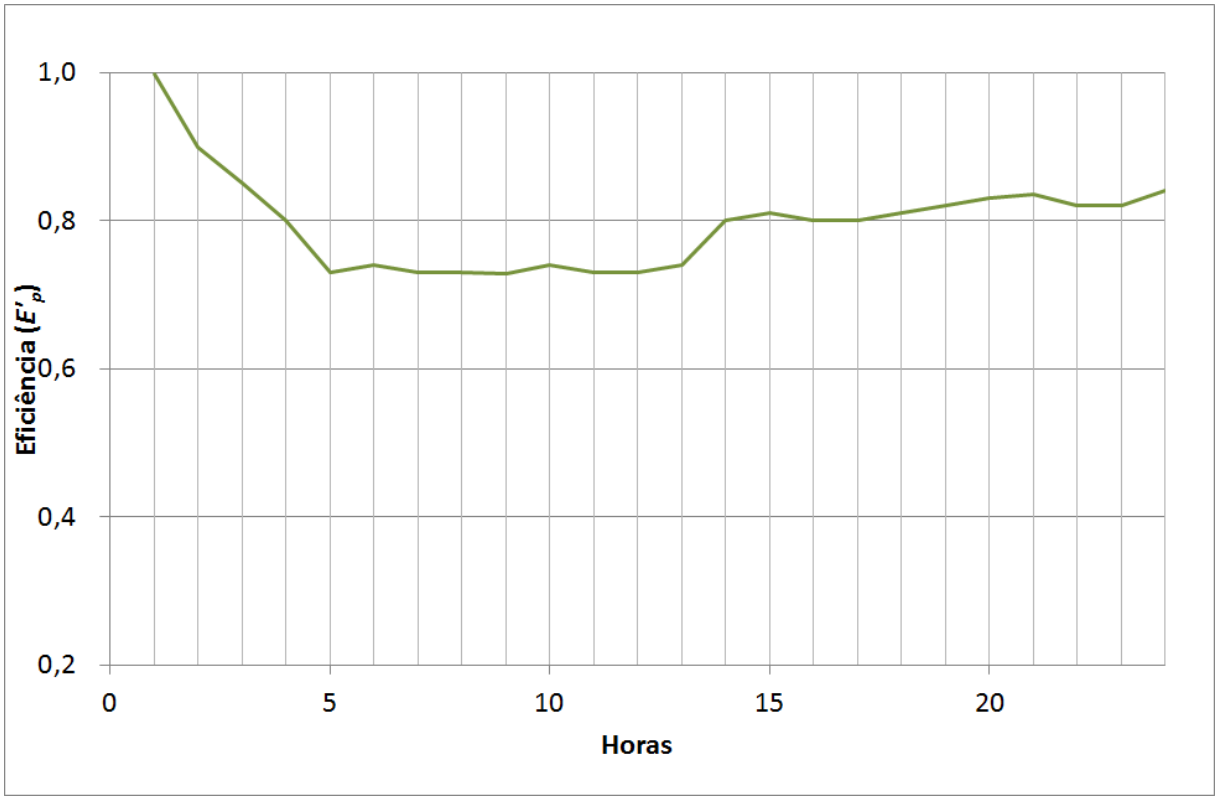

Figura 9: Eficiência parcial E'_p ao longo do tempo

Usando esta abordagem, a utilização de recursos do cluster parece ser mais eficiente, já que as atividades com prazos mais folgados podem consumir menos recursos para permitir que atividades com prazos mais apertados usem mais. O Heracles utiliza as métricas de eficiência parcial e taxa de tarefas completadas por unidade de tempo para analisar e decidir se ele pode reduzir o número de recursos sendo utilizados se o prazo especificado estiver tendendo a ser cumprido. Esta estratégia busca tornar a configuração da execução paralela do workflow mais simples e mais transparente para os cientistas, já que eles precisam definir

apenas o prazo de execução sem se preocupar com o número de processadores necessários para tal. O Heracles utiliza as métricas para expandir e contrair dinamicamente os recursos para atender as restrições de tempo da execução. Vale ressaltar que em alguns casos, os cientistas podem estipular prazos irreais para a execução de seus experimentos. Nesta situação, a abordagem Heracles pode não conseguir atender ao prazo, visto que a obtenção de recursos do cluster é limitada. Por maior que o cluster utilizado seja, os gerentes de recursos estabelecem políticas que limitam a quantidade de nós reservados por usuário.

4.2 Hierarquização de recursos

Geralmente, clusters computacionais possuem uma máquina de serviço que controla a submissão e execução de todos as tarefas do cluster sendo executadas nos nós de processamento, essa máquina de serviço é comumente chamada de *frontend* do cluster ou *head-node* (Sonmez et al. 2010). O nó de serviço é a porta de entrada para todos os serviços disponíveis no cluster. Essa abordagem torna mais simples a manutenção de uma máquina grande, mesmo que se tenha mais de um nó de serviço. Entretanto, abordagens centralizadas como esta não podem apresentar sobrecargas. Por mais que os nós de serviço sejam servidores poderosos capazes de lidar com a maioria das aplicações que demandem computação de alto desempenho, em um cenário que demande MTC, os nós de serviço podem ficar sobrecarregados como Sonmez et al. (2010) notou em seu trabalho sobre escalonamento de workflows em arquiteturas *multicluster*. Essa sobrecarga ocorre em função da grande quantidade de tarefas que uma atividade de workflow pode gerar. O nó de serviço pode também ter que lidar com transferências de arquivos durante o processo de *data staging* de cada tarefa. Dados de proveniência são coletados durante a execução das tarefas e, como a rede dos nós de processamento é normalmente privada, estes dados podem precisar transitar pela rede do nó de serviço. Dessa forma, em um cenário MTC, o nó de serviço de um cluster de larga escala pode ficar sobrecarregado com o alto número de tarefas e nós para gerenciar.

Muitos dos processos executados pelos nós de serviço não podem ser movidos para outro recurso, já que estes nós são exclusivamente responsáveis por alguns serviços do cluster. Entretanto, o escalonamento de aplicações de workflows não deve adicionar responsabilidades extras a estes nós. Portanto, o Heracles visa a aprimorar o balanceamento de carga da execução distribuindo o escalonamento de tarefas e a gerência de execução dentre

os próprios nós de execução. Para atingir este objetivo, o Heracles estabelece uma rede P2P hierárquica virtual dentre os recursos disponíveis reservados para executar suas tarefas.

Com o objetivo de distribuir a carga no escalonamento de tarefas geradas a partir de atividades de workflows, o Heracles constrói uma rede P2P virtual sobre os recursos de processamento do cluster. Geralmente, ao submeter um *job* em um cluster, um determinado número de nós de processamento é requisitado. Quando esse número requisitado de processadores encontra-se disponível, o gerente de recursos disponibiliza estes recursos para serem utilizados pelo *job*. Quando o gerente de recursos do cluster disponibiliza um conjunto de recursos ao Heracles, ele reorganiza esses recursos para formar uma rede P2P hierárquica com controle distribuído. A Figura 10 ilustra a visão da rede P2P virtual do Heracles sobre os recursos de processamento do cluster. Do ponto de vista físico, cada nó do cluster detém um processo Heracles e tais processos estabelecem a rede P2P de comunicação entre eles.

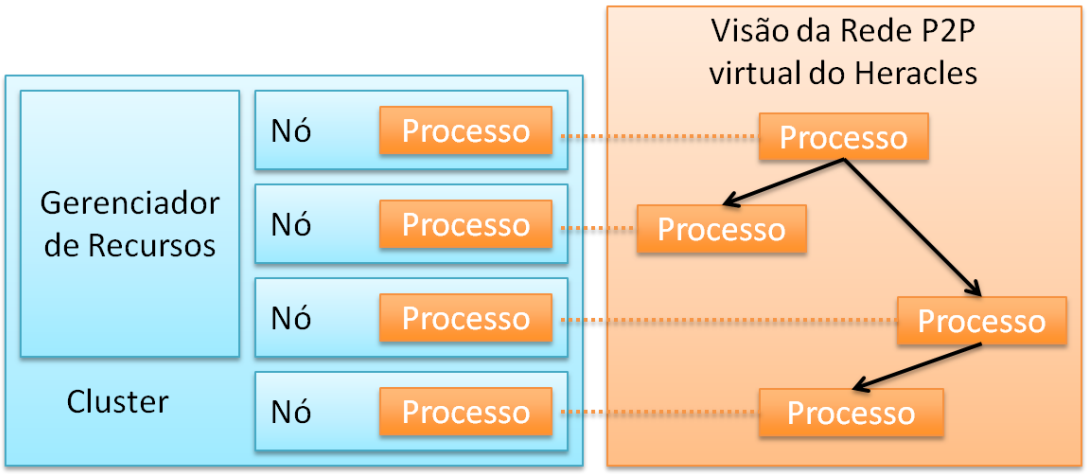

Figura 10: Visão da rede P2P virtual do Heracles sobre os recursos do Cluster

De acordo com o que foi discutido no capítulo 3, a abordagem Heracles utiliza a estratégia das redes P2P hierárquicas para estabelecer grupos dentre os processadores destinados às execuções de tarefas no cluster. O Heracles divide o conjunto de recursos em grupos e elege um processador do grupo para ser o líder. Nesta dissertação propomos uma solução para clusters com uma infraestrutura homogênea. Neste caso, a escolha do líder pode ser arbitrária, como, por exemplo, o primeiro núcleo de processamento alocado de cada nó. Entretanto, o paradigma MTC prevê a utilização de recursos heterogêneos para o processamento de tarefas. Neste caso, seria necessário um mecanismo de eleição de líder para

eleger um nó como líder do grupo. Essa questão, porém, está fora do escopo desta dissertação. Estes primeiros grupos se organizam de maneira hierárquica utilizando uma ordem arbitrária, porém seguindo um processo de construção bem definido, que será descrito adiante. Ao final, os processos são dispostos na forma de uma árvore binária. Embora uma hierarquia possa ser estabelecida com diferentes estruturas, a árvore binária foi escolhida por ser simples, fácil de manter e permitir uma primeira avaliação da abordagem proposta, estabelecendo-se uma linha base.

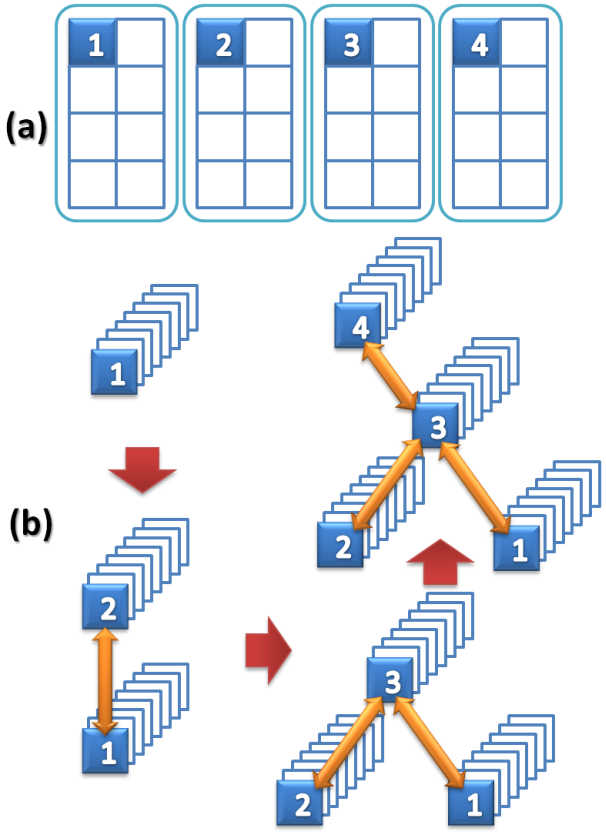

Figura 11: Representação (a) dos recursos computacionais do cluster e (b) a forma

Quando o conjunto de recursos do Heracles precisa ser expandido, novos grupos são formados. Quando um grupo novo entra na hierarquia, ele entra no nível hierárquico mais alto. Conforme novos grupos ingressam na rede, os grupos mais velhos começam a descer na hierarquia. Como os grupos mais antigos são mais propícios a deixar a rede P2P virtual quando ela sofrer uma redução, é interessante manter tais grupos nos níveis mais baixos da hierarquia. A Figura 11 (a) representa os recursos computacionais dados ao Heracles pelo gerente de recursos do cluster. Cada nó obtido possui oito núcleos de processamento. A

Figura 11 (b) mostra os processadores formando os grupos organizados de maneira hierárquica. Os números indicam a ordem que os grupos foram criados e os quadrados mais escuros são os processadores líderes de cada grupo. A hierarquia mantém os grupos mais velhos na base da árvore, enquanto os novos grupos são inseridos no topo da hierarquia, rebaixando o líder anterior. O antigo líder deve ocupar um espaço na hierarquia abaixo. Se não houver espaço na hierarquia abaixo, o grupo mais velho da hierarquia é rebaixado e assim sucessivamente.

A Figura 12 apresenta o processo de construção da árvore utilizada pelo Heracles. Cada número da árvore representa um grupo, e cada retângulo é um nível de hierarquia. Inicialmente, o grupo inicial é colocado no nível inferior da hierarquia (nível 1). O segundo grupo, ao ingressar na rede, observa que não há espaço na hierarquia inferior, portanto, ele é inserido na hierarquia intermediária (nível 2). Quando o terceiro grupo é inserido, ele toma a posição do grupo dois, que é rebaixado para o nível 1 da hierarquia. Em seguida, o quarto grupo é adicionado. A princípio, ele poderia tomar a posição do grupo 3, porém não há espaço no nível 1 para rebaixar o grupo 3, portanto, o grupo 4 é inserido na hierarquia superior (nível 3). Os outros grupos são adicionados de maneira similar. Embora a Figura 12 ilustre apenas três níveis hierárquicos, não há limites para a quantidade de níveis da estrutura.

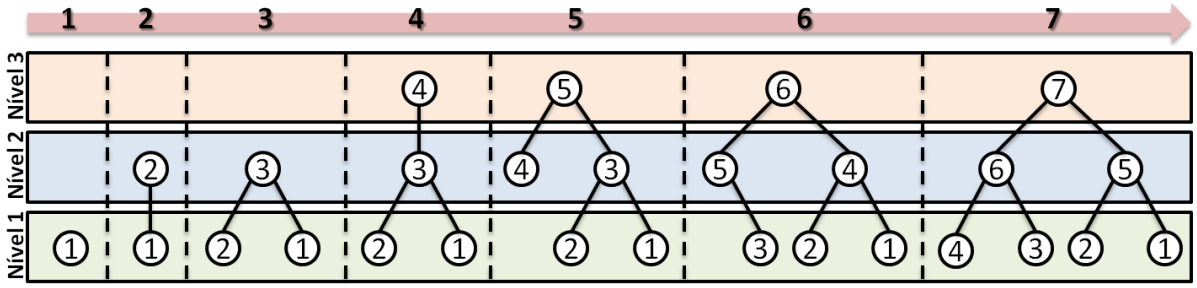

Figura 12: Processo de construção da rede hierárquica do Heracles.

Líderes de grupo mantêm uma lista de tarefas a serem processadas em uma tabela distribuída. Nesta tabela ficam registrados os metadados das tarefas, o status e informações de proveniência, pois os líderes também são responsáveis por coletar os dados de proveniência de cada tarefa. Centralizar o processo de coleta de dados no líder é importante para poupar o número de conexões com banco de dados e operações de escrita em disco dependendo de onde os dados serão escritos. Em algum momento do processo de execução, os dados da tabela distribuída são registrados em um banco de dados ou em arquivos.

Na tabela distribuída também fica estabelecido quais tarefas são consumidas por cada grupo. Quando um processador termina de processar uma dada tarefa, ele reporta o fato ao líder de seu grupo, que atualiza a tabela localmente. O líder, então, escalona uma nova tarefa da tabela distribuída para ser processada por aquele processador. O líder do grupo, em seguida, se comunica com os líderes de grupos adjacentes sobre as atualizações da tabela. É importante que a tabela esteja replicada nos líderes de grupo de forma que o Heracles a recupere em eventuais falhas de líderes de grupo, como descrito na próxima seção. O líder da hierarquia mais alta também é o responsável por medir a eficiência parcial da execução e tomar decisões a respeito das expansões e contrações dos recursos computacionais da execução seguindo os procedimentos descritos na seção 4.1.

Durante uma expansão dos recursos da execução, quando um novo grupo é criado, o líder do grupo recebe a tabela distribuída com a informação de quais tarefas aquele grupo deve consumir, a princípio. A tabela é transmitida ao novo líder seguindo um modelo baseado em *push* (Acharya et al. 1997). A decisão de quais tarefas ficam reservadas para aquele grupo consumir é decisão do nó superior da hierarquia que pode fazer uma divisão do conjunto de tarefas do grupo que possui mais tarefas pendentes.

Quando um nó de processamento está disponível para processar uma tarefa, ele faz uma requisição ao seu líder. A requisição de tarefas é feita seguindo o modelo *pull* (Acharya et al. 1997). Ou seja, o nó de processamento requisita uma tarefa para consumir. O líder do grupo, então, escalona uma tarefa da tabela para aquele nó. Se aquele grupo não tiver mais tarefas para processar, o líder requisita tarefas ao líder superior na hierarquia.

A hierarquização dos recursos busca auxiliar a gerência dinâmica de recursos e o mecanismo de tolerância a falhas. Como consequência, ela pode melhorar o balanceamento de carga distribuindo o controle sobre o escalonamento e execução. Os grupos de processadores tem uma autonomia descentralizada e respondem ao seu líder, enquanto o líder reporta suas atualizações aos líderes adjacentes, que propagam a informação para os demais líderes.

4.3 Qualidade de serviço

Em um sistema de cluster tradicional, se um processador estiver trabalhando em uma atividade e falhar, a atividade é abortada. Este procedimento é necessário se a atividade for composta, por exemplo, de um conjunto de processos MPI acoplados. Quando cientistas

submetem um grande conjunto de tarefas para serem executadas, alguns escalonadores conseguem distinguir que falhas singulares afetam somente àquela tarefa que executava no nó que falhou. Este é um cenário melhor, porém os cientistas ainda precisam checar as tarefas que falharam e então ressubmetê-las. Este tipo de esforço manual aumenta as chances de erros e, normalmente, os cientistas preferem resubmeter o conjunto completo de tarefas. Entretanto, o escalonador poderia reescalonar as tarefas que falharam automaticamente. As tarefas seriam, então, processadas por outro nó como acontece em um sistema P2P, por exemplo.

É possível relacionar falhas em clusters com eventos de *churn*, que são muito comuns em sistemas P2P. Em um cluster de larga escala, a frequência de falhas é menor se comparado a uma rede P2P tradicional. Entretanto, se os cientistas escalonarem atividades MTC utilizando um grande número de nós de processamento, é bem provável que eles tenham problemas com falhas de alguns nós ou processadores. Como essas falhas são esperadas, esta dissertação considera qualquer falha em uma tarefa sendo processada como um evento de *churn*. Após uma falha, a tarefa que havia sido designada para ser processada por um processador ou nó, que falhou, deve ser atribuída a outro processador ou nó. Mecanismos de tolerância a falhas disponíveis em sistemas P2P são úteis para lidar com falhas em cluster durante a execução de muitas tarefas. Portanto, com o intuito de melhorar a qualidade de serviço durante o escalonamento e execução de workflows em clusters, o Heracles utiliza um sistema de tolerância a falhas, descrito a seguir.

Cada núcleo de processamento disponível nos recursos reservados ao Heracles pode ser visto como um *peer*. Um *peer* obtém tarefas para processar através do líder de seu grupo, que possui a lista de tarefas em sua tabela distribuída. A tabela possui a lista de todas as tarefas, que podem estar marcadas como: pendentes, executando e terminadas. Uma tarefa muda seu estado para executando quando é escalonada para executar em um *peer*. Os líderes de grupo mantêm uma média de quanto tempo uma tarefa leva para completar. Se uma tarefa estiver demorando demais para terminar, é possível que o *peer* responsável tenha falhado. O líder pode, então, decidir reescalonar a tarefa mudando seu estado para pendente novamente. Esta decisão é baseada na média e no desvio padrão do tempo de execução das tarefas. Se uma tarefa for consideravelmente maior do que as demais, o líder pode tomar uma decisão errada ao decidir reescaloná-la. Entretanto, como as tentativas de execução estão registradas

na tabela, na segunda tentativa de execução da tarefa, o líder irá aguardar um tempo maior antes de classificar novamente a tarefa como falha.

Em um cenário mais crítico, o líder do grupo ou mesmo o grupo como um todo pode falhar. Isto é possível, pois todos os *peers* são núcleos de processamento (*cores*) que podem estar dentro do mesmo *chip* ou nó computacional. De qualquer maneira, a falha do líder é tão ruim quanto a falha do grupo como um todo, já que, sem um líder, os *peers* do grupo perdem a comunicação com o Heracles. Portanto, nestes cenários, o líder do grupo de um nível superior da hierarquia verifica que ocorreu uma falha naquele grupo e então reescalona todas as tarefas que não terminaram daquele grupo que falhou. Novamente, este reescalonamento pode ser feito apenas mudando o estado das tarefas para pendente. Entretanto, as tarefas também precisam ser designadas para serem processadas por outro grupo. Sendo assim, elas permanecem na tabela até que um grupo fique ocioso e requisite novas tarefas para processar. Se o Heracles expandir a rede de recursos, as tarefas pendentes também podem ser designadas para o novo grupo. Uma falha de um grupo completo é fácil de ser notada por outros líderes da hierarquia, pois eles mantêm contato através de mensagens de atualização tipo *push* para manter a tabela distribuída atualizada. Se um líder observa que outros líderes não estão executando suas tarefas e não respondendo as mensagens, ele entende isto como uma falha do grupo e pode decidir reescalonar as tarefas na tabela.

A falha de um grupo na execução do Heracles impacta na topologia da rede P2P virtual formada pelos grupos. Como descrito na seção 4.2, os grupos do Heracles são organizados na forma de uma árvore binária organizada pela ordem de entrada dos grupos na rede. O caso mais simples de recuperação de falha de grupo é quando este é um nó folha da árvore. Neste caso, nada precisa ser feito para reestruturar a árvore, pois ele apenas irá deixar vago um espaço na hierarquia mais baixa da árvore. Quando um novo grupo ingressar na árvore, o espaço vago será preenchido. A Figura 13 (a) ilustra este cenário onde o grupo 3 falha e deixa a árvore sem causar impacto nos grupos adjacentes.

Outra possibilidade é um grupo de um nível intermediário da hierarquia da árvore falhar. Neste caso, os nós filhos desse grupo perdem a ligação com o restante da árvore. Para corrigir este problema, o filho mais novo (*i.e.* que entrou por último na rede) é eleito para substituir o grupo que falhou. A Figura 13 (b) ilustra esta situação quando o grupo 6 falha e é substituído pelo grupo 4 na hierarquia. Entretanto, o pior cenário de falha é quando o grupo

líder da execução, posicionado na raiz da árvore, falha. Neste caso, o filho mais novo do grupo líder também será promovido. Se houver mais de dois níveis na hierarquia, a promoção ocorrerá em cascata como ilustra a Figura 13 (c). Nela, o grupo 7 falha e é substituído pelo grupo 6. Porém, quando o grupo 6 assume a liderança, o grupo 4 precisa ser promovido para substituí-lo e manter a árvore conexa.

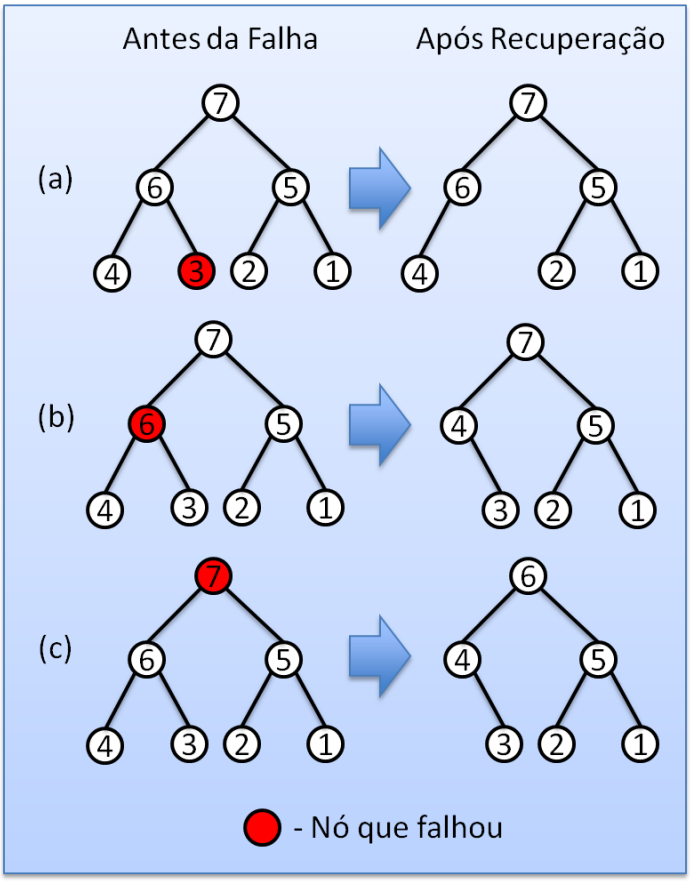

Figura 13: Recuperação de falha no Heracles para diferentes casos. (a) Falha em um nó folha, (b) falha em um nível intermediário e (c) falha no grupo líder da execução.

A estrutura hierárquica do Heracles e a tabela distribuída favorecem a construção deste modelo de tolerância a falhas. A não ser que todos os grupos falhem ao mesmo tempo, o reescalonamento automático das tarefas tem como objetivo garantir que as tarefas das atividades de workflow serão executadas em algum momento da execução. Este mecanismo busca melhorar a qualidade dos serviços de escalonamento e execução em grandes clusters.

Capítulo 5 – O Simulador do Heracles

O propósito da abordagem Heracles é controlar a execução paralela de atividades de workflow em clusters de grande porte. Portanto, para avaliar a abordagem seria necessária a execução de estudos em um cluster grande. Entretanto, usar uma infraestrutura de computação de alto desempenho de larga-escala com exclusividade pode ser inviável ou muito custoso.

O intuito do estudo proposto nesta dissertação é avaliar a viabilidade da abordagem Heracles. Ou seja, o estudo busca verificar se o Heracles atende aos requisitos de transparência sendo tolerante a falhas e realizando a gerência dinâmica de recursos. Dessa forma, acredita-se que estudos de simulação atendam ao propósito e apresentem o comportamento da abordagem Heracles no controle da execução de atividades de workflows científicos. Embora um estudo de simulação possa não refletir a realidade de maneira completa, ele apresenta tendências que possibilitam uma análise inicial do modelo simulado.

Em trabalhos anteriores (Ogasawara et al. 2010), um estudo de simulação foi realizado para avaliar a execução de atividades de workflows científicos em redes P2P utilizando a arquitetura SciMule. No trabalho mencionado foi utilizado o simulador SciMulator (Dias et al. 2010c) com o intuito de verificar o ganho de desempenho no paralelismo de dados através da distribuição de tarefas pelos *peers* da rede P2P. O estudo permitiu a verificação de características e melhorias necessárias na arquitetura SciMule, enquanto o SciMulator mostrou-se escalável no tempo de execução e consumo de memória, mesmo nos cenários com falhas. Em função dos resultados positivos obtidos com o SciMulator, desenvolvemos o HeracleSim como uma adaptação orientada a clusters, e não mais redes P2P, que utiliza a abordagem Heracles.

O HeracleSim é uma extensão do PeerSim (Jelasity et al. 2010). O PeerSim é um simulador de redes P2P desenvolvido na linguagem Java. Ele possui quatro componentes principais extensíveis: (i) o nó, que representa um processador na rede simulada e é modelado pela classe *GeneralNode*; (ii) uma abstração da rede que mantém informações sobre como os nós estão conectados; (iii) protocolos que são executados pelos nós e definem as ações que os nós executam na rede e (iv) elementos de controle que executam ações globais na rede, tais como eventos de falha e registro de *log* ou proveniência. Estes componentes foram estendidos com o intuito de criar o modelo de submissão de atividades de workflow e execução de

tarefas. Um desafio na construção do simulador foi, justamente, acrescentar componentes que permitissem a distribuição de atividades em tarefas pela rede, modelando o procedimento como ocorre em clusters voltados à computação de alto desempenho. Além disso, a execução das tarefas pelos nós foi feita através de protocolos PeerSim.

Este capítulo é dividido de acordo com os principais processos do simulador. A seção 5.1 descreve o processo de simulação do HeracleSim. A seção 5.2 descreve a arquitetura do simulador apresentando os principais componentes do HeracleSim e sua integração com o PeerSim. A seção 5.2.1 descreve o modelo de submissão de tarefas e gerência de recursos do simulador enquanto a seção 5.2.2 descreve o modelo de execução e o modelo de falhas.

5.1 O Processo de simulação do HeracleSim

O HeracleSim foi modelado usando uma abordagem síncrona (Barbosa 1996), baseada em ciclos, do PeerSim. Em um dado ciclo, todos os nós da rede executam um conjunto de instruções baseados em trocas de informações realizadas no ciclo anterior. Esta abordagem foi escolhida por garantir maior escalabilidade, embora possua simplificações na camada de transporte de rede. Entretanto, estas simplificações não impactam na simulação, pois a rede do cluster simulado utiliza um mecanismo de armazenamento de dados compartilhados. Além disso, a gerência de recursos do cluster é centralizada.

O processo da simulação ocorre em um determinado número de ciclos. A Figura 14 apresenta o diagrama de atividades do simulador. A inicialização da simulação realiza a configuração de uma topologia, que gera os nós e os dispõe na rede, formando o cluster. Em seguida, cada ciclo inicia com o registro de proveniência, que armazena informações a respeito das atividades executadas no cluster. Depois ocorre o processamento dos controles, que são ações globais realizadas na rede, tais como a submissão de novos *jobs* e o controle de falhas na rede. O controle de submissão de *jobs* define quantos *jobs* serão submetidos naquele ciclo. O controle do Heracles interage com o gerente de recursos do cluster com o intuito de requisitar nós para os processos Heracles ou liberar recursos. O escalonamento de *jobs* é o controle que verifica os *jobs* da fila e a quantidade de recursos disponíveis no cluster, de forma a verificar quais *jobs* podem iniciar sua execução naquele ciclo. O controle de falhas define quais nós deverão falhar durante a execução daquele ciclo. Após o término da execução dos controles, segue a execução dos protocolos por cada um dos nós da rede.

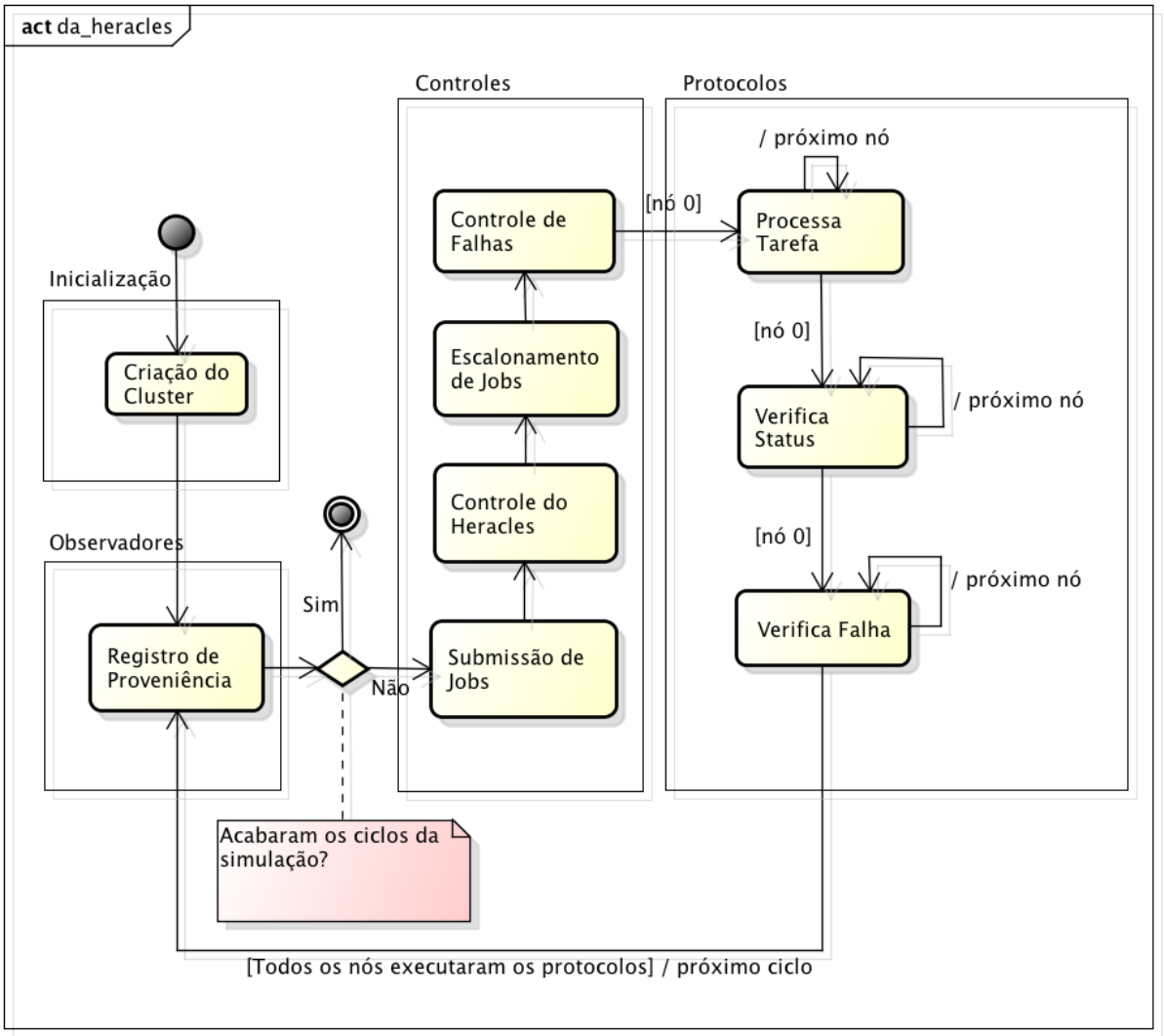

Figura 14: Diagrama de atividades da simulação com o HeracleSim

Cada nó inicia cada ciclo processando a tarefa que lhe foi escalonada, se o nó não estiver ocioso, naturalmente. No segundo protocolo, o nó verifica seu estado, isto é, verifica se a tarefa atual terminou e, se for o caso, o nó notifica o processo paralelo e tenta obter uma nova tarefa. Se não houver novas tarefas para processar, o nó se torna ocioso. No protocolo seguinte, o nó pode apresentar uma falha, acarretando na interrupção do processamento da tarefa atual. A ocorrência da falha de um nó em um determinado ciclo é definida pelo controle de falhas. Após a execução de todos os protocolos por todos os nós, dá-se inicio um novo ciclo da simulação. No último ciclo da simulação, após o registro de proveniência, a simulação é encerrada.

5.2 A arquitetura do simulador

O HeracleSim retrata o modelo de submissão de um cluster. Sendo assim, a submissão de *jobs* é feita através de uma fila. O gerente de recursos, ao detectar recursos suficientes para consumir determinado *job*, reserva os recursos para aquele *job* e permite que o escalonador dispare a execução do mesmo. Como o objetivo do estudo proposto nesta dissertação é avaliar o comportamento do Heracles quando comparado com uma abordagem tradicional, o simulador permite a submissão de processos simples e processos Heracles. O processo simples retrata uma atividade paralela comum que gera uma determinada quantidade de tarefas e consome uma determinada quantidade de recursos do cluster. As tarefas possuem um custo associado que indica por quanto tempo aquela tarefa ocupa um processador da rede. Já o processo Heracles é modelado para aplicar os conceitos da abordagem Heracles durante a execução de tarefas. Para o entendimento de como estes processos são submetidos para execução no cluster e como as tarefas são consumidas pelos nós, é necessário explicar os componentes do simulador de maneira mais detalhada. A Figura 15 apresenta o modelo conceitual UML das principais classes do HeracleSim.

A classe *Cluster* detém o conjunto de nós (*Core*) da rede simulada além de métodos para a gerência de recursos do cluster. É através dessa classe que é possível verificar se uma determinada quantidade de nós está disponível e alocá-los. Métodos como *grabCore* e *releaseCore* permitem marcar um nó como ocupado ou liberá-lo para ser utilizado por outro *Job*. Também é possível verificar se todos os processadores do cluster estão ocupados através do método *isFull*.

A classe *Core* representa um núcleo de processamento do cluster. Quando está ocupado, significa que o processador está executando alguma tarefa (*currentTask*) de um determinado processo (*workingProcess*) e ficará ocupado por um determinado número de ciclos (*busyCycles)* durante a simulação. O método *process* é o responsável por consumir a tarefa atual, alterando o valor da variável *busyCycles*, da classe *Core* e *remainingCost* da classe *Task*. A classe *Core* também guarda uma lista de tarefas pendentes (*pendingTasks*) de forma que, quando a tarefa atual do nó terminar, ele pode obter uma nova tarefa desta lista através do método *getNewTask*.

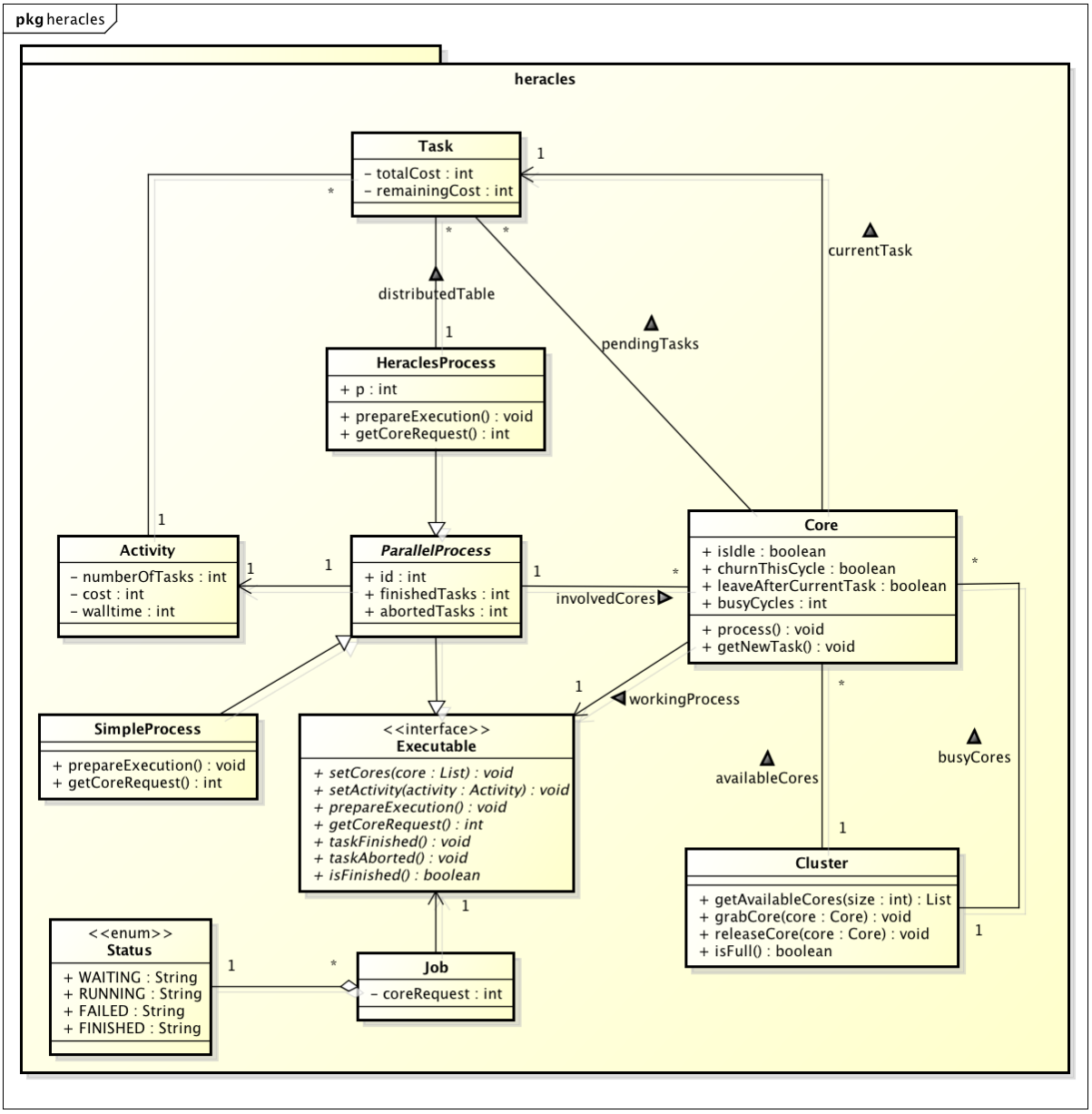

Figura 15: Principais componentes do HeracleSim

A classe *Activity* representa uma atividade do workflow que será executada em paralelo no cluster simulado. A paralelização é decorrente da construção de um conjunto de tarefas (*Tasks*) baseadas na atividade. Sendo assim, a classe *Activity* guarda o valor de quantas tarefas ele gera (*numberOfTasks)* e quanto tempo a atividade demora para executar, em ciclos (*cost*). Além disso, a atividade também guarda o prazo (*walltime*) estipulado para a execução daquela atividade.

A classe *Task* representa uma execução da atividade que lhe deu origem. O custo total da tarefa (*totalCost*) é igual ao custo da atividade, já que uma tarefa representa uma execução de um determinado programa definido pela atividade utilizando um conjunto de dados ou parâmetros específicos. A tarefa também guarda o quanto de seu custo total ainda precisa ser consumido para que ela termine (*remainingCost*).

A interface *Executable* representa o modelo necessário para um determinado processo ser executável pelo cluster simulado. Essa interface permite que diferentes tipos de processos, sejam eles processos tradicionais ou processos Heracles, possam ser submetidos no cluster e executados sem distinção. O gerente de recursos e o escalonador do cluster só precisam enxergar a interface *Executable* embora o que é executado, de fato, são as classes que a implementam.

A classe abstrata *ParallelProcess* é uma abstração de um processo paralelo genérico e, por tal motivo, não pode ser instanciada. Ela é a base para todo o processo que precisa ser submetido no cluster simulado. Ela implementa os métodos mais gerais da interface *Executable* tais como *setCores*, *setActivity*, *taskFinished*, *taskAborted* e *isFinished*. Este métodos não estão representados na classe *ParallelProcess* para não sobrecarregar o modelo, já que estão apresentados na interface. Os métodos *taskFinished* e *taskAborted* servem para que o nó envolvido na execução daquele processo notifique quando terminou de executar uma tarefa ou se ocorreu alguma falha e a tarefa foi abortada, respectivamente. O método *isFinished* verifica se o processo terminou, i.e., todas as tarefas da atividade foram executadas.

A classe *SimpleProcess* é uma extensão da classe abstrata *ParallelProcess* e implementa os métodos restantes da interface: o *prepareExecution* e o *getCoreRequest*. Estes dois métodos representam a lógica do processo paralelo e é o que diferencia um tipo de processo do outro. Em um processo simples, o método *prepareExecution* produz o conjunto de tarefas da atividade e os distribui uniformemente dentre os nós de processamento reservados. O método *getCoreRequest* calcula quantos processadores serão requisitados ao cluster para processar aquela atividade. Este cálculo é feito baseado no número de tarefas da atividade, gerando uma distribuição uniforme de possibilidades que são escolhidas aleatoriamente.

A classe *HeraclesProcess* é a extensão da classe *ParallelProcess* que implementa a abordagem do Heracles em um processo. Ele guarda o valor de *p* atual do processo que poderá ser modificado dinamicamente durante a execução. O método *prepareExecution* desta classe produz o conjunto de tarefas da atividade e os guarda na tabela distribuída (*distributedTable*), que será utilizada pelos nós para obter novas tarefas para processar. O método *getCoreRequest* calcula a requisição de nós para o processo baseado em dados de execuções passadas (dados de proveniência da simulação). Sendo assim, ele faz uma busca por execuções passadas de uma determinada atividade e obtém o valor médio de *p* armazenado.

A classe *Job* representa o elemento que é, de fato, submetido ao escalonador do cluster para permanecer em uma fila até poder obter um número desejado de nós e, então iniciar sua execução. Sendo assim, a classe *Job* mantém informações a respeito de quantos nós ele está requisitando (*coreRequest*), qual processo ele executa (*Executable*) e o seu status (*Status*) que pode ser: aguardando (*Waiting*), executando (*Running*), falho (*Failed*) e terminado (*Finished*).

5.2.1 Modelos de submissão e gerência de recursos

Embora os componentes apresentados na Figura 15 representem os principais componentes do Heracles, o processo de submissão e gerência de recursos é feito através de classes de controle. Essas classes estendem a interface de controle do PeerSim e exercem atos globais na rede. A Figura 16 apresenta o modelo conceitual de classes de controle do Heracles e sua interação com outros componentes do PeerSim e do próprio Heracles.

A classe *Scheduler* guarda uma fila de *Jobs* que aguardam ser escalonados para executar no cluster. O método *execute* é o método chamado pela classe de simulação do PeerSim a cada ciclo. A cada ciclo da simulação, essa classe verifica a quantidade de recursos disponíveis no cluster e escalona novos *Jobs*. Sendo assim, a classe *Scheduler* é responsável por verificar cada *Job* na fila e analisar a possibilidade de submetê-lo no cluster, caso haja nós de processamento suficientes disponíveis. A classe *Scheduler* também verifica se *Jobs* em execução já terminaram ou se falharam, mudando seu estado (*Status*) adequadamente.

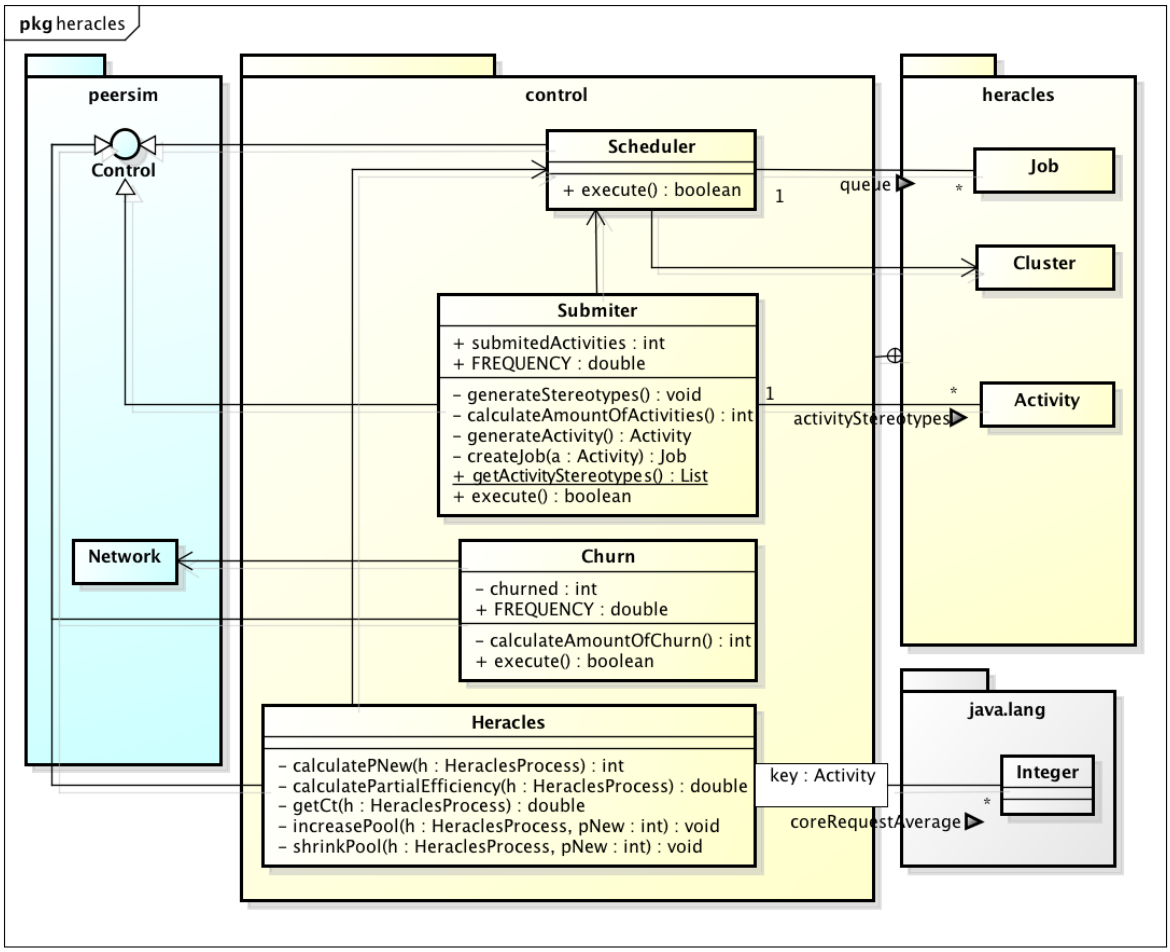

Figura 16: Classes de controle do Heracles

A classe *Submiter* é a responsável pela submissão de novos *Jobs* na fila da classe *Scheduler*. Ela faz o papel dos usuários cientistas do cluster submetendo seus experimentos para executar. Ao ser inicializada, a classe utiliza o método *generateStereotypes* para gerar uma lista de estereótipos de atividades que poderão ser submetidos para executar. Quando o simulador invoca o método *execute* do controle, ele calcula a quantidade de atividades que devem ser submetidas naquele ciclo através do método *calculateAmountOfActivities*. Com a quantidade de atividades estabelecida, ele gera tais atividades utilizando o método *generateActivity*, que cria uma atividade baseada em algum dos estereótipos presente na lista. A decisão de qual estereótipo de atividade escolher é feita de maneira aleatória. Em seguida, cada atividade é encapsulada em um processo paralelo (que pode ser um *SimpleProcess* ou *HeraclesProcess*) e atribuída a um novo *Job* através do método *createJob*. O *Job* criado é então submetido à fila da classe *Scheduler*.

A classe de controle de falhas *Churn* é a responsável por provocar as falhas no processamento de tarefas no cluster. O método *execute* calcula a quantidade de falhas que precisam ocorrer naquele ciclo através do método *calculateAmountOfChurn*. Em seguida ele sorteia os nós da rede que falham naquele ciclo e os marca para falhar. A falha só ocorrerá durante a execução dos protocolos pelos nós, que será explicada na seção 5.2.2.

A classe de controle *Heracles* é responsável por requisitar as expansões e contrações do conjunto de nós dos processos Heracles sendo executados. Por tal motivo, ela possui os métodos para o cálculo da eficiência parcial da execução e para obter o número de tarefas finalizadas por ciclo (*getCt*). O método *calculatePNew* é o responsável por estimar o possível novo valor de *p* para o processo Heracles. A partir desta estimativa, o classe irá requisitar uma expansão da quantidade de recursos do processo através do método *increasePool* ou uma redução através do método *shrinkPool*. A classe *Heracles* também guarda a tabela com os registros do valor médio de *p* em execuções passadas das atividades. Dessa forma, um novo processo Heracles pode consultar esta tabela e obter um bom valor inicial de nós para requisitar ao gerente de recursos do cluster.

5.2.2 Modelo de execução e falhas

Além dos procedimentos para submeter novos *Jobs* para execução no cluster e do modelo de gerência de recursos, foi necessário construir um modelo para execução de tarefas e falhas dos nós. Como a execução de tarefas e a ocorrência de falhas são procedimentos que ocorrem nos nós de maneira individual, foram construídos protocolos PeerSim para realizar tais procedimentos. Cada nó da rede simulada no PeerSim guarda um conjunto de protocolos. Cada nó, em cada ciclo, executa cada um dos seus protocolos. No HeracleSim foram elaborados três protocolos cujo modelo conceitual é apresentado na Figura 17.

Como os protocolos implementam a interface *Protocol* do PeerSim, todos eles possuem o método *nextCycle*. Este método contém a lógica de cada protocolo e pode alterar características e invocar métodos do nó (*n : Node*) que o detém.

O protocolo *Processor* é o responsável por processar a tarefa atual do nó. Em linhas gerais, ele verifica o estado do nó e, se ele estiver executando uma tarefa, o protocolo invoca o método *process* da classe *Core*. O método *process*, como explicado anteriormente, consome a tarefa atual do nó. Isto significa que, a cada ciclo, o protocolo irá consumir uma unidade do

custo restante da tarefa (*remainingCost*). Por exemplo, se uma tarefa tiver um custo inicial igual a 150 minutos, ela irá demorar 150 ciclos para terminar.

O protocolo *CheckStatus* é responsável por verificar se um nó terminou a execução de uma tarefa. Em caso positivo, o protocolo tenta obter uma nova tarefa para o nó. Se o nó terminou e não há tarefas pendentes, o nó é declarado como ocioso e é liberado para executar outro *Job*.

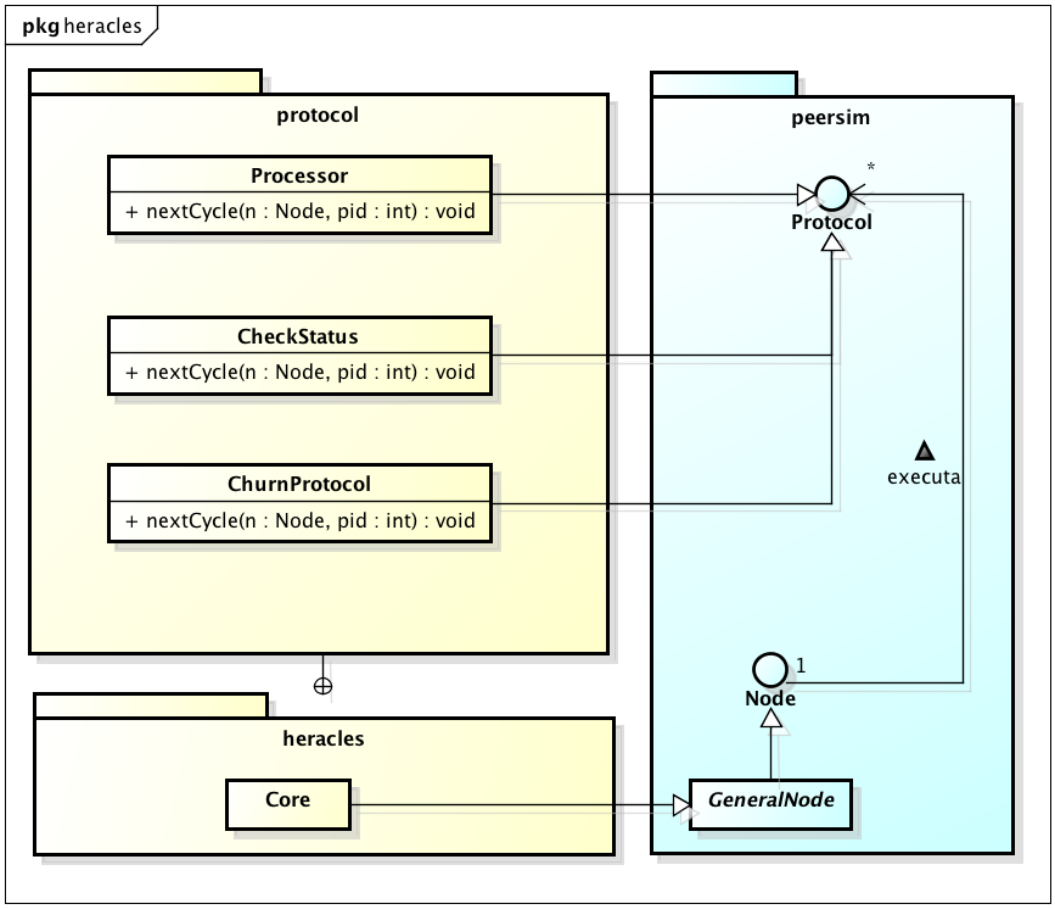

Figura 17: Modelo conceitual dos protocolos do HeracleSim

O protocolo *ChurnProtocol* é responsável pela ocorrência da falha no nó. A classe de controle *Churn*, descrita na seção 5.2.1, escolhe alguns nós para falhar durante aquele ciclo. Quando um nó escolhido executar o protocolo *ChurnProtocol* ele aborta a tarefa atual e abandona a execução do processo paralelo no qual ele está envolvido. Com isso, o processo paralelo é notificado que uma de suas tarefas foi abortada.

O HeracleSim foi programado de forma que as simulações pudessem explorar clusters com grande quantidade de nós por longos períodos de simulação. O HeracleSim atende a estes requisitos sendo capaz de simular clusters com mais de trinta mil nós durante períodos equivalentes a meses de simulação. Embora os estudos realizados nesta dissertação não demandem a simulação de um cluster tão grande por tanto tempo de simulação, a ferramenta HeracleSim pode ser aproveitada em estudos futuros envolvendo novas versões do Heracles ou abordagens equivalentes para o processamento de atividades de workflows científicos em clusters.

Capítulo 6 – Avaliação da abordagem Heracles

O Heracles foi projetado para tornar a execução distribuída de atividades de workflows científicos em clusters mais transparente. Isto significa que o Heracles precisa ser tolerante a falhas e capaz de gerenciar os recursos da execução de forma a atender o prazo definido para a atividade. A tolerância a falhas está relacionada com a transparência pelo fato do cientista executar seu workflow, e, ao término da execução, não se preocupar com quais atividades executaram com sucesso e quais não. A gerência de recursos torna a execução paralela mais transparente, pois o cientista não precisa configurar a execução no cluster especificando características como a quantidade de nós necessários para a execução. O Heracles deve ser capaz de estipular os valores ideais dos números de processadores necessários para processar determinada atividade no prazo especificado. Nesta dissertação, o Heracles é avaliado com o objetivo de verificar se ele consegue atender a estes requisitos de transparência.

Foi planejado um estudo envolvendo a simulação de um cluster voltado a computação de alto desempenho utilizando o simulador HeracleSim. Durante a simulação são submetidas atividades de workflow científico para serem processadas em paralelo. Essas atividades podem ser processadas através de um processo simples ou um processo Heracles. O processo simples constrói um conjunto de tarefas homogêneas a partir da atividade e as executa em paralelo após obter recursos computacionais do cluster. O processo Heracles também constrói o conjunto de tarefas homogêneas a partir da atividade, mas as executa em paralelo utilizando a abordagem Heracles. Os resultados da simulação permitem avaliar e comparar características de ambas as abordagens.

As seções seguintes deste capítulo descrevem o estudo realizado com o simulador. A seção 6.1 apresenta as definições do estudo e o planejamento. A seção 6.2 apresenta a execução do estudo e a seção 6.3 apresenta as análises avaliando os estudos realizados.

6.1 Definição e planejamento

O estudo realizado tem como objetivo *analisar* as abordagens de execução paralela de atividades de workflow científico *com o propósito de* verificar seu comportamento diante de falhas durante a execução e a capacidade de atender os prazos da execução *sob o ponto de vista* do cientista *no contexto* do Núcleo de Atendimento em Computação de Alto

Desempenho (NACAD) da UFRJ. Sendo assim optamos por simular um cluster equivalente ao atual maior cluster do NACAD, com 7168 núcleos de processamento. Entretanto, o gerente de recursos do cluster simulado permite que cada *Job* faça a requisição de no máximo 1024 nós de processamento. Como o ponto de vista do estudo é o do cientista, as métricas analisadas na simulação são métricas percebidas por cientistas quando executam suas atividades nos clusters, tais como a quantidade de atividades finalizadas e tempo total da execução.

A primeira avaliação (A1) realizada tem como propósito verificar a velocidade na qual o Heracles consegue equalizar suas métricas para iniciar a execução de uma atividade com uma quantidade suficiente de recursos para atender ao prazo especificado pelo cientista. Como apresentado no capítulo 4, o Heracles avalia a quantidade de recursos disponíveis durante a execução e, quando necessário, expande ou contrai a rede de recursos. Esse ajuste da quantidade do número de processadores fica registrado nos dados de proveniência e pode ser consultado pelo Heracles quando uma próxima atividade equivalente for submetida. Nesta situação, o Heracles inicia a execução com uma quantidade de processadores mais adequada de forma que precise fazer menos ajustes durante a execução. Como diferentes tipos de atividades são submetidos ao longo da simulação, é desejável verificar em quanto tempo o Heracles consegue estabelecer boas métricas para a quantidade inicial de recursos e atingir os prazos das atividades.

A segunda avaliação (A2) realizada tem como propósito comparar o desempenho do processo de execução simples com o processo Heracles no cenário com falhas para diferentes tipos de atividades. Para tal, mede-se a taxa de finalização de atividades submetidas ao cluster durante a execução. As falhas podem ser decorrentes de defeitos de hardware, falhas no sistema operacional ou na rede, por exemplo. O simulador não distingue os diferentes tipos de falha, pois, durante a simulação, quando uma falha acontece, uma determinada tarefa ou um conjunto de tarefas são abortados e o nó aonde a falha ocorreu é excluído da execução daquele processo.

A terceira avaliação (A3) realizada tem como propósito verificar a taxa de finalização de um determinado tipo de atividade para diferentes frequências de ocorrência de falhas no cluster. Essa avaliação é similar à avaliação anterior, porém é verificado o comportamento de ambas as abordagens ao aumentar a frequência de falhas no cluster.

Como o HeracleSim utiliza uma abordagem baseada em ciclos, todo o modelo do estudo precisou ser parametrizado para refletir medidas de um ambiente real. Como o ciclo é a unidade de tempo do simulador, estabelecemos que cada ciclo equivale a um minuto. Partindo desta premissa, estabelecemos os custos das atividades submetidas baseados em experimentos reais (Ogasawara et al. 2009a) executados em clusters do NACAD. Dessa forma, uma atividade ter o custo de 60 ciclos significa que suas tarefas demoram 60 minutos para serem executadas.

Durante a simulação, tipos diferentes de atividades são submetidos ao cluster. Os atributos das atividades podem variar com relação à quantidade de tarefas que a atividade gera e quanto ao custo dessas tarefas. Avaliamos atividades que geram tarefas pequenas, tarefas médias e tarefas grandes. Tarefas pequenas demoram uma hora para executar enquanto as tarefas médias demoram duas horas e meia e tarefas grandes demoram quatro horas. Ao mesmo tempo, uma atividade pode gerar uma quantidade de 256, 512, 1024 ou 2048 tarefas. Ao todo, o simulador pode submeter doze tipos diferentes de atividades. A frequência de submissão de *jobs* durante a simulação é de 170 atividades ao dia, o que dá em torno de sete atividades por hora. A distribuição de submissão de atividades não é uniforme, pois atividades mais custosas são mais raras que atividades menos custosas. Dessa forma, foi estipulada uma função de distribuição de probabilidade na submissão conforme é apresentada na Figura 18. O custo das tarefas apresentado no eixo de profundidade do gráfico é dado em minutos. Seguindo esta distribuição, por exemplo, a cada 24 submissões de atividades que geram 256 tarefas pequenas, é submetida em média uma atividade que gera 2048 tarefas grandes.

Outro fator importante da simulação é a frequência de falhas. Nas duas primeiras avaliações (A1 e A2) realizadas a frequência de falha foi de 1%. Isto significa que 14,4% dos nós de processamento podem falhar ao dia. Essa taxa de falhas é arbitrária e pode ser considerada pequena, visto que a infraestrutura de um cluster voltado à computação de alto desempenho costuma ser bem confiável para utilização. Entretanto, nenhum sistema, especialmente os sistemas de larga-escala, está livre de falhas. Na avaliação A3, o objetivo foi analisar as abordagens variando as frequências de falha de 1% a 3% em intervalos de 0,5%.

O simulador registra os dados da execução de cada atividade em um banco de dados PostgreSQL (PostgreSQL 2009) utilizando uma tabela simples onde cada tupla é uma atividade submetida ao cluster durante a simulação. Os diferentes atributos e características

registradas da atividade são registrados como atributos da tupla. A vantagem de armazenar os dados na tabela é a facilidade na realização de consultas através da linguagem SQL. As análises dos dados das avaliações A1, A2 e A3 foram feitas separadamente. Após a análise de cada avaliação, os dados eram guardados e removidos do banco de dados para dar lugar aos dados da análise seguinte.

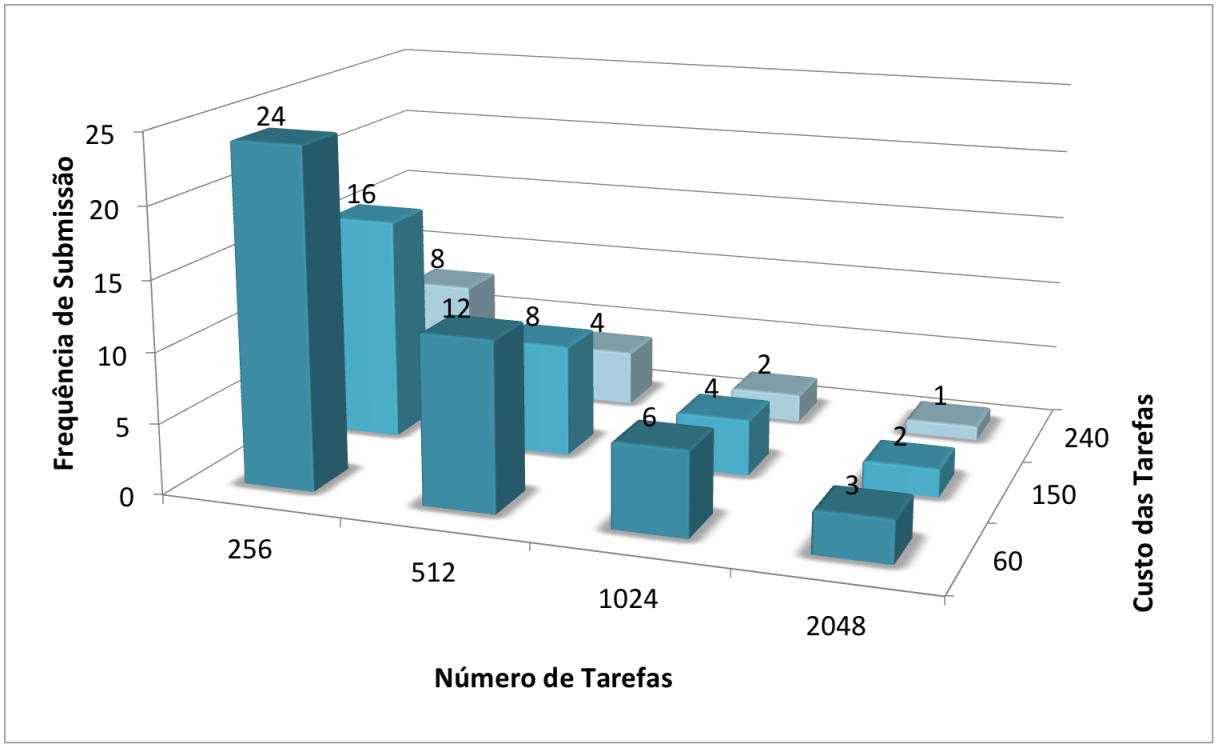

Figura 18: Distribuição de probabilidade de submissão de diferentes tipos de atividade.

6.2 Execução

A simulação executada para a avaliação A1 decorreu por quatro semanas, o que corresponde a, aproximadamente, um mês de simulação. Durante esta simulação, apenas *jobs* com processos Heracles foram submetidos. Como o intuito desta avaliação era verificar o tempo necessário para o Heracles equalizar suas métricas de forma a atender aos prazos das atividades, estipulamos prazos bem curtos para as atividades submetidas. Os prazos para executar todo o conjunto de tarefas produzidas por uma atividade variou uniformemente de uma vez o custo da atividade a duas vezes o custo. Isto significa que uma atividade com custo 150 minutos terá um prazo variando 150 a 300 minutos. Se esta atividade produzir 1024 tarefas, ela precisa de um número de processadores entre 1024 a 512, para poder atingir o

prazo, se ela utilizar uma abordagem de execução tradicional (processo simples). Porém, como o Heracles realiza a gerência dinâmica de recursos, ele pode ajustar a quantidade de processadores em tempo real. Entretanto, para prazos curtos, a escolha inicial do número de processadores pode ser fundamental.

Para a avaliação A2, foram realizadas duas execuções do simulador por quatro semanas. Uma simulação ocorreu com a submissão de *jobs* contendo processos simples, tradicionais. A outra simulação ocorreu com a submissão de *jobs* contendo processos Heracles. A separação em duas simulações foi feita para evitar qualquer tipo de interferência nos resultados de cada abordagem já que o comportamento da fila do cluster pode ser alterado dependendo do tipo processo que está executando. Nestas simulações, o prazo das atividades submetidas também foi mais folgado, variando de quatro a oito vezes o custo da atividade. Ao término da execução ficam registradas quantas atividades foram terminadas (*T*), quantas falharam (*F*) e quantas ainda estavam em execução (*E*) ao término da simulação. A taxa de atividades finalizadas é calculada como um percentual de quanto do total de tarefas terminou, i.e., *T* dividido pelo total (*T*+*F*+*E*).

Para a avaliação A3, foram aproveitados os resultados das simulações da avaliação A2, pois eles usaram uma taxa de frequência de falhas de 1%. Em seguida foram executadas novas simulações de quatro semanas para as frequências de 1,5%, 2,0%, 2,5% e 3,0%. Novamente, para não haver interferência nos resultados, as simulações com a abordagem tradicional (processos simples) ocorreram separadamente das simulações com a abordagem Heracles. A taxa de atividades finalizadas é calculada da mesma forma como na avaliação A2, porém filtrando os dados de acordo com a frequência de falha.

6.3 Análises

Na análise da avaliação A1, foram considerados apenas três tipos de atividade. Atividades (P) que geram 256 tarefas pequenas, atividades (M) que geram 512 tarefas médias e atividades (G) que geram 1024 tarefas grandes. Lembrando que uma tarefa pequena demora uma hora para ser consumida, enquanto a média demora duas horas e meia e a grande demora quatro horas. Para cada um desses tipos de atividades, calculou-se o tempo médio de execução e o tempo médio dos prazos destas atividades, agrupando pelo ciclo da simulação no qual tais atividades tiveram início. Dessa forma, é possível construir um gráfico de evolução

da simulação exibindo o tempo médio de execução gasto para processar as atividades iniciadas ao longo dos ciclos. Espera-se que, como os prazos das atividades são bem curtos, o Heracles não consiga atender os prazos inicialmente, mas consiga melhorar suas estimativas ao longo da simulação.

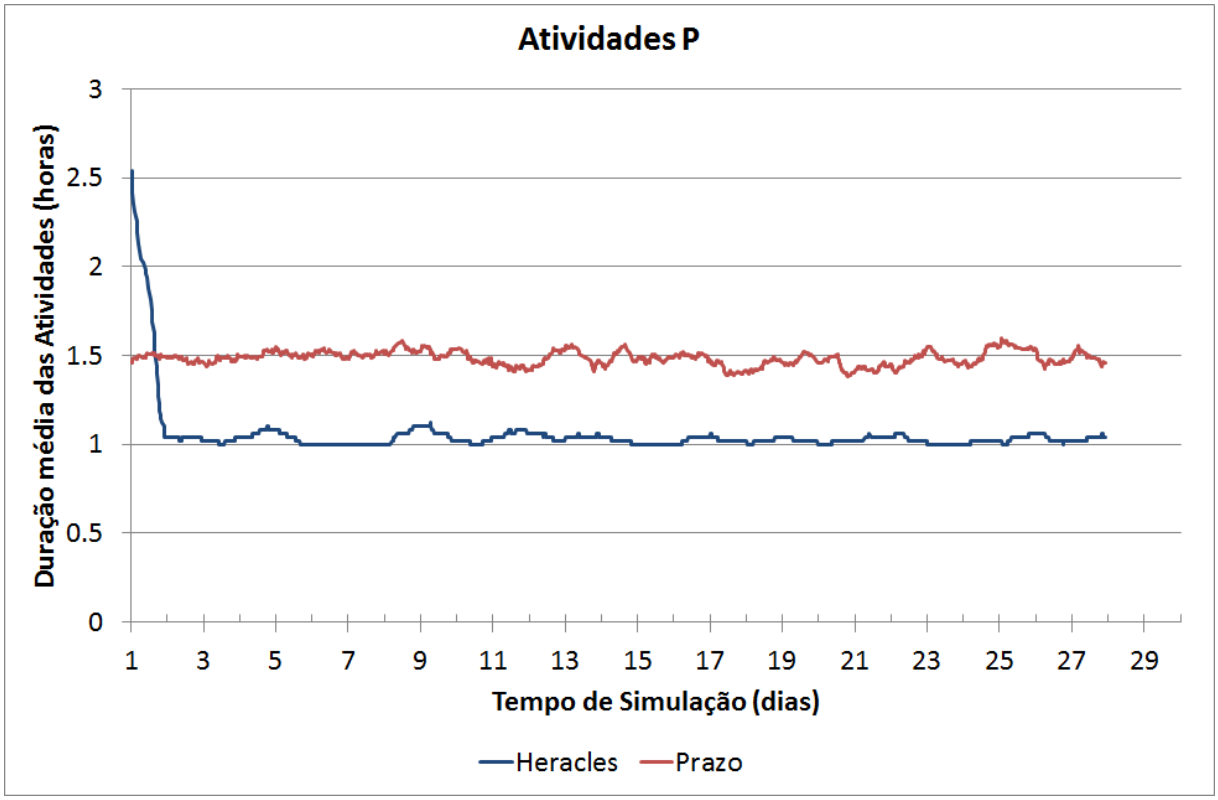

Figura 19: Evolução do tempo de execução de atividades P comparado ao prazo.

A Figura 19 apresenta o gráfico de evolução da simulação para atividades do tipo P, ou seja, que geram 256 tarefas pequenas. O decorrer da simulação é apresentado em dias, e não em ciclos, para facilitar o entendimento. É possível observar que em menos de dois dias o Heracles consegue equalizar as suas métricas e passa a atender os prazos das atividades, mesmo com prazos bem curtos. Outra observação importante é que neste caso, o Heracles estabiliza o tempo médio de execução das atividades P perto de uma hora e não próximo ao prazo médio. O Heracles faz sua estimativa inicial de processadores baseada nas atualizações da quantidade de processadores necessários para executar atividades passadas. No pior caso, o prazo de uma atividade P será de uma hora. Como este é um caso mais difícil de atender, o processo Heracles realiza mais atualizações do número de processadores durante as execuções. Com isso, ele faz a estimativa inicial de processadores tender ao prazo de pior

caso. Na prática isso não é ruim, pois atividades iguais ou similares tendem a ter prazos iguais ou similares. Portanto, o Heracles conseguirá atender aos prazos, mesmo que curtos.

Na Figura 20 é apresentado o mesmo gráfico de evolução da simulação, porém para atividades M. Neste caso é possível observar que o Heracles já consegue atender os prazos logo após completar um dia de simulação. Em um primeiro momento, não parece razoável que, em um cenário com atividades mais custosas, o Heracles equalize melhor as suas métricas. Entretanto, como a atividade M é mais longa, o Heracles consegue, durante a execução de uma única atividade, realizar mais atualizações do número de processadores. Com isso, a estimativa do número de processadores tende ao valor desejável com menos execuções da atividade. No caso das atividades P, o Heracles precisa executar diversas atividades para equalizar a estimativa inicial. Além disso, semelhante ao que aconteceu com as atividades P, o tempo médio de execuções das atividades tende a ficar próximo das duas horas e meia, pois é próximo do pior caso dos prazos das atividades M.

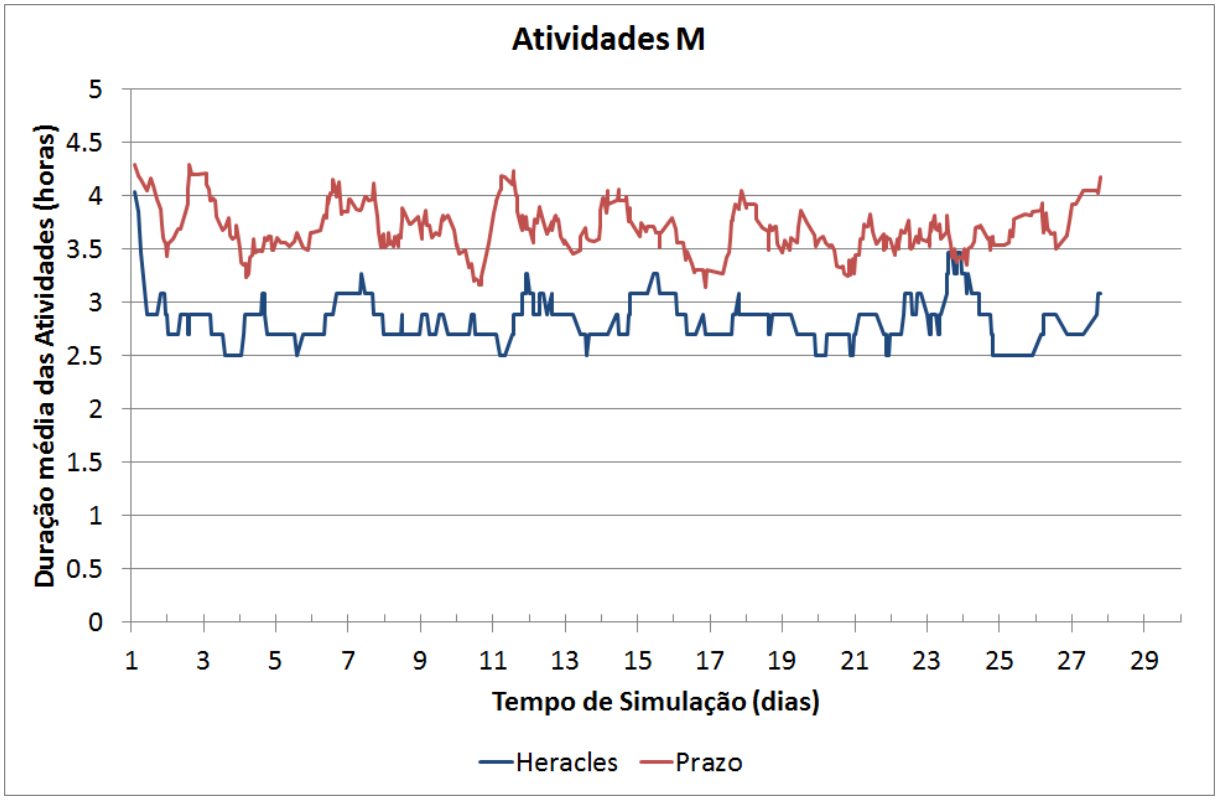

Figura 20: Evolução do tempo de execução de atividades M comparado ao prazo.

A Figura 21 apresenta o gráfico de evolução da simulação para atividades do tipo G. Nela o Heracles também demora menos de dois dias para começar a atender aos prazos.

Entretanto, observamos uma maior dificuldade do Heracles para atender prazos ao longo da simulação. Diferente do que acontece com as atividades do tipo M, o Heracles não atinge a estimativa ideal tão rapidamente. Esse resultado é esperado em função da quantidade de tarefas custosas gerada pela atividade. Embora as atividades do tipo G demorem mais para executar, dando mais oportunidades para o Heracles atualizar suas métricas, elas também são submetidas com menos frequência conforme mostra a Figura 18. Além disso, o Heracles pode sofrer com a escassez de recursos do cluster, já que ele precisa de uma grande quantidade de processadores para atender uma atividade tipo G. Porém o cluster pode não possuir a quantidade de recursos disponíveis. Ainda assim, os resultados apresentados no gráfico são positivos. A tendência da linha da execução é ficar abaixo da média do prazo, em função do prazo de pior caso. Vale ressaltar que, como as atividades do tipo G são menos frequentes, a amostragem apresentada na Figura 21 é bem menor se comparada com a amostragem da Figura 19. A amostragem reduzida afeta a qualidade dos resultados, pois pode apresentar mais desvios em torno da média.

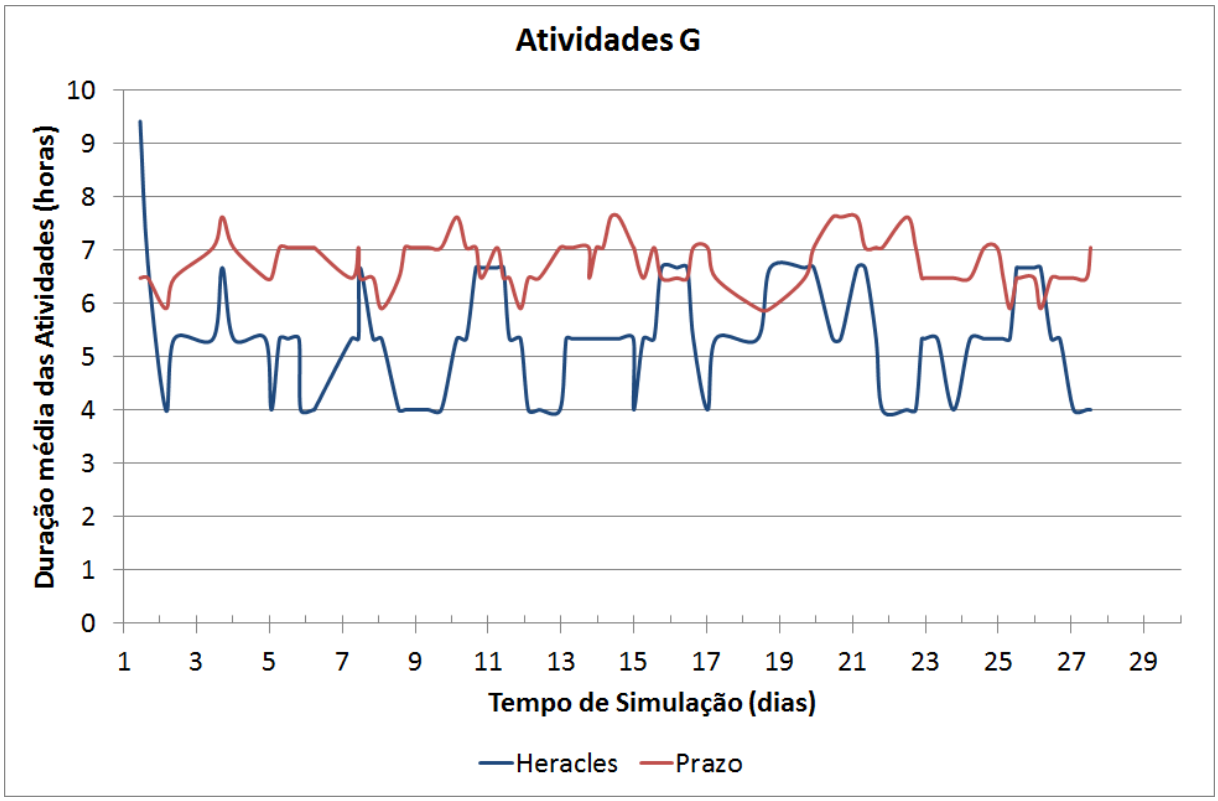

Figura 21: Evolução do tempo de execução de atividades G comparado ao prazo.

Na análise da avaliação A2, foram considerados todos os doze tipos diferentes de atividades. Os dados utilizados são as taxas de finalização de cada um desses tipos de atividades. Como esta avaliação foi realizada para os dois tipos de abordagem de execução, os resultados apresentados buscam comparar o desempenho da abordagem tradicional (processo simples) com o a abordagem Heracles. Para simplificar a visualização, os gráficos foram agrupados pelo tamanho das tarefas, i.e, tarefas pequenas, médias e grandes.

A Figura 22 apresenta os resultados para as atividades que produzem tarefas pequenas. Nele já é possível observar que com o aumento do número de tarefas por atividade, a taxa de atividades completas sofre uma queda na abordagem tradicional enquanto o impacto na abordagem Heracles é praticamente imperceptível.

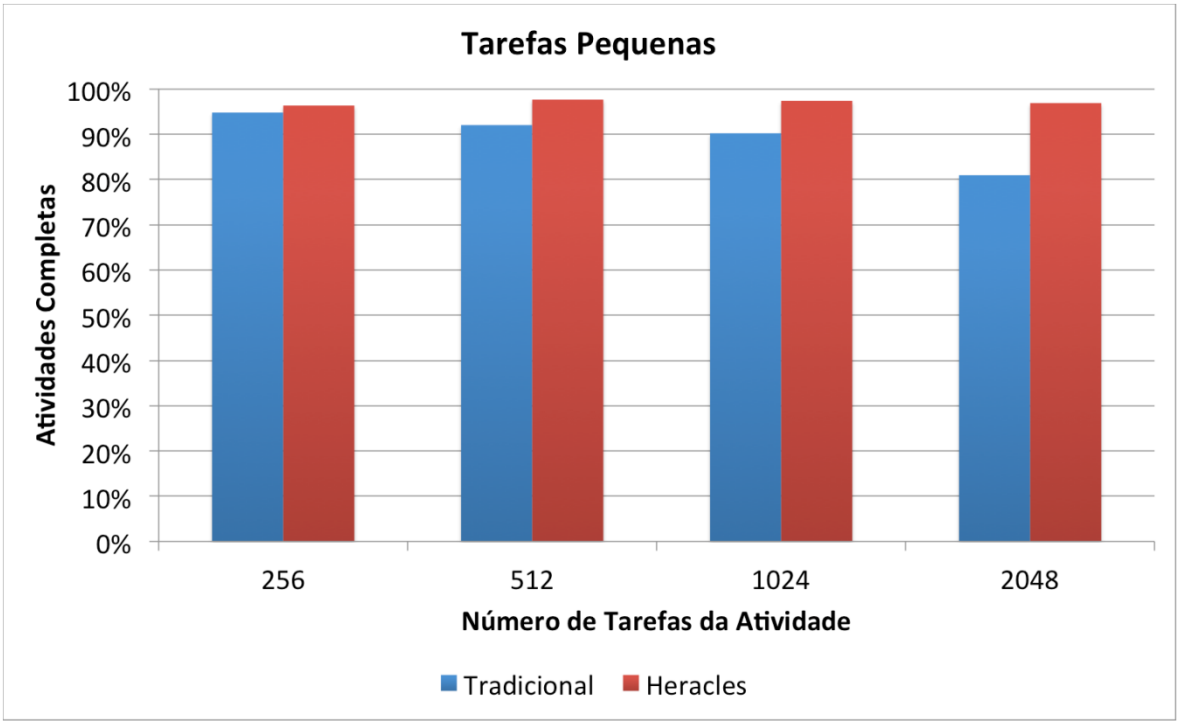

Figura 22: Taxa percentual de atividades de custo pequeno completas.

A Figura 23 mostra os resultados para atividades que produzem tarefas médias. Novamente, a abordagem Heracles mostra-se mais estável com o aumento do número de tarefas por atividade, enquanto a abordagem tradicional chega a quase 50% das tarefas submetidas completas.

A Figura 24 apresenta os resultados da avaliação A2 para atividades que produzem tarefas grandes. Neste caso, podemos observar um impacto na abordagem Heracles com o

aumento do número de tarefas por atividade. Para as atividades que produzem 2048 tarefas, a taxa de atividades completas com o Heracles ficou próximo aos 80%. Já o impacto na abordagem tradicional, é muito maior, já que a taxa de finalização de atividades ficou abaixo dos 40%. Com estes resultados, acreditamos que a capacidade do Heracles em concluir atividades, mesmo em um cenário com falhas, é muito superior ao de uma abordagem tradicional.

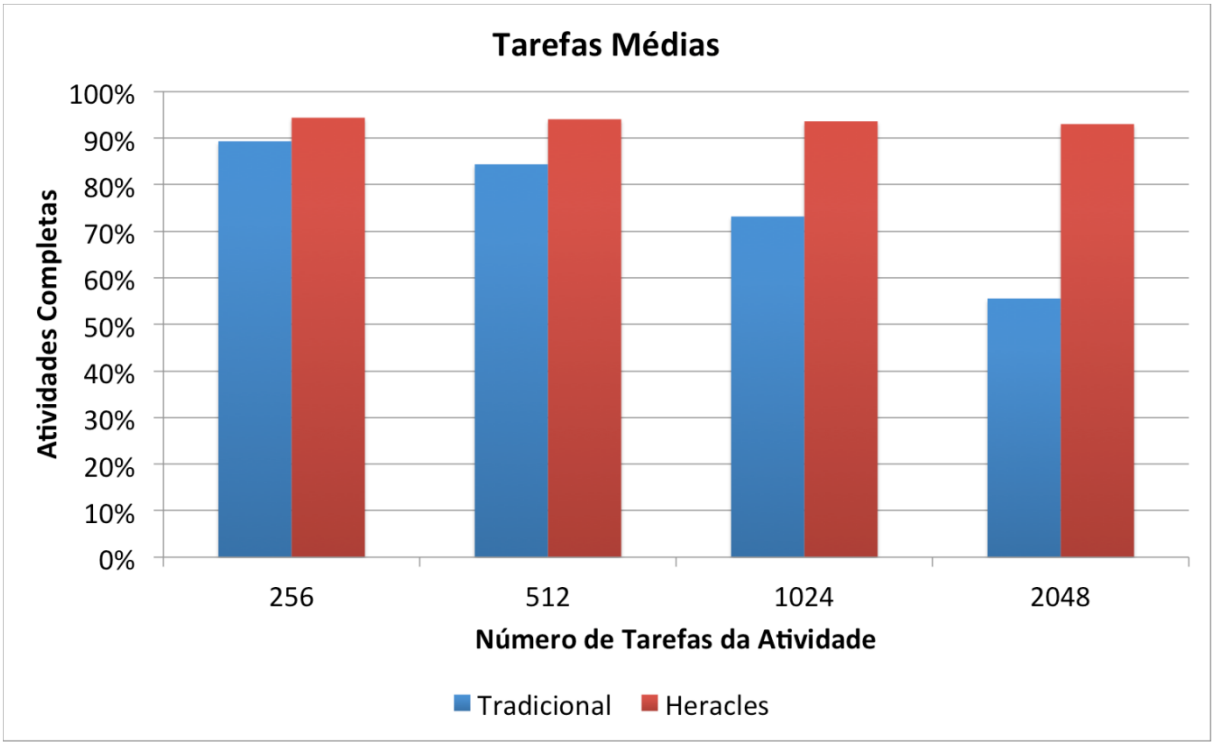

Figura 23: Taxa percentual de atividades de custo médio completas.

Para realizar a análise da avaliação A3, foi considerado apenas um tipo de atividade. O tipo escolhido foi um intermediário, onde a atividade gera 1024 tarefas médias, que demoram duas horas e meia para concluir. A taxa percentual de atividades completas foi medida nas simulações com frequências de falhas variando de 1% a 3% em intervalos de 0,5%. Novamente, o resultado da abordagem tradicional é comparado com a abordagem Heracles. Porém, ao invés do aumento do número de tarefas, como acontece nos gráficos da avaliação A2, aumenta-se a frequência de falhas.

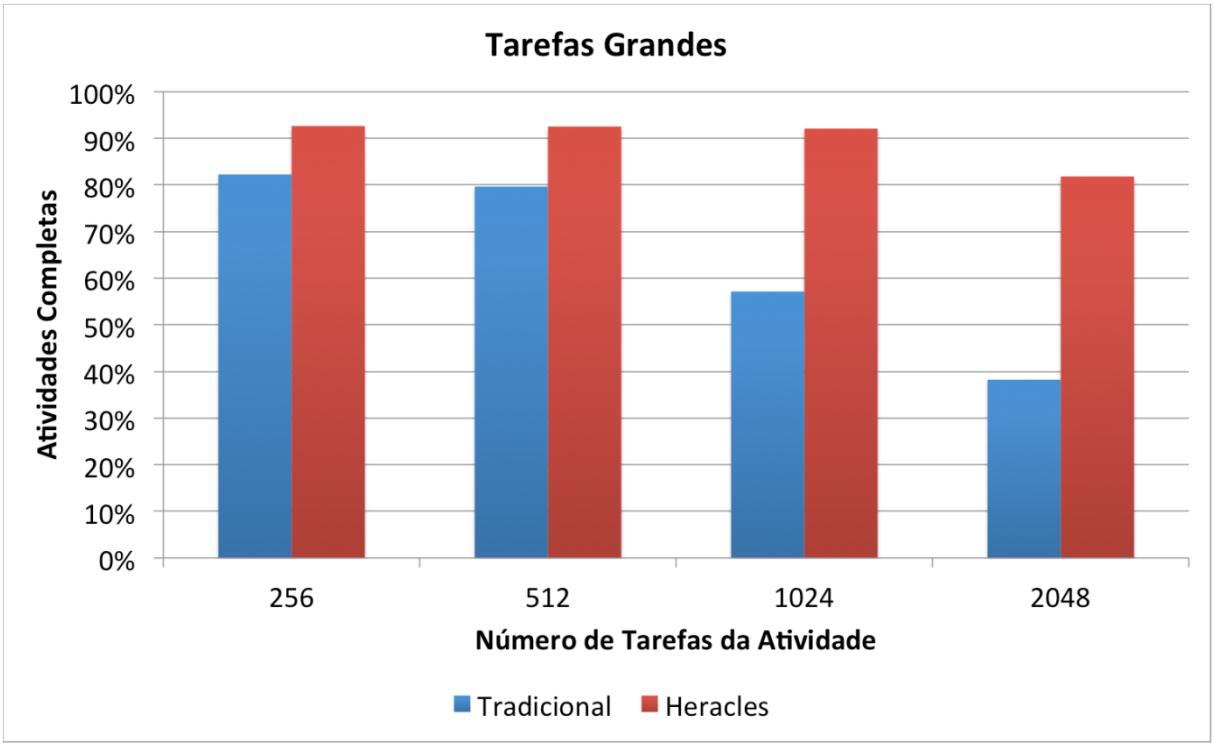

Figura 24: Taxa percentual de atividades de custo grande completas.

A Figura 25 apresenta os resultados coletados para a avaliação A3. Podemos observar que o desempenho da abordagem tradicional também cai com o aumento da frequência das falhas. A taxa percentual de finalização fica próxima aos 40% enquanto a abordagem Heracles mantém a taxa de finalização de atividades acima dos 90%. O desempenho consideravelmente maior do Heracles não se deve somente ao fato dele conseguir redistribuir as tarefas para outros nós, mas também à gerência dinâmica de recursos realizado por ele. Na hipótese onde vários processadores envolvidos na execução de uma determinada atividade falhem e abandonem a execução do processo Heracles, o mesmo poderá estimar uma nova quantidade de processadores necessários para concluir a tarefa dentro do prazo e então requisitar mais recursos ao cluster. Dessa forma, o processo de entrada e saída dos nós da rede de recursos do Heracles passa a ser algo natural. Vale lembrar que, muitas vezes, o próprio Heracles pode reduzir o tamanho da própria rede, se o prazo estipulado tender a ser atendido. Sendo assim, a falha na execução de tarefas impacta pouco na execução da atividade como um todo.

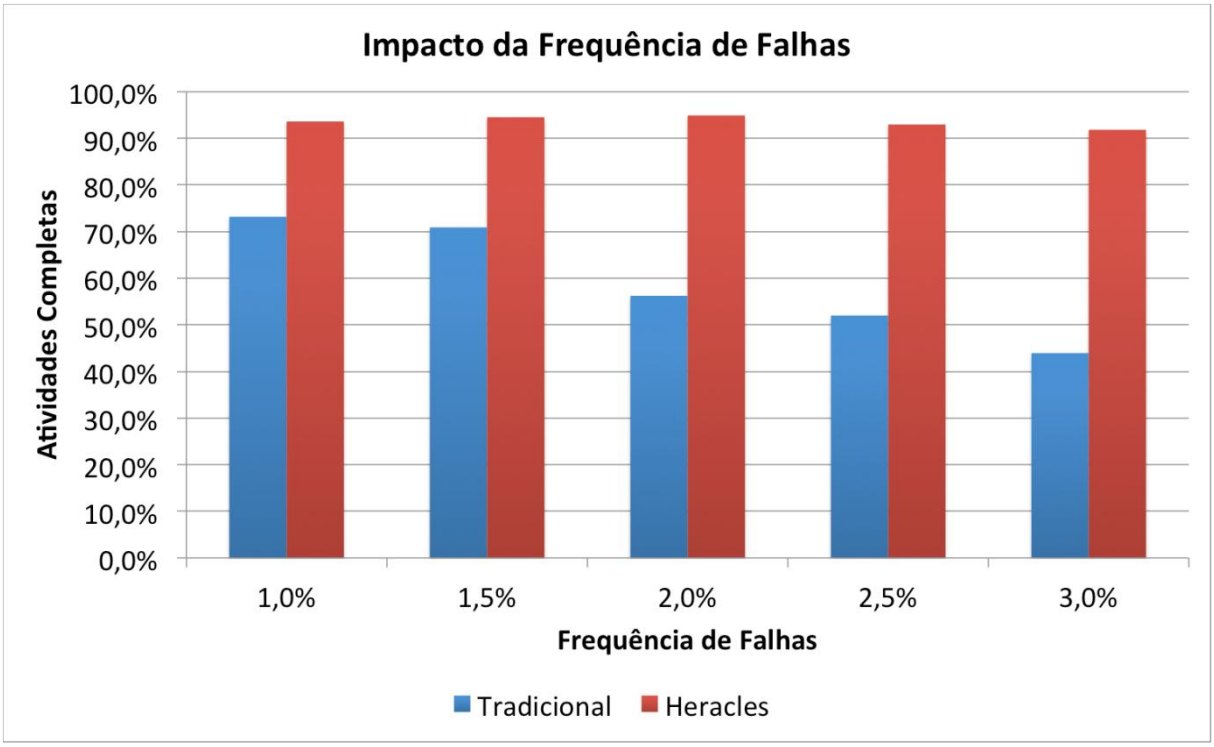

Figura 25: Análise do impacto da falha na taxa de finalização de atividades.

Capítulo 7 – Conclusões

A experimentação científica é a maior ferramenta da ciência. É através dos experimentos científicos que é possível avaliar hipóteses e testar condições à cerca de um problema. Seguindo esta linha, as simulações computacionais trouxeram novas oportunidades na ciência para avaliar cenários complexos como análises meteorológicas, bioinformática e exploração de petróleo. Como estas simulações são modelos de problemas envolvendo análises encadeadas e sistemáticas, a utilização de workflows científicos para representar e gerenciar tais experimentos é frequente e desejável. Os SGWfC existentes oferecem diferentes formas de modelagem e execução de workflows com foco em diferentes funcionalidades, como ferramentas de modelagem gráfica e componentes para execução distribuída.

Dentre as fases do ciclo de vida de um experimento científico, a fase de execução pode envolver o processamento de atividades de workflow computacionalmente custosas repetidas vezes. O alto custo computacional destes workflows reforça a necessidade de paralelização destas atividades em clusters computacionais de alto desempenho. Tais clusters, atualmente, possuem um grande conjunto de nós de processamento, que estão propensos a falhas. Além disso, a configuração necessária para obter a quantidade necessária de recursos do cluster para atender a um determinado prazo pode não ser trivial para alguns cientistas. Com o objetivo de melhorar o escalonamento e execução de tarefas de workflows científicos, técnicas P2P podem auxiliar com mecanismos de tolerância a falhas e gerência dinâmica de recursos.

Nesta dissertação, apresentamos a abordagem Heracles que utiliza técnicas P2P tais como tabelas distribuídas, topologias P2P hierárquicas e recuperação de falhas de maneira transparente para o usuário. O Heracles busca lidar com a execução de muitas tarefas de workflows científicos em recursos distribuídos utilizando técnicas P2P. Seu objetivo é aprimorar o paralelismo de dados em workflows que demandem varredura de parâmetros e fragmentação de dados. O Heracles pode ser incorporado em escalonadores de workflow, tais como o Hydra ou o Falkon. O Heracles assume o controle da execução de tarefas e coleta os dados de proveniência para retorná-los ao escalonador, que deverá encaminhá-los ao SGWfC.

O Heracles busca prover transparência na execução de workflows, já que ele exige apenas que os cientistas informem o prazo da execução. A partir deste prazo, o Heracles

decide a quantidade inicial de processadores que precisa requisitar. Avaliando métricas durante a execução, como eficiência parcial da execução e quantidade de tarefas processadas por unidade do tempo, o Heracles pode avaliar se é necessário expandir a quantidade de recursos computacionais envolvidos no processamento ou se pode reduzir o número de nós reservados. O Heracles organiza os processadores reservados para execução em uma estrutura hierárquica P2P, que visa tornar o mecanismo de gerência dinâmica de recursos mais eficiente e facilitar o processo de recuperação de falhas. Como consequência, o mecanismo hierárquico também provoca um melhor balanceamento de carga, pois distribui operações de controle entre alguns dos nós de processamento.

Para verificar se a abordagem Heracles é capaz de atender aos requisitos de transparência, que incluem a gerência dinâmica de recursos e tolerância a falhas, foi necessário realizar alguns estudos. A avaliação de uma solução como o Heracles envolve a utilização de recursos computacionais de grande porte. É muito custoso utilizar um cluster de maneira exclusiva para realizar estudos de viabilidade de uma solução. Sendo assim, optou-se por uma abordagem de simulação capaz de verificar o funcionamento da proposta Heracles em um cenário de submissão de atividades de workflows em recursos computacionais distribuídos. Para tal, foi elaborado o HeracleSim, como uma extensão do PeerSim para submissão de atividades de workflow em clusters utilizando a abordagem Heracles. O simulador foi calibrado baseado em dados reais de experimentos realizados em clusters do NACAD da UFRJ.

Para os estudos de simulação, foi considerado um cluster de 7168 processadores, que corresponde ao número de núcleos de processamento do atual maior cluster do NACAD. Foi simulado um período equivalente a quatro semanas com uma frequência de submissão de 170 atividades por dia. As características dessas atividades poderiam variar com relação ao número de tarefas que a atividade gerava e ao custo destas tarefas. Foi assumido que o cluster simulado sofre falhas, ao processar tarefas, com uma frequência de 1%. Isto significa que 14,4 execuções de tarefas no cluster podem falhar ao dia. Neste cenário, foi avaliada a solução tradicional de execução de tarefas de atividades de workflow e a abordagem Heracles. O objetivo do estudo foi avaliar o comportamento do Heracles ao gerenciar os recursos dinamicamente e reescalonar tarefas ao ocorrer uma falha na execução.

Para avaliar a gerência dinâmica de recursos do Heracles, realizou-se um primeiro estudo de simulação para verificar o tempo gasto pelo Heracles para equalizar suas métricas o suficiente para atender aos prazos estipulados das atividades. O Heracles estabelece o número de processadores iniciais para execução de atividades baseados em dados de proveniência de atividades passadas. Para avaliar a capacidade do Heracles em estabelecer uma quantidade ideal de processadores para iniciar uma atividade de forma a atender um prazo, foram submetidas diferentes atividades com prazos bem curtos. Os resultados mostraram que o Heracles, mesmo em cenários com prazos curtos, consegue atingir um valor ideal de processadores em menos de dois dias para iniciar a execução de uma atividade e cumprir o prazo estipulado.

Um segundo estudo de simulação buscou medir a taxa de finalização de tarefas da abordagem de execução tradicional de tarefas comparada com a abordagem Heracles. Os resultados mostraram que o impacto da falha na abordagem de execução tradicional é grande já que o percentual de atividades completas pode chegar a somente 38% das atividades submetidas no caso de atividades que geram muitas tarefas grandes. Ao utilizar o Heracles, no mesmo cenário, a taxa de finalização de atividades fica acima dos 80%. As atividades que não terminaram, no caso do Heracles, foram as que estavam em execução ao término da simulação. O Heracles mostrou-se superior na taxa de finalização de atividades para todos os tipos de atividades, quando comparado com a abordagem tradicional.

O último estudo foi realizado para avaliar o comportamento da abordagem tradicional e do Heracles em cenários com diferentes frequências de falhas no cluster. Neste estudo foram consideradas apenas as atividades que geravam 1024 tarefas onde cada tarefa demorava cerca de duas horas e meia para executar. A frequência de falhas variou de 1% a 3% em intervalos de 0,5%. Nos resultados a taxa de finalização de atividades no Heracles manteve-se acima dos 90% em todos os casos. Já na abordagem tradicional, a taxa de finalização chegou aos 43,9%, no pior caso. Com estes resultados, o Heracles mostrou-se capaz de se recuperar de falhas com eficiência, mesmo com o aumento da frequência destas falhas. Isto ocorre não só por que o Heracles é capaz de redistribuir tarefas interrompidas, mas também por ser capaz obter mais recursos dinamicamente, caso parte dos processadores reservados falhe durante a execução.

Os resultados dos estudos realizados foram positivos, de forma que o Heracles mostrou-se promissor como abordagem de escalonamento e execução de atividades de workflows científicos com paralelismo de dados. O Heracles pode, futuramente, ser incorporado ao Hydra com o intuito de proporcionar mecanismos de tolerância a falhas e gerência dinâmica de recursos. Dessa forma, a execução paralela de experimentos científicos com o Hydra fica mais transparente para o cientista, que deixa de se preocupar com questões técnicas da infraestrutura. Esforços manuais de verificação de quais tarefas executaram e consequente ressubmissão de atividades deixa de ser um problema recorrente na execução de experimentos científicos.

Como trabalhos futuros, o Heracles pode experimentar novas técnicas para a tolerância a falhas e gerência dinâmica de recursos. Uma opção para aprimorar a tolerância a falhas é explorar execuções redundantes (Dean e Ghemawat 2008) e mecanismos de *checkpoint* (Plank e Thomason 2001). Melhorias na gerência dinâmica de recursos podem avaliar princípios de localidade (Dick et al. 2009) dos dados levando em consideração a movimentação de dados (*data staging)* para os novos recursos a serem alocados. No modelo de execução, estratégias alternativas como *work stealing* (Pezzi et al. 2007) podem ser interessantes de ser adotadas no Heracles.

O Heracles, ao ser incorporado a um escalonador de workflows, poderá beneficiar inúmeras pesquisas que executem grandes simulações que demandem paralelismo de dados científicos. O Heracles proporciona melhor usabilidade na submissão de atividades em clusters, pois exige que os cientistas configurem apenas o prazo para aquela atividade ser executada. Assim, Heracles torna a execução de atividades de workflows científicos com paralelismo de dados mais transparente. Além disso, a gerência dinâmica de recursos do Heracles pode proporcionar uma utilização mais coerente dos recursos do cluster já que atividades com prazos menos rígidos podem consumir menos recursos enquanto atividades com prazos apertados consumam mais. Dessa forma, os recursos computacionais de um cluster tendem a ser mais bem aproveitados.

Referências Bibliográficas

Acharya, S., Franklin, M., Zdonik, S., (1997), "Balancing push and pull for data broadcast". In: *ACM SIGMOD Record*, p. 183–194, New York, NY, USA.

Akbarinia, R., Pacitti, E., Valduriez, P., (2007), "Data currency in replicated DHTs". In: *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, p. 211-222, Beijing, China.

Almeida, E. C. D., Sunyé, G., Traon, Y. L., Valduriez, P., (2008), "A Framework for Testing Peer-to-Peer Systems". In: *Proceedings of the 2008 19th International Symposium on Software Reliability Engineering*, p. 167-176, Los Alamitos, CA, USA.

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

Ames, M., Barreto, M., Navaux, P., (2006), "A P2P-based model for high-performance grid computing.". *Congresso Internacional de Telemática y Telecomunicaciones*, p. 10-20, Havana.

Barbosa, V. C., (1996), *An Introduction to Distributed Algorithms*. The MIT Press.

Blythe, J., Jain, S., Deelman, E., Gil, Y., Vahi, K., Mandal, A., Kennedy, K., (2005), "Task scheduling strategies for workflow-based applications in grids". In: *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium onCluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, p. 759-767 Vol. 2

Bosch, J., (2000), *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Addison-Wesley Professional.

Bradley, D., Sfiligoi, I., Padhi, S., Frey, J., Tannenbaum, T., (2010), "Scalability and interoperability within glideinWMS", *Journal of Physics: Conference Series*, v. 219, n. 6, p. 062036.

Briquet, C., Dalem, X., Jodogne, S., Marneffe, P. D., (2007), "Scheduling data-intensive bags of tasks in P2P grids with bittorrent-enabled data distribution". In: *Proceedings of the second workshop on Use of P2P, GRID and agents for the development of content*

networks, p. 39-48, Monterey, California, USA.

Brown, D. A., Brady, P. R., Dietz, A., Cao, J., Johnson, B., McNabb, J., (2007), "A Case Study on the Use of Workflow Technologies for Scientific Analysis: Gravitational Wave Data Analysis", *Workflows for e-Science*, Springer, p. 39-59.

Buchmann, E., Bohm, K., (2004), "How to run experiments with large peer-to-peer data structures". In: *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th InternationalParallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, p. 27

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

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

Cera, M., Pezzi, G., Mathias, E., Maillard, N., Navaux, P., (2006), "Improving the dynamic creation of processes in MPI-2", *Lecture Notes in Computer Science*, v. 4192/2006, p. 247-255.

Chamberlin, D., (2003), "XQuery: a query language for XML". In: *Proceedings of the 2003 ACM SIGMOD international conference on Management of dataACM SIGMOD international conference on Management of data*, San Diego, California, USA.

Chan, P., Abramson, D., (2008), "A Programming Framework for Incremental Data Distribution in Iterative Applications". In: *Proceedings of the 2008 IEEE International Symposium on Parallel and Distributed Processing with Applications*, p. 244-251

Chandra, R., (2001), *Parallel programming in OpenMP*. Morgan Kaufmann.

Cirne, W., Brasileiro, F., Costa, L., Paranhos, D., Santos-Neto, E., Andrade, N., Rose, C. D., Ferreto, T., Mowbray, M., et al., (2004), "Scheduling in Bag-of-Task Grids: The PAUÁ Case". In: *Proceedings of the 16th Symposium on Computer Architecture and High Performance Computing*, p. 124-131

Coutinho, F., Ogasawara, E., Oliveira, D., Braganholo, V., Lima, A. A. B., Dávila, A. M. R.,

Mattoso, M., (2010), "Data Parallelism in Bioinformatics Workflows Using Hydra". In: *ECMLS 2010*, Chicago, Illinois, United States.

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

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

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

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

Dias, J., Aveleda, A., (2010), "HPC Environment Management: New Challenges in the Petaflop Era". In: *9th International Meeting on High Performance Computing for Computational Science*, Berkeley, CA, USA.

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

Dias, J., Ogasawara, E., Oliveira, D., Pacitti, E., Mattoso, M., (2010b), "Improving Many-Task Computing in Scientific Workflows Using P2P Techniques". In: *MTAGS 2010*, p. 31-40, New Orleans, LA, USA.

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

Dick, M. E., Pacitti, E., Kemme, B., (2009), "Flower-CDN: a hybrid P2P overlay for efficient query processing in CDN". In: *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, p. 427-438, Saint Petersburg, Russia.

Dun, N., Taura, K., Yonezawa, A., (2010), "Easy and Instantaneous Processing for Data-Intensive Workflows". In: *3rd IEEE Workshop on Many-Task Computing on Grids*

and SupercomputersSuperComputing 2010, New Orleans, LA, USA.

eMule, (2010). eMule Project. Disponível em: http://www.emule-project.net/. Acesso em: 5 Mar 2010.

Falkon Project, (2010). Falkon Project. Disponível em: http://dev.globus.org/wiki/Incubator/Falkon#Instructions. Acesso em: 27 Dez 2010.

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

Ganesan, P., Gummadi, K., Garcia-Molina, H., (2004), "Canon in G Major: Designing DHTs with Hierarchical Structure". In: *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, p. 263-272, Tokyo, Japan.

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

Garcés-Erice, L., Biersack, E. W., Felber, P. A., Ross, K. W., Urvoy-Keller, G., (2003), "Hierarchical Peer-to-peer Systems". In: *Proceedings of ACM/IFIP International Conference on Parallel and Distributed Computing*, p. 643-657, Klagenfurt, Austria.

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

Gropp, W. D., (2001), "Learning from the Success of MPI", *cs/0109017* (Set.)

Ioannidis, Y. E., Wong, E., (1987), "Query optimization by simulated annealing". In: *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, p. 9-22, San Francisco, California, United States.

Jelasity, M., Montresor, A., Jesi, G. P., Voulgaris, S., (2010), *The PeerSim simulator*, http://peersim.sourceforge.net.

Jun Yan, Yun Yang, Raikundalia, G., (2006), "SwinDeW-a p2p-based decentralized workflow management system", *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, v. 36, n. 5, p. 922-935.

Juve, G., Deelman, E., Vahi, K., Mehta, G., (2010), "Experiences with resource provisioning for scientific workflows using Corral", *Sci. Program.*, v. 18, n. 2, p. 77-92.

Kazaa, (2011). Kazaa Media Desktop. Disponível em: http://www.kazaa.com/. Acesso em: 12 Jan 2011.

Liang, J., Kumar, R., Ross, K. W., (2006), "The FastTrack overlay: A measurement study", *Computer Networks*, v. 50, n. 6 (Abr.), p. 842-858.

Lua, E. K., Crowcroft, J., Pias, M., Sharma, R., Lim, S., (2005), "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes", *IEEE Communications Surveys and Tutorials*, v. 7, p. 72-93.

Martinez-Yelmo, I., Cuevas, R., Guerrero, C., Mauthe, A., (2008), "Routing Performance in a Hierarchical DHT-based Overlay Network". In: *Parallel, Distributed, and Network-Based Processing, Euromicro Conference on*, p. 508-515, Los Alamitos, CA, USA.

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

Meshkova, E., Riihijärvi, J., Petrova, M., Mähönen, P., (2008), "A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks", *Comput. Netw.*, v. 52, n. 11, p. 2097-2128.

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

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

Ogasawara, E., Dias, J., Oliveira, D., Rodrigues, C., Pivotto, C., Antas, R., Braganholo, V., Valduriez, P., Mattoso, M., (2010), "A P2P Approach to Many Tasks Computing for Scientific Workflows". In: *9th International Meeting on High Performance Computing for Computational Science*, Berkeley, CA, USA.

Ogasawara, E., Oliveira, D., Chirigati, F., Barbosa, C. E., Elias, R., Braganholo, V., Coutinho,

A., Mattoso, M., (2009a), "Exploring many task computing in scientific workflows". In: *MTAGS 09*, p. 1-10, Portland, Oregon.

Ogasawara, E., Paulino, C., Murta, L., Werner, C., Mattoso, M., (2009b), "Experiment Line: Software Reuse in Scientific Workflows". In: *Scientific and Statistical Database Management*, p. 264–272, New Orleans, LA.

Oliner, A. J., Sahoo, R. K., Moreira, J. E., Gupta, M., (2005), "Performance Implications of Periodic Checkpointing on Large-Scale Cluster Systems". In: *Parallel and Distributed Processing Symposium, International*, p. 299b, Los Alamitos, CA, USA.

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

Oram, A., (2001), *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, Inc.

Ozsu, M. T., Valduriez, P., (1999), *Principles of Distributed Database Systems*. 2 ed. Prentice Hall.

Pacitti, E., Valduriez, P., Mattoso, M., (2007), "Grid Data Management: Open Problems and New Issues", *Journal of Grid Computing*, v. 5, n. 3, p. 273-281.

Pezzi, G. P., Cera, M. C., Mathias, E., Maillard, N., Navaux, P. O. A., (2007), "On-line Scheduling of MPI-2 Programs with Hierarchical Work Stealing". In: *Computer Architecture and High Performance Computing, Symposium on*, p. 247-254, Los Alamitos, CA, USA.

Plank, J. S., Thomason, M. G., (2001), "Processor Allocation and Checkpoint Interval Selection in Cluster Computing Systems", *Journal of Parallel and Distributed Computing*, v. 61, n. 11 (Nov.), p. 1570-1590.

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

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

Raicu, I., Zhao, Y., Dumitrescu, C., Foster, I., Wilde, M., (2007), "Falkon: a Fast and Light-weight tasK executiON framework". In: *SC07*, p. 1-12, Reno, Nevada.

Ranjan, R., Rahman, M., Buyya, R., (2008), "A Decentralized and Cooperative Workflow Scheduling Algorithm". In: *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium onCluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on*, p. 1-8

Rodrigues, R., Druschel, P., (2010), "Peer-to-peer systems", *Communications of the ACM*, v. 53, n. 10, p. 72.

Romano, P., Bartocci, E., Bertolini, G., De Paoli, F., Marra, D., Mauri, G., Merelli, E., Milanesi, L., (2007), "Biowep: a workflow enactment portal for bioinformatics applications", *BMC Bioinformatics*, v. 8, n. Suppl 1, p. S19.

Rowstron, A., Druschel, P., (2001), "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems", *Middleware 2001*, , chapter 2218, Berlin, Heidelberg: Springer Berlin Heidelberg

Silva, F. A., Carvalho, S., Hruschka, E. R., (2004), "A Scheduling Algorithm for Running Bag-of-Tasks Data Mining Applications on the Grid", *Euro-Par 2004 Parallel Processing*, , p. 254-262.

Smanchat, S., Indrawan, M., Ling, S., Enticott, C., Abramson, D., (2009), "Scheduling Multiple Parameter Sweep Workflow Instances on the Grid". In: *Fifth IEEE International Conference on e-Science, 2009e-Science, 2009. e-Science '09. Fifth IEEE International Conference on*, p. 300-306

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

Spooner, D. P., Cao, J., Jarvis, S. A., He, L., Nudd, G. R., (2005), "Performance-Aware Workflow Management for Grid Computing", *Comput. J.*, v. 48, n. 3, p. 347-357.

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

Stevens, R., Zhao, J., Goble, C., (2007), "Using provenance to manage knowledge of In Silico experiments", *Brief Bioinform* (Maio.), p. bbm015.

Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., Balakrishnan, H., (2003), "Chord: a scalable peer-to-peer lookup protocol for internet applications", *IEEE/ACM Trans. Netw.*, v. 11, n. 1, p. 17-32.

Stout, L., Murphy, M. A., Goasguen, S., (2009), "Kestrel: an XMPP-based framework for many task computing applications". In: *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, p. 1-6, Portland, Oregon.

Team, C., (2005), *DAGMan: A Directed Acyclic Graph Manager, July 2005*.

TOP500, (2010), *TOP500 Supercomputing Sites*, http://www.top500.org/.

Travassos, G. H., Barros, M. O., (2003), "Contributions of In Virtuo and In Silico Experiments for the Future of Empirical Studies in Software Engineering". In: *Proc. of 2nd Workshop on Empirical Software Engineering the Future of Empirical Studies in Software Engineering, Roma*, p. 117-130

Valduriez, P., Pacitti, E., (2005), "Data Management in Large-Scale P2P Systems", *High Performance Computing for Computational Science - VECPAR 2004, ,* p. 104-118.

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

Welch, V., Siebenlist, F., Foster, I., Bresnahan, J., Czajkowski, K., Gawor, J., Kesselman, C., Meder, S., Pearlman, L., et al., (2003). Security for Grid Services. *ArXiv Computer Science e-prints*. Disponível em: http://adsabs.harvard.edu/abs/2003cs.....6129W. Acesso em: 27 Dez 2010.

Wu, D., Tian, Y., Ng, K., Datta, A., (2008), "Stochastic analysis of the interplay between object maintenance and churn", *Computer Communications*, v. 31, n. 2 (Fev.), p. 220-239.

Xie, C., Chen, G., Vandenberg, A., Pan, Y., (2008), "Analysis of hybrid P2P overlay network topology", *Comput. Commun.*, v. 31, n. 2, p. 190-200.

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

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

Zhao, B. Y., Kubiatowicz, J., Joseph, A. D., (2001), "Tapestry: An infrastructure for fault-tolerant wide-area location and routing". , University of California at Berkeley Berkeley, CA, USA.

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